ECE 271, Design Project, Group 4

Nicholas Kim, Julian Brinkley

December 3rd, 2021

Contents

1 Project Description

Inputs: This design reads a PS/2 keyboard. The PS/2 keyboard device is powered by a MAX-10 FPGA host using wires for a 5V source and ground. The PS/2 keyboard communicates with the host using the clock and data lines.

Outputs: This design outputs a frequency onto a speaker. The speaker is connected to a source with a digital frequency and a ground line from the host.

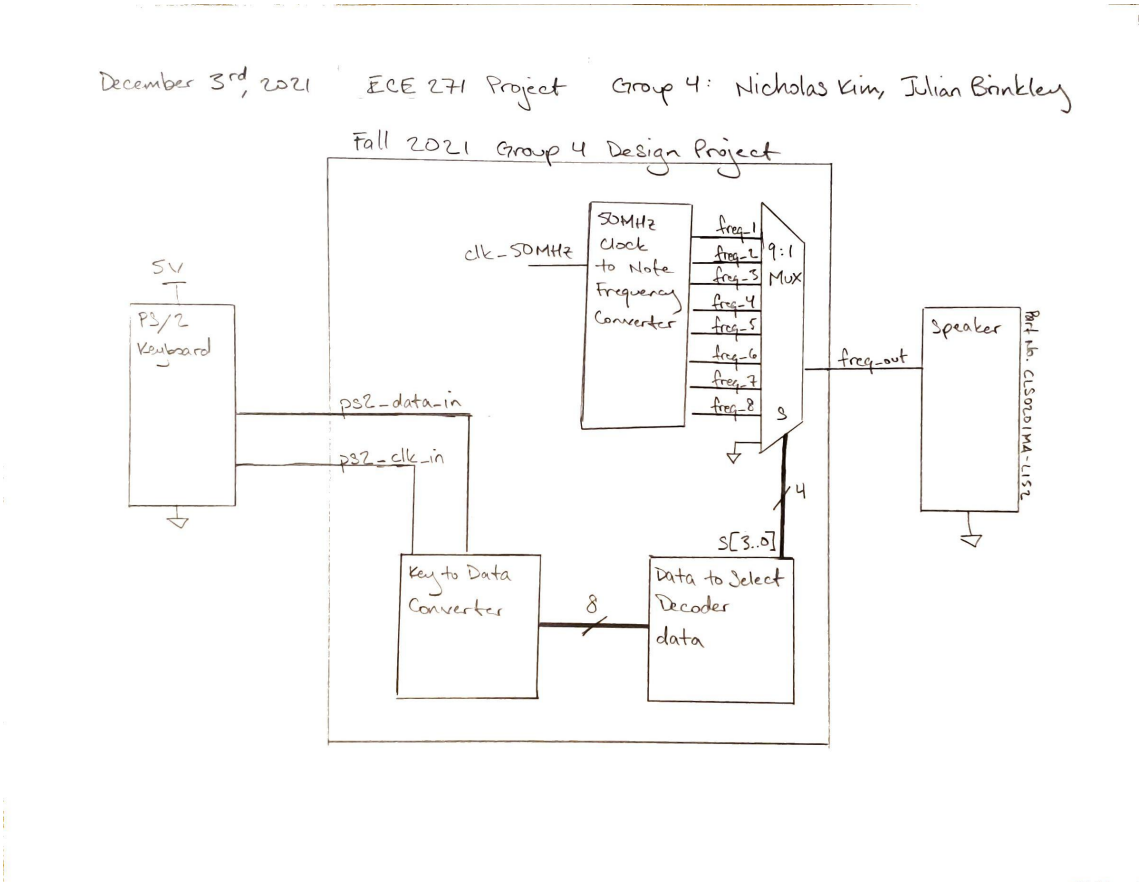

Figure 1: This is an overview of the piano project design.

The hardware modules used include a PS/2 keyboard and a speaker. The hardware diagram in figure 2 shows the pin connections between the FPGA and the hardware modules. Table 1 is an interface definition for each of the inputs and outputs on the FPGA.

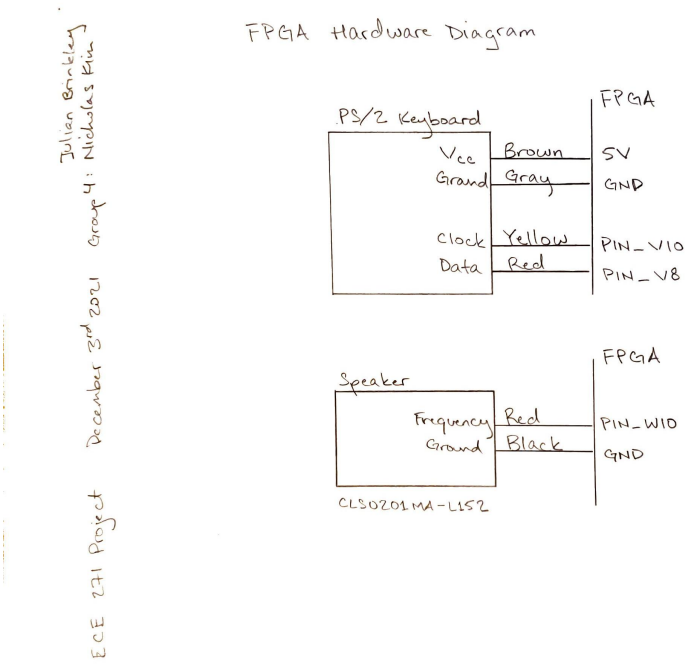

Figure 2: The hardware diagram shows which pins are used on the FPGA to supply power to and communicate with the PS/2 keyboard and the speaker. The speaker is powered by a digital frequency which also produces the sound.

Name	**I/O**	**Function**	**FPGA Pin(s)**
clk_50MHz	In	50MHz signal is divided into the desired note frequencies for the piano	P11 (Clock)
reset	In	Resets PS/2 interpreter and all frequency converters	C10 (Switch)
enable	In	Enables PS/2 interpreter and all frequency converters	C11 (Switch)
ps2_clk_in	In	Clock signal from the PS/2 keyboard which allows data to be interpreted	V10 (GPIO)
ps2_data_in	In	Data signal from the PS/2 keyboard which sends the key data in accordance with the PS/2 clock	V8 (GPIO)
freq_out	Out	Digital note frequency that is sent to the speaker to play	W10 (GPIO)
pulse_count[4..0]	Out	Troubleshooting: Counts the number of 11-bit data signals sent by the keyboard	F18, F21, B20, C18, C14, C10 (Hex 0's)
data[9..1]	Out	Troubleshooting: 8 data bits and odd parity bit sent by the keyboard and combined into a bus by the PS/2 interpreter	A8, A11, D14, E14, C13, D13, B10, A10, A9 (LED's)
s[3..0]	Out	Troubleshooting: Decoded select signal from the PS/2 decoder which selects the note frequency on the 9:1 Mux	D22, A19, A16, D15 (Hex DP's)

Table 1: Interface Definition for the FPGA inputs and outputs

2 High Level Description

Inputs: This reads a clock and data signal from a PS/2 keyboard. It also reads the internal 50MHz clock and two input switches for reset and enable.

Outputs: A digital note frequency is the main output for the design. Additionally, there are troubleshooting outputs for the pulse count, key data, and select signal, though these outputs are not shown in top level diagram.

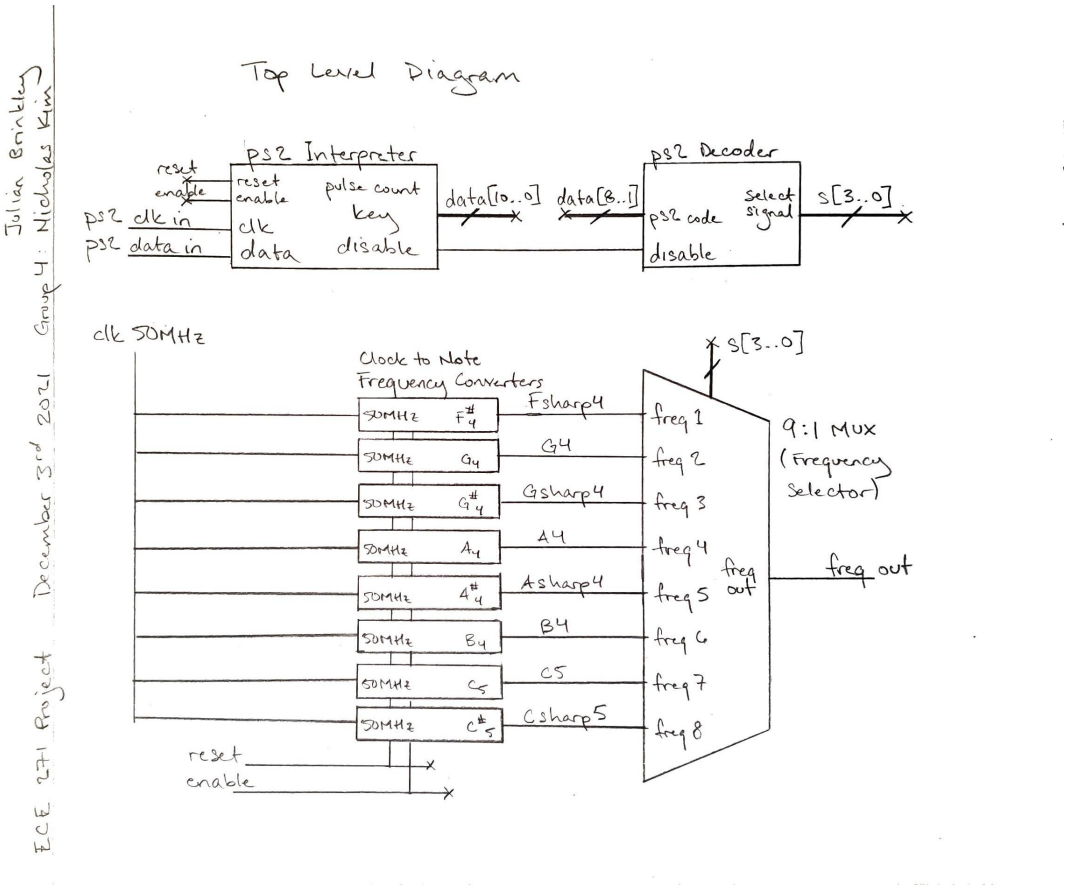

Figure 3: This is the top level diagram for the piano project. For simplicity, the troubleshooting outputs were excluded from this diagram.

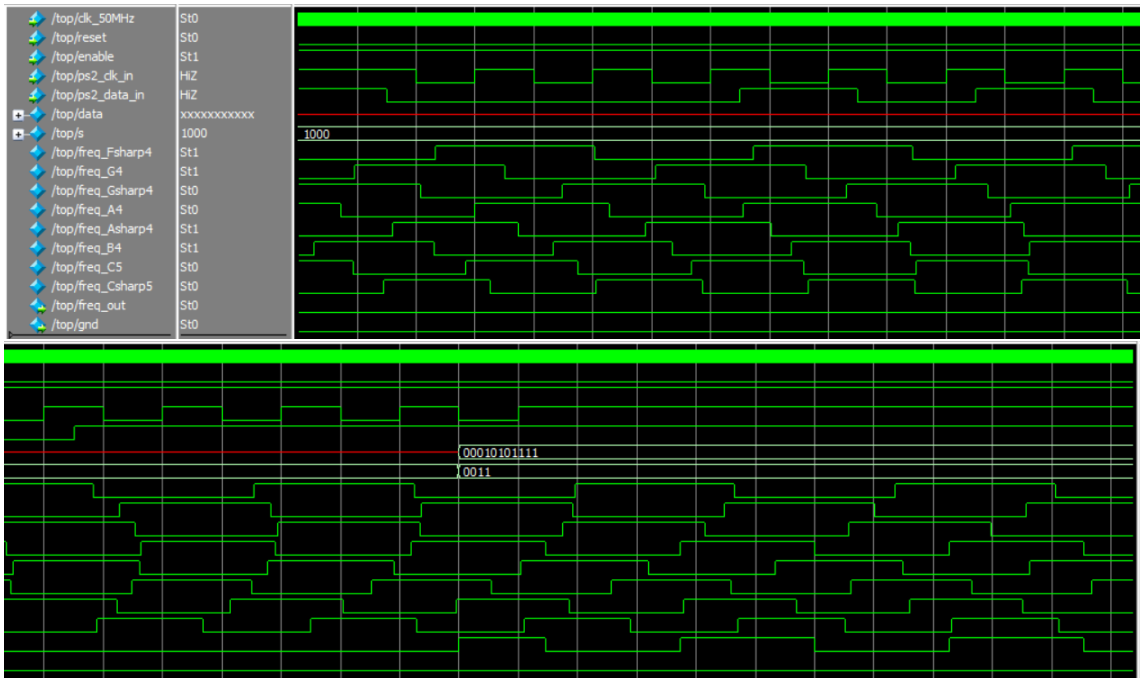

Figure 4: Top level simulation results from ModelSim

The simulation above demonstrates that the piano driver functions as intended. The clock and data signals are forced to timing specific values to simulate 11 bits of data sent on 11 clock pulses from the PS/2 keyboard. The PS/2 input signals were simulated in accordance with the PS/2 keyboard protocol.[1]

Following the 11 clock pulses, the PS/2 interpreter correctly outputs the 11 data bits as an 11-bit bus. The PS/2 decoder correctly decodes the 8 key related data bits into a 4-bit select signal. Information on the PS/2 decoder and select signal can be found in table 2.

The lower wave plots show the note frequencies as derived from the 50MHz clock input. The output frequency remains off until the select signal switches to 0011, representing the fourth input on the 9:1 multiplexer. The fourth input frequency is A4, and the output frequency matches that A4 frequency correctly.

2.1 PS/2 Interpreter

Inputs: This reads two inputs: clk and data. These are the two signals sent by the keyboard. The data is a string of 11 bits representing the pressed key, and the clock signal oscillates from high to low 11 times while each piece of the data is sent.

Outputs: An 11-bit string, representing the pressed key. The 11 bits are the same 11 bits sent through the data line, but stored as a bus. There is a disable output, which goes high only when the previous code observed is equal to "F0", which is half of the code for a key release.[2] This is used later by the decoder.

[1] http://www.burtonsys.com/ps2_chapweske.htm
[2] https://techdocs.altium.com/display/FPGA/PS2+Keyboard+Scan+Codes

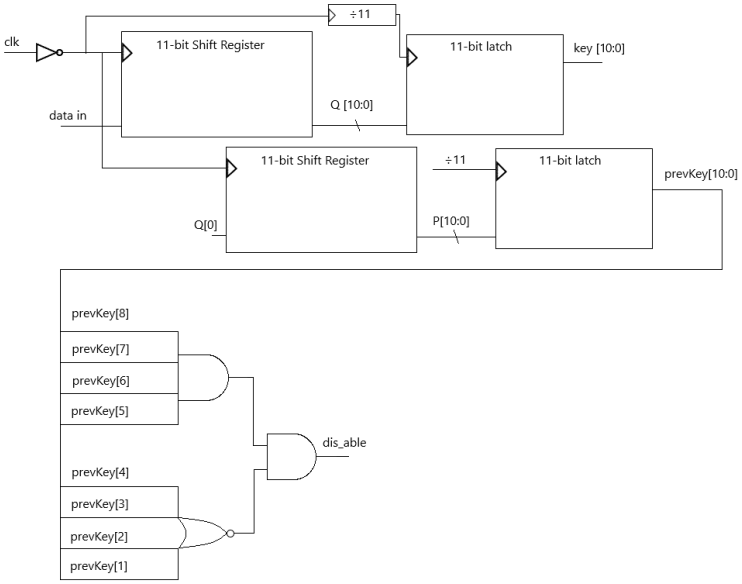

Figure 5: This is an overview of the PS/2 interpreter block that reads the data stream from the keyboard and releases it as 11-bit busses.

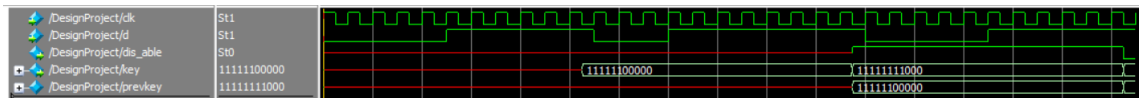

Figure 6: Interpreter simulation results from ModelSim

2.1.1 Shift Register 11-bit

Inputs: Data is a single bit that changes between different values at each clock cycle, and clk is the clock signal coming from the keyboard.

Outputs: 11 bits of a q bus, equal to the last 11 values that "data" had.

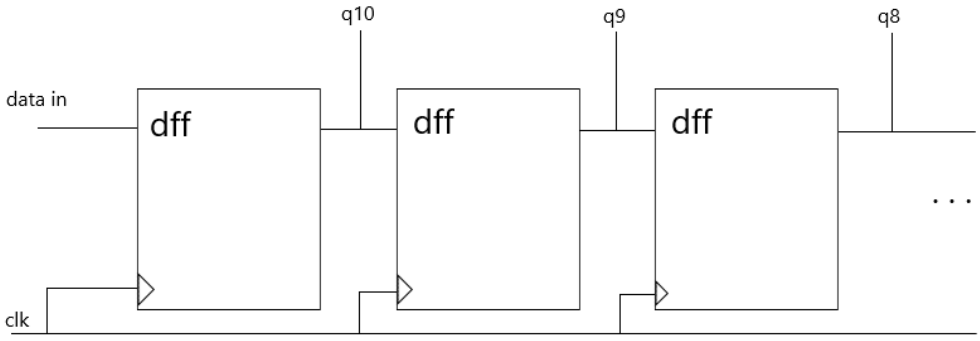

Figure 7: This is an overview of the 11-bit shift register. Chaining D Flip-Flops is a standard design for a shift register.

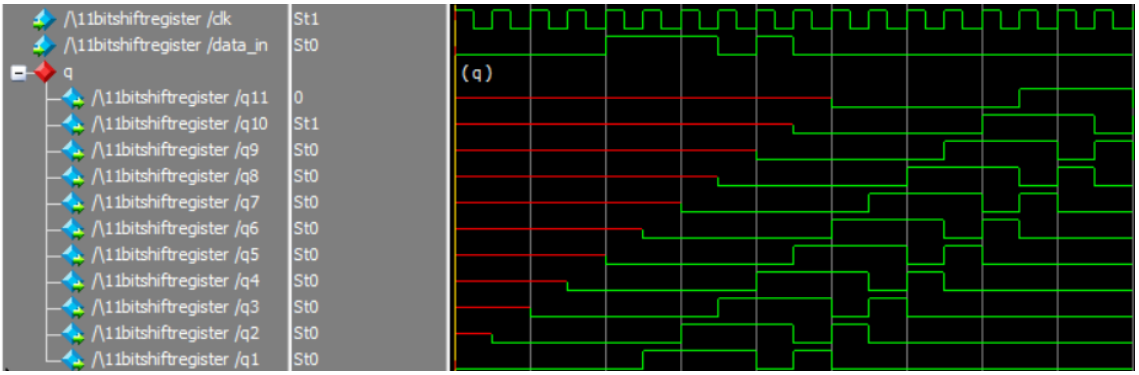

Figure 8: Register simulation results from ModelSim. The first bit of the data input reaches the first bit (q1) of the data output after one clock cycle, then the second bit (q2) after two cycles. It steps down the output ladder on each cycle. The next bit of the input follows, one cycle behind. The rest of the data continues during following cycles.

2.1.2 Latch 11-bit

Inputs: Each q input represents a different bit of the pressed key's code. Clk is the clock signal that causes the latches to release the signals in the q inputs. (Where this module is used, clk is the keyboard's clock divided by 11.)

Outputs: An 11 bit bus, containing all 11 q inputs. Output is only updated on the rising edge of clk.

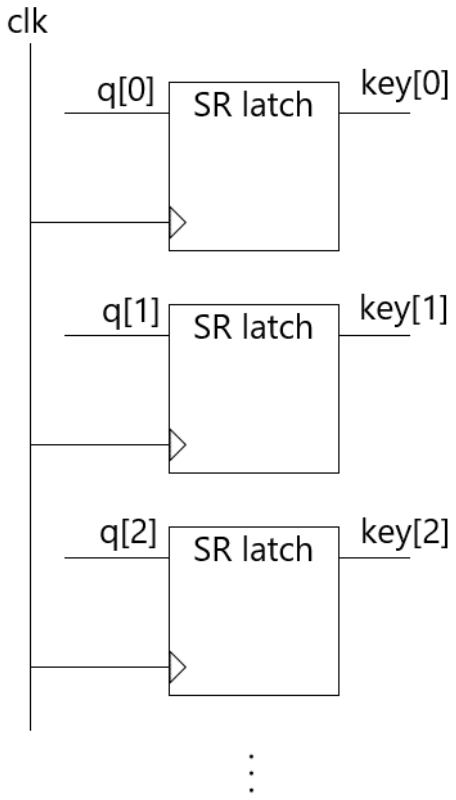

Figure 9: This is an overview of the 11-bit latch. This module simplifies having 11 separate SR latches by compressing them into one module.

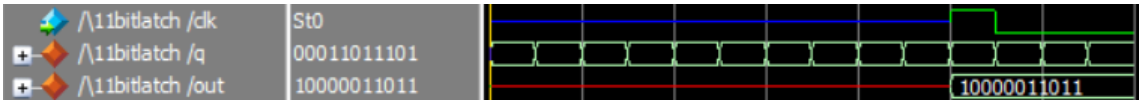

Figure 10: Latch simulation results from ModelSim

2.1.3 Clock Divider

Inputs: A clock signal. For the place it is used, it takes in the clock signal that the keyboard sends.

Outputs: A signal "clk_divided_11" that pulses HIGH every 11 clock cycles of the input clock. The "count" bus output shows how many times the divided clock signal has pulsed and is used for testing purposes.

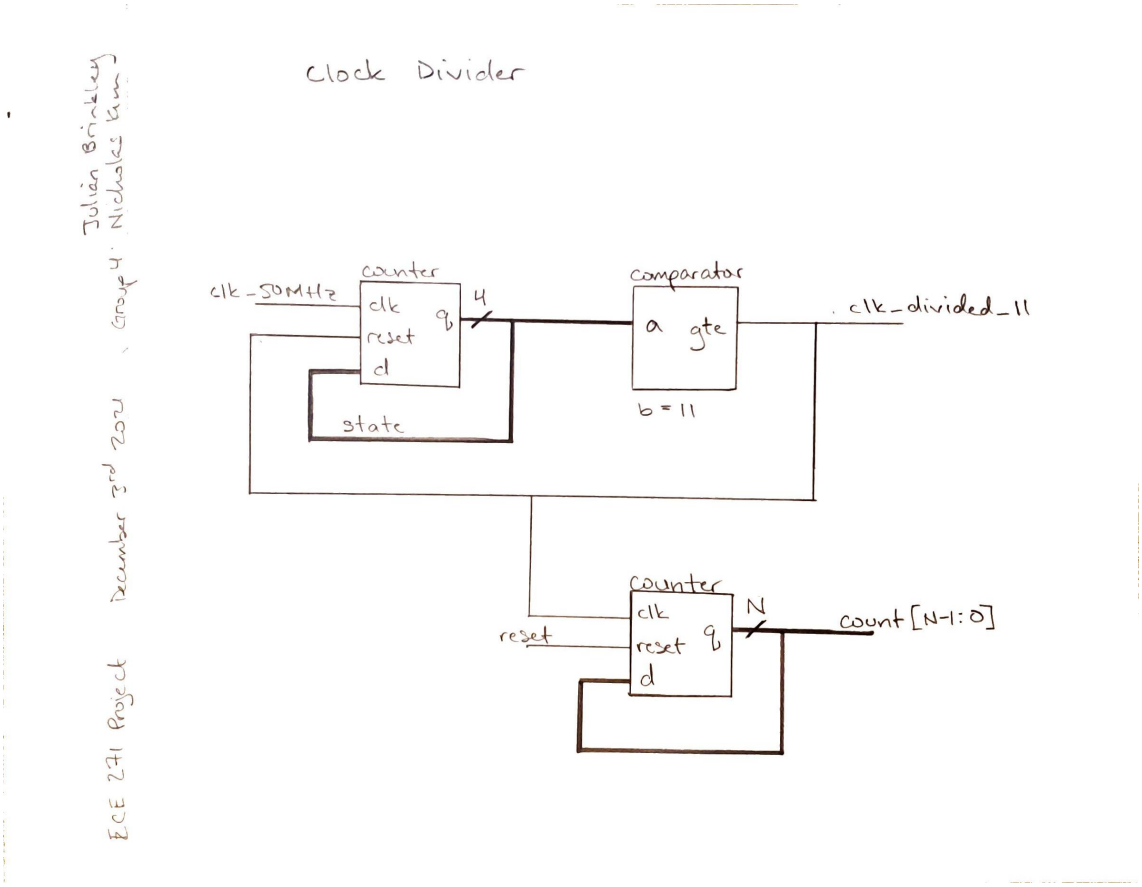

Figure 11: This is an expanded view of the clock divider block shown in the PS/2 Interpreter block diagram. The counter and comparator blocks are shown in figures 13 and 15.

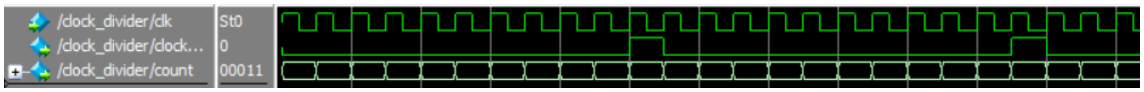

Figure 12: Clock divider simulation results from ModelSim

2.1.4 Counter

Inputs: This individual takes a clock, reset, enable, and multi-bit "d" value representing the previous state of the counter. The clock signal counts up the counter on the rising edge. The "d" value is incremented by 1 on the rising edge.

Output: A multi-bit "q" output is a value that is 1 higher than the "d" input and represents the current state of the counter.

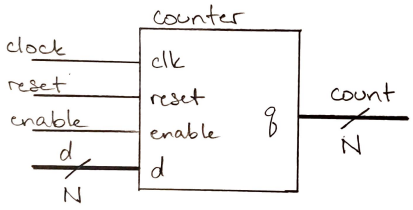

Figure 13: The counter block was written in System Verilog for Lab 5. The counter is also used in the Clock to Note Frequency converter shown in figure 23.

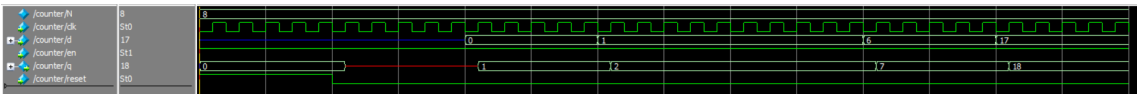

Figure 14: Counter simulation results from ModelSim

2.1.5 Comparator

Input: A multi-bit value "a" is taken in and compared to a parameter value "b".

Output: A logic value "gte" represents whether the comparison "a is greater than or equal to b" is true (HIGH) or false (LOW).

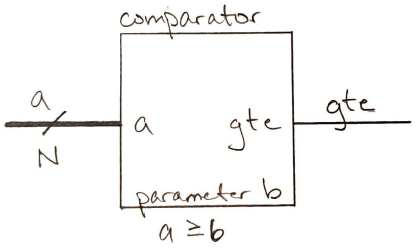

Figure 15: The comparator block was written in System Verilog for Lab 5. The comparator is also used in the Clock to Note Frequency converter shown in figure 23.

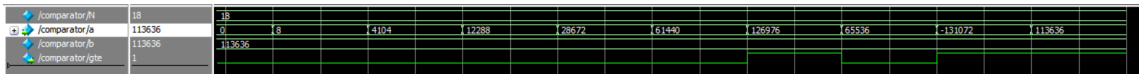

Figure 16: Comparator simulation results from ModelSim

2.2 PS/2 Data to Select Decoder

Inputs: This reads an 8 bit value – the 8 bits out of the 11 bit bus that identify each key. It also reads a "disable" selector, which is active when the previous key read by the interpreter was "F0."

Outputs: This outputs a shorter code that represents one of the 8 keys used for the piano. Each key used in the application gets a different code. When a key that is unused by the piano application has been pressed, or the "disable" input is high, the output is 1000.

The simulation results can be seen in figure 18, and the specific outputs are outlined in this table (Note: DC = don't care). Any key outside of the first 8 in the table will provide an output of 1000.

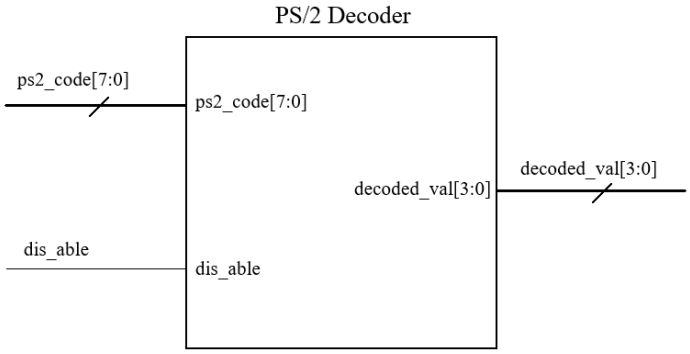

Figure 17: The PS/2 decoder module, written in System Verilog.

Key	PS/2 Code	Disable	Output
A	00011100	0	0000
S	00011011	0	0001
D	00100011	0	0010
F	00101011	0	0011
J	00111011	0	0100
K	01000010	0	0101
L	01001011	0	0110
;	01001100	0	0111
DC	DC	1	1000
T	00101100	DC	1000
U	00111100	DC	1000

Table 2: PS/2 Key Data inputs and decoded select signal outputs

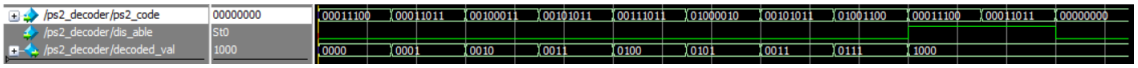

Figure 18: PS/2 Decoder simulation results from ModelSim. The first 8 values for ps2code in the simulation are the values for the 8 keys on the keyboard, so the block outputs select values 0000 to 0111. The next two tests have a high disable value, so the output is 1000. The last test is an unknown key value, so even with a low disable output, the result is still 0000.

2.3 9:1 Multiplexer (Frequency Selector)

Inputs: This functional unit takes 8 input signals that represent digital note frequencies. The 9th input is grounded internally to represent an OFF signal. A 4-bit select bus input determines which of the 9 inputs pass through to the output.

Output: One of the nine input signals goes to the output depending on the value of the select bus. This output signal represents the selected digital frequency that will be played by the speaker.

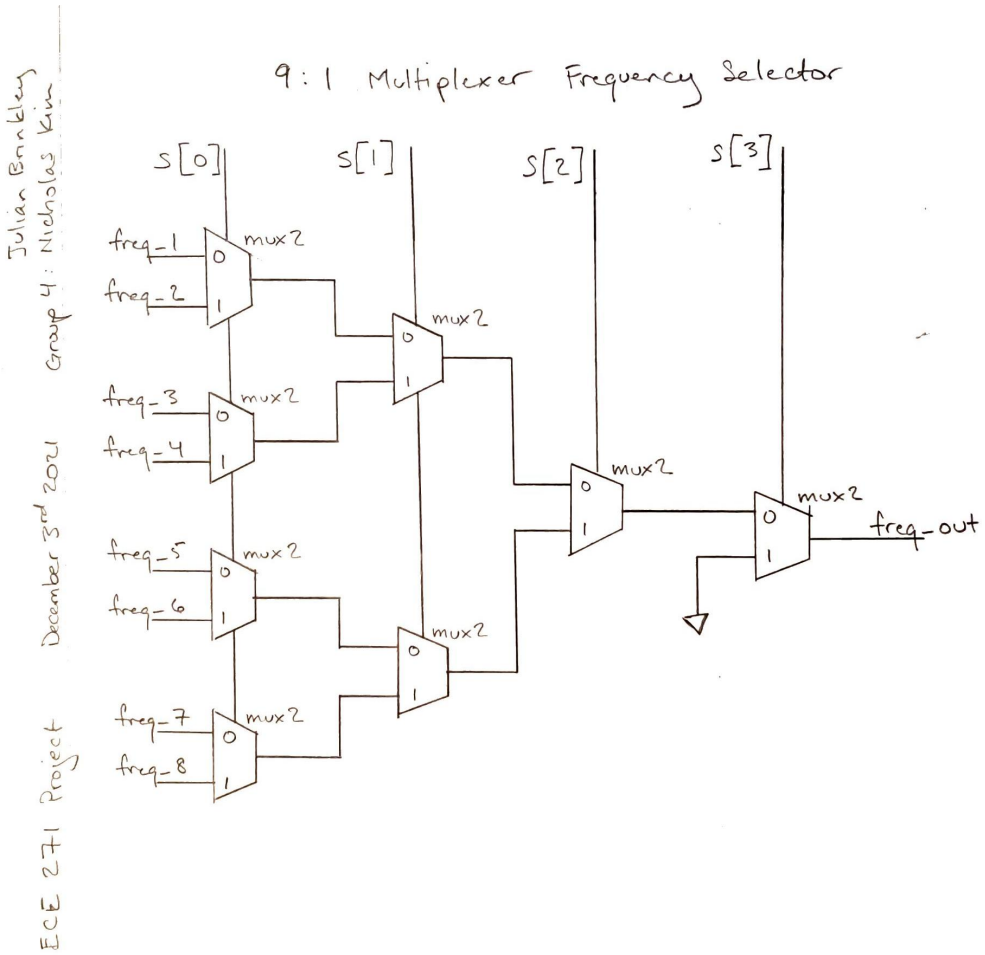

Figure 19: This is an expanded view of the 9:1 multiplexer block shown in the top level diagram.

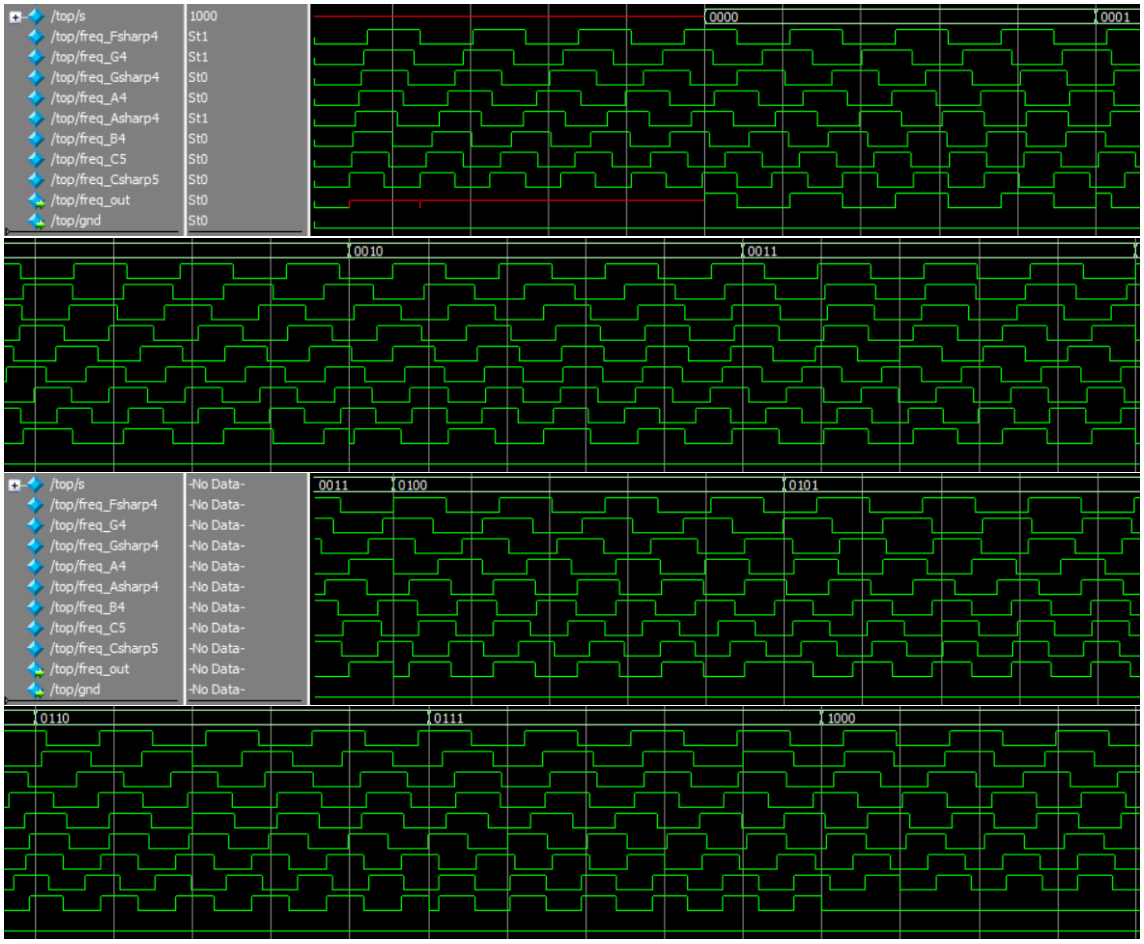

Figure 20: 9:1 Multiplexer (Frequency Selector) simulation results from ModelSim

The note frequencies used in the design were included in this simulation to demonstrate that the output frequency matches the correct note frequency in accordance with the select signal.

2.3.1 2:1 Multiplexer

Inputs: This individual block takes one 1-bit select signal and two 1-bit inputs to select from.

Output: The select signal determines which of the two 1-bit inputs is directed to the 1-bit output.

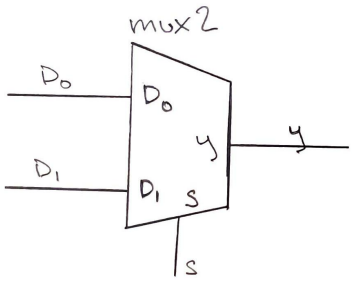

Figure 21: This block was written in System Verilog using the conditional operator.

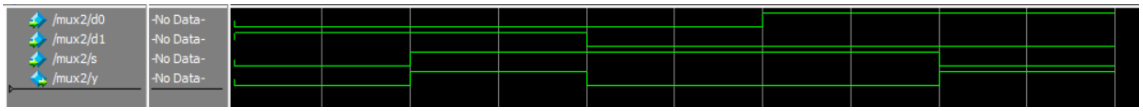

Figure 22: 2:1 Multiplexer simulation results from ModelSim

2.4 50MHz Clock to Note Frequency Converter

Input: This functional unit takes a 50MHz clock signal input.

Output: The 50MHz clock signal input is deliberately divided by two comparators to produce a wave pattern that rises and falls at specific times to match the frequency of any given note.

Parameters: This design includes 8 instances of this module template with varying parameter values on the comparator blocks based on the desired note frequency output. The parameter for the half wave comparator instance describes how many 50MHz clock cycles must pass before rising from LOW to HIGH, representing half of the note frequency wave length. The parameter for the full wave comparator instance describes how many 50MHz clock cycles pass in one cycle of the note frequency.

The full wave parameter b_{full} is calculated by dividing the frequency of the input clock by the desired output frequency f_{note} in Hz. The half wave parameter b_{half} is found by dividing the full wave parameter by 2.

$$b_{full} = \frac{50MHz}{f_{note}}$$

$$b_{half} = \frac{50MHz}{f_{note}} * \frac{1}{2}$$

Table 3 shows the note frequencies of the 8 notes that are implemented in the design and the calculated parameters for the 8 instances of the frequency converter.

In the wave plot simulations in figures 24 and 25, the 50MHz clock was forced LOW at 0ns, HIGH at 10ns, and repeating this pattern every 20ns to accurately represent the timing of a 50MHz clock. The period of a 50MHz clock is $\frac{1}{50MHz} = 20 * 10^{-9}$ seconds (20ns).

Note	Frequency (Hz)	Full Wave parameter	Half wave parameter	Period (ns)
$F_4^{\#}$	369.99	135139	67569	2702776
G_4	392.00	127551	63776	2551020
$G_4^{\#}$	415.30	120395	60197	2407898
A_4	440.00	113636	56818	2272727
$A_4^{\#}$	466.16	107259	53630	2145186
B_4	493.88	101239	50620	2024783
C_5	523.25	95557	47778	1911132
$C_5^{\#}$	554.37	90192	45096	1803849

Table 3: This table shows the note frequencies and instance parameters used in the frequency converter. The period of each note is included for simulation purposes in figures 24 and 25.

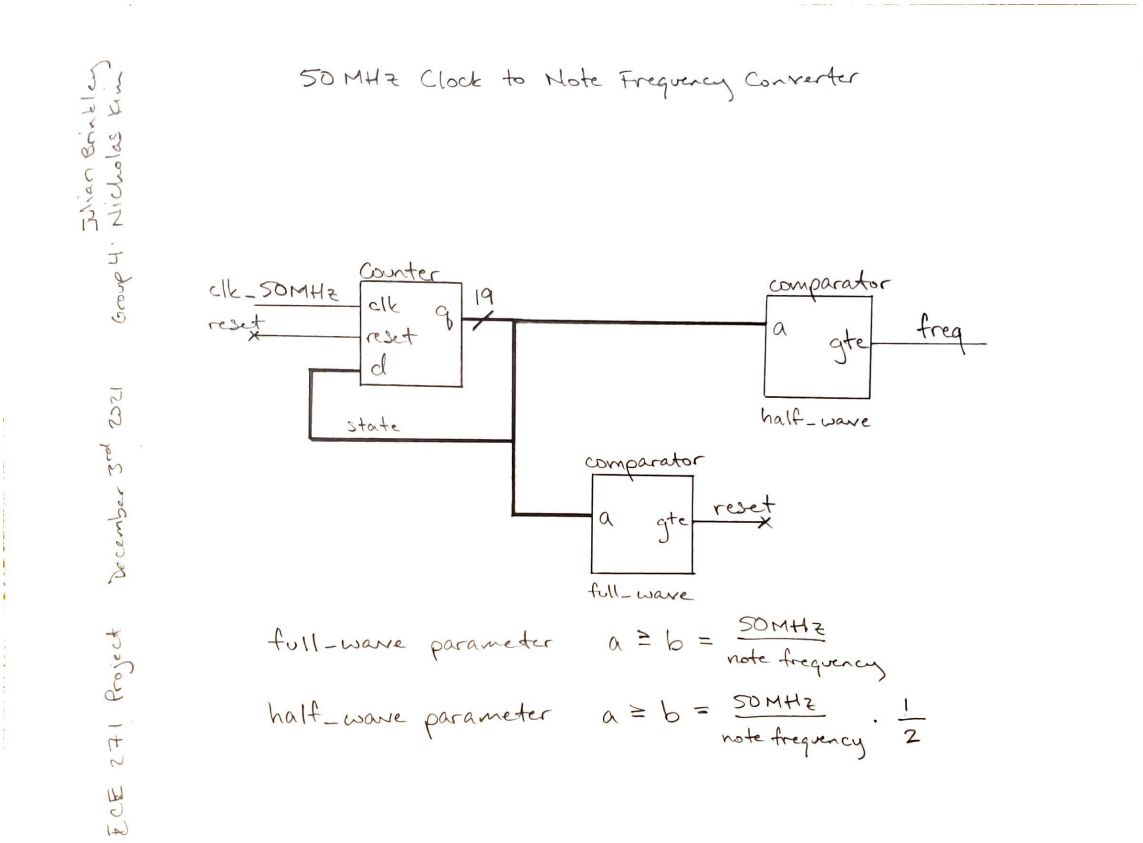

Figure 23: This is an expanded view of the frequency converter block shown in the top level diagram. This block is a template for each note frequency, which is determined by adjusting the parameters in both comparator instances. Refer to figures 13 and 15 for the counter and comparator blocks.

These simulations include a timestamp at the end of the first output cycle to show that the output frequency matches with the expected period of the desired note. For example, one wave cycle for G4 is correctly measured to be 2552750ns as predicted in table 3.

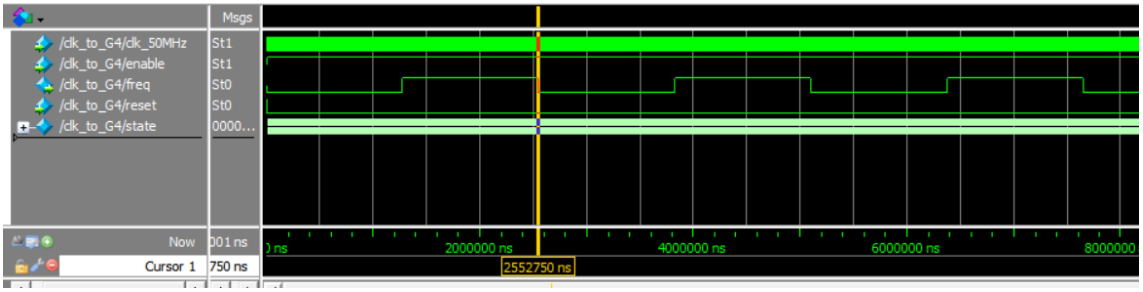

Figure 24: Clock to G4 simulation results from ModelSim

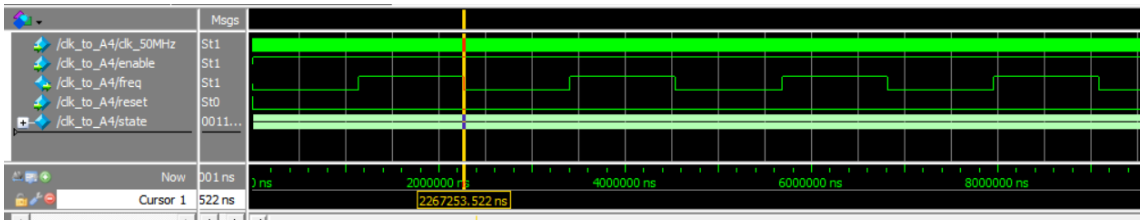

Figure 25: Clock to A4 simulation results from ModelSim

3 FPGA Implementation

The following link is a YouTube video demonstrating the FPGA implementation of the piano driver functioning as intended.

https://youtu.be/ZEomzQGTn8U

A SystemVerilog Files

A.1 Top Level

```
1  // Copyright (C) 2018  Intel Corporation. All rights reserved.
2  // Your use of Intel Corporation's design tools, logic functions
3  // and other software and tools, and its AMPP partner logic
4  // functions, and any output files from any of the foregoing
5  // (including device programming or simulation files), and any
6  // associated documentation or information are expressly subject
7  // to the terms and conditions of the Intel Program License
8  // Subscription Agreement, the Intel Quartus Prime License Agreement,
9  // the Intel FPGA IP License Agreement, or other applicable license
10 // agreement, including, without limitation, that your use is for
11 // the sole purpose of programming logic devices manufactured by
12 // Intel and sold by Intel or its authorized distributors.  Please
13 // refer to the applicable agreement for further details.
14
15 // PROGRAM                "Quartus Prime"
16 // VERSION                "Version 18.0.0 Build 614 04/24/2018 SJ Lite Edition"
17 // CREATED                "Sun Nov 28 16:12:36 2021"
18
19 module top(
20         clk_50MHz,
21         reset,
22         enable,
23         ps2_clk_in,
24         ps2_data_in,
25         gnd,
26         freq_out
27 );
28
29
30 input wire      clk_50MHz;
31 input wire      reset;
32 input wire      enable;
33 input wire      ps2_clk_in;
34 input wire      ps2_data_in;
35 output wire     gnd;
36 output wire     freq_out;
37
38 wire    [10:0] data;
39 wire    freq_A4;
40 wire    freq_Asharp4;
41 wire    freq_B4;
42 wire    freq_C5;
43 wire    freq_Csharp5;
44 wire    freq_Fsharp4;
45 wire    freq_G4;
46 wire    freq_Gsharp4;
47 wire    [3:0] s;
48
49 assign  gnd = 0;
50
51
52
53
54 clk_to_Csharp5  b2v_inst(
55         .clk_50MHz(clk_50MHz),
56         .reset(reset),
57         .enable(enable),
58         .freq(freq_Csharp5));
59
60
61 ps2_decoder     b2v_inst13(
62         .ps2_code(data[9:2]),
63         .decoded_val(s));
64
65
66 ps2_data        b2v_inst14(
67         .clk(ps2_clk_in),
68         .d(ps2_data_in),
69         .reset(reset),
70         .key(data));
71
72
```

```
clk_to_Fsharp4  b2v_inst25(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_Fsharp4));


clk_to_G4       b2v_inst26(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_G4));


clk_to_Gsharp4  b2v_inst27(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_Gsharp4));


clk_to_A4       b2v_inst28(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_A4));


clk_to_Asharp4  b2v_inst29(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_Asharp4));


clk_to_B4       b2v_inst31(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_B4));


clk_to_C5       b2v_inst32(
        .clk_50MHz(clk_50MHz),
        .reset(reset),
        .enable(enable),
        .freq(freq_C5));


mux9    b2v_inst35(
        .freq_A4(freq_Fsharp4),
        .freq_B4(freq_G4),
        .freq_C5(freq_Gsharp4),
        .freq_D5(freq_A4),
        .freq_E5(freq_Asharp4),
        .freq_F5(freq_B4),
        .freq_G5(freq_C5),
        .freq_A5(freq_Csharp5),
        .s(s),
        .freq_out(freq_out));


endmodule
```

A.2 Clock Divider

```
module clock_divider(input  logic clk,
                                                output logic clock_divided_11,
                                                output logic [4:0] count = 0);
        always @(posedge clk)
        begin
                if (count == 10)
                begin
                        clock_divided_11 <= 1;
                        count <= 0;
                end
                else
                begin
                        clock_divided_11 <= 0;
                        count <= count + 1;
                end
        end
endmodule
```

A.3 PS/2 Data to Select Decoder

```
module ps2_decoder_v2(input  logic [7:0] ps2_code,
                                                input  logic       dis_able,
                                                output logic [3:0] decoded_val);
        always_comb
                if(dis_able) begin
                        decoded_val = 4'b1000;
                end else begin
                        case(ps2_code)
                                8'b00011100 : decoded_val = 4'b0000;
                                8'b00011011 : decoded_val = 4'b0001;
                                8'b00100011 : decoded_val = 4'b0010;
                                8'b00101011 : decoded_val = 4'b0011;
                                8'b00111011 : decoded_val = 4'b0100;
                                8'b01000010 : decoded_val = 4'b0101;
                                8'b01001011 : decoded_val = 4'b0110;
                                8'b01001100 : decoded_val = 4'b0111;
                                default     : decoded_val = 4'b1000;
                        endcase
                end
endmodule
```

A.4 2:1 Multiplexer

```
module mux2 (input      logic   d0, d1,
                                input   logic   s,
                                output  logic   y);

        assign y = s ? d1 : d0;
endmodule
```

A.5 Counter

```
module counter  #(parameter N=8)
                                        (input logic clk, reset, en,
                                         input logic[N-1:0]     d,
                                         output         logic[N-1:0]     q);

        always_ff@(posedge clk, posedge reset)
                if(reset)       q <= 0;
                else if(en)     q <= d+1;

endmodule
```

A.6 Comparator

```
module comparator #(parameter N=18, b=113636)
                                                (input logic [N-1:0]a,
                                                 output logic gte);

        //assign eq  = (a == b);
        //assign neq = (a != b);
        //assign lt  = (a <  b);
        //assign lte = (a <= b);
        //assign gt  = (a >  b);
        assign gte = (a >= b);
endmodule
```