# Final report – ee.ele.250

Rasmus Sorila
rasmus.sorila@tuni.fi
150267845

## Description

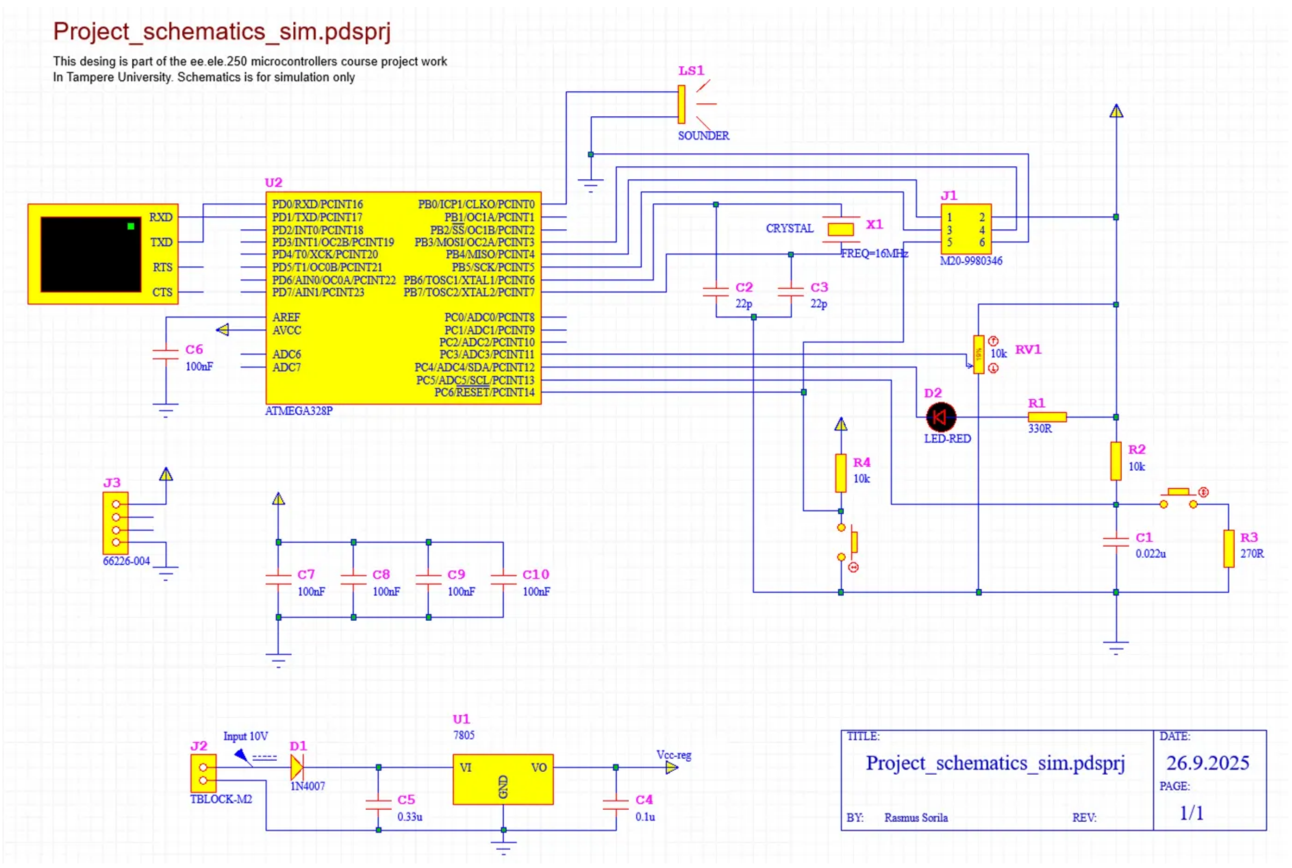
The purpose of this project is to design, program, and simulate a microcontroller-based board using the ATmega328P. The system generates an audible signal with a frequency range of 50–1000 Hz, which can be adjusted either with a 10 kΩ potentiometer or by sending commands via the terminal.

## Schematics

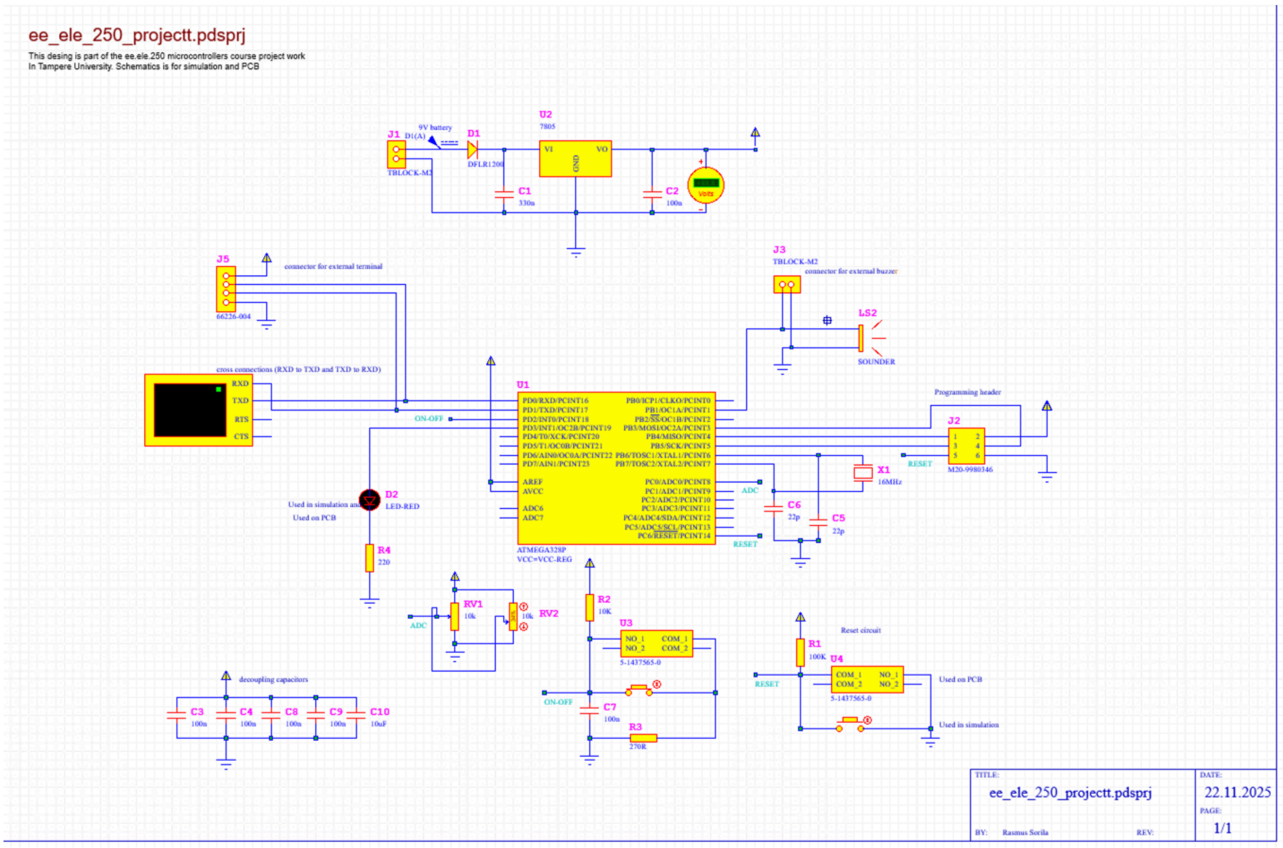
The circuit is powered by a 7805-voltage regulator, which provides a stable 5 V supply. A 16 MHz crystal oscillator with two 22 pF capacitors is used to generate the system clock.
A sounder is connected to the microcontroller output and serves as the main output device for frequency generation. Additional components include:

- Reset button with pull-up resistor
- Soft on/off button
- Red LED with series resistor for status indication
- ISP header (2x3) for programming
- Serial communication header (1x4)

This project is implemented entirely in Proteus simulation. For better visualization, interactive components such as the potentiometer, LED, button, and sounder are selected from the Proteus virtual/animated library. First simulation (Photo 1) works only for simulation, but while prepping schematic for PCB layout I found a way to create schematic that works in sim and PCB (Picture 2). In schematic 2 there is also used modularity for better readability and some pin changes for easier access.



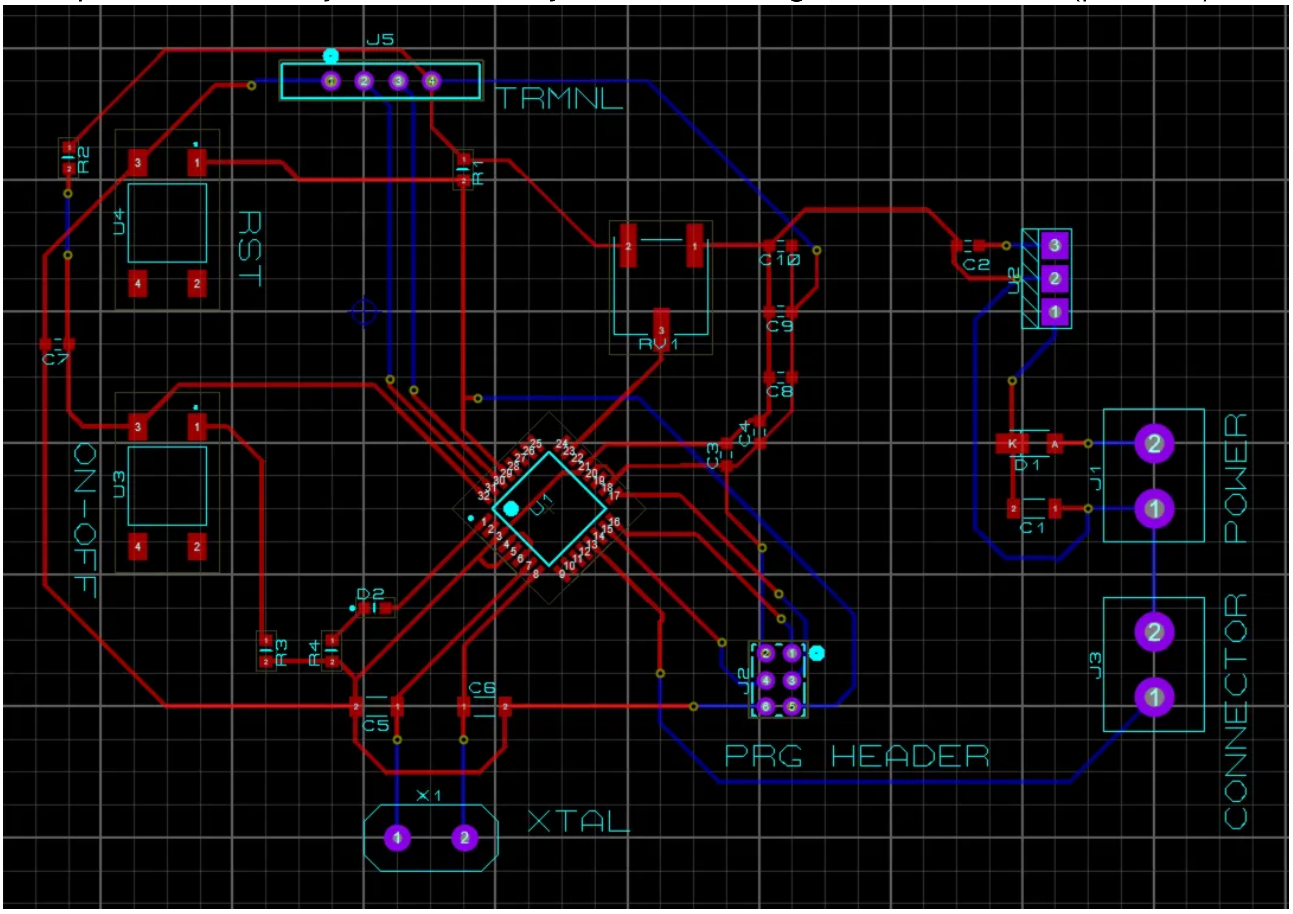
picture 1: first schematic (only for simulation use)



Picture 2: Schematic 2 (sim & PCB)

## PCB layout

The layout process began by correctly identifying and assigning all microcontroller pins according to the schematic and documentation. The ATmega328P was placed at the center of the board to minimize routing complexity and ensure short, direct connections to components. Once the microcontroller was positioned, the associated components such as the crystal oscillator with its load capacitors, reset circuitry, and voltage regulator were arranged close to their corresponding pins. Resistors, capacitors, and connectors were then placed logically around the board. After component placement, routing was performed by connecting signal traces. The programming header, terminal connectors, and on/off control were positioned for easy access. PCB layout is made using second schematic (picture 2).



Picture 3: PCB layout

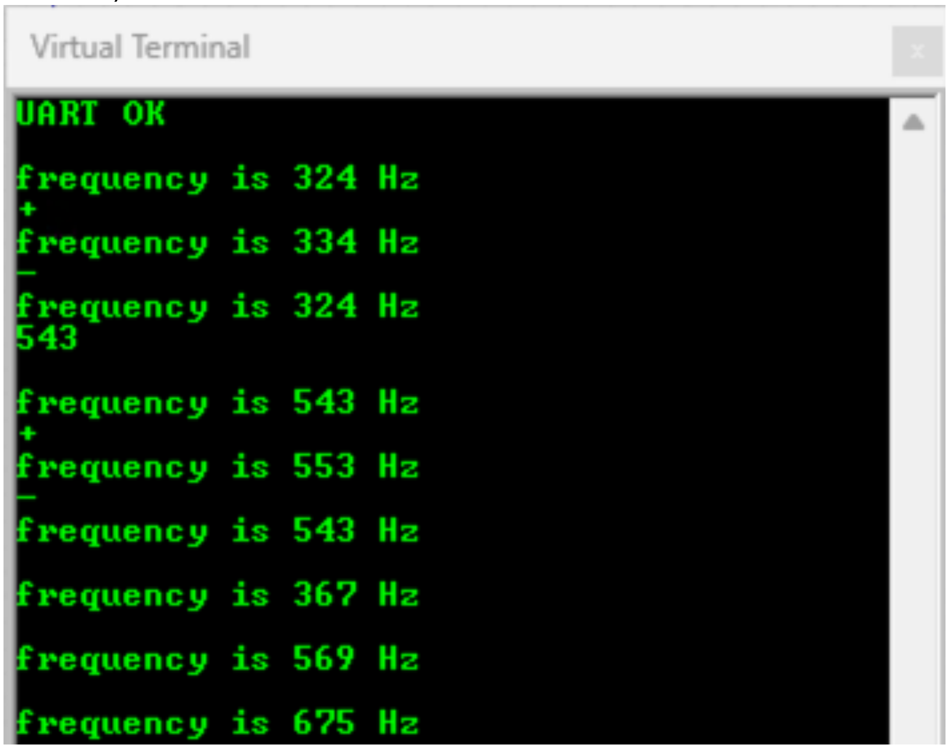
## Programming

The program generates a square wave sound with a variable frequency that can be controlled either by a potentiometer or via UART commands, and it also drives an LED with a fading effect. Originally there was a problem, the system allowed only one control source at a time the potentiometer or UART input because the frequency was assigned directly from one source, overriding the other. This was corrected by introducing a base frequency determined primarily from the potentiometer, with any UART exact frequency input or fine adjustments applied on top of it. As a result, the potentiometer and UART controls now work together, and the + and - keys on UART can adjust the frequency without interfering with the other inputs.

The ADC reads the potentiometer and scales its value to a frequency range of 50–1000 Hz. Timer1 generates a square wave at the calculated frequency, while Timer2 produces a PWM signal for the LED, synchronized with sound activity. An on/off button is monitored; when the button is off, the system stops sound and LED output and enters a low-power sleep mode. When the switch is turned on, normal operation resumes.

The current frequency is sent to the UART terminal every two seconds, allowing real-time monitoring in the Proteus virtual terminal. In picture 4 is virtual terminal, first frequency is changed by terminal (+/-) commands, after that accurate frequency is settled by typing it (543) in terminal and confirmed by enter, last the frequency is changed using potentiometer.

Software's were used on TUNI virtual desktop and there are lags in demonstration video because of that, also the timing in simulation is much slower than real time (time can be seen in bottom left corner)



```
UART OK

frequency is 324 Hz
+
frequency is 334 Hz
-
frequency is 324 Hz
543

frequency is 543 Hz
+
frequency is 553 Hz
-
frequency is 543 Hz

frequency is 367 Hz

frequency is 569 Hz

frequency is 675 Hz
```

Picture 4: Virtual terminal

```c
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdint.h>

#define MIN_FREQ 50
#define MAX_FREQ 1000

#define SOUND_PIN PB1
#define LED_PIN   PD3
#define SWITCH_PIN PD2

#define SWITCH_IS_ON (!(PIND & (1<<SWITCH_PIN)))

// UART variables
volatile int16_t uart_adj = 0;       // fine adjustment via +/-
volatile int16_t uart_set_value = 0; // exact frequency set via UART
volatile uint8_t uart_updated = 0;

// Frequency and previous ADC value
volatile int16_t frequency = MIN_FREQ;
volatile int16_t adc_old = 0;

// --------------------------------------------------------
// UART initialization
// --------------------------------------------------------
void UART_Init(void) {
    uint16_t ubrr = F_CPU/16/9600-1;
    UBRR0H = ubrr>>8;
    UBRR0L = ubrr;
    UCSR0B = (1<<TXEN0)|(1<<RXEN0)|(1<<RXCIE0);
    UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}

void UART_SendText(const char *s) {
    while(*s) {
        while(!(UCSR0A & (1<<UDRE0)));
        UDR0 = *s++;
    }
}
```

```c
// ----------------------------------------------------------
// UART interrupt for fine adjustment (+/-) and exact frequency input
// ----------------------------------------------------------
ISR(USART_RX_vect) {
    static char buf[6];
    static uint8_t idx = 0;

    char c = UDR0;

    // Fine adjustment always applies
    if(c == '+') uart_adj += 10;
    else if(c == '-') uart_adj -= 10;

    // Number input for exact frequency
    else if(c >= '0' && c <= '9') {
        if(idx < sizeof(buf)-1) {
            buf[idx++] = c;
            buf[idx] = 0; // null terminate
        }
    }
    // Enter key signals end of input
    else if(c == '\r' || c == '\n') {
        uart_set_value = atoi(buf); // convert string to integer
        if(uart_set_value < MIN_FREQ) uart_set_value = MIN_FREQ;
        if(uart_set_value > MAX_FREQ) uart_set_value = MAX_FREQ;
        uart_updated = 1;
        idx = 0;
        buf[0] = 0;
    }
    else { // ignore other characters
        idx = 0;
        buf[0] = 0;
    }
}

// ----------------------------------------------------------
// ADC initialization and read
// ----------------------------------------------------------
void ADC_Init(void) {
    ADMUX = (1<<REFS0); // Avcc as reference
    ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); // prescaler 128
}

uint16_t ADC_Read(void) {
    ADCSRA |= (1<<ADSC);
    while(ADCSRA & (1<<ADSC));
    return ADC;
}
```

```c
// ---------------------------------------------------------
// Sound generation using Timer1
// ---------------------------------------------------------
void Sound_SetFreq(uint16_t f) {
    if(f < MIN_FREQ) f = MIN_FREQ;
    if(f > MAX_FREQ) f = MAX_FREQ;

    uint32_t ocr = F_CPU / (2UL ∗ 8 ∗ f) − 1;
    if(ocr > 0xFFFF) ocr = 0xFFFF;

    OCR1A = (uint16_t)ocr;
    TCCR1A = (1<<COM1A0);
    TCCR1B = (1<<WGM12)|(1<<CS11);
}

void Sound_Off(void) { TCCR1B = 0; }

// ---------------------------------------------------------
// LED PWM using Timer2
// ---------------------------------------------------------
void LED_PWM_Init(void) {
    TCCR2A = (1<<WGM21)|(1<<WGM20)|(1<<COM2B1);
    TCCR2B = (1<<CS22);
    OCR2B = 0;
}

// ---------------------------------------------------------
// External interrupt
// ---------------------------------------------------------
ISR(INT0_vect) {}

// ---------------------------------------------------------
// Power down function
// ---------------------------------------------------------
void power_low(void) {
    EICRA &= ~((1<<ISC01)|(1<<ISC00));
    EIFR = (1<<INTF0);
    EIMSK |= (1<<INT0);

    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_enable();
    sei();
    sleep_cpu();

    sleep_disable();
    EIMSK &= ~(1<<INT0);
}
```

```c
// ---------------------------------------------------------
// Main loop
// ---------------------------------------------------------
int main(void) {
    // Pin setup
    DDRB |= (1<<SOUND_PIN);
    DDRD |= (1<<LED_PIN);
    DDRD &= ~(1<<SWITCH_PIN);
    PORTD |= (1<<SWITCH_PIN);

    UART_Init();
    ADC_Init();
    LED_PWM_Init();
    sei();

    Sound_Off();
    OCR2B = 0;

    uint16_t timer_ms = 0;
    uint8_t led_level = 0;
    int8_t led_dir = 1;

    UART_SendText(”UART OK\r\n”);

    while(1) {
        // ---------------- Power off if switch released ----------------
        if(!SWITCH_IS_ON) {
            Sound_Off();
            OCR2B = 0;
            _delay_ms(10);
            power_low();
            _delay_ms(50);
            continue;
        }

        // ---------------- Read ADC ----------------
        int16_t adc_value = ADC_Read();
        int16_t adc_freq = (int32_t)adc_value * (MAX_FREQ-MIN_FREQ)/1023 + MIN_FREQ;
        // ---------------- Determine base frequency ----------------
        int16_t base_freq = adc_freq;

        // UART exact frequency has priority if potentiometer not moved significantly
        if(uart_updated) {
            if(abs(adc_freq – adc_old) <= 3) {
                base_freq = uart_set_value;
                } else {
                uart_updated = 0; // ADC takes over if potentiometer moved
            }
        }

        adc_old = adc_freq;

        // ---------------- Apply fine adjustment (+/–) on top of base ----------------
        frequency = base_freq + uart_adj;

        // Clamp frequency
        if(frequency < MIN_FREQ) frequency = MIN_FREQ;
        if(frequency > MAX_FREQ) frequency = MAX_FREQ;

        // ---------------- Apply sound ----------------
        Sound_SetFreq(frequency);

        // ---------------- LED fading blinking effect ----------------
        led_level += led_dir;
        if(led_level == 255 || led_level == 0) led_dir = –led_dir;
        OCR2B = led_level;

        // ---------------- Periodic UART output ----------------
        timer_ms += 10;
        if(timer_ms >= 2000) {
            char buf[16];
            itoa(frequency, buf, 10);
            UART_SendText(”\r\nfrequency is ”);
            UART_SendText(buf);
            UART_SendText(” Hz\r\n”);
            timer_ms = 0;
        }

        _delay_ms(10);
    }
}
```

Actual code

There is added extra white space in code for better readability in this document.