ECE 271: Design Project Report

Chase Denecke, Chris Pavlovich, John Davis, Yang Yang

December 1, 2017

1 Introduction

The job of our final project group was to attach various controllers to a speaker so that the speaker could be made to play various frequencies using the controllers. Our three controllers were a button board with 8 buttons, a PS2 keyboard, and an SNES controller. We used two intermediate pieces of hardware to connect the controllers to the speaker: A Field Programmable Gate Array (FPGA), and a Digital-to-Analog-Converter (DAC). Our main task was to write code in SystemVerilog that would take input signals from the controllers and convert those signals into an output to the DAC.

The idea of this project was to test our FPGA programming skills in a more rigorous way than they had been in our labs. I think we can all attest that this project did so.

2 High Level

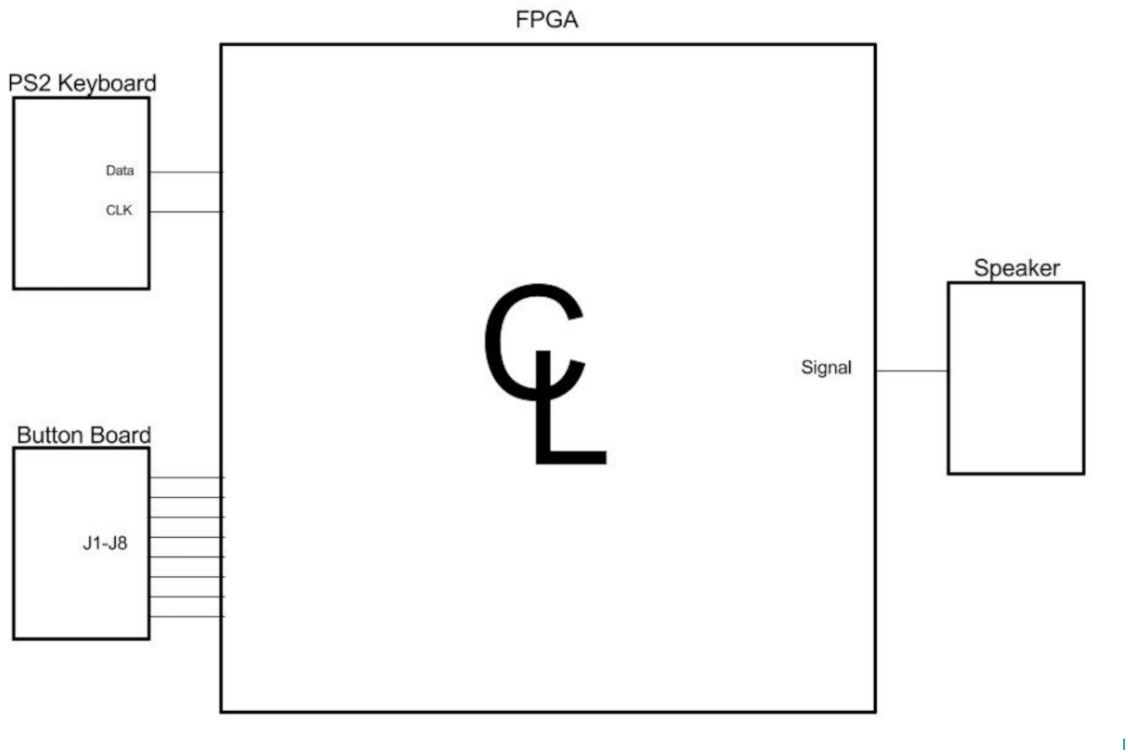

Figure 1: High Level Representation

Inputs: This reads signals from either a PS_2 keyboard or an 8-button push button board, the one provided in the lab portion of this class.

Outputs: This design outputs a sine wave signal to a speaker which will play one of the eight preset frequencies, the signal of which is selected by the inputs.

Description: As shown in the diagram above, the signals are taken from either of the two input options and then sent to the FPGA. In the case of the PS2 keyboard, a data line connects the keyboard to the FPGA using one of the input pins on the FPGA. This data line provides a flow of binary inputs that are read into the logic on the FPGA. In the case of the button board, the buttons J1-J8 are connected to 8 of the input pins on the FPGA, when pressed they will send a logic low reading to the FPGA, if they are not pressed they will send a logic high. When a button is pressed, the FPGA will choose which frequency is played based on the origin of the logic low reading. The speaker has one connection with the FPGA which with send a sine wave through one of the output pins on the FPGA.

3 Input Boards

3.1 PS2 Keyboard

Figure 2: PS_2 Keyboard

The first input that our design takes in is a PS2 keyboard (pictured above). The keyboard has one wire that connects to board itself to the logic. This wire is a data wire that sends data bit by bit to the logic. This type of communication protocol is known as parallel communication. Since the data is sent bit by bit, a shift register is required to pick up all the bits that are sent. This shift register then sends the N-bits of data when the register is full, then wipes the registers clear and starts reading bits in again. The shift register in our design is 11-bits because 11 bits of data represents a pressed key. Three of those eleven bits are erranous and don't have a particular use in our design. The middle eight bits hold the information for which key was pressed. However, the data does not start flowing instantly upon connection. A clock value has to be started in order for the data wire to start sending signals. The same clock value is used in the register to update the inputted data.

3.2 Button Board

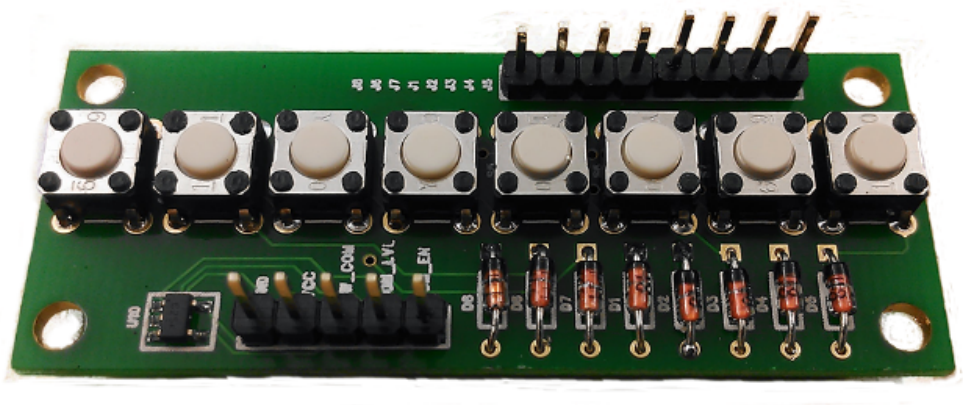

Figure 3: buttonBoard

The second input that our design is capable of reading is the 8-button push-button board. The picture above is the same model as the device used in the lab portion of this class. This is a very simple implementation of parallel communication protocol which can send 8 bits of information at a time. The controller is used in this design to switch between the 8 pre-set frequency values. There is also a command from the button board to pause and play the currently selected frequency through the speaker.

4 Modules

4.1 Top Module

Inputs: This module intakes a 4-bit number representing the decimal value of the current digit being displayed.

Outputs: This module outputs a 7-bit code that is sent to the 7-segment display that controls which of the segments on the display are illuminated.

Description: The module is built from a case statement that looks at what the input value represents in decimal form then decides which code to send to the 7-segment display to turn on the right segments to display the decimal number.

4.2 Register

Inputs:
regInData from the PS2 keyboard;
Clk generated from the PS2 keyboard;
A reset input from clock sync module.

Outputs: 11-bit data line.

Description: The register takes the keyboard input regInData which comes 1 byte at a time; the register also takes a clk input from the keyboard. It stores 1 byte each time at the rising clock edge; after 11 ticks the 11-bit data line is output as regOutData. It also has a reset input so it can reset the values if the input data is invalid.

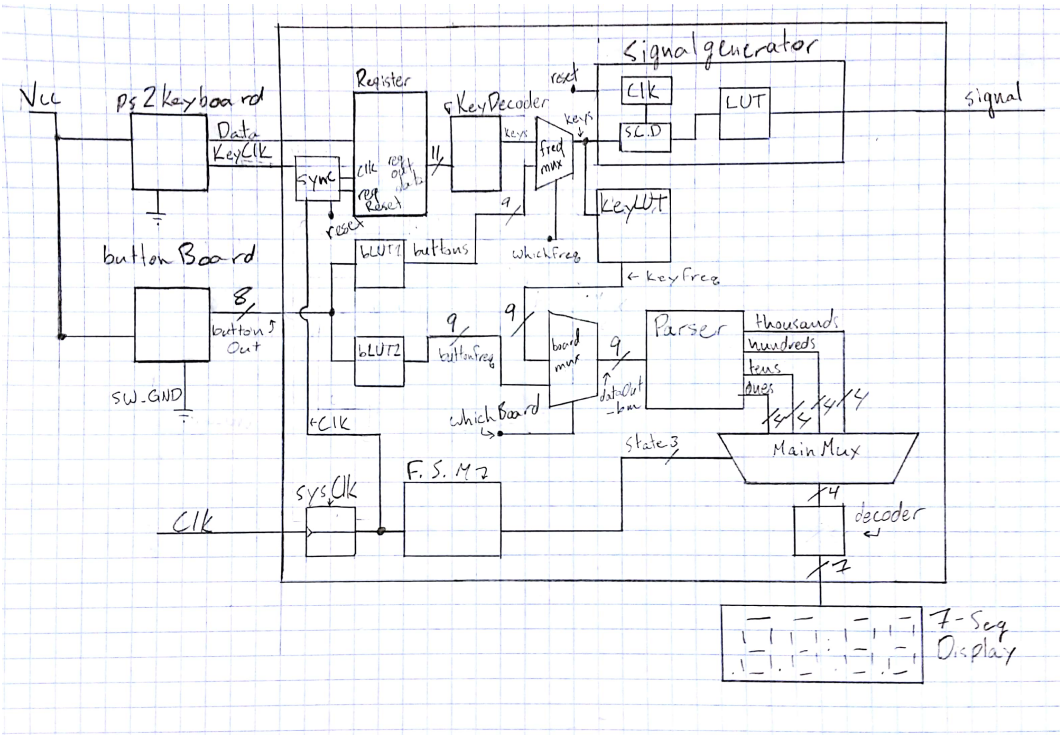

Figure 4: Top Module Schematic

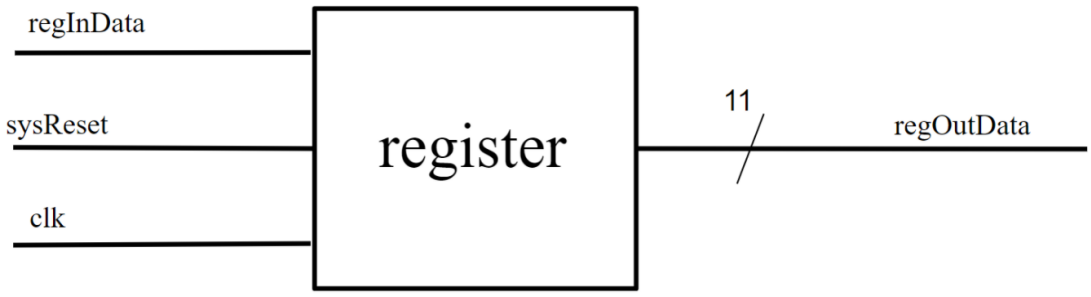

Figure 5: Register

4.3 Keyboard Decoder

Inputs: This module intakes a 4-bit number representing the decimal value of the current digit being displayed.

Outputs: This module outputs a 7-bit code that is sent to the 7-segment display that controls which of the segments on the display are illuminated.

Description: The module is built from a case statement that looks at what the input value represents in decimal form then decides which code to send to the 7-segment display to turn on the right segments to display the decimal number.

4.4 Key Look Up Table

Inputs: This module inputs a 9-bit frequency key that is related to an actual frequency value.

Outputs: This module outputs a 9-bit number that represents the decimal value for the selected frequency.

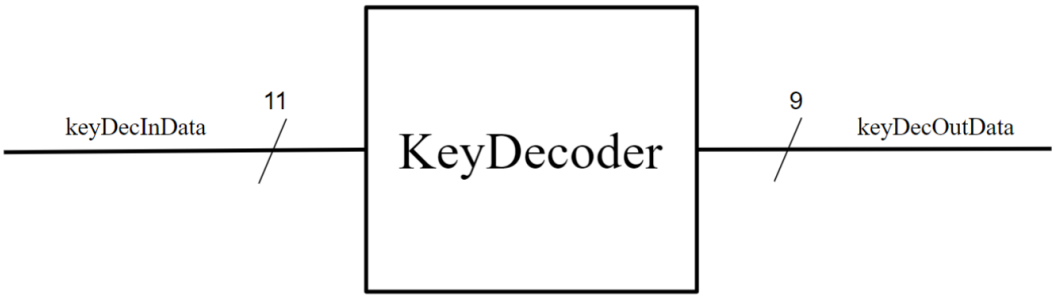

Figure 6: KeyDecoder

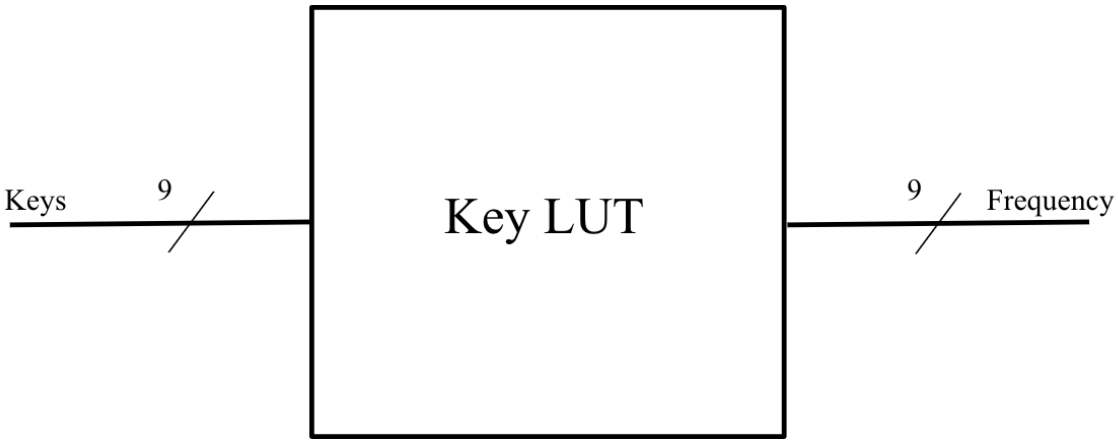

Figure 7: keyLUT

Description: The module is constructed from a case statement that selects which frequency value to send out based on what frequency key was inputted. If the frequency key was inputted for the first frequency, then the output of the module would be the binary representation of the decimal value for the first frequency.

4.5 Button Look Up Table 1

Inputs: This module intakes 8-bits of data from the button board.

Outputs: This module outputs a 9-bit frequency key value

Description: The module is built from a case statement that looks at the buttons pressed, and then sends a key value to the signal generator which lets the signal generator which frequency to output.

4.6 Button Look Up Table 2

Inputs: This module intakes 8-bits of data from the button board.

Outputs: This module outputs a 9-bit value that represents the decimal value of the frequency selected.

Description: The module is built from a case statement that looks at what buttons are pressed,

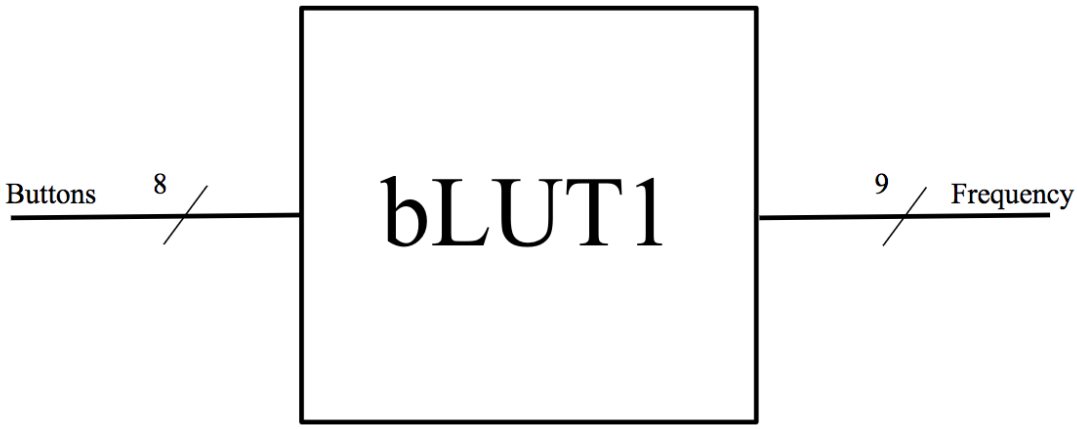

Figure 8: buttonLUT1

then decides which decimal frequency value corresponds with the button pressed, then sends the frequency value in binary form so it can be displayed by the 7-segment display decoder.

4.7 Frequency Mux

4.8 Board Mux

4.9 Signal Generator Top Level

The signal generator is told which key is pressed by the decoder. It then outputs an 8 bit binary number to the DAC representing the height of the sound wave at each successive moment in time. A new 8 bit value is sent to the DAC a few hundred thousand times per second.

4.10 Signal Clock Divider

The signal clock divider sub-module takes the built-in 2.08 Mhz clock signal and slows it down by an amount dependent on which frequency was selected. It accomplishes this by counting a specific number of clock edges from the 2.08 Mhz clock, and flipping its own output clock signal when it reaches that specific number.

This specific number is determined with a look-up table: each button corresponds with a single entry in the lookup table.

4.11 Signal Look Up Table

The signal look-up table submodule takes the slow clock as an input and outputs the 8-bit number representing the height of the sine wave. Filling the entries of the lookup table was one of the hardest parts of the project, because there is no way to directly compute sine using SystemVerilog.

Our solution was to write a program in C++ to generate the verilog code. The code from that file was then copied and pasted into our program. The C++ code that generated said code is included below.

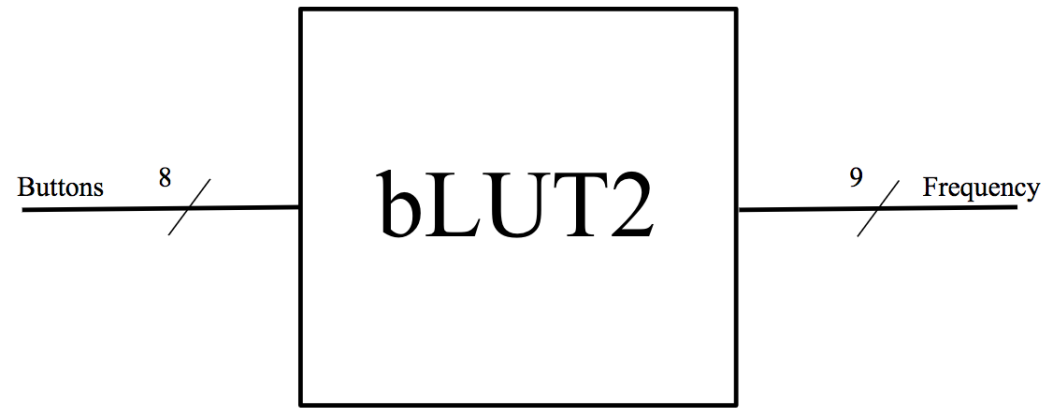

Figure 9: buttonLUT2

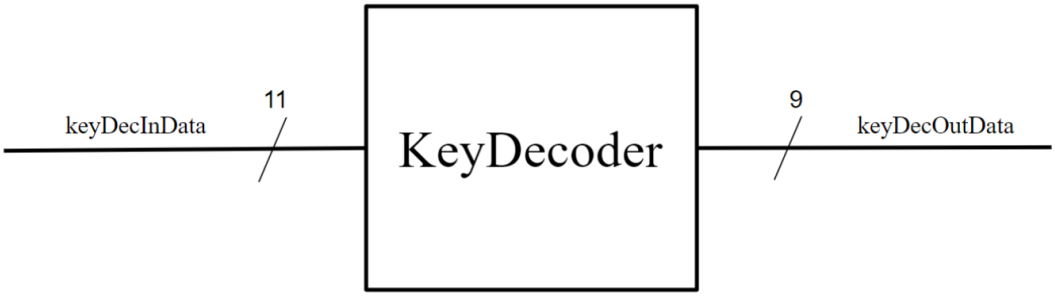

Figure 10: Key Decoder

4.12 Sync

4.13 System Clock

4.14 Free State Machine

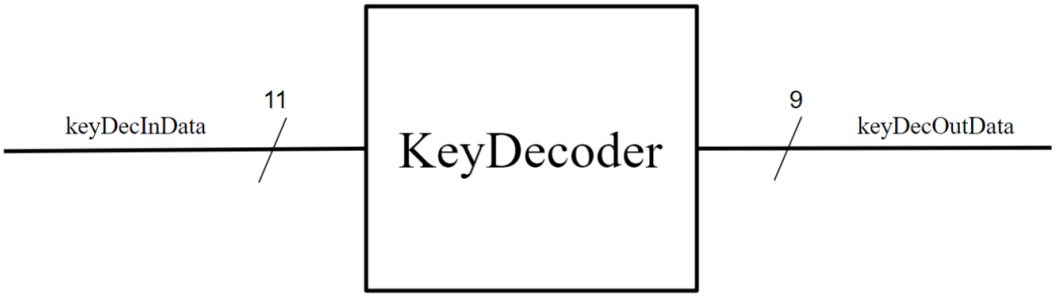

Figure 11: Key Decoder

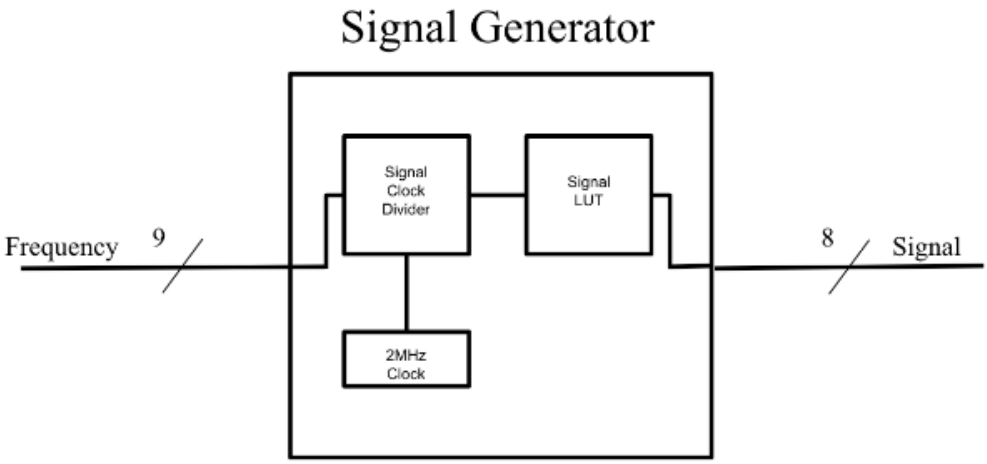

Figure 12: keyDec

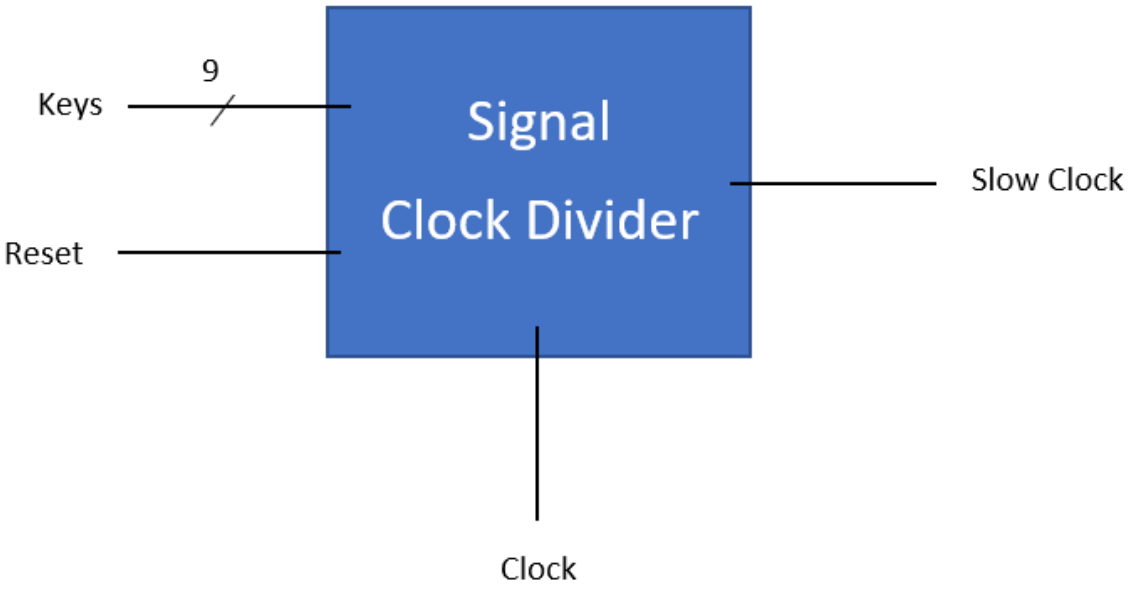

Figure 13: Signal Clock Divider

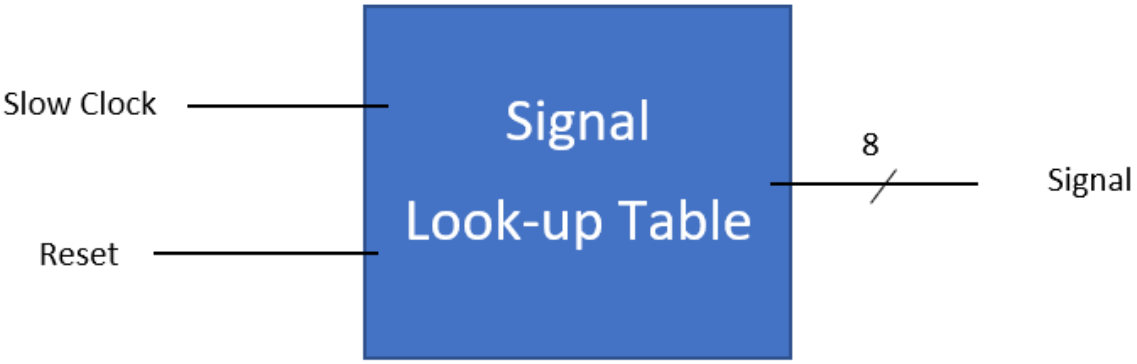

Figure 14: sigLUT

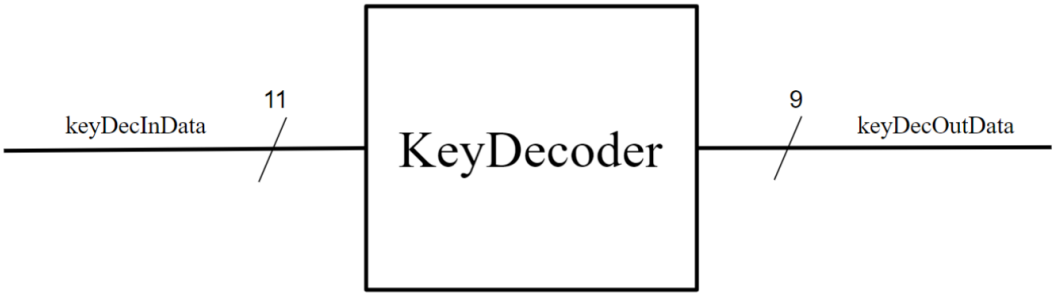

Figure 15: Sync

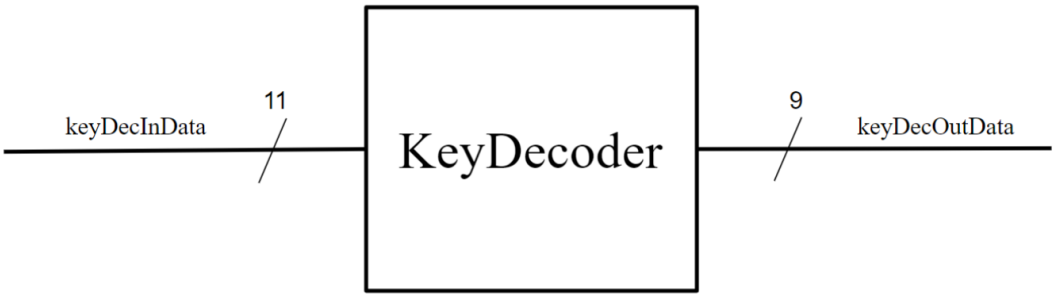

Figure 16: Sync

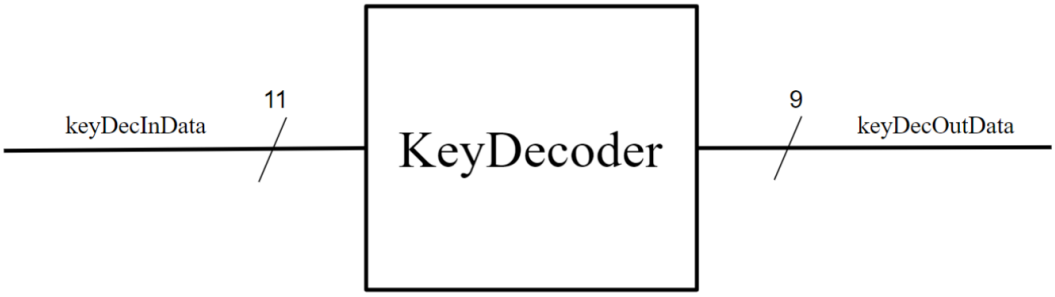

Figure 17: Sync

5 Appendix