12/08/2021

Digital Alarm Clock

Digital Design Lab Final Project

Sarah Brown

I. INTRODUCTION

This course focused on various features and applications of the LPC1769, this project works to tie this experience together into one cohesive design. The project serves as a prototype for a digital alarm clock. Based on the features selected from the requirements, this prototype is able to accept user input with a series of switches. This user input is used for various features and centers on keeping track of time and related functions. One display uses input to set the time to the correct local time. An alarm clock can be set to be triggered at a specific time, once it is triggered it will display a different screen alerting the user to the alarm and play the song Oklahoma (like the OU clock tower). In addition, another display serves as a stopwatch and shows the elapsed time. This allows the user to start, stop, and reset the timer.

Points	Function	Description / Minimum Requirements	Status
1	Time of Day	Keep track the current time and be able to display it as decimal hours, minutes, and seconds.	Working
1	Alarm Clock	When the current time matches a user specified time, generate an alarm	Working
1	Elapsed Time	Display elapsed time. The user should be able to start, stop, and reset the timer. Must display at least minutes and seconds and go up to at least 99 minutes and 59 seconds.	Working
0.5	Character LCD	Use an LCD module with an HD44780 (or compatible) Controller	Working
0.5	Serially Controlled Display	Use SPI or I2C interface to read at least 4 switches	Working
0.5	Serially Interfaced Switches	Use SPI or I2C interface to read at least 4 switches	Working
0.5	Melody Alarm	Play a simple melody for the alarm condition (at least 8 notes and 4 frequencies)	Working
5			

Table 1: Summary of selected and implemented features

II. DIRECTIONS FOR USE

The alarm clock prototype has three main parts of interest for users: the LCD display and its contrast control, the speaker and its volume control, and four different buttons. To begin use of the clock, power must be supplied. Power is supplied via the USB connection on the LPC1769 for the 3.3 V components and via an USB power supply for the 5V components. To ensure no faults occur, users should make sure that the 3.3 V and 5 V power rails do not come in contact in the prototype.

After the clock prototype is powered, the clock is initialized at 0 hours, 0 minutes, and 0 seconds and begins to measure time. With the use of the serially interfaced switches, different displays can be reached. The main display, shown on start-up, shows the

current time and what other displays the blue, white, and green buttons can navigate to. On the main display, the red button simply returns the user to the main display to ensure continuity across the displays and thus has no function.

From the main display, the blue button can be used to navigate to a settings menu to set the clock's time. On this display, the clock time is paused so time will not pass. However, the user can use the blue button to set the seconds, the white button to set the minutes, and the green button to set the hours. The user input is display on the LCD display to help the user select the proper time. Once the desired time is set, the red button is then pressed to return to the main display and to resume the clock.

From the main display, the white button can be used to navigate to a settings menu to set the time of the alarm. Here the user can set the alarm to a given hour and minute combination. For the alarm, the second value is preset to 0 seconds to allow for the blue button to be used for other input. The blue button is used to toggle the alarm on and off, the white button is used to set the minute value of the alarm, and the green button is used to set the hour value. The set alarm time and the alarm status are both displayed on the LCD screen. The red button can be used to return to the main display.

From the main display, the green button can be used to navigate to the stopwatch function. Here the user can start, stop, and reset a counter displaying the elapsed time. The blue button is used to toggle the stopwatch status from counting to stopped and vice versa, the white button is used to reset the amount of time that has elapsed, and the green button is unused. The elapsed time is shown on the LCD screen. The red button can be used to return to the main display.

	Red Button	Blue Button	White Button	Green Button
Main Display	Return to Main Display	Set Time	Set Alarm	Stopwatch
Set Time	Return to Main Display	Set Seconds	Set Minutes	Set Hours
Set Alarm	Return to Main Display	Toggle Alarm On/Off	Set Minutes	Set Hours
Stopwatch	Return to Main Display	Toggle Stopwatch	Reset Time Elapsed	Unused

Table 2: Summary of directions for use

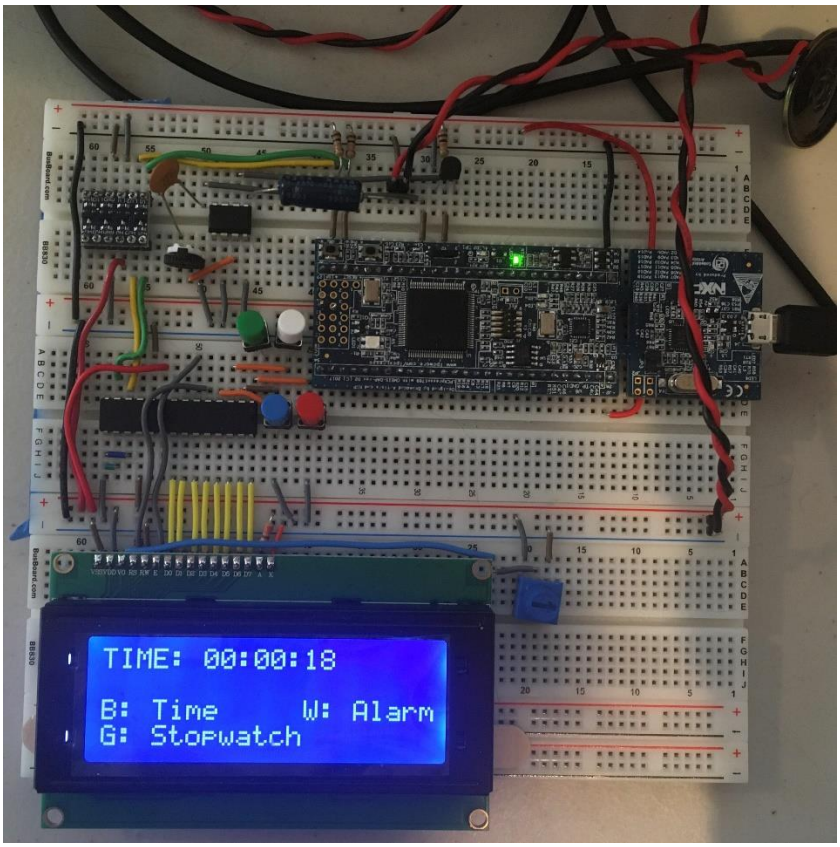


Figure 1: Picture of clock prototype

III. Design Analysis

The clock prototype incorporates various design elements from the different labs this semester. However, due to the different requirements for this project, many of the variables had to be recalculated. These values were related to the core building blocks of the software solution and include values to change the CPU clock frequency, values to control I2C communication, values to setup the RIT and generate various frequency notes, and values to control the RTC and its interrupts. In addition, values had to be determined for the various components for the hardware solution. These values were pull-up resistors, the resistor for the LCD backlight, and the amplifier circuit.

Part	Quantity
LPCXpresso1769/CD	1
MCP23017 I/O Expander	1
TC2004A-01 20x4 LCD Display	1
4-channel I2C-safe Bi-directional Logic Level Converter - BSS138	1
LM386 Amplifier	1
Speaker	1
TP2104N3	1
Push Button	4
10k Pull-up Resistors	3
330 Pull-up Resistor	1
100k Potentiometer	1
10k Potentiometer	1
100 uF Capacitor	1
0.1 uF Capacitor	1
5v Power Supply	1

Table 3: Bill of Materials

A. Hardware Design

The hardware solution for the alarm clock prototype provides support for the LCD display to show output to the user while also interfacing with user input to change different clock variables. The LCD display operates off of 5V while the LPC1769 operates off of 3.3V. Therefore, to use I2C communication between the LPC1769 and the LCD display, a logic shifter is included to shift the SDA and SCL lines from 3.3 V to 5 V. This is done with an Adafruit level shifter that uses BSS138 FETs with 10K pull-up resistors. This chips shifts to logic levels which are then used to communicate with the MPC23017 expander chip which then interfaces with the input buttons and the LCD display.

The LCD display includes a backlight, this backlight has an operating voltage of 4.2 V with a forward current of 180 mA. A resistor can then be selected from these specifications to ensure that the backlight LED is within its safety limits.

$$180\ mA = \frac{5V - 4.2\ V}{R}$$

$$\text{R} = 4.4\ \Omega$$

As any resistor greater than 5 Ω will be within the safety limits, a 330 Ω resistor is selected due to preference in backlight brightness. This results in a current of 2.4 mA which is within the safety limits.

$$I_{330\Omega} = \frac{5V - 4.2\ V}{330\Omega} = 2.4\ mA$$

The MCP23017 has internal weak pull-up resistors (100 kΩ) which can be enabled to internally pull up inputs. This allows for the four input buttons to be pulled up via the MCP23017 instead of with additional resistors.

Two pull-up resistors are used on the SDA and SCL lines to pull the lines up to 3.3 V. In addition, one pull-up resistor is used with a TP2104 FET to control the power to the amplifier circuit. This FET is used to limit noise produced by the speaker due to feedback from the I2C communication. All three of these pull-up resistors were selected to be a standard value of 10k. The standard value of 10k meets the specifications for I2C communication for the pull-up resistors for SDA and SCL. This is computed to be within two bounds:

$$\frac{V_{DD}}{I_{OL}} < R \ll \frac{t_{cycle}}{3C} = \frac{1}{3fC}$$

Here the I2C clock frequency must be set to an appropriate value (less than or equal to 100 kHz). C is the effective capacitance between the 3 devices tied in parallel to SCL, assuming each device is 10pF, this results in a C of 30pF for the 3 devices. V_{DD} is 3.3V and the smallest I_{OL} for the various devices is 3mA (the TC74 and the MCP23017 both at 3mA and the LPC1769 at 4mA). This results in the following bounds:

$$\frac{3.3V}{3mA} < R \ll \frac{1}{3(100kHz)(30pF)}$$
$$1.1\ kOhm < R \ll 111\ kOhm$$

Selecting a R value in the middles of 10 kΩ, gives the following values.

$$t_r \approx 3RC$$
$$t_r \approx 3(10000)(30 * 10^{-12})$$
$$t_r \approx 900\ nanoseconds$$
$$t_r \ll \frac{1}{f}$$
$$900\ ns \ll \frac{1}{100{,}000}$$
$$900\ ns \ll 10\ \mu s$$

And:

$$I = \frac{3.3V}{10\ kOhm} = 0.33mA$$

Finally, the last of the hardware calculations were to calibrate the gain of an amplifier circuit. This amplifier circuit is used with the speaker to produce the alarm sound. This is done with an LM386. However, the LM386's default gain of 21 would result in clipping so a voltage divider was utilized to avoid clipping. By using a 100 kΩ potentiometer, the gain can be adjusted to adjust the volume output. This allows to adjust to minimize and remove clipping. In addition, this allows for quieter volume levels for testing.

B. Software Design

The software solution for the alarm clock prototype controls the LCD display to show output to the user while also receiving input from the user to change various options within the clock.

The CPU clock frequency is set to 100 MHz to allow for precision with frequencies generated with the RIT. By using the main oscillator as the PLL0 clock source in the LPC1769, it is possible to start with a square signal of 12 MHz. This can be selected by setting the CLKSRCSEL value to 01 (page 36 in documentation). There is also an internal PLL feature that can be used to divide and multiply the signal. The 12 MHz signal is used as a base and the final signal is output on pin 1.27 on PAD10 so this must be appropriately configured as an output.

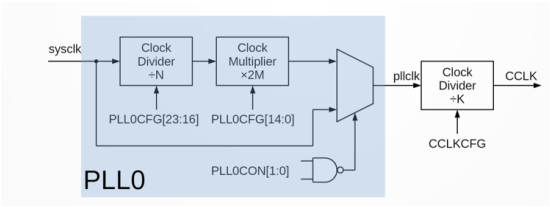


Figure 2: Details about the internal PLL in the LPC1769.

Examples in the LPC1769 documentation can be followed to calculate appropriate values to plug into the code and the registers.

The PLL inputs and settings must meet the following:

- F_{IN} is in the range of 32 kHz to 50 MHz
- F_{CCO} is in the range of 275 MHz to 550 MHz

FCCO can be found as follows, but as we want it to be a multiple of 100 so by selecting it as 300 MHz the equivalent N can be found. Since N, M, and K are required to be an integer values, N can be found by inspection. As 300/2=150/12=12.5 and 12.5*2 = 25 which is an integer. This lets us select M to be 25 and N to be 2.

$$F_{CCO} = \frac{(2)(M)(F_{in})}{N}$$

$$N = \frac{(2)(M)(12\ MHz)}{300\ MHz}$$

$$2 = \frac{(2)(25)(12\ MHz)}{300\ MHz}$$

K can be found as follows:

$$K = \frac{(F_{CCO})}{(F_{out})}$$

$$K = \frac{(300 * 10^6)}{(100 * 10^6)}$$

$$K = 3$$

However, since the code is 0 indexed each parameter should have an one subtracted from it leaving the values at: M = 25-1=24, K = 3-1=2, and N = 2-1=1. These values produce a clock value of 100 MHz.

The I2C data rate is controlled by I2CxSCLH and I2CxSCLL. This frequency is standard at 100 kHz so as it is based on the PCLK value and thus the CCLK value, it must be divided as the current clock CPU frequency is 100 MHz.

$$Freq = \frac{PCLK}{I2CxSCLH + I2CxSCLL}$$

PCLK is set to CCLK value/4 by default. This leaves it at PCLK = 100 MHz/4 = 25 MHz. Then I2CxSCLH and I2CxSCLL are set equal to each other to ensure a duty cycle of 50%. Setting these values both to 125 gives:

$$Freq = \frac{25\ MHz}{125 + 125} = 100\ kHz$$

The RIT is then configured to allow use of interrupts at a given match frequency. The interrupt creates a square wave at that given match frequency and decreases the wave amplitude over a short period of time. This period of time is dependent on whether or not the note is a whole note, a half note, a quarter note, or an eighth note. In addition, the same function that selects the note frequency and note duration, selects between whether the note is a rest note of a given size instead of producing a note. This allows for various notes to be played and can be used to create a melody alarm.

The variable for the RICOMP for these notes is calculated with the following:

$$Match\ Frequency\ Value = \frac{CCLK}{Note\ Freq * 2}$$

Using this equation, variables were found for the middle c octave as well as some notes from the fifth octave. Using simple sheet music, song functions were created to play various alarm melodies. The sheet music can be seen in Appendix 3: Sheet Music. Song functions were created to play the introduction of Oklahoma, to intimate the OU clock tower, and to play the introduction of the Pokémon theme song. Which melody is played during the alarm is currently hardcoded into the alarm clock prototype, but in a continued design could have additional songs as well as options to select between different songs.

The Real Time Clock is used to set and then measure the passage of time. This is done with various RTC control and data registers. The RTC also has interrupts for a set alarm and a counter interrupt that can be set to trigger on the increase of a unit of time (e.g. seconds). The use of these interrupts makes it possible to set and trigger an alarm and to measure and display the elapsed time.

Finally, in various initialization functions and while writing to the LCD display, waits are needed. The wait_ticks function from lab 1 was recalibrated to account for the 100 MHz clock. This resulted in a linear function as shown below.

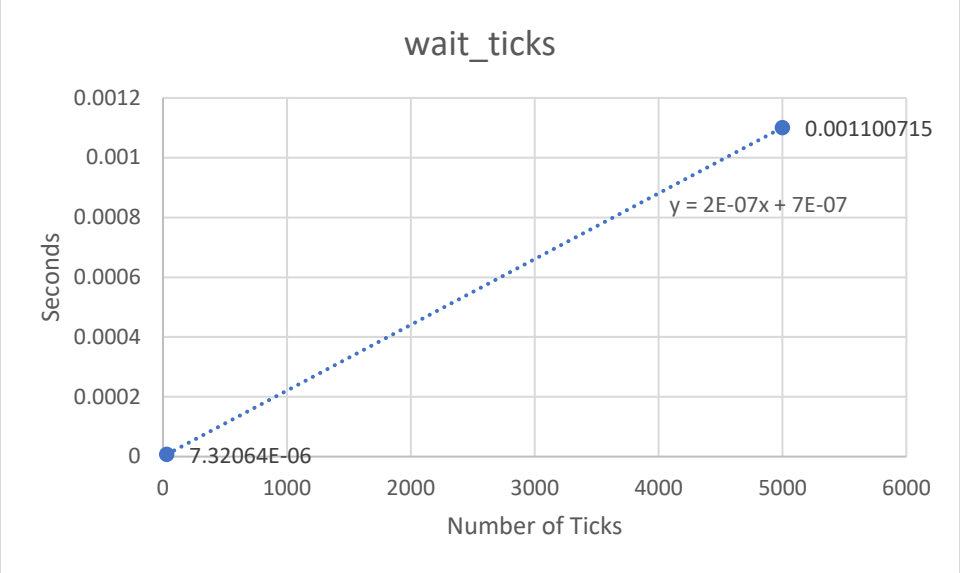


Figure 3: A graph created in Excel showing the equation for the wait_ticks() function.

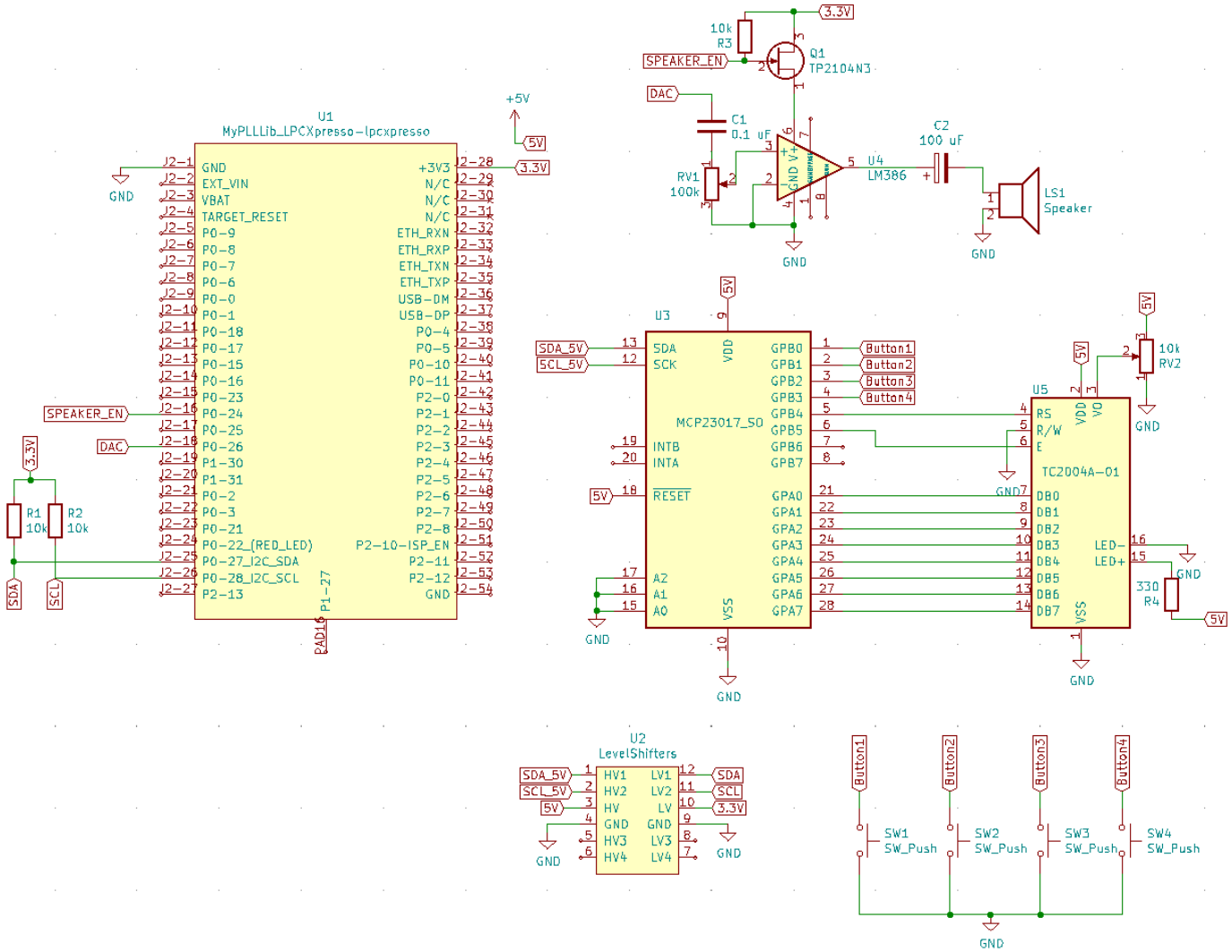
$$y = 2 * 10^{-7} + 7 * 10^{-7}$$

IV. Conclusion

With the use of knowledge from throughout the semester it is possible to create a prototype for a digital clock with various features. This clock can be used as an alarm clock as well as a stopwatch to measure the elapsed time. Future developments of this prototype could include using a different display to allow for pixel level control as well as using a different microcontroller to allow for a smaller device than possible with the LPC1769. The complete schematic and the complete software are shown in the appendix.

V. Appendix

1) Schematic



2) Code

```
/*
================================================================================
 Name        : define.h
 Author      : Sarah Brown
 Description : Main header file for including other headers and constants
================================================================================
*/

// other headers
#include "stdio.h"
#include "string.h"
#include "i2c.h"
#include "lcd.h"
#include "timer.h"
#include "button.h"
#include "alarm.h"
#include "display.h"
#include "melody.h"


// define LPC1769 I2C registers
#define PINSEL1  (*(volatile unsigned int *)0x4002C004) // pin function select
#define PCLKSEL0        (*(volatile unsigned int *)0x400FC1A8) // peripheral clock select
```

```
#define I2C_SCLH        (*(volatile unsigned int *)0x4001C010) // sclh duty cycle
#define I2C_SCLL        (*(volatile unsigned int *)0x4001C014) // scll duty cycle
#define I2C_CONSET      (*(volatile unsigned int *)0x4001C000) // control set
#define I2C_CONCLR      (*(volatile unsigned int *)0x4001C018) // control clear
#define I2C_DAT  (*(volatile unsigned int *)0x4001C008) // data
#define I2C_STAT        (*(volatile unsigned int *)0x4001C004) // status

// define clock/pll registers
#define CCLKCFG (*(volatile unsigned int *) 0x400FC104)
#define CLKSRCSEL (*(volatile unsigned int *) 0x400FC10C)
#define PLL0CFG (*(volatile unsigned int *) 0x400FC084)
#define PLL0CON (*(volatile unsigned int *) 0x400FC080)
#define PLL0FEED (*(volatile unsigned int *) 0x400FC08C)
#define PLL0STAT (*(volatile unsigned int *) 0x400FC088)
#define CLKOUTCFG (*(volatile unsigned int *) 0x400FC1C8)
#define SCS (*(volatile unsigned int *) 0x400FC1A0)

// define RIT/interupt registers
#define PCLKSEL1 (*(volatile unsigned int *) 0x400FC1AC)
#define PCONP (*(volatile unsigned int *) 0x400FC0C4)
#define RICOMP (*(volatile unsigned int *) 0x400B0000)
#define RICTRL (*(volatile unsigned int *) 0x400B0008)
#define RICOUNTER (*(volatile unsigned int *) 0x400B000C)
#define DACR (*(volatile unsigned int *) 0x4008C000)
#define PCONP (*(volatile unsigned int *) 0x400FC0C4)

// define pin registers
#define PINSEL3 (*(volatile unsigned int *) 0x4002C00C)
#define FIO0DIR (*(volatile unsigned int *)0x2009c000)
#define FIO0PIN (*(volatile unsigned int *)0x2009c014)

// RTC
#define CCR  (*(volatile unsigned int *) 0x40024008) // clock control reg
#define CIIR (*(volatile unsigned int *) 0x4002400C) // counter increment interrupt reg
#define AMR  (*(volatile unsigned int *) 0x40024010) // alarm mask reg. sec: bit 0, min: bit 1, hour: bit 2
#define SEC (*(volatile unsigned int *) 0x40024020)
#define MIN (*(volatile unsigned int *) 0x40024024)
#define HOUR (*(volatile unsigned int *) 0x40024028)
#define ISER0 (*(volatile unsigned int *) 0xE000E100)

// Alarm
#define ILR (*(volatile unsigned int *) 0x40024000)
#define ALSEC (*(volatile unsigned int *) 0x40024060)
#define ALMIN (*(volatile unsigned int *) 0x40024064)
#define ALHOUR (*(volatile unsigned int *) 0x40024068)

// I2C addresses
#define MCP_ADDR_W 0x40// MCP I2C ADDR write
#define MCP_ADDR_R 0x41// MCP I2C ADDR read
#define MCP_IODIRA 0x0 // MCP IO Pins DirA
#define MCP_IODIRB 0x1 // MCP IO Pins DirB
#define MCP_GPIOB 0x13 // GPIOB
#define MCP_GPIOA 0x12 // GPIOA
#define MCP_GPPUB 0x0D // pull-up resistor reg A

// macros
#define PLL_FEED() PLL0FEED = 0xAA; PLL0FEED = 0x55

// PLL variables for CLK init
#define K 2
#define M 24
#define N 1

// various magic numbers
#define TRUE 1
#define FALSE 0

#define COMMAND 0
#define DATA 1

#define PRESSED 0
```

```
#define UNPRESSED 1

#define MAIN_DISPLAY 0
#define SET_TIME 1
#define SET_ALARM 2
#define STOPWATCH 3
#define ALARM 4

#define RED 0
#define BLUE 1
#define WHITE 2
#define GREEN 3

#define SET_SECOND BLUE
#define SET_MINUTE WHITE
#define SET_HOUR GREEN

#define DAC_PIN 20 // pin p0.26
#define SPEAKER_EN_PIN 24 // pin p0.24
#define CLEAR 0x01

// frequencies for melody
// 100 MHz/(freq*2) (for 50% square)
#define FREQ_C 191110
#define FREQ_C_SHARP 180388
#define FREQ_D 170265
#define FREQ_D_SHARP 160710
#define FREQ_E 151685
#define FREQ_F 143172
#define FREQ_F_SHARP 135139
#define FREQ_G 127551
#define FREQ_G_SHARP 120395
#define FREQ_A 113636
#define FREQ_A_SHARP 88314
#define FREQ_B 101239

#define FREQ_C_OCT5 95557
#define FREQ_C_SHARP_OCT5 90192
#define FREQ_D_OCT5 85131
#define FREQ_D_SHARP_OCT5 80354
#define FREQ_E_OCT5 75844

#define EIGTH_NOTE 700
#define QUARTER_NOTE 500
#define HALF_NOTE 300
#define WHOLE_NOTE 1

/*
================================================================================
 Name        : alarm.h
 Author      : Sarah Brown
 Description : Header for alarm functions
================================================================================
*/

void RTC_IRQHandler();
void toggle_alarm();
void alarm_init();

/*
================================================================================
 Name        : button.h
 Author      : Sarah Brown
 Description : Header for button functions
================================================================================
*/

int button_press();
int which_button(int button, int prev_button);

/*
```

```
/*
===============================================================================
 Name        : display.h
 Author      : Sarah Brown
 Description : Header for display functions
===============================================================================
*/

void alarm_display();
void set_alarm();
void set_time();
void main_display();
void stopwatch_display();

/*
===============================================================================
 Name        : i2c.h
 Author      : Sarah Brown
 Description : Header for i2c functions
===============================================================================
*/

void i2c_start();
void i2c_write(int data);
int i2c_read(int stop);
void i2c_stop();
void i2c_init();

/*
===============================================================================
 Name        : lcd.h
 Author      : Sarah Brown
 Description : Header for lcd functions
===============================================================================
*/

void lcd_write(int command, int isData);
void lcd_init();
void display_string(char *str);
void display_char(char c);
void display_digits(int time_unit);

/*
===============================================================================
 Name        : melody.h
 Author      : Sarah Brown
 Description : Header for song functions
===============================================================================
*/

void play_note();
void play_freq();
void play_song_oklahoma();
void play_song_pokemon();
void disable_speaker();
void enable_speaker();

/*
===============================================================================
 Name        : timer.h
 Author      : Sarah Brown
 Description : Header for rit and rtc functions
===============================================================================
*/

void clk_init();
void rit_init();
void rit_enable();
void rit_disable();
void dac_init();
void RIT_IRQHandler();
void rtc_enable();
```

```c
void wait_ticks(int ticks);

/*
===============================================================================
 Name        : main.c
 Author      : Sarah Brown
 Description : Main definition and control for clock
===============================================================================
*/

#include "define.h"
int cur_mode = MAIN_DISPLAY;
int alarm_status = FALSE;
int button=0xF; int prev_button=0xF;
int dac_waveup = 0;
int wave_amp = 0;
int counter = 0;
int freq = 0;
int stopwatch_status = FALSE;
int stopwatch_counter = 0;

// setup function to perform various inits
void setup() {
        clk_init();
        rit_init();
        dac_init();
        i2c_init();
        lcd_init();
        rtc_enable();
        alarm_init();
        FIO0DIR |= (1 << SPEAKER_EN_PIN); // set speaker enable pin to output
        disable_speaker(); // turns off speaker so it does not produce unwanted noise
        wait_ticks(500); // to ensure proper setup
}

int main(void) {
        setup(); // calls setup function

    while(1) {
        button = button_press(); // updates button status
        if (cur_mode == ALARM) { // if the alarm interrupt was triggered
                alarm_display(); // go to alarm display
        }

        else {
                        cur_mode = which_button(button, prev_button); // updates current mode based on button and prev
button status

                        switch(cur_mode){
                                case MAIN_DISPLAY:
                                        main_display();
                                        break;
                                case SET_TIME:
                                        set_time();
                                        break;
                                case SET_ALARM:
                                        set_alarm();
                                        break;
                                case STOPWATCH:
                                        stopwatch_display();
                                        break;
                                case ALARM:
                                        alarm_display();
                                        break;
                                default:
                                        main_display();
                                        break;
                        }
        }
        prev_button = button; // updates prev button
    }
```

```
}

/*
============================================================================
 Name        : alarm.c
 Author      : Sarah Brown
 Description : Various functions for controlling the alarm
============================================================================
*/

#include "define.h"
extern int cur_mode;
extern int alarm_status;
extern int stopwatch_status;
extern int stopwatch_counter;

void RTC_IRQHandler() {
        if ((ILR & 1) == 1) {
                if (stopwatch_status) { // if stopwatch is unpaused
                        stopwatch_counter++; // increment
                }

                ILR |= (1 << 0); // resets interrupt
        }
        else { // otherwise interrupt is alarm
                cur_mode = ALARM;
                ILR |= (1 << 1); // resets interrupt
        }
}

void toggle_alarm(int alarm) {
        alarm_status = !alarm_status;

        if (alarm_status) {
                AMR = 0xF8; // sets mask to only check hour, min, sec
        }

        else {
                AMR = 0xFF; // turns off alarm
        }
}

void alarm_init() {
        ALSEC = 0;
        ALMIN = 0;
        ALHOUR = 7; // initializes alarm to 7am
        AMR = 0xFF; // initializes alarm to off
        alarm_status = FALSE;
}

/*
============================================================================
 Name        : button.c
 Author      : Sarah Brown
 Description : Various functions for control of button
============================================================================
*/

#include "define.h"

// retrieves data from MCP expander for buttons with i2c
int button_press() {
        int data;
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(MCP_GPIOB);
        i2c_start();
        i2c_write(MCP_ADDR_R);
        data = i2c_read(TRUE);
        i2c_stop();
```

```
        return (data & 0x0F); // masks data to only buttons
}

// compares button to prev button to determine which button was pressed
int which_button(int button, int prev_button) {
        if ((button & 0b1) == PRESSED && (prev_button & 0b1) == UNPRESSED) {
                return RED;
        }

        else if (((button>>1) & 0b1) == PRESSED && ((prev_button>>1) & 0b1) == UNPRESSED) {
                return BLUE;
        }

        else if (((button>>2) & 0b1) == PRESSED && ((prev_button>>2) & 0b1) == UNPRESSED) {
                return WHITE;
        }

        else if (((button>>3) & 0b1) == PRESSED && ((prev_button>>3) & 0b1) == UNPRESSED) {
                return GREEN;
        }

        return -1; // defaults to -1 if no change between button and prev button
}

/*
================================================================================
 Name        : display.c
 Author      : Sarah Brown
 Description : Various functions to control lcd display output
================================================================================
*/

#include "define.h"
extern int cur_mode;
extern int alarm_status;
extern int button; extern int prev_button;
extern int stopwatch_status ;
extern int stopwatch_counter;

// main clock display
void main_display() {
        int cur_sec = SEC;
        int cur_min = MIN;
        int cur_hour = HOUR;

        lcd_write(0x80, COMMAND); // move cursor to line 1
        display_string("TIME: ");
        display_digits(cur_hour);
        display_char(':');
        display_digits(cur_min);
        display_char(':');
        display_digits(cur_sec);

        lcd_write(0x94, COMMAND); // move cursor to line 3
        display_string("B: Time     W: Alarm");
        lcd_write(0xD4, COMMAND); // move cursor to line 4
        display_string("G: Stopwatch");
}

// display for setting time
void set_time(){
        lcd_write(0x01, COMMAND); // clear display
        CCR &= ~(0 << 0); // time counter is disabled so it may be initialized
        int cur_sec = SEC; int cur_min = MIN; int cur_hour = HOUR;

        while (((button >> SET_TIME) & 1) == PRESSED) { // waits for button to be released from entering function
                button = button_press();
        }

        int increment_time = which_button(button, prev_button);
```

```
        while (increment_time != RED) { // red button to exit back to main display
                switch (increment_time) {
                        case SET_SECOND:
                                cur_sec++;
                                if (cur_sec > 59) cur_sec=0;
                                break;
                        case SET_MINUTE:
                                cur_min++;
                                if (cur_min > 59) cur_min=0;
                                break;
                        case SET_HOUR:
                                cur_hour++;
                                if (cur_hour > 23) cur_hour=0;
                                break;
                }
                lcd_write(0x80, COMMAND); // move cursor to line 1
                display_string("TIME: ");
                display_digits(cur_hour);
                display_char(':');
                display_digits(cur_min);
                display_char(':');
                display_digits(cur_sec);

                prev_button = button;
                button = button_press();
                increment_time = which_button(button, prev_button);
        }

        cur_mode = MAIN_DISPLAY;
        SEC = cur_sec;
        MIN = cur_min;
        HOUR = cur_hour;
        CCR |= (1 << 0); // time counter is enabled so it may be initialized
        lcd_write(0x01, COMMAND); // clear display
}

void set_alarm() {
        lcd_write(0x01, COMMAND); // clear display
        int alarm_min = ALMIN; int alarm_hour = ALHOUR;

        while (((button >> SET_ALARM) & 1) == PRESSED) { // waits for button to be released from entering function
                button = button_press();
        }

        int increment_time = which_button(button, prev_button);

        while (increment_time != RED) {  // red button to exit back to main display
                switch (increment_time) {
                        case BLUE:
                                toggle_alarm();
                                break;
                        case SET_MINUTE: // white
                                alarm_min++;
                                if (alarm_min > 59) alarm_min=0;
                                break;
                        case SET_HOUR: // green
                                alarm_hour++;
                                if (alarm_hour > 59) alarm_hour=0;
                                break;
                }
                lcd_write(0x80, COMMAND); // move cursor to line 1
                display_string("TIME: ");
                display_digits(alarm_hour);
                display_char(':');
                display_digits(alarm_min);
                lcd_write(0xC0, COMMAND); // move cursor to line 2
                if (alarm_status) display_string("Alarm: On ");
                else display_string("Alarm: Off");

                prev_button = button;
                button = button_press();
```

```
                increment_time = which_button(button, prev_button);
        }

        cur_mode = MAIN_DISPLAY;
        ALMIN = alarm_min;
        ALHOUR = alarm_hour;
        lcd_write(0x01, COMMAND); // clear display
}

void alarm_display() {
        lcd_write(0x01, COMMAND); // clear display
        lcd_write(0x80, COMMAND); // move cursor to line 1
        display_string("ALARM ALARM ALARM");
        //play_song_pokemon();
        play_song_oklahoma();

        prev_button = button;
        button = button_press();

        int check_button = which_button(button, prev_button);

        while (check_button != RED) {  // red button to exit back to main display
                lcd_write(0x80, COMMAND); // move cursor to line 1
                display_string("ALARM ALARM ALARM");

                prev_button = button;
                button = button_press();
                check_button = which_button(button, prev_button);
        }

        cur_mode = MAIN_DISPLAY;
        lcd_write(0x01, COMMAND); // clear display
}

void stopwatch_display() {
        lcd_write(0x01, COMMAND); // clear display
        int stopwatch_min = 0; int stopwatch_hour = 0;

        while (((button >> STOPWATCH) & 1) == PRESSED) { // waits for button to be released from entering function
                button = button_press();
        }

        int check_button = which_button(button, prev_button);

        while (check_button != RED) {  // red button to exit back to main display
                switch (check_button) {
                        case BLUE:
                                stopwatch_status = !stopwatch_status;
                                break;
                        case WHITE:
                                stopwatch_counter = 0;
                                stopwatch_min = 0;
                                stopwatch_hour = 0;
                                break;          }

                if (stopwatch_counter > 59) {
                        stopwatch_counter = 0;
                        stopwatch_min++;
                }

                if (stopwatch_min > 59) {
                        stopwatch_min = 0;
                        stopwatch_hour++;
                }

                if (stopwatch_hour > 23) {
                        stopwatch_hour = 0;
                }

                lcd_write(0x80, COMMAND); // move cursor to line 1
                display_string("TIME: ");
```

```
                display_digits(stopwatch_hour);
                display_char(':');
                display_digits(stopwatch_min);
                display_char(':');
                display_digits(stopwatch_counter);

                prev_button = button;
                button = button_press();
                check_button = which_button(button, prev_button);
        }

        cur_mode = MAIN_DISPLAY;
        lcd_write(0x01, COMMAND); // clear display
}

/*
===============================================================================
 Name        : i2c.c
 Author      : Sarah Brown
 Description : Various functions for use of i2c
===============================================================================
*/

#include "define.h"

// I2C Start
void i2c_start() {
        I2C_CONSET = (1 << 3); // set SI bit in i2c cont
        I2C_CONSET = (1 << 5); // set STA bit in i2c cont
    I2C_CONCLR = (1 << 3); // clear SI bit

    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

    I2C_CONCLR = (1 << 5); // clear STA bit
}

// I2C Write
void i2c_write(int data) {
        I2C_DAT = data;
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

}

// I2C Read
int i2c_read(int stop) {
        // wait for data
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 3) & 1) == 0) {} // wait for SI bit to return to 1

        int data = I2C_DAT; // save the data

        if (stop) { // send nack
        I2C_CONCLR = (1 << 2);
        }

        else { //send ack
        I2C_CONSET = (1 << 2);
        }

        return data;
}

// I2C Stop
void i2c_stop() {
        I2C_CONSET = (1 << 4); // set STO bit in i2c cont
    I2C_CONCLR = (1 << 3); // clear SI bit
    while (((I2C_CONSET >> 4) & 1) == 0) {} // wait for STO bit to return to 1
}

// I2C Init
```

```
void i2c_init() {
	PINSEL1 |= (1 << 22); PINSEL1 |= (1 << 24); // sets scl and sda

	// want i2c freq to be 100 kHz, clock is at 100 MHz
	// PCLK is CCLK/4 by default, divide by 250 to be 100 kHz
	I2C_SCLL = 125; I2C_SCLH = 125; // configs i2c freq, pg 459

	I2C_CONCLR = 0x38; I2C_CONSET = 0x40;  // inits i2c

	// setup mcp
	// set dir a to all output
	i2c_start();
	i2c_write(MCP_ADDR_W);
	i2c_write(MCP_IODIRA);
	i2c_write(0x00); // set a to output
	i2c_stop();

	// set all dir b to output and input
	i2c_start();
	i2c_write(MCP_ADDR_W);
	i2c_write(MCP_IODIRB);
	i2c_write(0x0F); // set b0-b3 to input, b4-b7 to output
	i2c_stop();

	// GPPU
	i2c_start();
	i2c_write(MCP_ADDR_W);
	i2c_write(MCP_GPPUB);
	i2c_write(0x0F); // set GPIO PULL-UP RESISTOR REGISTER for B
	i2c_stop();

}

/*
===============================================================================
 Name        : lcd.c
 Author      : Sarah Brown
 Description : Various functions for use of the lcd display
===============================================================================
*/

#include "define.h"

void lcd_write(int command, int isData) {
	// update dbo-db7 to match command code
	i2c_start();
	i2c_write(MCP_ADDR_W);
	i2c_write(MCP_GPIOA);
	i2c_write(command);
	i2c_stop();

	// drive r/~w low (but its grounded so don't need to)

	if (isData) {
		// drive rs low to indicate this is a command
		i2c_start();
		i2c_write(MCP_ADDR_W);
		i2c_write(MCP_GPIOB);
		i2c_write(0x30);
		i2c_stop();
	}

	else {
		// drive rs low to indicate this is a command
		i2c_start();
		i2c_write(MCP_ADDR_W);
		i2c_write(MCP_GPIOB);
		i2c_write(0x20);
		i2c_stop();
	}
```

```
        // drive E high than low to generate the pulse
        // cpu is faster than 4MHz so might need a short delay after E is high to meet min pulse width req
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(MCP_GPIOB);
        i2c_write(0x00);
        i2c_stop();

        wait_ticks(500);
        i2c_start();
        i2c_write(MCP_ADDR_W);
        i2c_write(MCP_GPIOB);
        i2c_write(0x20);
        i2c_stop();
        // wait 100 us for command to complete
        wait_ticks(500);
}


void lcd_init() {
        wait_ticks(20000); // wait 4ms after control signals and data signals are configured (configured in i2c setup)
        lcd_write(0x38, COMMAND); // selects full 8-bit bus and 2 line display
        lcd_write(0x06, COMMAND); // sets cursor to auto advance to the right
        lcd_write(0x0c, COMMAND); // enables display only
        lcd_write(0x01, COMMAND); // clears display and moves cursor to upper left
        wait_ticks(20000); // wait 4ms
}

void display_string(char *str) {
        int len = strlen(str);

        for (int i = 0; i < len; i++) {
                char cur_char = str[i];
                lcd_write(cur_char, DATA);
        }
}

void display_char(char c) {
        lcd_write(c, DATA);
}

void display_digits(int time_unit) {
        display_char(time_unit/10 + '0');
        display_char(time_unit%10 + '0');
}

/*
===============================================================================
 Name        : melody.c
 Author      : Sarah Brown
 Description : Various functions to output songs from speaker for alarm
===============================================================================
*/

#include "define.h"

extern int dac_waveup;
extern int wave_amp;
extern int counter;
extern int freq;

// uses fet to disable speaker amp power
void enable_speaker() {
        FIO0PIN &= ~(1 << SPEAKER_EN_PIN); // 0 enables
}

// uses fet to disable speaker amp power
void disable_speaker(){
        FIO0PIN |= (1 << SPEAKER_EN_PIN); // 1 disables
}
```

```
// sets the note freq
void play_freq() {
        rit_disable();
        RICOMP = freq;
        wave_amp = 1023;
        counter = 0;
        RICOUNTER = 0;
        rit_enable();
}

// plays a note of a given frequency and size. or is silent for the length of a rest note
void play_note(int note_freq, int note_size, int rest) {
        if (rest) {
                wave_amp = 1023;
                if (note_size == WHOLE_NOTE) {
                        while (wave_amp != 0) wave_amp--;
                }

                else if (note_size == HALF_NOTE) {
                        while (wave_amp > HALF_NOTE) wave_amp--;
                }

                else if (note_size == QUARTER_NOTE) {
                        while (wave_amp > QUARTER_NOTE) wave_amp--;
                }

                else if (note_size == EIGTH_NOTE) {
                        while (wave_amp > EIGTH_NOTE) wave_amp--;
                }
        }

        else {
                freq = note_freq;
                play_freq();

                if (note_size == WHOLE_NOTE) {
                        while (wave_amp != 0);
                }

                else if (note_size == HALF_NOTE) {
                        while (wave_amp > HALF_NOTE);
                }

                else if (note_size == QUARTER_NOTE) {
                        while (wave_amp > QUARTER_NOTE);
                }

                else if (note_size == EIGTH_NOTE) {
                        while (wave_amp > EIGTH_NOTE);
                }
        }
}

// plays the song oklahoma
void play_song_oklahoma() {
        enable_speaker();
        play_note(FREQ_C, WHOLE_NOTE, FALSE); // OOOOk
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // la
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // hom
        play_note(FREQ_E, QUARTER_NOTE, FALSE); // a
        play_note(FREQ_D, QUARTER_NOTE, FALSE); // where
        play_note(FREQ_C, QUARTER_NOTE, FALSE); // the
        play_note(FREQ_D, HALF_NOTE, FALSE); // wind
        play_note(FREQ_G, HALF_NOTE, FALSE); // comes
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // sweep
        play_note(FREQ_F_SHARP, QUARTER_NOTE, FALSE); // in'
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // down
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // the
        play_note(FREQ_G, WHOLE_NOTE, FALSE); // plain
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // and
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // the
```

```
        play_note(FREQ_E, HALF_NOTE, FALSE); // wav
        play_note(FREQ_G, HALF_NOTE, FALSE); // in'
        play_note(FREQ_G, HALF_NOTE, FALSE); // wheat
        play_note(FREQ_E, QUARTER_NOTE, FALSE); // can
        play_note(FREQ_D, HALF_NOTE, FALSE); // sure
        play_note(FREQ_F, HALF_NOTE, FALSE); // smell
        play_note(FREQ_F, HALF_NOTE, FALSE); // sweet
        play_note(FREQ_E, QUARTER_NOTE, FALSE); // when
        play_note(FREQ_D, QUARTER_NOTE, FALSE); // the
        play_note(FREQ_E, HALF_NOTE, FALSE); // wind
        play_note(FREQ_F, HALF_NOTE, FALSE); // comes
        play_note(FREQ_G, HALF_NOTE, FALSE); // right
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // be
        play_note(FREQ_B, QUARTER_NOTE, FALSE); // hind
        play_note(FREQ_C_OCT5, QUARTER_NOTE, FALSE); // the
        play_note(FREQ_D_OCT5, WHOLE_NOTE, FALSE); // rain
        disable_speaker();
}

// plays the pokemon theme song
void play_song_pokemon(){
        enable_speaker();
        play_note(FREQ_A, EIGTH_NOTE, FALSE); // i wanna be the very best like no one ever was
        play_note(FREQ_A, EIGTH_NOTE, FALSE); // wan
        play_note(FREQ_A, EIGTH_NOTE, FALSE); //na
        play_note(FREQ_A, QUARTER_NOTE, FALSE); //be
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // the
        play_note(FREQ_G, QUARTER_NOTE, FALSE);// ver
        play_note(FREQ_E, QUARTER_NOTE, FALSE); //y
        play_note(FREQ_C, QUARTER_NOTE, FALSE); // best
        play_note(FREQ_C, QUARTER_NOTE, FALSE); // like
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // no
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // one
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // ev
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // er
        play_note(FREQ_G, HALF_NOTE, FALSE); // was
        play_note(FREQ_G, WHOLE_NOTE, TRUE); // REST
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // to
        play_note(FREQ_A_SHARP, QUARTER_NOTE, FALSE); // catch
        play_note(FREQ_B, QUARTER_NOTE, FALSE); // them
        play_note(FREQ_B, QUARTER_NOTE, FALSE); // is
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // my
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // real
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // test
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // to
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // train
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // them
        play_note(FREQ_G, QUARTER_NOTE, FALSE); // is
        play_note(FREQ_F, QUARTER_NOTE, FALSE); // my
        play_note(FREQ_A, WHOLE_NOTE, FALSE); // cause
        play_note(FREQ_A, HALF_NOTE, TRUE); // REST
        play_note(FREQ_A, EIGTH_NOTE, FALSE); // po
        play_note(FREQ_C_OCT5, EIGTH_NOTE, FALSE); // ke
        play_note(FREQ_D_OCT5, EIGTH_NOTE, FALSE); // mon
        play_note(FREQ_A, EIGTH_NOTE, TRUE); // REST
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // got
        play_note(FREQ_A, QUARTER_NOTE, FALSE); // ta
        play_note(FREQ_B, QUARTER_NOTE, FALSE); // catch
        play_note(FREQ_D_OCT5, QUARTER_NOTE, FALSE); // em
        play_note(FREQ_D_OCT5, QUARTER_NOTE, FALSE); // all
        disable_speaker();
}

/*
================================================================================
 Name        : timer.c
 Author      : Sarah Brown
 Description : Various functions for use of rit and rtc
================================================================================
*/
```

```c
#include "define.h"

extern int dac_waveup;
extern int wave_amp;
extern int counter;
extern int freq;


void clk_init() {
        // step 0 - enable main oscillator
        SCS |= (1 << 5);
        while (!(SCS & (1 << 6)));

        // step 1 - disconnect PLL0 with one feed sequence
        PLL0CON &= ~(1 << 1);
        PLL_FEED();

        // step 2 - disable PLL0 with one feed sequence
        PLL0CON &= ~(1);
        PLL_FEED();

        // step 3 - change cpu clock divider to speed up operation without PLL0, if desired
        // step 4 - write to the clock source selection control register to change the clock source if needed
        CLKSRCSEL = 1;

        // step 5 - write to PLL0CFG and make it effective with one feed sequence (can only be updated when PLL0
disabled)
        PLL0CFG = (N << 16) | M; // write MSEL0 to bits 14:0 and write NSEL0 to bits 23:16 in PLL0CFG
        PLL_FEED();

        // step 6 - enable PLL0 with one feed sequence
        PLL0CON |= (1 << 0);
        PLL_FEED();

        // step 8 - wait for PLL0 to achieve lock
        // step 8 is swapped with 7 per rec of powerpoint slides
        while (!(PLL0STAT & (1 << 26)));

        // step 7 - change CPU clock divider setting for operation with PLL0
        CCLKCFG = K;

        // step 9 - connect PLL0 with one feed sequence
        PLL0CON |= (1 << 1) | (1 << 0);
        PLL_FEED();
}

void rit_init() {
        PCLKSEL1 |= (01 << 26); //pclk = clk, page 59
        PCONP |= (1 << 16);
        RICOMP = freq;
        RICTRL |= (1<<3) | (1<<1);
        RICOUNTER = 0;
}

void rit_enable() {
        ISER0 |= (1<<29);
}

void rit_disable() {
        ISER0 &= ~(1<<29);
}

void dac_init() {
        PINSEL1 = (PINSEL1 & ~(0b11 << DAC_PIN)) | (0b10 << DAC_PIN); // clears pins and then sets them to function 10
}

void RIT_IRQHandler() {
        dac_waveup = !dac_waveup;

        if (dac_waveup) {
                DACR = (wave_amp << 6);
```

```
        }

        else {
                DACR = 0;
        }

        if (counter == 5 && wave_amp > 10) {
                wave_amp -= 10;
                //wave_amp *= 0.99;
                counter = 0;
        }

        else if (wave_amp < 10 && wave_amp > 0) {
                wave_amp = 0;
        }

        counter++;
        RICTRL |= 1; // clears the interupt
}

void rtc_enable() {
        PCONP |= (1 << 9); // turns on power to rtc
        CCR &= ~(0 << 0); // time counter is disabled so it may be initialized
        SEC = 0;
        MIN = 0;
        HOUR = 0;
        CCR |= (1 << 0); // time counter is enabled so it may be initialized
        ISER0 |= (1 << 17); // enable rtc interupts, table 52
        CIIR = 1; // generate an interupt every second
}

/*
 * Function to waste time
 *
 * @param ticks - x variable in function
 * y = 2*10^-7*x+7*10^-7
 */
void wait_ticks(int ticks) {
        volatile int count;
        for (count=0; count<ticks; count++) {
                //do nothing
        }
}
```

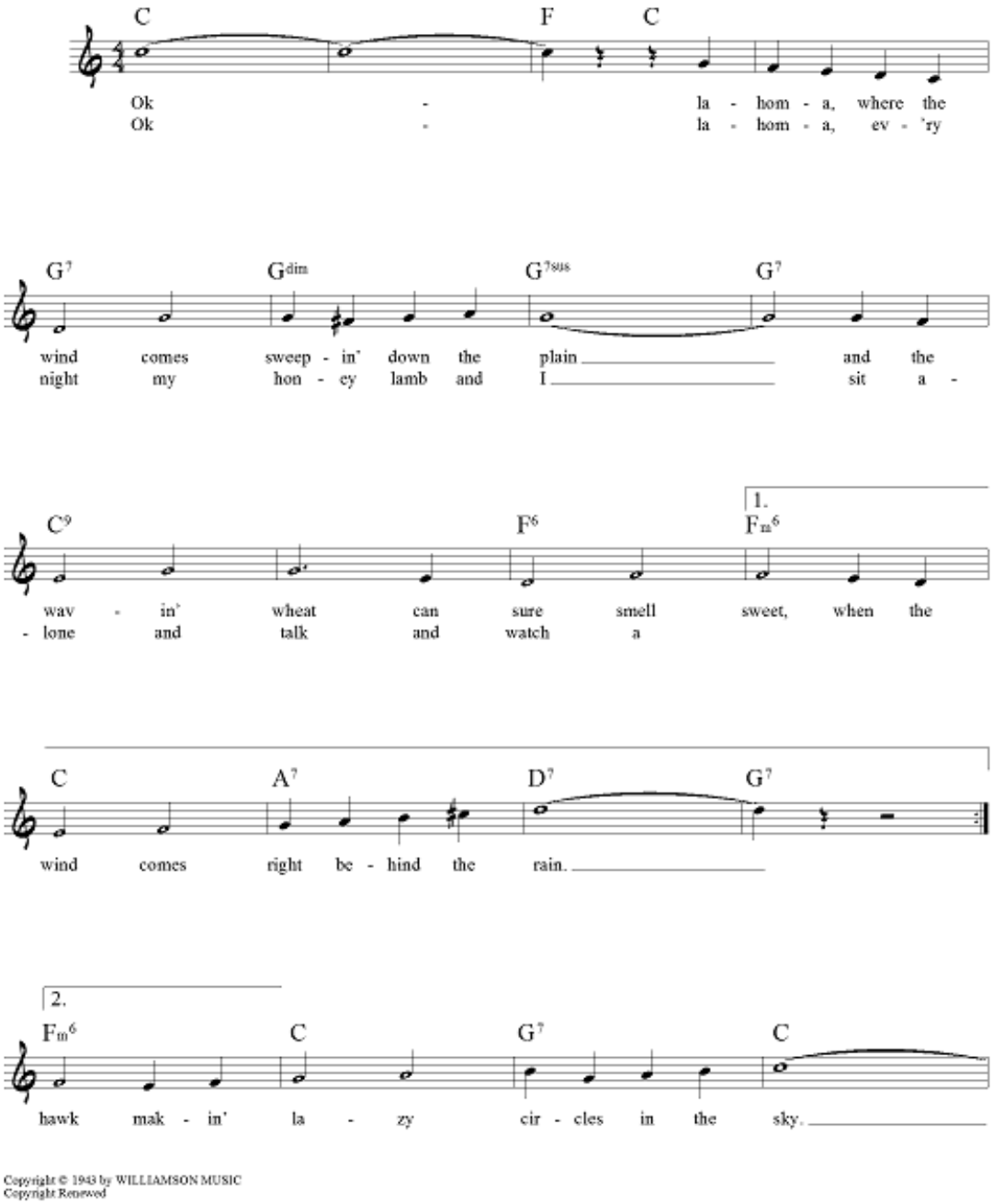
3) Sheet Music

OKLAHOMA

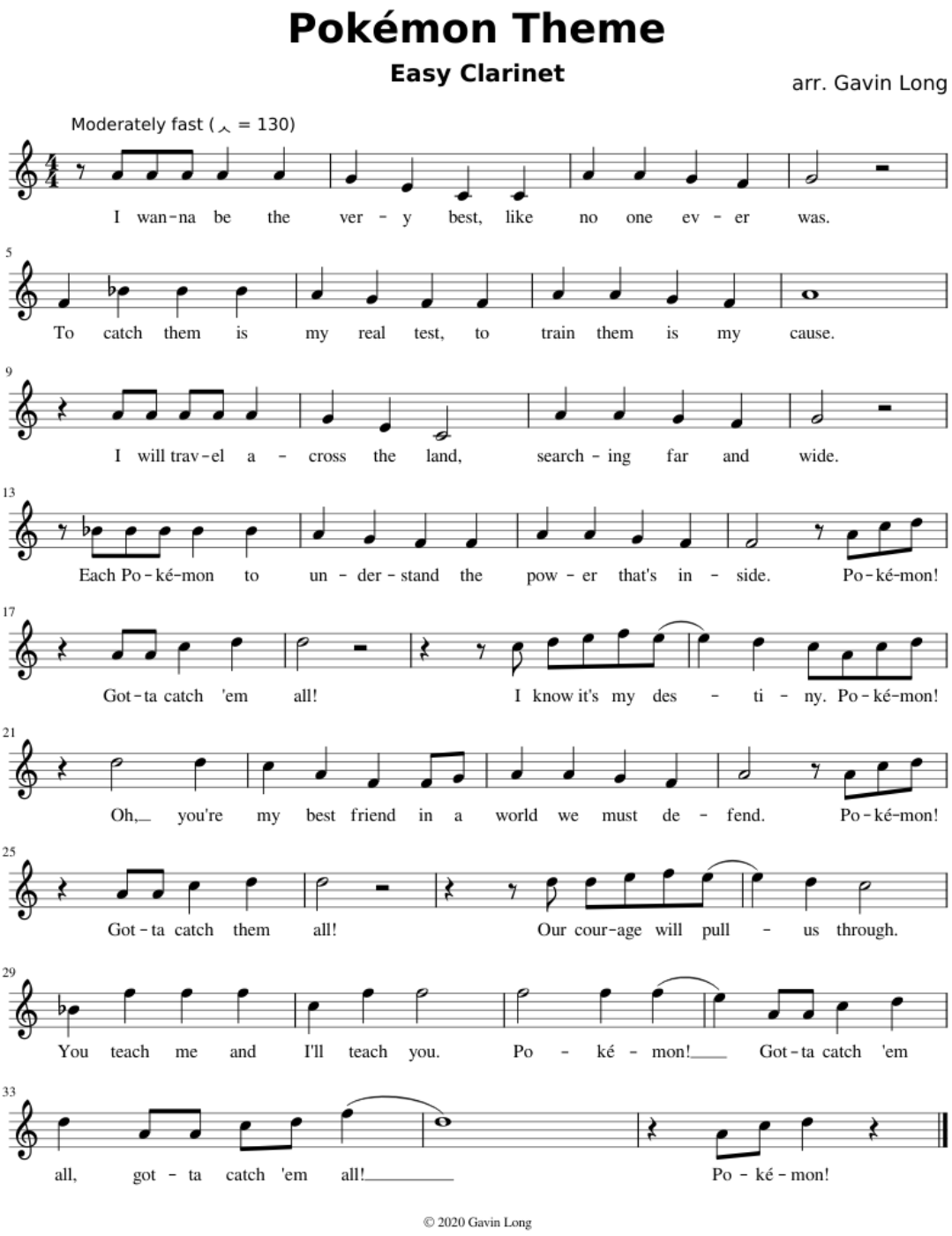
Lyrics by
OSCAR HAMMERSTEIN II

Music by
RICHARD RODGERS

Brightly
This big ENSEMBLE number is begun by AUNT ELLER.



Appendix 3, Figure 1: Oklahoma Sheet Music from https://www.sheetmusicnow.com/products/oklahoma-p456931



Appendix 3, Figure 2: Pokemon Theme Song Sheet Music from https://musescore.com/gavin/pokemon-theme