Main Program

```
/*
	CE2812 Lab 5
	Songs
	Evan Heinrich
	1/14/2022
*/

/*
 * My personal experience with this lab
 *
 * This lab wasn't too terribly difficult to implement since we had already
 * worked with interrupts in the previous quarter. The hardest parts for me
 * personally were debugging the ISR that I wrote. I met with Dr. Rothe during
 * one of his office hours, and it turns out that the main issue I was facing
 * was due to not clearing the status register for timer 5, causing note timing
 * issues. Besides this, I also had messed up the RCC struct. I had thought I
 * incorrectly laid out the register map in the struct, but it turns out that I
 * just had the incorrect base address for the RCC.
 */

#include <music.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "uart_driver.h"
#include "delay.h"
#include "music.h"
#include "memconsole.h"

#define F_CPU 16000000UL

// Rip and tear until it is done
#define doomTempo 1500000 // Technically this track should be 240bpm but this sounds right
static note atDoomsGate[] = {
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{C, 5, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{B, 4, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{A, 4, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{Fs, 4, doomTempo>>3},
			{0, 0, doomTempo>>5},
			{F, 2, doomTempo>>3},
```

```
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{A, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{C, 5, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{B, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{Fs, 4, (doomTempo>>2)+(doomTempo>>3)},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{C, 5, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{B, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{A, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{Fs, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{F, 2, doomTempo>>3},
{0, 0, doomTempo>>5},
{G, 4, doomTempo>>3},
{0, 0, doomTempo>>5},
```

```
            {A, 4, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {C, 5, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {B, 4, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {G, 4, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {F, 2, doomTempo>>3},
            {0, 0, doomTempo>>5},
            {Fs, 4, (doomTempo>>2)+(doomTempo>>3)},
            {0, 0, 0}
};

// "130bpm"
#define zeldaTempo 800000        // This totally isn't 130bpm but it sounds right
#define betweenNotes 46000
static note zelda[] = {
            {A, 4, zeldaTempo>>1},
            {0, 1, zeldaTempo>>2},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 0, betweenNotes},
            {B, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 1, zeldaTempo>>2},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
            {0, 0, betweenNotes},
            {B, 4, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {A, 4, zeldaTempo>>2},
```

```
{0, 1, zeldaTempo>>2},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{B, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>2},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>2},
{0, 0, betweenNotes},
{E, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>3},
{0, 0, betweenNotes},
{A, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{B, 4, zeldaTempo>>4},
{0, 0, betweenNotes},
{C, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{D, 5, zeldaTempo>>4},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>1},
{0, 1, zeldaTempo>>3},
{D, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{E, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{F, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{G, 5, zeldaTempo>>3},
{0, 0, betweenNotes},
{B, 6, zeldaTempo>>5},
{0, 0, betweenNotes},
{D, 6, zeldaTempo>>3},
{0, 0, betweenNotes},
```

```
            {B, 6, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {G, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {F, 5, zeldaTempo>>1},
            {0, 0, betweenNotes},
            {G, 5, zeldaTempo>>4},
            {0, 1, zeldaTempo>>4},
            {F, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>1},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>2},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {F, 5, zeldaTempo>>1},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>3},
            {0, 0, betweenNotes},
            {C, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {D, 5, zeldaTempo>>4},
            {0, 0, betweenNotes},
            {E, 5, zeldaTempo>>1},
            {0, 0, 0}
};
```

```
// main
int main(void){
	init_usart2(115200,F_CPU);

	delay_Init();

	music_Init();

	// Blank string for input
	char input[30] = "";

	// Address to interact with
	uint32_t* address = 0;

	// Command variable
	int command = -1;

	// Last argument, either length to read or value to write
	uint32_t argument = 0;

	// Welcome message
	printf("Evan's Memory Management Console\n\r");
	printf("Type \'?\' for help\n\r");

	// Infinite loop for program
	while(1==1) {
		// Prompt
		printf("> ");
		fgets(input, 29, stdin);
		// First token, determines command
		char* token = strtok(input, " ");

		// Second token, determines address
		char* arg1 = strtok(NULL, " ");

		// Third token, optional third argument, required for wmw, optional for dm
		char* arg2 = strtok(NULL, " ");

		// If there is an extracted command
		if(token != NULL) {
			// Attempt to parse the command
			command = parseCommand(token);

			// Attempt to parse address
			if(arg1 != NULL) {
				address = parseAddress(arg1);
			}

			// Attempt to parse second argument
			if(arg2 != NULL) {
				argument = parseArgument(arg2);
			}

			// Switch case for reported commands
			switch (command) {
			// Help command
			case 0:
				help();
				break;

			// Dump memory command
```

```
case 1:
	if(arg1 != NULL) {
		if(arg2 == NULL) {
			memdmpDefault((uint8_t*)address);
		} else {
			memdmp((uint8_t*)address, argument);
		}
	} else {
		printf("No address provided\n\r");
	}
	break;

// Read word command
case 2:
	if(arg1 != NULL) {
		memwrd(address);
	} else {
		printf("No address provided\n\r");
	}
	break;

// Write word command
case 3:
	if(arg1 != NULL) {
		if(arg2 != NULL) {
			wmemwrd(address, argument);
		} else {
			printf("No value to write provided\n\r");
		}
	} else {
		printf("No address provided\n\r");
	}
	break;

// Music command
case 4:
	// Determine song to be played
	if(strcmp(arg1, "doom") == 0 || strcmp(arg1, "doom\n") == 0) {

		// Play background/foreground accordingly
		if(strcmp(arg2, "background\n") == 0) {
			music_Background(atDoomsGate);
		} else {
			music_Play(atDoomsGate);
		}

	} else if(strcmp(arg1, "zelda") == 0 || strcmp(arg1, "zelda\n") == 0) {

		// Play background/foreground accordingly
		if(strcmp(arg2, "background\n") == 0) {
			music_Background(zelda);
		} else {
			music_Play(zelda);
		}

	} else {
		printf("Invalid song\n");
	}
	break;
```

```
                default:
                        printf("Invalid command\n\r");
                }

        } else {
                printf("No input\n\r");
        }

        // fgets again because it will read the newline from previous entry
        fgets(input, 29, stdin);

        // Clear the input string
        memset(input, 0, strlen(input));

        }

    exit(EXIT_SUCCESS);
    return 0;
}
```

Updated Music API (for some reason, special formatting was lost)

```
// Kinda recursive-y since that's the header for this but
// I need the anonymous struct for notes
#include "music.h"
#include "registers_new.h"
#include "delay.h"
#include <stdio.h>

/**
 * Global variable that holds the next note index for the
 * current background song
 */
static uint32_t musicIndex;


/**
 * Global variable that points to the current song playing
 */
static note* backgroundSong;

/**
 * Flag variable to show if a note is playing in the background
 */
static uint8_t backgroundPlaying = 0;

/**
 * Plays individual notes using a busy wait
 */
void note_Play(uint32_t period, uint32_t duration) {
	volatile TIMER* TIM3 = (TIMER*) TIM3_BASE;
	if(period == 0) {
		// If period = 0, just do a delay
		delay_us(duration);
	} else {
		// Load period / 2 into CCR
		TIM3->CCR1 = period >> 1;

		// Load period - 1 into ARR
		TIM3->ARR = period - 1;

		// Start playing the note
		TIM3->CR1 |= 1;

		// Delay for the proper time
		delay_us(duration);

		// Stop playing note
		TIM3->CR1 &= ~(1);
	}

	return;
}
```

```
/**
 * Initializes the appropriate timers and GPIO port
 */
void music_Init() {
	// Pointers for all used peripherals
	// TIM5 will be a simple down-counter which will trigger interrupts
	//     each time a note completes playing in the background.
	volatile RCC* RCC_Target = (RCC*) RCC_BASE;
	volatile GPIO* GPIOB = (GPIO*) GPIOB_BASE;
	volatile TIMER* TIM3 = (TIMER*) TIM3_BASE;
	volatile TIMER* TIM5 = (TIMER*) TIM5_BASE;
	volatile NVIC* NVIC_Target = (NVIC*) NVIC_BASE;

	// Enable GPIOB in RCC
	RCC_Target->AHB1ENR |= RCC_GPIOBEN;

	// Enable TIM3 and TIM5 in RCC
	RCC_Target->APB1ENR |= (RCC_TIM3EN | RCC_TIM5EN);

	// Set PB4 to alternate function
	GPIOB->MODER = GPIO_ALTFUN << 8;

	// Set AFRL such that PB4 is connected to TIM3
	GPIOB->AFRL = PB4_PIEZO;

	// Set TIM3 & TIM5 prescale to 16, AKA 1 count = 1us
	TIM3->PSC = 16;
	TIM5->PSC = 16;

	// Prescale fix
	// Forces an event to be generated and then
	// clears it right away which tricks the timer
	// into applying the prescale somehow
	TIM3->EGR = 1;
	TIM3->SR &= ~(1);
	TIM5->EGR = 1;
	TIM5->SR &= ~(1);

	// Configure CCMR for PWM mode
	TIM3->CCMR1 |= (OC1M_PWM | OC1M_PE);

	// Enable in CCER
	TIM3->CCER |= CCER_CC1E;

	// Assert not counting in CR1
	TIM3->CR1 &= ~(TIM_CEN);
	TIM5->CR1 &= ~(TIM_CEN);

	// Enable interrupts for TIM5
	TIM5->DIER |= TIM_UIE;

	// Set TIM5 to count-down and one-pulse
	TIM5->CR1 |= (TIM_OPM | TIM_DIR);

	// Enable the TIM5 interrupt in ISER, slot 50
	//     aka 1<<18 in ISER1
	NVIC_Target->ISER[1] = 1<<18;

	return;
}
```

```
/**
 * Iterates through the provided array of notes until a null
 * note is found
 */
void music_Play(const note song[]) {
	// Index counter
	int i = 0;

	if(backgroundPlaying == 0) {
		// Loop through array until we find a note with 0 period and 0 length
		while(!(song[i].period == 0 && song[i].length == 0)) {
			// Attempt to put these in a register for passing to the funct
			register uint32_t length = song[i].length;
			register uint32_t period = song[i].period;

			uint32_t octave = song[i].octave;
			period = period >> octave;
			note_Play(period, length);
			i++;
		}
	} else {
		printf("Cannot play a song while one is playing in the background\n");
	}

	return;
}

/**
 * Stops a background song that is playing
 */
void music_StopBackground() {
	volatile TIMER* TIM3 = (TIMER*) TIM3_BASE;
	volatile TIMER* TIM5 = (TIMER*) TIM5_BASE;

	// Stop the timers
	TIM5->CR1 &= ~(TIM_CEN);
	TIM3->CR1 &= ~(TIM_CEN);

	// Reset controls
	backgroundPlaying = 0;
	musicIndex = 0;

	return;
}
```

```
/**
 * Starts the provided song playing in the background
 */
void music_Background(note song[]) {
	volatile TIMER* TIM3 = (TIMER*) TIM3_BASE;
	volatile TIMER* TIM5 = (TIMER*) TIM5_BASE;

	// Stop background song
	music_StopBackground();

	// Set flag to show music is playing
	backgroundPlaying = 1;

	// Set new address and reset note index
	// note index gets set to one because this method
	// plays the 0th note on its own, then the interrupts
	// take over
	backgroundSong = song;
	musicIndex = 1;

	// Load the period and length
	uint32_t period = song[0].period;
	period = period >> song[0].octave;
	uint32_t length = song[0].length;

	// Write values
	TIM5->CNT = length;
	TIM3->ARR = period - 1;
	TIM3->CCR1= period >> 1;

	// Start timers
	TIM5->DIER|= TIM_UIE;
	TIM3->CR1 |= TIM_CEN;
	TIM5->CR1 |= TIM_CEN;

	return;
}
```

```
/**
 * ISR
 * Triggers when TIM5 hits zero;
 * loads next note from the global song array into appropriate
 * timers and increments the global note index
 */
void TIM5_IRQHandler() {
	volatile TIMER* TIM3 = (TIMER*) TIM3_BASE;
	volatile TIMER* TIM5 = (TIMER*) TIM5_BASE;


	// Clear the register that triggers interrupts
	TIM5->SR = 0;

	// Stop the frequency timer
	TIM3->CR1 &= ~(TIM_CEN);

	// Load current note info
	note current = backgroundSong[musicIndex];
	uint32_t period = current.period;
	period = period >> current.octave;
	uint32_t length = current.length;

	// If the note isn't a terminator
	if(!(period == 0 && length == 0)) {
		// Write the period values accordingly
		TIM3->ARR = period - 1;
		TIM3->CCR1 = period >> 1;
		// Write the note duration
		TIM5->CNT = length;

		// Increment the note counter
		musicIndex++;

		// Enable both timers
		TIM3->CR1 |= TIM_CEN;
		TIM5->CR1 |= TIM_CEN;

	// If the note is a terminator
	} else if(period == 0 && length == 0) {
		// Clear the note counter
		musicIndex = 0;

		// Clear the flag for background songs
		backgroundPlaying = 0;

		// Disable timer 5's interrupts
		TIM5->DIER &= ~(TIM_UIE);

		// Clear the timers
		TIM3->ARR = 0;
		TIM3->CCR1 = 0;
		TIM5->CNT = 0;
	}

	return;
}
```