Octave Keyboard with AutoPlay

Daniel Chen and Vivian Hu

August 26, 2014

The goal of this project was to create a simple one octave keyboard mapped to buttons with the additional capability to autoplay "Kids" by MGMT when a switch is turned on. When the buttons corresponding to notes are pressed, the appropriate notes are played through the speaker, and LEDs which correspond to the keys light up. The LEDs can be disabled using a switch.

This project was created by Vivian Hu and Daniel Chen in Summer 2014 at Dartmouth College for the Digital Electronics (ENGS031/COSC056) course. This report goes over the implementation, design, and usage of the final product, which implements all of these features.

Contents

1 Introduction: The Problem **2**

2 Design Solution **2**
2.1 Specifications . . . 2
2.2 Operating Instructions . . . 3
2.3 Theory of Operation . . . 3
2.4 Construction and Debugging . . . 3

3 Evaluation of Design **3**

4 Conclusions and Recommendations **3**

5 Acknowledgments **3**
5.1 Vivian's Contributions . . . 4
5.2 Daniel's Contributions . . . 4

6 References **4**

7 Appendices **5**
7.1 System level diagrams . . . 5
7.1.1 Front Panel . . . 5
7.1.2 Block Diagram . . . 6
7.1.3 Schematic Diagram . . . 6
7.1.4 Package Map . . . 6
7.1.5 External Components . . . 6
7.2 Programmed Logic . . . 6
7.2.1 State Diagrams . . . 6
7.2.2 VHDL Code . . . 6
7.2.3 Resource utilization . . . 10
7.3 Memory Map . . . 10
7.4 Timing Diagram . . . 10

1 Introduction: The Problem

The problem that this project solves is the creation of a one-octave keyboard, and the ability to produce certain sounds through circuit logic. An additional issue is representing the song that will be autoplayed.

2 Design Solution

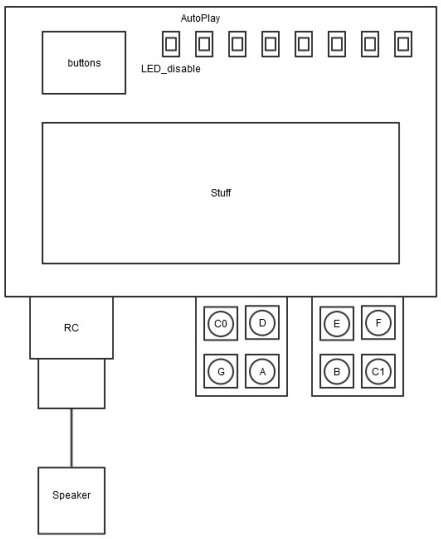
2.1 Specifications

The inputs to this circuit are:

- 8 Buttons that map to the notes to play (bottom right of image to the right).
- An LED disable switch which disables LED output.
- An AutoPlay switch which enables the playing of "Kids" by MGMT.

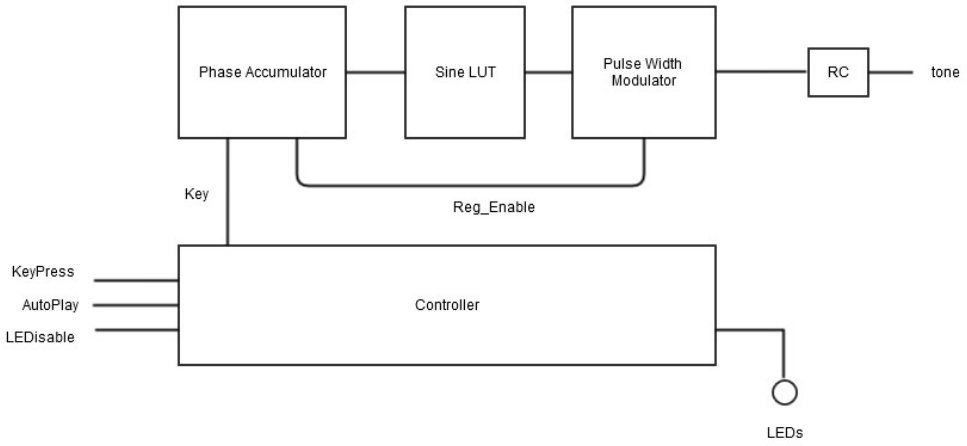
The outputs to the circuit are:

- 8 LEDs which correspond to the notes being played.
- A speaker which outputs the notes appropriate to the song or specified by the keys.



The picture shown below displays the datapath and control of the circuit. The controller (a finite state machine) takes in all of the inputs and outputs the LEDs. It emits a signal corresponding to the note that needs to be output (a copy of the LED output) which is the Phase Accumulator receives. The data from the Phase Accumulator transfers to the Sine LUT, which goes to the Pulse Width Modulator...

For more information on how the circuit works, see section 2.3 "Theory of Operation."



2.2 Operating Instructions

Set up Circuit

To set up the one-octave keyboard circuit, you will need:

- Design of
- The second item
- The third etc

Here are the steps to get the circuit running:

1. (If no bit file) Open the project in Xilinx, generate the programming file.
2. Plug in FPGA (Spartan 6) into computer.
3. Open Digilent Adept, select bit file to program the FPGA.

Playing notes

To play notes, press the buttons on the FPGA corresponding to the notes that you want to play. There are two rows of four buttons. The top four buttons represent low c, d, e and f, while the bottom four buttons are g, a, b and high c.

Only one note can be played at a time (first note pressed takes priority), and user-inputted notes only play when the auto-play switch is off.

Auto-play "Kids" by MGMT

To play "Kids" by MGMT, simply flip the AutoPlay switch on. You cannot create additional notes at this time using the buttons.

Disable LEDs

To disable the LEDs that light up corresponding to the notes that are pressed, simply turn on the LED disable switch.

2.3 Theory of Operation

2.4 Construction and Debugging

3 Evaluation of Design

4 Conclusions and Recommendations

The original goal of our project was to simply make a one-octave keyboard that plays all the notes in C major. LEDs would light up

5 Acknowledgments

We would like to thank Eric Hansen and Dave Picard for their support and mentorship throughout not only this project but also the course. We would also like to thank the other students of Digital Electronics as well as the TAs. We'd also like to thank MGMT for an awesome song that conveniently contains itself to a single octave.

5.1 and 5.2 list the contributions for each partner. Although both partners had their hands in most aspects of the project, each component generally had one partner who was more involved in its creation.

5.1 Vivian's Contributions

- Circuit Design
- The second item
- The third etc

5.2 Daniel's Contributions

- Majority of the controller, including auto-play
- Design of song, state diagram
- Top level
- LaTeX for report
- Git creation/management

6 References

7 Appendices

List of Figures

1 Simple FPGA Diagram 5
2 Block Diagram 6
3 VHDL for Controller 6
4 VHDL for DDS 7
5 VHDL for PWM 7
6 VHDL for PlayCount 8
7 VHDL for FreqLUT 9

7.1 System level diagrams

7.1.1 Front Panel

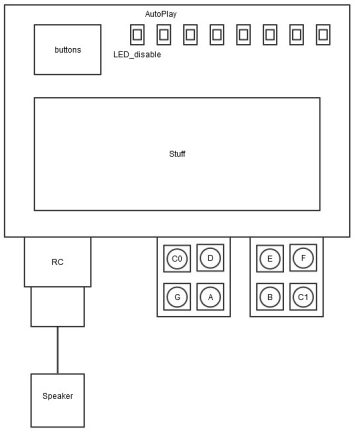


Figure 1: Simple FPGA Diagram

7.1.2 Block Diagram

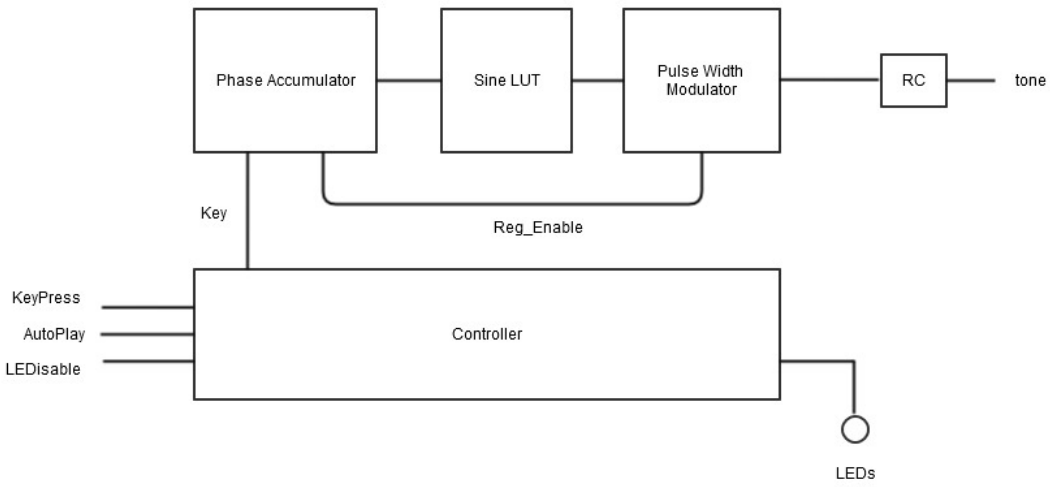


Figure 2: Block Diagram

7.1.3 Schematic Diagram

7.1.4 Package Map

7.1.5 External Components

- Design of
- The second item
- The third etc

7.2 Programmed Logic

7.2.1 State Diagrams

7.2.2 VHDL Code

Header comments removed.

Figure 3: VHDL for Controller

```
process
begin
  CLK <= '1'; wait for 10 NS;
  CLK <= '0'; wait for 10 NS;
end process;
```

Figure 4: VHDL for DDS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity DDS is
   Generic (  ACCUMSIZE : integer := 13;
          INDEXSIZE : integer := 8;
          CLKFREQ     : integer := 100000000);

    Port (  clk     : in  STD_LOGIC;
        step    : in  STD_LOGIC_VECTOR(ACCUMSIZE-1 downto 0);
        clk10   : in  STD_LOGIC;
        phase   : out STD_LOGIC_VECTOR(INDEXSIZE-1 downto 0));
end DDS;

architecture Behavioral of DDS is
  signal curr_phase : unsigned(ACCUMSIZE-1 downto 0) := (others => '0');
begin

  AccumPhase: process(clk, clk10)
  begin
    if (rising_edge(clk)) then
      if (clk10 = '1') then
        curr_phase <= curr_phase + unsigned(step);
      end if;
    end if;
  end process AccumPhase;

  phase <= std_logic_vector(curr_phase(ACCUMSIZE-1 downto ACCUMSIZE-INDEXSIZE));
end Behavioral;
```

Figure 5: VHDL for PWM

```
  process
  begin
    CLK <= '1'; wait for 10 NS;
    CLK <= '0'; wait for 10 NS;
  end process;
```

Figure 6: VHDL for PlayCount

```
process
begin
  CLK <= '1'; wait for 10 NS;
  CLK <= '0'; wait for 10 NS;
end process;
```

Figure 7: VHDL for FreqLUT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity FreqLUT is
   Generic (  ACCUMSIZE : integer := 13;
          CLKFREQ     : integer := 10000);

    Port ( clk : in  STD_LOGIC;
           key_in : in  STD_LOGIC_VECTOR (7 downto 0);
           increment : out  STD_LOGIC_VECTOR (ACCUMSIZE-1 downto 0));
end FreqLUT;

architecture Behavioral of FreqLUT is
  constant PHASECONSTANT : integer := 2**ACCUMSIZE;
  constant LOWC : integer := 262;
  constant D : integer := 294;
  constant E : integer := 330;
  constant F : integer := 349;
  constant G : integer := 392;
  constant A : integer := 440;
  constant B : integer := 494;
  constant HIGHC : integer := 523;
begin

  getIncrement: process(key_in)
  begin

    if (key_in(7) = '1') then
      increment <= std_logic_vector(to_unsigned(LOWC * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(6) = '1') then
      increment <= std_logic_vector(to_unsigned(D * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(5) = '1') then
      increment <= std_logic_vector(to_unsigned(E * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(4) = '1') then
      increment <= std_logic_vector(to_unsigned(F * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(3) = '1') then
      increment <= std_logic_vector(to_unsigned(G * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(2) = '1') then
      increment <= std_logic_vector(to_unsigned(A * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(1) = '1') then
      increment <= std_logic_vector(to_unsigned(B * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    elsif (key_in(0) = '1') then
      increment <= std_logic_vector(to_unsigned(HIGHC * PHASECONSTANT / CLKFREQ, ACCUMSIZE));
    else
      increment <= (others => '0');
    end if;

  end process getIncrement;
end Behavioral;
```

7.2.3 Resource utilization

7.3 Memory Map

7.4 Timing Diagram