Team Report

CPE 3150 KEYBOARD

LanChau LeTran, Joshua Cash, Alan Truong

**Architecture:**

* **Set up for music notes:**

The following tables shows the frequency from the speaker for each note and the preload value that we need for each frequency. This is the formula that we used to get the preload value:

This formula was used in excel to quickly calculate all the notes of each octave.

We have two octaves for each note in order to play the songs correctly and emulate reading from a music sheet. The frequency increases as we go up the octaves.

This table is the lower octave that we used. It contains the frequency and the preload value that we calculated.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C4-B6 | FOUND AT: | <https://pages.mtu.edu/~suits/notefreqs.html> | | | |
| Notes | Frequency | N | Size | N(floored) | Size |
| C | 261.63 | 7045.25475 | LONG | 7045 | LONG |
| C# | 277.18 | 6650.01082 | LONG | 6650 | LONG |
| D | 293.66 | 6276.81673 | LONG | 6276 | LONG |
| D# | 311.13 | 5924.37245 | LONG | 5924 | LONG |
| E | 329.63 | 5591.87574 | LONG | 5591 | LONG |
| F | 349.23 | 5278.04026 | LONG | 5278 | LONG |
| F# | 369.99 | 4981.8914 | LONG | 4981 | LONG |
| G | 392 | 4702.16837 | LONG | 4702 | LONG |
| G# | 415.3 | 4438.35781 | LONG | 4438 | LONG |
| A | 440 | 4189.20455 | LONG | 4189 | LONG |
| A# | 466.16 | 3954.11447 | LONG | 3954 | LONG |
| B | 493.88 | 3732.18191 | LONG | 3732 | LONG |

* Size identifies the amount of bits in which the preload value is. SHORT is within 8 bits, and LONG is within 16 bits.

This table is the higher octave that we used. It contains the frequency and the preload value that we calculated.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | 523.25 | 3522.6947 | LONG | 3522 | LONG |
| C# | 554.37 | 3324.94543 | LONG | 3324 | LONG |
| D | 587.33 | 3138.35493 | LONG | 3138 | LONG |
| D# | 622.25 | 2962.23383 | LONG | 2962 | LONG |
| E | 659.25 | 2795.98028 | LONG | 2795 | LONG |
| F | 698.46 | 2639.02013 | LONG | 2639 | LONG |
| F# | 739.99 | 2490.91204 | LONG | 2490 | LONG |
| G | 783.99 | 2351.11417 | LONG | 2351 | LONG |
| G# | 830.61 | 2219.15219 | LONG | 2219 | LONG |
| A | 880 | 2094.60227 | LONG | 2094 | LONG |
| A# | 932.33 | 1977.03603 | LONG | 1977 | LONG |
| B | 987.77 | 1866.07206 | LONG | 1866 | LONG |

The two octaves of notes are put into two separate arrays and are called by passing the index value for the corresponding note we want. Notes4 is the array of the lower octave and notes5 is the array of the higher octave.

Next we calculated the amount of time needed to play each type of note (i.e. a quarter note) based on the tempo of the song. When reading sheet music, each duration is based on the tempo for that song. For Hot Cross Buns, the tempo we used is 120 bpm. We looked up online for the duration or period (in seconds) of each type. Whole, half, quarter, and eighth notes are the types we use in our calculation.

This equation was used in excel to calculate N for each different types of notes.

The following table gives the preload value for tempo 120 used for Hot Cross Buns.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Time Wanted |  | N | Size | SHRINK N | New Size |
| whole | 1.99 |  | 7336135 | HUGE | 32605 | LONG |
| half | 0.99 |  | 3649635 | HUGE | 16220 | LONG |
| quarter | 0.49 |  | 1806385 | HUGE | 8028 | LONG |
| eighth | 0.24 |  | 884760 | HUGE | 3932 | LONG |

The original calculation for N was of size that was larger than 16 bits. Because of this, we then shrink the value to be able to use it as a 16 bit number. Shrink N is the value of N divided by 225 to get down to a 16 bit value. To get the correct tempo, we wrap the shrunken value in a loop that loops 225 times in order to emulate the N value.

This next table is at a tempo of 144 for Tetris.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| whole | 1.658 |  | 6112217 | HUGE | 27165 | LONG |
| half | 0.824 |  | 3037676 | HUGE | 13500 | LONG |
| quarter | 0.407 |  | 1500406 | HUGE | 6668 | LONG |
| eighth | 0.208 |  | 766792 | HUGE | 3407 | LONG |

These values were put into an array named beats and accessed when the note of that type is played.

|  |
| --- |
| Code unsigned int beats[] = { |
|  | 163, // Articulation pause !!Don't use this!! Internal only. |
|  | 3932, // Eighth Note. |
|  | 8028, // Quarter Note. |
|  | 16220, // Half Note. |
|  | 32605, // Whole Note. |
|  | 163, // Articulation pause !!Don't use this!! Internal only. |
|  | 3407, // Eighth Note. |
|  | 6668, // Quarter Note. |
|  | 13500, // Half Note. |
|  | 27165 // Whole Note. |
|  | }; |

**How notes are actually played:**

There are two functions within the code that works together to play the notes: playNote and holdNote.

playNote takes in three arguments: the preload value for the note, the type of note (i.e. half, whole, quarter etc.), and the tempo. Using timer 0, the loops 225 times, and the tempo determine which preload value is used for the loop.

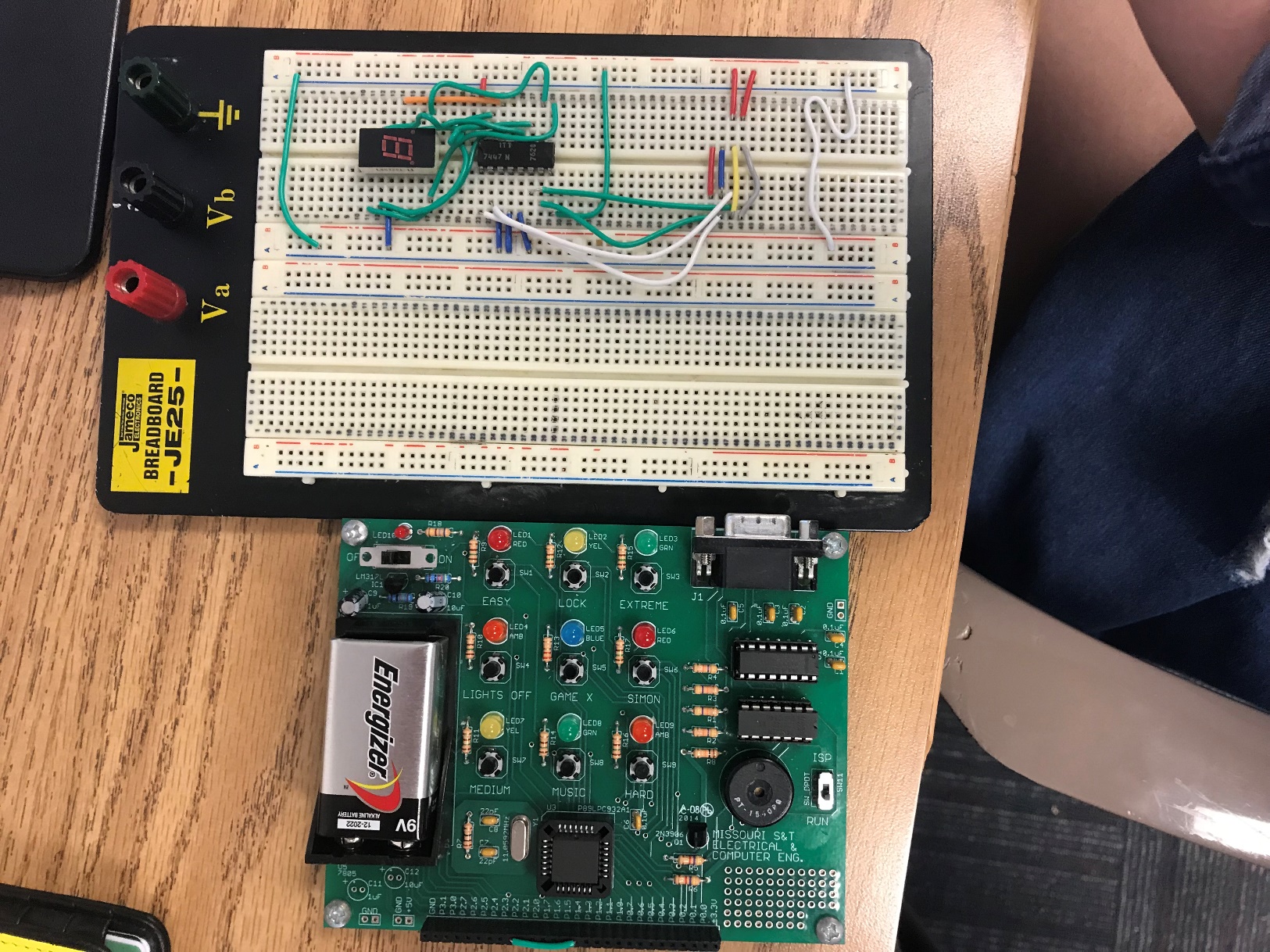
Within playNote, holdNote is called in order to actually send a frequency to the speaker to play the actual notes. The preload value for the note is passed from playNote to holdNote. This function uses timer 1 to send a square wave to the speaker. This square wave is sent continuously for the entire duration of the playNote loop.

At the end of playNote, the timeover function is called. This function uses timer 0 to pause the speaker or create an articulation pause.

**Switching Modes using interrupt and Breadboard:**

We created a global variable flag that is set to 0. When switch 9 or P2.2 is pressed, external interrupt 1 is triggered and flag is set to high. The flag is detected in our main function, and when it is high, the flag will be reset to low and the mode will be incremented.

updateMode is called to ensure that the right mode is being displayed. This function alters the binary code being sent to a decoder that then sends the correct signals to the 7-Segment display. The mode is then detected within a switch case to call different functions that correspond to each mode.



**Displaying the song title using Serial Communication:**

We created two arrays that each contain the title of a song. Mode 1 and mode 2 play each respective song. For each, displaySong is called. It takes in the array of song title and the size of that array. This function calls a different function, display, which loops through each element of the array and transmits the letter to terminal via UART.

**Project Descriptions:**

* **Mode 0 (Simon Board Keyboard):**

Allow the user to use the first row of the Simon2 Board to play a spontaneous tune using the three buttons.

When mode 0 is detected, the code will jump to the keyboard function. The entirety of the contents of the function is wrapped in a while-loop that waits for the flag variable to be triggered. This function will continue to run as long as the mode is not changed.

The three switches used are assigned to a variable to be used within the code so it can be determined which button is pressed.

// First button of the Keyboard.

sbit BTN1 = P2^0;

// Second button of the Keyboard.

sbit BTN2 = P0^1;

// Third button of the Keyboard.

sbit BTN3 = P2^3;

BTN1 plays C, BTN2 plays D, and BTN3 plays E in notes.

Within a switch statement, if one of these buttons are pressed, the function holdNote is called to send a square wave to the speaker to play the note.

* **Mode 1 (Song 1: Hot Cross Buns):**

When mode 1 is detected, the board will play Hot Cross Buns. We call playNote and pass the corresponding note we want to play, the note type, and the first tempo.

* **Mode 2 (Song 2: Tetris):**

When mode 2 is detected, the board will play Tetris. We call playNote and pass the corresponding note we want to play, the note type, and the second tempo.

**Problems:**

* **Running out of memory.** There was a point where we tried to add more notes or additional features and we simply couldn’t due to lack of memory. To fix this, we used the keyword code to store the arrays we needed into code memory instead of ROM.
* **Only one character was transmitting.** When we originally try to serially transmit the title of the current song that is playing, we could only transmit the first character and nothing else. We switched from external interrupt 0 to external interrupt 1 and it fixed it.
* **Multifunctional.** The pitch of the note would sometimes change when we tried to press another button or change mode. We isolated the keyboard functionality to its own while loop to avoid this pitch issue.
* **Unknown port functionality.** When we tried to breadboard to display the mode with the 7 segments display, we realized that some of the pins couldn’t be used. Some pins on the board were used by other processes that were not ours. By trial and error, we ended up using only p3.0 and p3.1 to use the display as their only functionality were external clock input and output.
* **Mode freeze up.** Sometimes when we were changing between modes, the whole board would freeze up until we restarted the board. To fix this issue, when we use the external interrupt to change mode, we made it a falling edge trigger rather than a level low.

**Efforts:**

We worked on this together but here’s the breakdown of how things were done:

**Joshua Cash 40%:** Responsible for interrupts, use of timers and breadboard. Worked on hot cross buns and displaying title in terminal.

**LanChau LeTran 30%:** Worked on Tetris and some of the note playing functionalities. Responsible for documentation and the project report.

**Alan Truong 30%:** Worked on mode 0. And all LED functionalities.

**Simon Board:**

**P2.2:** Switch 9 used to switch modes in main function.

**P3.1 and P3.0**: Used to connect to breadboard to display the mode on 7 segments display.

**P1.7:** Pin for the speaker, assigned to a variable named SPEAKER. Use this variable to turn on and off the speaker for different functions as well as sending it different square waves.

**P2.0:** Switch 1 used for the C note for keyboard function.

**P0.1:** Switch 2 used for the D note for keyboard function.

**P2.3:** Switch 3 used for the E note for keyboard function.

All the pins for the LED are used to display which notes are playing as followed:

sbit LEDC = P2^4;

sbit LEDD = P0^5;

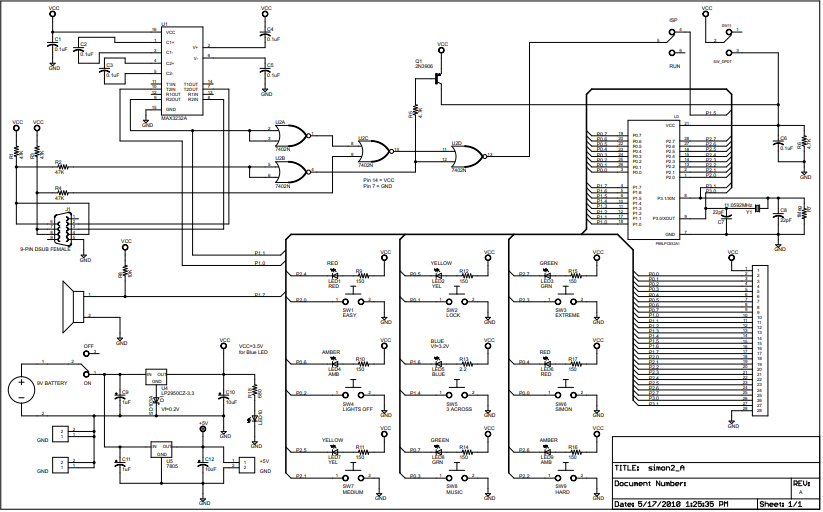
sbit LEDE = P2^7;

sbit LEDF = P0^6;

sbit LEDG = P1^6;

sbit LEDA = P0^4;

sbit LEDB = P2^5;

****

**Project Code:**

//#include <reg51.h>

#include <reg932.h>

#include "uart.h"

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*GLOBAL\*VARIABLES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Current mode of device.

unsigned char mode = 0;

// The variable of the speaker for sending sounds.

sbit SPEAKER = P1^7;

// First button of the Keyboard.

sbit BTN1 = P2^0;

// Second button of the Keyboard.

sbit BTN2 = P0^1;

// Third button of the Keyboard.

sbit BTN3 = P2^3;

// Flag to control when to switch modes. (set by External Interrupt).

bit flag = 0;

sbit LEDC = P2^4;

sbit LEDD = P0^5;

sbit LEDE = P2^7;

sbit LEDF = P0^6;

sbit LEDG = P1^6;

sbit LEDA = P0^4;

sbit LEDB = P2^5;

// Control the mode displayed on the 7-segment display two bit number.

// Bit 1 of the number.

sbit DISPLAYCONTROL1 = P3^0;

// Bit 2 of the number.

sbit DISPLAYCONTROL2 = P3^1;

// Arrays of the character that make up the song names.

// Name for Song 1.

code unsigned char HotxBuns[] = "Hot Cross Buns";

// Name for Song 2.

code unsigned char Tetris[] = "Tetris Theme";

// Array of Arrays of characters for each note in an octative.

code unsigned char notes[12][2] ={"C ", "C#", "D ", "D#", "E ", "F ", "F#", "G ", "G#", "A ", "A#", "B "};

// Notes Preload values for Octative 4, C4-B4 (includes sharps).

code unsigned int notes4[] = {

7045, // C 0

6649, // C# 1

6276, // D 2

5924, // D# 3

5591, // E 4

5278, // F 5

4981, // F# 6

4702, // G 7

4438, // G# 8

4189, // A 9

3954, // A# 10

3732 // B 11

};

// Notes Preload values for Octative 5, C5-B5 (includes sharps).

code unsigned int notes5[] = {

3522, // C 0

3324, // C# 1

3138, // D 2

2962, // D# 3

2795, // E 4

2639, // F 5

2490, // F# 6

2351, // G 7

2219, // G# 8

2094, // A 9

1977, // A# 10

1866 // B 11

};

// Tempo Control Preload values to "Play" a note for a certain type of note. AKA a quarter note.

// Add 4 to index to switch from bps = 120 to bps = 144.

//1 = Eighth note, 2 = Quarter note, 3 = Half note, 4 = Whole note, 0 = Articulation pause.

code unsigned int beats[] = {

163, // Articulation pause !!Don't use this!! Internal only.

3932, // Eighth Note.

8028, // Quarter Note.

16220, // Half Note.

32605, // Whole Note.

163, // Articulation pause !!Don't use this!! Internal only.

3407, // Eighth Note.

6668, // Quarter Note.

13500, // Half Note.

27165 // Whole Note.

};

// Timer Wait Values.

// One Second Wait Preload Value.

unsigned int SEC = 16384;

// 0.2 Second Wait Preload Value for Multiple Button Press Shielding.

unsigned int BUTTONPAUSE = 3276;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*OF\*VARIABLES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

\* Description: Updates the 7-Segment Display to the new mode.

\* Requires: None.

\* Returns: None.

\*/

void updateMode() {

// Start off at Zero.

DISPLAYCONTROL1 = 0;

DISPLAYCONTROL2 = 0;

// Turn on bits to make number in binary.

switch(mode) {

case 1:

DISPLAYCONTROL1 = 1;

break;

case 2:

DISPLAYCONTROL2 = 1;

break;

case 3:

DISPLAYCONTROL1 = 1;

DISPLAYCONTROL2 = 1;

break;

default:

// If it is not Case 1-3 Then it should be Zero (this keeps the mode in a loop).

mode = 0;

break;

}

}

/\*

\* Description: Sends a string via Serial Communication via UART to be displayed on terminal.

\* Requires: Array of characters to be sent and size of that array.

\* Returns: None.

\*/

void display(unsigned char text[], unsigned char size) {

unsigned char k = 0;

// Loop through each character in the array.

if(flag) return;

for(;k < size; k++) {

// Transmit the character.

uart\_transmit(text[k]);

}

}

/\*

\* Description: Display Function That will send the Song name given via UART to Terminal.

\* Requires: Array of characters of song name and size of the array.

\* Returns: None.

\*/

void displaySong(unsigned char song[], unsigned char size) {

// Display Song

display(song, size);

// Move to the next line

uart\_transmit('\r');

uart\_transmit('\n');

}

/\*

\* Description: Turns off all the LEDs for notes.

\* Requires: None.

\* Returns: None.

\*/

void shutOffLED() { LEDA = LEDB = LEDC = LEDD = LEDE = LEDF = LEDG = 1; }

/\*

\* Description: Turns on the LED corresponding to the note being played.

\* Requires: The index of the note.

\* Returns: None.

\*/

void turnOnLED(unsigned char note) {

// Turn off LEDs.

shutOffLED();

// Turn on the light corresponding to the note.

switch(note) {

case 0:

case 1:

LEDC = 0;

break;

case 2:

case 3:

LEDD = 0;

break;

case 4:

LEDE = 0;

break;

case 5:

case 6:

LEDF = 0;

break;

case 7:

case 8:

LEDG = 0;

break;

case 9:

case 10:

LEDA = 0;

break;

case 11:

LEDB = 0;

break;

}

}

/\*

\* Description: Display Function to display a note being played via UART to Terminal.

\* Requires: index of the note being played. valid if between 0 - 11.

\* Returns: None.

\*/

void displayNote(unsigned char note) {

display(notes[note], 2);

turnOnLED(note);

}

/\*

\* Description: Expects Timer 0 to not be in use. For Timer Load Values greater than 65535.

\* Requires: Timer Load Value that the actual desired value is 225 \* value but value is less than 16 bit size.

\* Returns: None.

\*/

void timerover(unsigned int t) {

// Looping over causes inexact timing due to loop nature.

unsigned char loop = 225;

// Make sure not to override the Timer1 settings.

//(Hopefully this makes sure it doesn't mess up if it is running).

unsigned int currentTMOD = TMOD >> 4 << 4;

// Attach the timer 0 mode we want.

currentTMOD += 0x01;

// Set TMOD to that mode.

TMOD = currentTMOD;

// Run the pause.

for(; loop > 0; loop--) {

TH0 = -t >> 8;

TL0 = -t;

TR0 = 1;

while(TF0 == 0);

TR0 = 0;

TF0 = 0;

}

}

/\*

\* Description: Does one Cycle of playing a note at a Frequency. Repeat to create a audible sound.

\* Requires: A Preload Value for a note that is less than 16 bit and is already set to 1/2 Duty.

\* Returns: None.

\*/

void holdNote(unsigned int note) {

// Make sure speaker is off.

SPEAKER = 0;

// Load the note value up.

TH1 = -note >> 8;

TL1 = -note;

TR1 = 1;

// Turn On SPEAKER.

SPEAKER = 1;

while(TF1 == 0);

// Turn Off SPEAKER.

SPEAKER = 0;

TR1 = 0;

TF1 = 0;

}

/\*

\* Description: Uses Timer 0 AND Timer 1 to play a note.

\*\*\* assumes that both timers are not in use.

\*\*\* note is preload for square wave for that frequency of that note.

\*\*\* type is the type of note: 1 = eighth note, 2 = quarter note, 3 = half note, 4 = whole note.

\* Requires: A Note Preload Value that is less than 16 bit and is already set to 1/2 Duty.

\*\*\* The type of note to be playing and the tempo to be set at.

\* Returns: None.

\*/

void playNote(unsigned int note, unsigned char type, bit tempo) {

// Looping over causes imprecise timing due to loop nature.

unsigned char loop = 225;

// Set the tempo.

unsigned int timerLoad = (tempo == 0 ? beats[type] : beats[type + 4]);

// Run the pause.

for(; loop > 0; loop--) {

TH0 = -timerLoad >> 8;

TL0 = -timerLoad;

TR0 = 1;

// Run while timer goes, speaker may slightly change.

while(TF0 == 0) {

// Check to see if we are playing a rest ( note = 0) or if the mode change flag has been set.

if(note > 0 && !flag)

holdNote(note);

// If the mode change flag has been set, leave the playNote Function.

if(flag)

return;

}

TR0 = 0;

TF0 = 0;

}

// Articulation Pause.

timerover(beats[0]);

}

/\*

\* Description: Function to play redundant tune for Song 1.

\* Requires: None.

\* Returns: None.

\*/

void song1Repeat() {

displayNote(4);

playNote(notes4[4], 2, 0);

displayNote(2);

playNote(notes4[2], 2, 0);

displayNote(0);

playNote(notes4[0], 3, 0);

}

/\*

\* Description: Song #1 - Hot Cross Buns.

\* Requires: None.

\* Returns: None.

\*/

void playSong1() {

// Measure 1.

song1Repeat();

// Measure 2.

song1Repeat();

// Measure 3.

displayNote(0);

playNote(notes4[0], 1, 0);

displayNote(0);

playNote(notes4[0], 1, 0);

displayNote(0);

playNote(notes4[0], 1, 0);

displayNote(0);

playNote(notes4[0], 1, 0);

displayNote(2);

playNote(notes4[2], 1, 0);

displayNote(2);

playNote(notes4[2], 1, 0);

displayNote(2);

playNote(notes4[2], 1, 0);

displayNote(2);

playNote(notes4[2], 1, 0);

// Measure 4.

song1Repeat();

// Only Delay the Next Play of the song if We are not switching modes.

if(!flag)

timerover(SEC);

}

/\*

\* Description: Song #2 - Tetris Theme.

\* Requires: None.

\* Returns: None.

\*/

void playSong2() {

// Measure 1.

displayNote(4);

playNote(notes5[4], 2, 1);

displayNote(11);

playNote(notes4[11], 1, 1);

displayNote(0);

playNote(notes5[0], 1, 1);

displayNote(2);

playNote(notes5[2], 2, 1);

displayNote(0);

playNote(notes5[0], 1, 1);

displayNote(11);

playNote(notes4[11], 1, 1);

// Measure 2.

displayNote(9);

playNote(notes4[9], 2, 1);

displayNote(9);

playNote(notes4[9], 1, 1);

displayNote(0);

playNote(notes5[0], 1, 1);

displayNote(4);

playNote(notes5[4], 2, 1);

displayNote(2);

playNote(notes5[2], 1, 1);

displayNote(0);

playNote(notes5[0], 1, 1);

// Measure 3.

displayNote(11);

playNote(notes4[11], 2, 1);

playNote(0, 1, 1);

displayNote(0);

playNote(notes5[0], 1, 1);

displayNote(2);

playNote(notes5[2], 2, 1);

displayNote(4);

playNote(notes5[4], 2, 1);

// Measure 4.

displayNote(0);

playNote(notes5[0], 2, 1);

displayNote(9);

playNote(notes4[9], 2, 1);

displayNote(9);

playNote(notes4[9], 3, 1);

// Only Delay the Next Play of the song if We are not switching modes.

if(!flag)

timerover(SEC);

}

/\*

\* Description: Keyboard Function. Allows user to use the first row of buttons to play notes.

\* Requires: None.

\* Returns: None.

\*/

void keyboard() {

// Keep looping till we change modes.

while(!flag) {

// Trigger note for corresponding button press. (Doesn't allow two to try and run at the same time).

if(!BTN1)

{

turnOnLED(0);

holdNote(notes5[0]);

}

else if(!BTN2)

{

turnOnLED(2);

holdNote(notes5[2]);

}

else if(!BTN3)

{

turnOnLED(4);

holdNote(notes5[4]);

}

shutOffLED();

}

}

/\*

\* Description: Function to handle when the mode switch button has been pressed.

\*\*\* Is wired up to receive press signal from SW9 on the board.

\* Requires: EX1 to be enabled.

\* Returns: None.

\*/

void ExternInterrupt() interrupt 2 {

// Set our Global Flag to let everything else to gracefully switch modes.

flag = 1;

// Multi Button Protection.

timerover(BUTTONPAUSE);

}

/\*

\* Description: Clears the Terminal Via UART Escape characters.

\* Requires: None.

\* Returns: None.

\*/

void clearScreen() {

// Google is my Friend.

// Excape Character.

uart\_transmit(27);

// Clear Screen Code.

display("[1J", 3);

// Excape Character.

uart\_transmit(27);

// Curser to top Code.

display("[H", 2);

}

/\*

\* Description: Keyboard Function 2. Allows User to use 'ASDF' as buttons to play notes via Serial Communication Via UART.

\* Requires: None.

\* Returns: None.

\*/

void keyboard2() {

// Stays here till mode changes.

while(!flag) {

// Get note to play and play the assigned note.

switch(uart\_get()) {

case 'a':

turnOnLED(11);

holdNote(notes4[11]);

break;

case 's':

turnOnLED(0);

holdNote(notes5[0]);

break;

case 'd':

turnOnLED(2);

holdNote(notes5[2]);

break;

case 'f':

turnOnLED(4);

holdNote(notes5[4]);

default:

break;

}

shutOffLED();

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*END\*OF\*FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void main() {

// Loop variable.

unsigned char k = 0;

// Initialization.

P1M1 = 0x00;

P2M1 = 0x00;

P0M1 = 0x00;

P3M1 = 0x00;

TMOD = 0x11;

EX1 = 1;

IT1 = 1;

mode = 0;

EA = 1;

uart\_init();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TO-DO List

-------------------

[X]Play two Short Tunes(~4 seconds long).

[X]Display Song Title of Each on PC Term via serial.

[X]Make keyboard with at least 3 buttons like keys.

[X]Use 1-2 buttons to control mode of opp(options above are modes).

- must be able to change modes in middle of stored tune is playing.

[X]use 8051 ports to connect a secondary device via breadboard to do some operation.

- Using seven-segment display to show mode.

[X]Each Member had their own feature.

[X] Comment Code.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

while(1) {

// Handle flag triggered.

if(flag) {

flag = 0;

mode++;

}

// Update Mode and 7-segment Display.

updateMode();

// Clear Screen before each mode.

clearScreen();

// Controls what mode we do.

switch(mode) {

case 0:

keyboard();

break;

case 1:

displaySong(HotxBuns, 14);

playSong1();

shutOffLED();

break;

case 2:

displaySong(Tetris, 12);

playSong2();

shutOffLED();

break;

case 3:

keyboard2();

break;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*EOF\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*