Final Project
Digital Synthesizer

NAME: Ryan Barker

CLASS: ECE 372 Section 006

DATE: November 29th, 2014

OBJECTIVE: As outlined in my proposal, the goal of my experiment was to produce an interface that would allow a user to interface with a 4x4 matrix keypad and play different musical notes at varying lengths utilizing the on-board buzzer. This device was to have each button of the keypad assigned to a separate note, and each note would play for as long as its respective button was held down. Additionally, the LCD screen of the Freescale board was to display each note name for as long as the button was held down and clear immediately after release. The speed notes can be played was dependent on the processor used and musical chords were not to be supported. When completed, the interface acted like a very basic digital synthesizer and allowed the user to play simple compositions. Operation and interfacing between the keypad and the buzzer was done by a C program.

EQUIPMENT USED: Freescale PBMCUSLK Student Learning Kit
4x4 Matrix Keypad
On-Board LCD Display
On-Board Buzzer
Lab Workstation with CodeWarior C/C++ IDE

PROCEDURE: After establishing my design specification and completing my project proposal I began design in lab by figuring out and wiring the hardware design. This was rather simple for the LCD display and keypad. Because the display is built into the board itself, it can be interfaced by manipulating pins four through seven of Port B, leaving no wiring for me. Wiring a 4x4 keypad to the board is very simple, and just requires eight wires from the keypad to the eight pins of Port T.

However, almost immediately, I noticed that using the buzzer and keypad together poised an interesting problem: The buzzer is normally set up with jumpers to operate off of Port T pin 0, but I needed to use all of the pins of Port T with the keypad, since Port T was the only available port with enough pins to support the keypad. Two devices obviously cannot run off of the same pin of the microprocessor, so solving the problem required finding and moving the jumper from the buzzer off of Port T to another pin on the processor. Unfortunately, the only jumper pin near the buzzer's pin is Port T0's, but I solved this by putting one half of a jumper on the buzzer's pin and connecting a wire into the other half and connecting that to Port M pin 0. Note that this was very much a temporary solution to the problem, and is not something I would ever do in designing a real product, but for the scope of the project, it was a quick and easy fix to a problem that would have otherwise made it impossible to proceed.

Once the hardware was laid out, I tested it for proper operation by running the Lab 4 code and the Lab 8 code (Changing all references to Port T to Port M in the Lab 8 code). Once these programs worked to design specification, I knew that I was ready to proceed with the actual project.

At this point, before I began coding, I grabbed a piece of paper and planned out how each of the sixteen keys on the 4x4 keypad would map to each note the synthesizer could play. I originally planned to utilize all sixteen keys of the keypad, and mapped out a design that made sense using all of them, but later had to change my design to only use twelve keys because the "A", "B", "C", and "D" keys of the keypad were producing unpredictable behavior. Nevertheless, it was easy to map the buttons of the keypad to each note used in either circumstance. The final design that was used had every key except A, B, C, and D and started at an F# on key 1 and worked all the way to a F on key #, incrementing each note by a half step for each move to the right and down.

Once I had established the above, I began coding and decided to break my design into two parts: Reading from the keypad and printing the names of each note the display as they were pressed and actually making the buzzer sound for each note. These parts were completed and tested separately.

DESCRIPTION OF CODE: The quick and dirty description of what I have designed is a relatively straightforward combination of labs 4 and 8. To be more precise though, as mentioned above, the software for this one can be broken into two parts based on the two pieces of hardware used.

I began the code by referencing the note periods from Lab 8 and adding them as constants to the top of main. Back in Lab 8, I obtained these numbers by calculating the periods of each note in milliseconds based on their known musical frequencies. These were then slightly corrected for my machine's processing speed by getting them as close as my ear could discern to a half step apart. This left the frequencies of each note played by the machine as close to their actual values as possible, but the numbers used were still rough approximations.

Regardless of this, as usual, the first step main took in the program was variable definitions and register initializations. The keypad portion of the program required the same masking matrix from Lab 4 for reading and an associated note name matrix for printing. The note name matrix had to be a string array, since some note names included a sharp symbol (#) and were thus two characters. The buzzer required a unique period for each of the notes to be played by the program, so a note periods matrix was set up. Further, since the pin out of the 4x4 keypad isn't streamlined, each position in the three matrices didn't match the assumed bit mask and instead followed a more unique pattern given in Lab 4. I noted this and used the piece of paper from design with the keys mapped to each note to make sure each note was placed in the appropriate position of each matrix with each appropriate mask.

The last few variables set up were a counter variable for a logical loop in main, a temporary variable to hold a note name (N), and a temporary variable to hold a note period (P). The functions of these variables will be described below.

Register initialization required setting the first four pins of port T for output and the last four for input, since it was hooked to the keypad. Pin 0 of Port M was set for output, since it was hooked by the wire and jumper to the buzzer. Each input of Port T was initially pulled high by PPST (After the functionality was enabled in PERT) as expected for operation with a 4x4 keypad. The TSCR1 timer enable bit was

flipped on, the TIOS_IOS1 bit was set high, and the TC1 register was set to TCNT, since the buzzer was being used. Finally the LCD was initialized with the associated LCD.h function.

The first part of the main program was almost identical to the Lab 4 code: It consisted of a simple logical loop that set each of the output pins in port T to the each mask's value and checking the port until the input bits for the port matched their specified values in one of the masks. At this point, the index in the mask matrix was used to get the note name that was about to be played and send it to the LCD screen with LCDPutString(). The code for the buzzer than executed, and after it completed, LCDClearDisplay() was used to take the note name off of the LCD.

The second part of main simply checked that port T was still set to the mask value that match for printing the note name (I.E.- The key was still pressed) and then started the buzzer. This process is identical to the process used in Lab 8, where the period of the note is added to and stored in TC1, and the buzzer bit is toggled after half the period of the note elapses. This process continues until the state of Port T changes, at which point the buzzer is killed by writing a zero to Port M pin 0.

Note that in Lab 8, a delay matrix was used to determine how long each note the buzzer used was played. This was not necessary here, since pressing a key on the keypad determined how long each note should be played.

OBSERVATIONS AND DISCUSSIONS: As mentioned several times above, the design initially used all 16 keys of the matrix keypad, but I experienced difficulties testing the interface with the A, B, C, and D keys, as they would cause the buzzer to sputter a note rather than hold it when they were pressed down for a length of time. I initially attempted to fix this by re-wiring the keypad altogether with different leads, but when that failed me, my only choice was to redo the design without A, B, C, or D. This had minimal impact on the software, but decreased the range my synthesizer could play by four notes. I set the program to print a 0 and play a note with period 0 ms when these four keys were pressed to make it as obvious as possible to the user that these buttons were not meant to be pressed.

The jumper work-around and the odd behavior of the A, B, C, and D keys were my only real problems in the whole design, as everything else worked as expected. Once the design was complete, the interface met the specifications laid out in the "Final Result" section of my proposal, and I even got a little creative, playing some Bon Jovi, Boston, and Bastille on the keypad to prove that it could actually be used to play a song though the interface was clunky. Notes could only play so fast, so the speed of riffs was limited, but the synth was more flexible than I originally expected it to be. Chords were not implemented to the software, since unfortunately, the math behind them is a far more advanced than the buzzer can handle.

CONCLUSIONS: As a whole, this semester has taught me that you mind is the limit when designing equipment in the really world to interface with micro-controllers: If you have an idea, it probably isn't that hard to implement. This project specifically has shown me that even things I thought were very complex, like the basic digital instrument I built, are really just the rudimentary building blocks of interfacing stacked in a unique order. I learned that it is easy to interface the buzzer and keyboard together, and, as already mentioned, that interfacing any two devices on the Freescale board is easier than it seems. The project also forced me to think outside the box and troubleshoot at two points (with the jumper and the A-D keys), which are both key skills to have in Engineering.

SIGNATURE: *"This report is accurate to the best of my knowledge and is a true representation of my laboratory results."*

Ryan Barker

C CODE

```
/* ALWAYS INCLUDE THESE LINES OF CODE */
#include <hidef.h>          /* common defines and macros */
#include <mc9s12dt256.h>     /* derivative information */
#include "pbs12dslk.h"   /* I/O definitions and init routines */
#include "lcd.h"            /* LCD definitions and init routines *,

#pragma LINK_INFO DERIVATIVE "mc9s12dt256"
/* END ALWAYS INCLUDE THESE LINES OF CODE */

#define FS 2710
#define G 2551
#define GS 2409
#define A 2272
#define AS 2145
#define B 2020
#define C 1906
#define CS 1805
#define D 1700
#define DS 1608
#define E 1515
#define F 1432

void main(void) ETX
{
  unsigned char mask[16]={0xEE,0xDE,0xBE,0x7E,
                          0xED,0xDD,0xBD,0x7D,
                          0xEB,0xDB,0xBB,0x7B,
                          0xE7,0xD7,0xB7,0x77};

  signed char *notes[16]={"0","F#","G","G#",
                          "0","A","A#","B",
                          "0","C","C#","D",
                          "0","D#","E","F"};

  int periods[16] = {0, FS, G, GS,
                     0, A, AS, B,
                     0, C, CS, D,
                     0, DS, E, F};

  int i, P;
  char *N;


  DDRT = 0x0F;
  DDRM = 0x01;
  PERT = 0xF0;
  PPST = 0x00;
  TSCR1_TEN = 1;
  TIOS_IOS1 = 1;
  TC1 = TCNT;
  LCDInit();
```

```
52
53    do {
54      for(i = 0; i < 16; i++) {
55        PTT = mask[i];
56
57        if(PTT == mask[i]) {
58          /* Print note name to screen */
59          N = notes[i];
60          LCDPutString(N);
61        }
62
63        while(PTT == mask[i]) {
64          /* Start TC1 */
65          P = periods[i];
66          TC1 = TC1 + P;
67
68          /* Only take action if half the period has elapsed */
69          TFLG1 = TFLG1_C1F_MASK;
70          while(TFLG1_C1F == 0);
71
72          /* Set PTM_PTM0 to the proper value */
73          if((PTM & 0x01) == 0) {
74              PTM_PTM0 = 1;
75          } else {
76              PTM_PTM0 = 0;
77          }
78
79          /* Recalculate TC1 */
80          TC1 = TC1 + P;
81        }
82
83        /* Clear display and stop buzzer */
84        LCDClearDisplay();
85        PTM_PTM0 = 0;
86      }
87
88    } while(1);
89  }
90
```