# *Assignment 5: Lab Report*

Digital Design Lab

Sarah Brown

## I. Introduction

This lab focused on using the digital-to-analog and analog-to-digital subsystems of the LPC1769. These subsystems were used to implement a controller for a house entryway. This house entryway features a doorbell and an automatic light based on ambient light levels. The doorbell has two chimes that resonate at different frequencies that decay overtime to 0 amplitude. The automatic light uses a phototransistor to determine lighting conditions and turns on when it is dark and off when it is light. These values are determined with two separate thresholds to allow for hysteresis of the system.

## II. Hardware Solution

The hardware solution for this lab focuses mainly on the audio amplifier with the LM386 as well as the phototransistor hooked up to the LPC1769.

470 ohms was selected for the green LED to meet its current limits. 100k ohms was selected for the photoresistor to meet its current limits and as it is a standard pull-up resistor. The values in the amplifier were selected as standard values for this design.

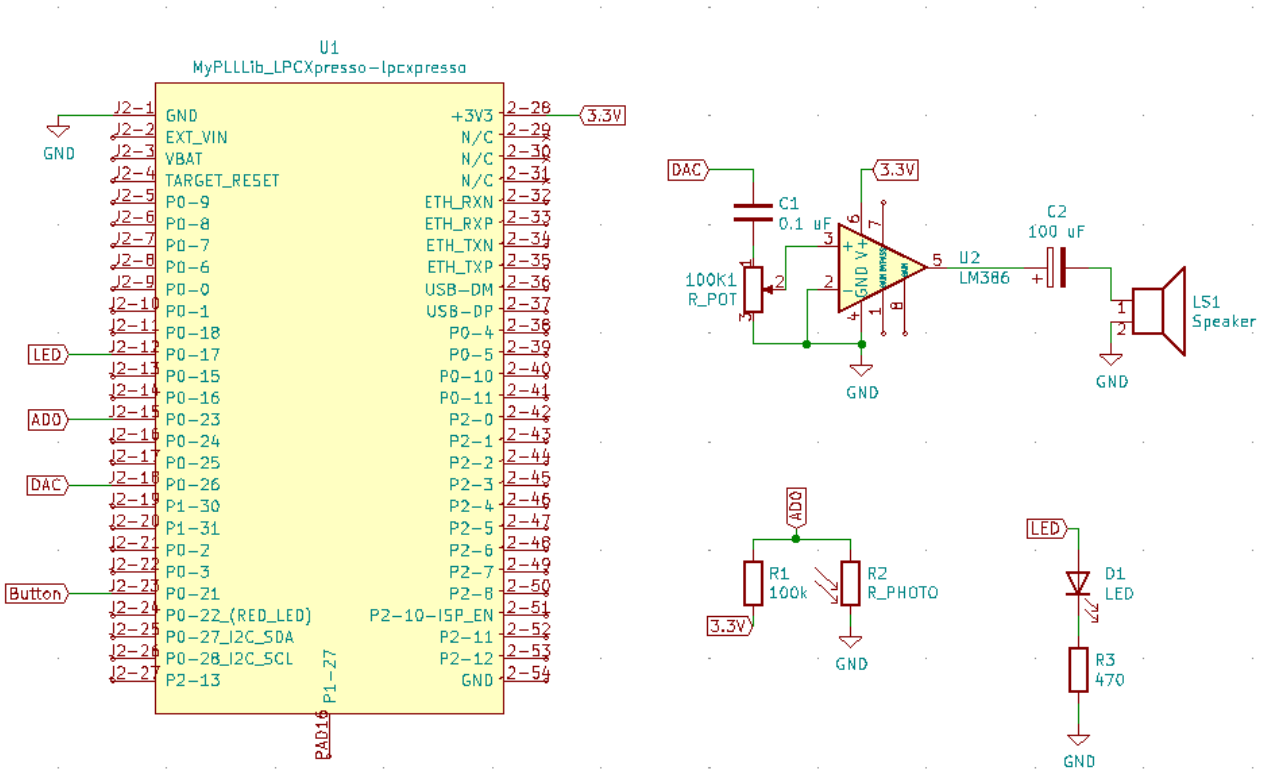


*Figure 1: Schematic of hardware design*

## III. SOFTWARE SOLUTION

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.

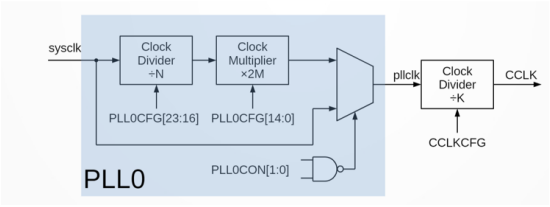


*Figure 2: Details about the internal PLL in the LPC1769.*

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.

To produce the 659 Hz and the 523 Hz signals, interrupts were used with the 100 MHz clock as an input. Using the 100 MHz clock allowed for more precise output signals. By using the interrupts, the interrupt

handler is called which allows for a function to be called. This function toggles a bit back and forth to produce a square wave by turning the DACR to the wave amplitude or 0 otherwise. This function has a counter section that decays the amplitude overtime.

The light is measured using the analog-to-digital subsystem, which was configured following the steps in the lecture slides. By use of the debugger function of the LPC1769, different lighting conditions were tested to determine appropriate thresholds for the LED to turn on and off with my lighting setup. The use of two thresholds ensures that the LED does not immediately turn on or off when the conditions change slightly.

Below are screenshots of the scope for the doorbell chimes.

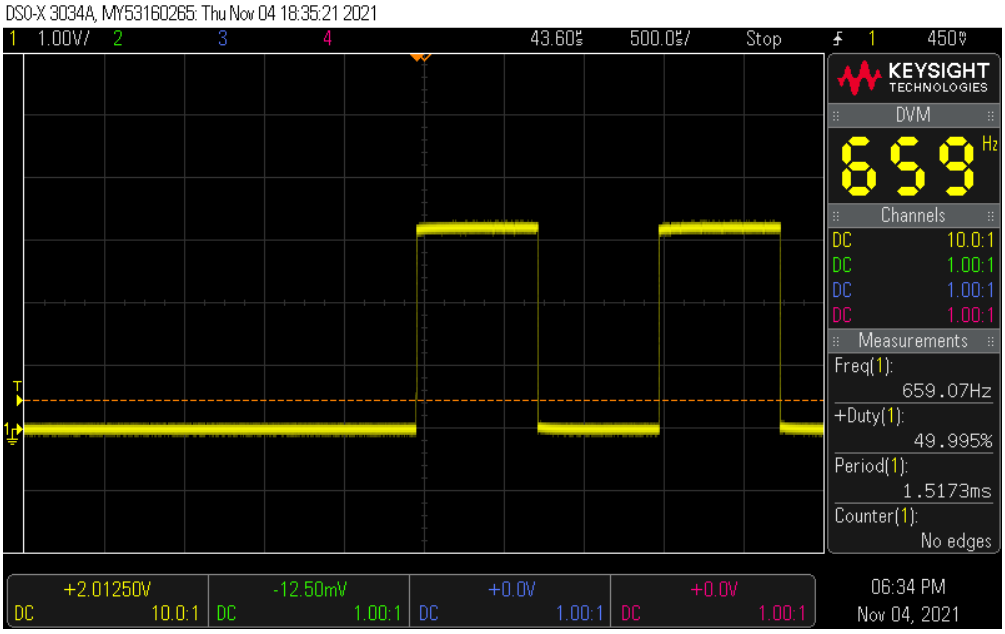


*Figure 3: Doorbell chime 1: 659 Hz*

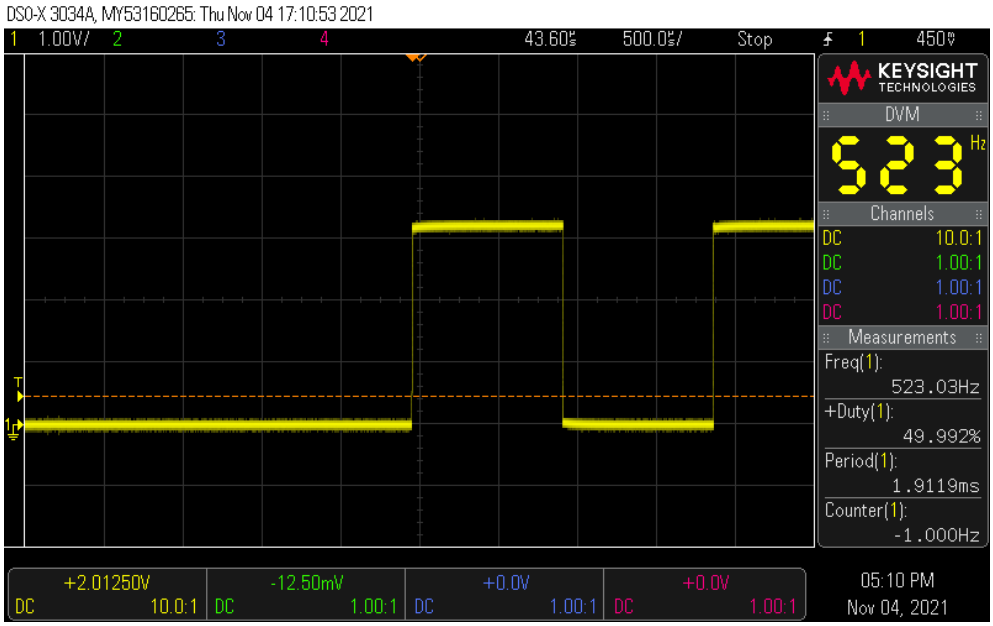


*Figure 4: Doorbell chime 2: 523 Hz*

These doorbells chimes are required to be within 1% of the value which can be determined with the following formula:

$$\left|\frac{Actual - Expected}{Expected}\right| \cdot 100\%$$

Doorbell chime 1: 659 Hz

$$\left|\frac{659.07 - 659}{659}\right| \cdot 100\% = 0.01\%$$

Doorbell chime 1: 523 Hz

$$\left|\frac{523.03-523}{523}\right| \cdot 100\% = 0.006\%$$

Therefore, both doorbell chimes are within the acceptable error range.

In addition, a zoomed-out scope shows that the doorbell chimes decay linearly over the time period of a few seconds.

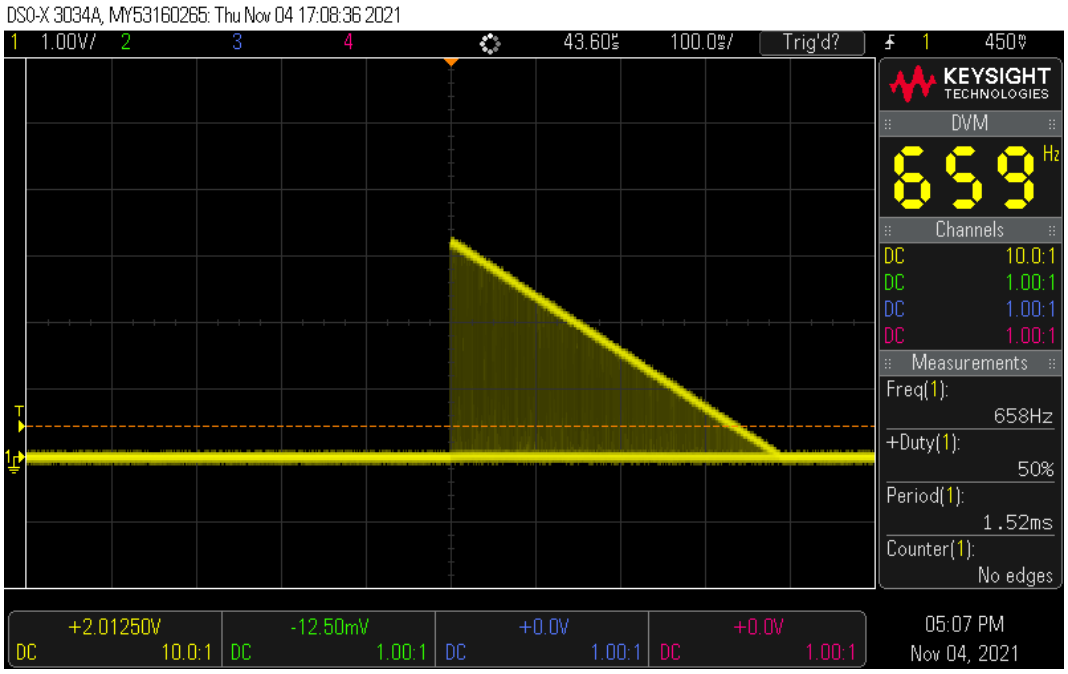


*Figure 5: Doorbell chime 1 decaying*


*Figure 5: Doorbell chime 2 decaying*

# IV. Appendix

## A. Code

```
// 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 ISERO0 (*(volatile unsigned int *) 0xE000E100)
#define DACR (*(volatile unsigned int *) 0x4008C000)
#define AD0CR (*(volatile unsigned int *) 0x40034000)
#define PCONP (*(volatile unsigned int *) 0x400FC0C4)
#define AD0DR0 (*(volatile unsigned int *) 0x40034010)

#define MATCH_FREQ_659 75873
#define MATCH_FREQ_523 95602

// define pin registers
#define PINSEL1 (*(volatile unsigned int *) 0x4002C004)
#define PINSEL3 (*(volatile unsigned int *) 0x4002C00C)
#define FIO0DIR (*(volatile unsigned int *)0x2009c000)
#define FIO0PIN (*(volatile unsigned int *)0x2009c014)

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

#define K 2
#define M 24
#define N 1

#define DAC_PIN 20 // pin p0.26
#define AD0_PIN 14 // pin p0.23
#define BUTTON_PIN 21 // P0.22
```

```
#define LED_PIN 17 // P0.17

#define LIGHT_THRESH_OFF 300
#define LIGHT_THRESH_ON 700


#define PRESSED 0 // active low
#define UNPRESSED 1

int dac_waveup = 0;
int wave_amp = 0;
int counter = 0;

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); // page 59
         PCONP |= (1 << 16);
         RICOMP = MATCH_FREQ_659;
         RICTRL |= (1<<3) | (1<<1);
         RICOUNTER = 0;
         ISERO0 |= (1<<29);
}

void rit_enable() {
         ISERO0 |= (1<<29);
}

void rit_disable() {
         ISERO0 &= ~(1<<29);
}

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

void adc_init() {
         // set PCONP bit 12
         PCONP |= (1 << 12);

         // set AD0CR bit 21
         AD0CR |= (1 << 21);

         // change AD0CR to reduce conversion clock to less than 13 MHz
         AD0CR |= (10 << 8);
```

```
        // enable ADC0 pin0
        PINSEL1 = (PINSEL1 & ~(0b11 << AD0_PIN)) | (0b01 << AD0_PIN); // clears pins and then sets them to function 11
}

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 IO_init() {
        FIO0DIR &= ~(1 << BUTTON_PIN); // configure port 0 bit 22 as input
    FIO0DIR |= (1 << LED_PIN); // configure port
}

int button_press() {
        int button = (FIO0PIN>>BUTTON_PIN) & 1; // configure port 0 bit 22 as input
        return button;
}

int measure_light() {
    AD0CR |= (1 << 24); // starts conversion
    while(!((AD0DR0 >> 31) & 1)); // waits for conversion to finish
    int light_measure = ((AD0DR0 >> 4) & (0x7FF)); // reads data, 0x7FF masks the 11 bits
    AD0CR &= ~(1 << 24); // stops conversion
    return light_measure; // returns value
}

int main(void) {
        clk_init();
        dac_init();
        adc_init();
        rit_init();
        IO_init();

        int button = UNPRESSED; int prev_button = UNPRESSED; int light = 0;

        //CLKOUTCFG = (1 << 8) | (10 << 4); // enables clkout for debugging reasons and divides it by 10 for readability
        //CLKOUTCFG = (1 << 8) | (9 << 4); // enables clkout for debugging reasons and divides it by 10 for readability
        //PINSEL3 |= (01<<22); // configure clkout as output

    while(1) {
        button = button_press();

        light = (99*light + measure_light())/100;

        if (light <= LIGHT_THRESH_OFF) { // turn off
           FIO0PIN &= ~(1 << LED_PIN);

        }

        else if (light >= LIGHT_THRESH_ON){ // turn on
                  FIO0PIN |= (1 << LED_PIN);
        }

        if (button == PRESSED && prev_button == UNPRESSED) {
                rit_disable();
                RICOMP = MATCH_FREQ_659;
                wave_amp = 1023;
                counter = 0;
                RICOUNTER = 0;
                rit_enable();
```

```
        }

        else if (button == UNPRESSED && prev_button == PRESSED){
                  rit_disable();
                  RICOMP = MATCH_FREQ_523;
                  wave_amp = 1023;
                  counter = 0;
                  RICOUNTER = 0;
                  rit_enable();
        }

        prev_button = button;
    }
}
```