**Project Description**

The purpose of this project was to create a program with at least two modes: one to play two songs coded to memory and have their title transmitted to the screen serially, and one to play a spontaneous tune using buttons as keys on a piano. An additional task included connecting the Simon Board to a breadboard and control an accessory device. Our group was given one seven segment display to connect to.

*Song Process*

First we created structures to hold the info for ‘notes’ played out of the 8051 and enumerated types to store note names (A4, C5, D1, etc..) and rhythm values (Whole Note, Quarter Note, Eighth, etc). The first struct variable “NoteName” holds the values placed in the timer to produce the corresponding note to the speaker. The second variable “Rhythm”, held the value for the length of the note. Finally we had a variable a char letter, that stored the letter name of the note stored. This overall structure (struct) is what made up an actual ‘note’ of the song. Each ‘song’ that we programmed was stored as an array of ‘Note’ structures for easy storage and playback.

Our first step was to create an enum that stored the half frequency value for the note we wanted to play back. So, if the goal was to play the A at 440 Hz initially we stored 220 with a value of A4. We quickly discovered that the float pointing arithmetic required to convert the frequency value to the initial timer value obscene in both code space cost and data memory cost. We then had the revelation that since the frequency values were constant the starting timer values were also constant. Unfortunately we discovered this after we had coded the frequency values. Since two members in our group are computer science majors, and as such programmers, inherently lazy, rather than do the conversion by hand we wrote a regular expression and a python script to do the conversion for us and output the resulting C code. We prefered to store the starting values as negative numbers, but for some of the lower frequencies had to use positive numbers because the int data type doesn’t allow for us to use initialization values beyond 32,000, which was half the address space allocated to TL0 and TH0 in 16 bit timer mode.

Once the song was working, serial transmission was implemented. This is fairly simple, originally we stared the song titles in arrays and loops through them, displaying each item in the array using the function *uart\_transmit()*. However we wanted to use the terminal to also display the different modes we were in. This lead to a function that could be used anywhere in project3.c that took in a pointer to a char, essentially a string, and while string is not a null character (\0) call *uart\_transmit()* and iterate through the string.

To facilitate playback of music we utilized Timer 0 to oscillate the waveform to the speaker pin and Timer 1 to subdivide the note values as indicated by the Rhythm enum into 16th notes. We set Timer 1 to the appropriate value to rollover every millisecond. We created a variable called tempo to store the number of milliseconds per 16th note, and when the timer rolled over the number times indicated in tempo, the assigned number of 16th notes indicated by the note value was decremented by 1. When the this number became zero, the timer interrupt service routine moved on to the next note in the song and updated necessary values to change the frequency being output by speaker and Timer 0 to the new note. By doing this, changing the tempo value was all that was required to make the song playback slower or faster. This process was somewhat counter intuitive because decrementing the tempo made the song playback faster.

As an added bonus and to utilize a secondary device, we created the char in our Note struct specifically for communication with a seven segment display, as the song plays the corresponding note that is playing is displayed on the display to the beat of the song. To do this we created a function called display(), inside this function is a switch statement with cases for letters a-g and numbers 1-5 with hard coded values to each segment of the seven segment display.

*Keyboard Mode Process*

To implement keyboard mode we created a key array of type NoteName, size 3, and three random note values, we chose A4, G4, and F4. When button 4 is pressed from the main menu, buttons 7, 8, and 9 can then be used to play a spontaneous tune until the button is released. A special feature was implement to bind the ‘keys’ to different note values.

*Issues Encountered*

During this project we encountered a considerable amount of errors. Our first several plans for how to play an actual song out of the 8051 failed, consuming a lot of our time and energy. Secondly, when flashing from our personal computer about 20% of the time FlashMagic failed and would not flash, which just took up time.

The 8051 has limited external pins that can be accessed, and on top of this these external pins are also the same pins as many of the switches, meaning you cannot use the port and the switch at the same time. We discovered that P1.0-P1.3, and P1.5 were open for use. This is only 5 pins, and you need at least 7 to fully use the seven segment display. Our main issue was not the limited amount of pins, but the fact that one of our free pins was actually broken, P1.1 which was originally assigned to segment B, would not work. We ended up using the pins for Switches 7 - 9 to connect to the seven segment display.

Another issue we ran into was calling a uart function outside of main. In the code, we originally had a statement only the lines of

if (sw1==0){

play\_song(0);

for(i = 0; i < 28; i++){

uart\_transmit(songTitle1[i]);

}

}

But when implementing the different modes, this code was moved into a function outside of main and stopped working. This confused us since uart interrupts work independently of the timer and the move should not have caused any issues. We eventually found a solution and the transmission works from anywhere inside project.c.

Using the two timers to control the waveform and tempo led us to discover that the individual notes lacked articulation when played back at any reasonable speed. For long notes such as half notes and quarter notes this was marginally acceptable, but shorter notes like eighth notes this result was a garbled mess. To overcome this we inserted a 2 millisecond delay between each note when the timer one interrupt moved on to the next note in the song.

We had hoped to manage the rhythm and the waveform only using 1 timer, but found that by attempting to do so either the loss of precision required to manage the frequency values and/or the rapid rollover required to to store such values in the address space allotted by the datatypes available to us caused the note frequencies to be far lower than they should be.

A major issue encountered limits in the amount of code memory used. The free version of keil available for download on personal computers only compiles up to 2048 code space while

Uvision2 in the ECE building does not have this limit. The Simon Board has 8k on code memory on chip. We found that we just needed to compile and flash from the computer lab for the latter part of the project.

*Features*

To provide a visual representation of the different modes and options available we created a ‘menu’ feature which used corresponding Leds and Switches to show and navigate through different options. In each mode except the main menu, SW1 acted as an escape key when in a selected mode which returned to the previous menu. The menu options are listed as follows:

Main Menu:

1. SW1 - Song Playback Mode
2. SW4 - Keyboard Mode
3. SW7 - Special Feature (Planned as Scale Mode - Katherine)

Song Playback

1. SW1 - ESC
2. SW2 - Song1 (Another One Bites The Dust)
3. SW3 - Song2 (Jeopardy Theme)
4. SW6 - Tempo Control (Special - Jacques)

Keyboard Mode

1. SW1 - ESC
2. SW7 - Key1
3. SW8 - Key2
4. SW9 - Key3
5. SW3 - Keybind Feature (Special - Evan)

*Frequency*

To calculate each frequency we used the equation . All values placed into the timer were negative except for some lower notes, whose values were too large to be store, these values were converted to their equivalent positive values. List of all notes available in frequency.h in project zip

*Work Distribution*

All work excluding special features were done as a group effort. We accomplish tasks best by collaboratively creating the entire program together, however if one person had a little bit more knowledge or a better grasp on topic we were implementing to the board, said person would take the lead of the group work.

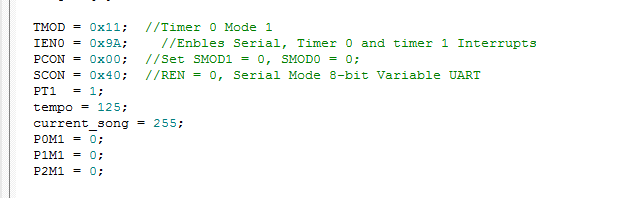
Special features were thought up and implemented by each individual, some troubleshooting help was given from other team members if asked.

Jacques — 35%

Evan— 33%

Katherine— 32%

**Use of the 8051 Architecture in Project Design**

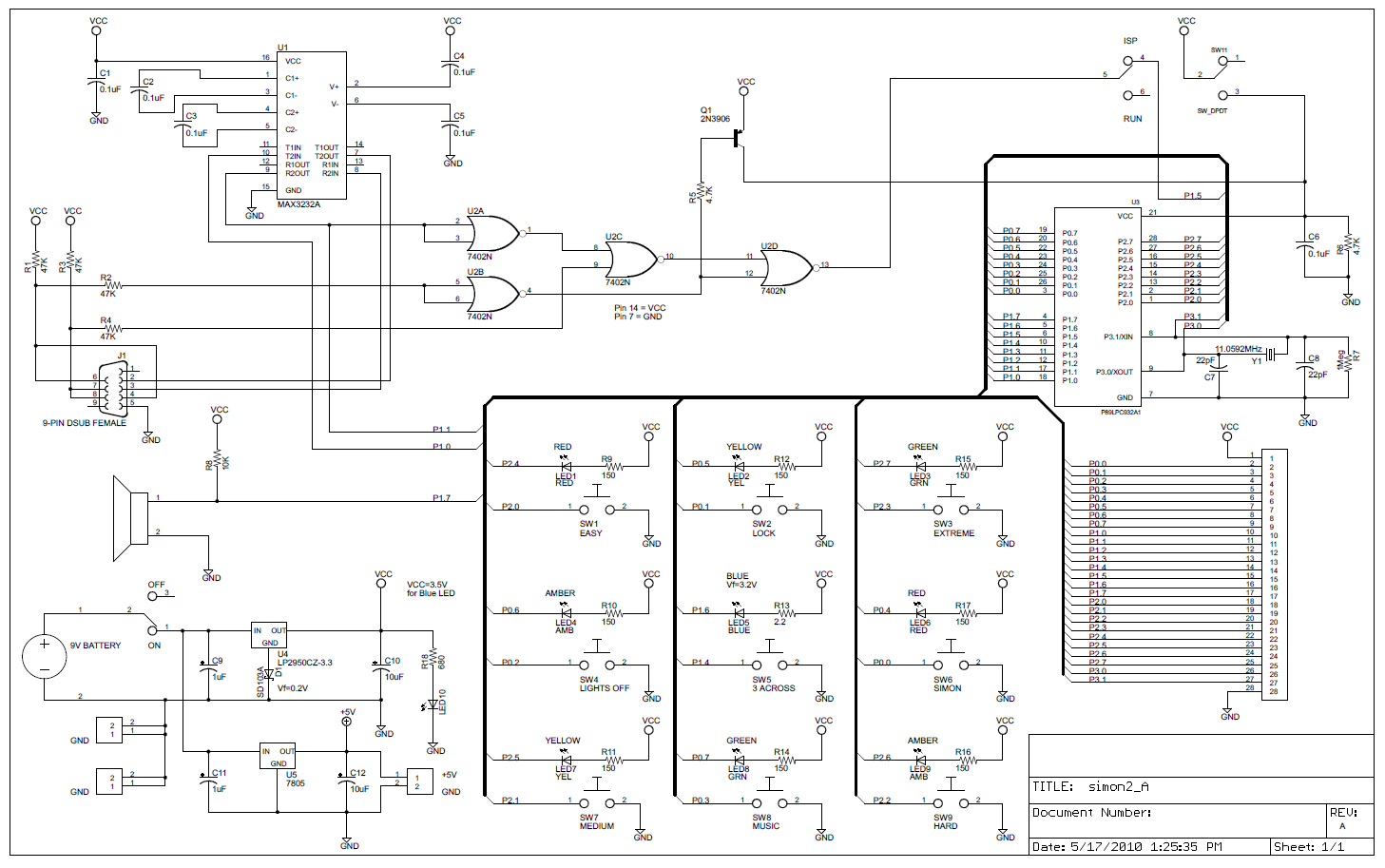
For this project Timer 0 and Timer 1 and their interrupts were used, equation for calculations for timer values can be seen in the frequency section of this report, further explanation on how the timers were used can be seen above in the project description. Here is the section of the code where the values regarding the timers are set. ****

**Simon Board**

*Ports and functions*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Port | Function | Port | Function | Port | Function |
| P2.0 | sw1 | P2.4 | led1 | P1.0 | segA |
| P0.1 | sw2 | P0.5 | led2 | P1.4 | segB |
| P2.3 | sw3 | P2.7 | led3 | P1.2 | segC |
| P0.2 | sw4 | P0.6 | led4 | P1.3 | segD |
| P1.4 | sw5 | P1.6 | led5 | P2.1 | segE |
| P0.0 | sw6 | P0.4 | led6 | P0.3 | segF |
| P2.1 | sw7 | P2.5 | led7 | P2.2 | segG |
| P0.3 | sw8 | P0.7 | led8 | P1.5 | DP (Not used) |
| P2.2 | sw9 | P2.6 | led9 | P1.7 | spkr |

*Schematic*



**Project Code**

* All project code can be found in project3.zip
* Files include:

Reg932.h //standard header file used for Simon Board

Uart.h //Given header file for uart features

Uart.c //Given uart code for serial communication with Simon Board

Project3.h //Header file for our project, contains ‘global’ variables and function prototypes

Project3.c //Main project file, contains main and function definitions

Frequency.h //Header file that holds structs for create of a Note

Frequency.c //Obligated accompanying c file to frequency.h