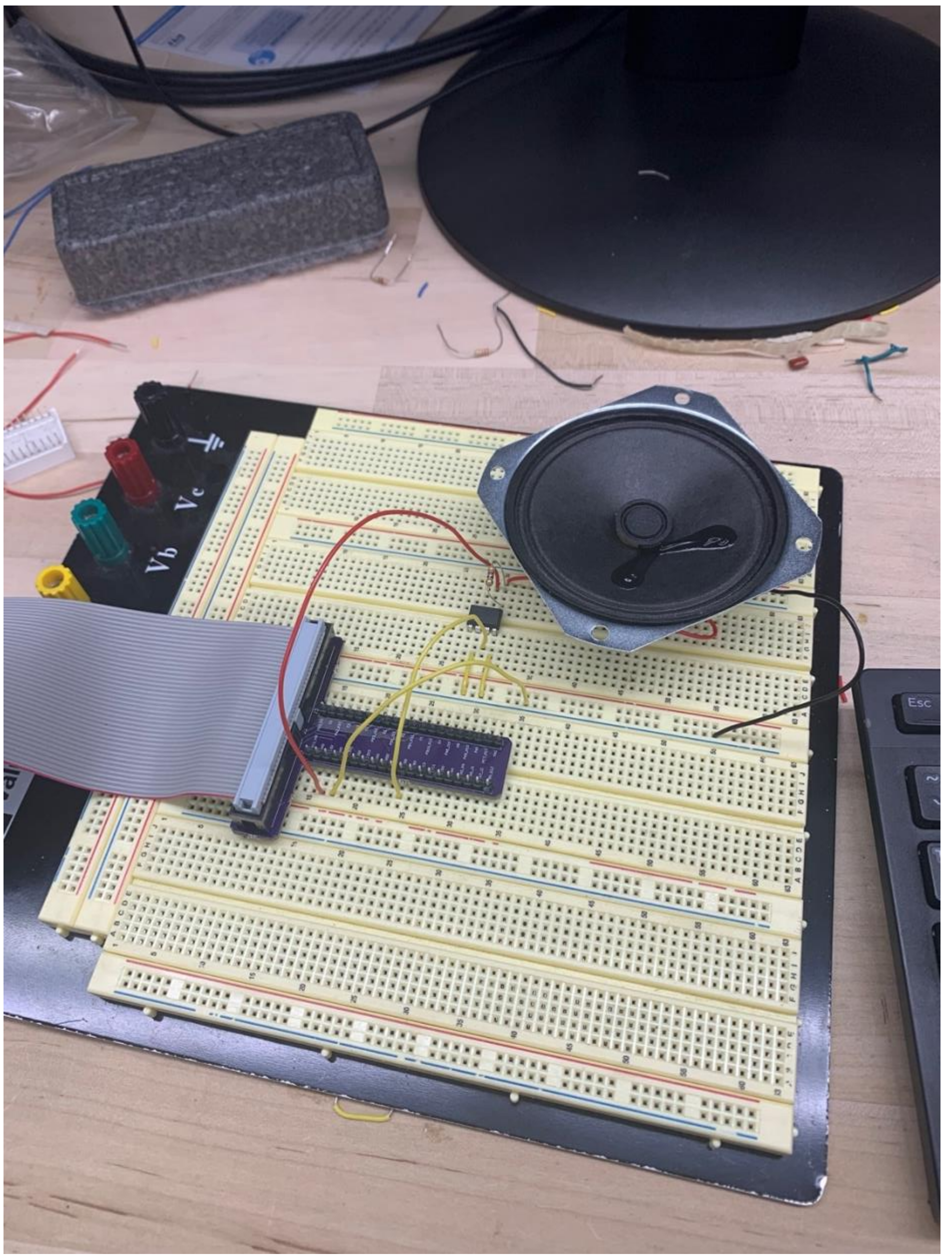
Digital Audio

Engineering 155 Lab IV Report
October 5th, 2021

John Hearn



Introduction:

The goal of this lab was to build a circuit to enable an I/O pin from the MCU to drive an 8-ohm speaker. We were to implement the timer functionality for this purpose by reading the datasheet and writing our own library in C using PlatformIO. Using this design, we were to read in an array of notes in the form of frequencies and durations to play Fur Elise.

Design Methodology:

The main component of the design for this lab came from the software side. It was determined that it would be effective to use two timers, TIM10 and TIM11, for calculating both the duration of the notes being played and the pitches that they would be played at.

It was necessary to create structs to access each field of the timers, shown in the Technical Documentation section. Looking through the reference manual, it was determined that the fields used within the configPitchT() and configLenT() functions were necessary for the configuration of the two timers.

Because TIM10 was used for the pitches, it determined that the timer frequency should be scaled down to 14 MHz from 84 MHz to produce a frequency able to be picked up by humans. To achieve this, we divided 84 MHz by 6, which meant setting the prescaler to 5, as shown below in Figure 1.

range of freq. for humans: 220Hz – 880Hz

$CNT = 2^{16}$;

$2^{16} \cdot x = 220Hz$] What frequency x to count by to count up to 2^{16} at 220Hz?

$x = 14.4\,MHz \rightarrow 14MHz$

$$PSC = \frac{84MHz}{14MHz} - 1 = 6 - 1$$

$PSC = 5$

Fig. 1: PSC calculation TIM10

TIM11 was used for the note durations and similarly, it was determined that 1 MHz was the easiest frequency to make calculations with. We divided 84 MHz by 84 to achieve this frequency, meaning a prescaler value of 83. The calculation is shown below in Figure 2.

given range of durations for notes: 2 Hz - 32 Hz.

$CNT = 2^{16};$

$$2^{16} \cdot \frac{1}{x} = 500ms$$

$$\frac{1}{x} = \frac{500ms}{2^{16}}$$

$$x = 2^{17} + 1$$

Figure 2: PSC calculation TIM11

After this, it was necessary to create functions updating the duration and pitch based on the notes passed in. These functions were used within a for-loop that iterated through all the notes to play the correct note for the correct durations.

Technical Documentation:

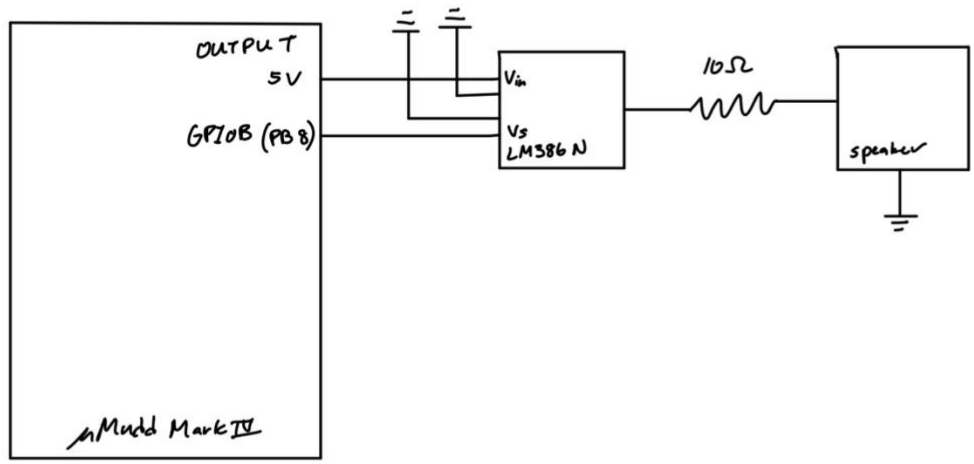


Figure 3: Circuit schematic

```c
// STM32F401RE_GPIO.c
// Source code for GPIO functions

#include "STM32F401RE_GPIO.h"

void pinMode(GPIO * GPIOX, int pin, int function) {
    switch(function) {
        case GPIO_INPUT:
            GPIOX->MODER &= ~(0b11 << 2*pin);
            break;
        case GPIO_OUTPUT:
            GPIOX->MODER |= (0b1 << 2*pin);
            GPIOX->MODER &= ~(0b1 << (2*pin+1));
            break;
        case GPIO_ALT:
            GPIOX->MODER &= ~(0b1 << 2*pin);
            GPIOX->MODER |= (0b1 << (2*pin+1));
            break;
        case GPIO_ANALOG:
            GPIOX->MODER |= (0b11 << 2*pin);
            break;
    }
}

int digitalRead(GPIO * GPIOX, int pin) {
    return ((GPIOX->IDR) >> pin) & 1;
}

void digitalWrite(GPIO * GPIOX, int pin, int val) {
    GPIOX->ODR |= (1 << pin);
}

void togglePin(GPIO * GPIOX,  int pin) {
    // Use XOR to toggle
    GPIOX->ODR ^= (1 << pin);
}
```

Figure 4: GPIO.c file

```c
// STM32F401RE_RCC.c
// Source code for RCC functions

#include "STM32F401RE_RCC.h"

void configurePLL() {
    // Set clock to 84 MHz
    // Output freq = (src_clk) * (N/M) / P
    // (8 MHz) * (336/16) / 4 = 42 MHz
    // M:16, N:336, P:4, Q:7
    // Use HSE as PLLSRC

    RCC->CR.PLLON = 0; // Turn off PLL
    while (RCC->CR.PLLRDY != 0); // Wait till PLL is unlocked (e.g., off)

    // Load configuration
    RCC->PLLCFGR.PLLSRC = PLLSRC_HSE;
    RCC->PLLCFGR.PLLM = 8;
    RCC->PLLCFGR.PLLN = 336;
    RCC->PLLCFGR.PLLP = 0b01;
    RCC->PLLCFGR.PLLQ = 4;

    // Enable PLL and wait until it's locked
    RCC->CR.PLLON = 1;
    while(RCC->CR.PLLRDY == 0);
}

void configureClock(){
    // Turn on and bypass for HSE from ST-LINK
    RCC->CR.HSEBYP = 1;
    RCC->CR.HSEON = 1;
    while(!RCC->CR.HSERDY);

    // Configure and turn on PLL
    configurePLL();

    // Select PLL as clock source
    RCC->CFGR.SW = SW_PLL;
    while(RCC->CFGR.SWS != 0b10);
}
```

Figure 5: RCC.c file

```c
// STM32F401RE_FLASH.c
// Source code for FLASH functions

#include "STM32F401RE_FLASH.h"

void configureFlash() {
    FLASH->ACR.LATENCY = 2; // Set to 2 waitstates
    FLASH->ACR.PRFTEN = 1; // Turn on the ART
}
```
Figure 6: Flash.c file

```c
#include "STM32F401RE_TIM.h"


void configPitchT() {

    TIM10->cr1Bits.ARPE = 1;



    TIM10->ccmr1Bits.OC1M = 0b110;

    TIM10->ccerBits.CC1E = 1;

    // TIM10->cntBits.CNT = 0b0000;

    TIM10->pscBits.PSC = 0b0101;

    TIM10->arrBits.ARR = 0xFFFF;

    // TIM10->ccr1Bits.CCR1 = 32767;
    uint32_t* access = (uint32_t*) 0x40014434UL;
    *access = 32767;
    // ((uint32_t*)(0x40014400 + 0x34))* = 32767;
    TIM10->egrBits.UG = 1;
    TIM10->cr1Bits.CEN = 1;
}

void configLenT() {
    TIM11->cr1Bits.CEN = 1;
    TIM11->cr1Bits.ARPE = 1;

    TIM11->egrBits.UG = 1;

    TIM11->ccmr1Bits.OC1M = 0b110;

    TIM11->ccerBits.CC1E = 1;

    // TIM11->cntBits.CNT = 0b0000;

    TIM11->pscBits.PSC = 640;

    TIM11->arrBits.ARR = 0xFFFF;

    TIM11->ccr1Bits.CCR1 = 32767;
}
```

Figure 7: TIM.c file

```c
// Pass in the duration, do math
int newDuration(int duration) {
    TIM11->cntBits.CNT = 0;
    TIM11->egrBits.UG = 1;
    return (131072 * .001 * duration);
}

// Pass in the frequency of the note, do math
void changePitch(int freq) {
    volatile int newCount = (freq != 0) ? (14e6 / freq) : 0;
    TIM10->cntBits.CNT = 0;
    TIM10->arrBits.ARR = newCount;
    TIM10->ccr1Bits.CCR1 = newCount / 2;  // The duty cycle is 50%
    TIM10->egrBits.UG = 1;
}

int main(void) {
    // Enable the GPIOA pins, both timers (for pitch and duration)
    RCC->AHB1ENR.GPIOAEN = 1;
    RCC->AHB1ENR.GPIOBEN = 1;
    RCC->APB2ENR.TIM10EN = 1;
    RCC->APB2ENR.TIM11EN = 1;

    // Set output pins
    GPIOB->AFRH |= (1 << 1) | (1 << 0);

    // Pinmodes
    pinMode(GPIOB, 8, GPIO_ALT);

    // configure flash, clock, pitch & length timers
    configureFlash();
    configureClock();
    configPitchT();
    configLenT();

    volatile int currDuration;
    for(volatile int i = 0; i < 108; i++) {
        currDuration = newDuration(notes[i][1]);
        changePitch(notes[i][0]);
        while(TIM11->cntBits.CNT < currDuration);
    }
    changePitch(0);
    newDuration(0);
    while(1);
}
```

Figure 8: main.c file

Results and Discussion:

The song played. The ending note lingers but otherwise it plays perfectly with no issues of tone.

If I were to redo this lab I would start earlier. This lab took everything from me to complete.

Conclusion:

I was successful in creating a system that plays Fur Elise through the speaker. This lab took 21 hours over two days to complete.