Ptuj, 10.1.2025

Tim Kosec

LED Light dimmer

Table of Contents

List of Figures

1 Introduction

This report describes the complete process of creating electronics from idea to finished product. It includes the design and development of a PCB for dimming DC-powered LED lights.

The first part explains the design theory. This section describes the process of going from problem to project idea and how to plan it. Then, the schematic design is described, including choosing the right components and designing the circuit in a CAD program. During this, issues like component availability and learning the Altium environment are addressed. Finally, the PCB is assembled and tested. A microcontroller program is also included in the report.

The whole process is quite straightforward, so anyone with some interest can build this project. It includes a wide range of knowledge, is fun to make, and useful in practice.

The report includes:

- Conceptual approach to implementing electronics
- Block diagram
- Component selection
- Schematic design
- Manufacturer design rule setup
- PCB layout
- Bill of Materials (BOM)
- Gerber file generation
- PCB soldering
- Microcontroller programming
- Electronics testing
- Program (C++)

Software used:

- Altium Designer
- Arduino IDE

Suppliers:

- Farnell
- JLC PCB

2 Design

2.1 Theory

For my project, I chose to design electronics for dimming DC LED lights. The reason was a practical issue in my room — poor lighting in the workspace and unwanted shadows. To fix this, I used a 12 V LED strip, placing it on optimal surfaces. While this improved lighting, full brightness wasn't always needed, hence the need for a dimmer.

A dimmer is a device that reduces voltage either digitally or analogically, thereby lowering the brightness of the light source. In my project, a digital method is used, described further below.

I needed a cost-effective, compact, and practical solution, so I decided to build the circuit myself. The first step was to understand how it would work in theory. I set a goal that the circuit should have an input interface, a display, and an output driver. The most logical input method was a simple, cheap, and precise solution — a potentiometer.

For the display, the same principles applied: a minimalist, affordable solution. I chose a 10-segment red LED bar graph. To control brightness, Pulse Width Modulation (PWM) was an obvious choice, offering high efficiency, compact size, and simple implementation (only one transistor required).

Once the main components were selected, I chose supporting parts. For control, I opted for a microcontroller due to its flexible logic capabilities. I selected the ATtiny family, specifically the ATtiny85, which was familiar and met all my requirements. A minor issue with the number of output pins was resolved using a SIPO (Serial-In Parallel-Out) register. I also chose to dim the display itself.

Naturally, the circuit needed power. The requirement was to power everything from a single 12 V source. To convert this to the microcontroller's voltage, I chose a digital step-down converter over a linear one for its better efficiency. It works using a PWM signal, inductor, and feedback loop, adjusting the pulse width to maintain the desired output voltage, which is then smoothed with capacitors. More details are found in the chip’s datasheet.

All selected components were checked on Farnell's website for availability and price, staying under the €20 budget. Part numbers were recorded for the BOM, which lists which parts, how many, and what values are needed to assemble the PCB.

2.2 Block diagram

The idea was also sketched as a block diagram. This provides a high-level overview of the circuit's functionality without specific components or connections. It's a minimal schematic useful for PCB design and understanding how it works. It's also helpful during troubleshooting.

The diagram includes inputs, signal processing stages, control circuitry, and outputs. Signal directions are also marked.

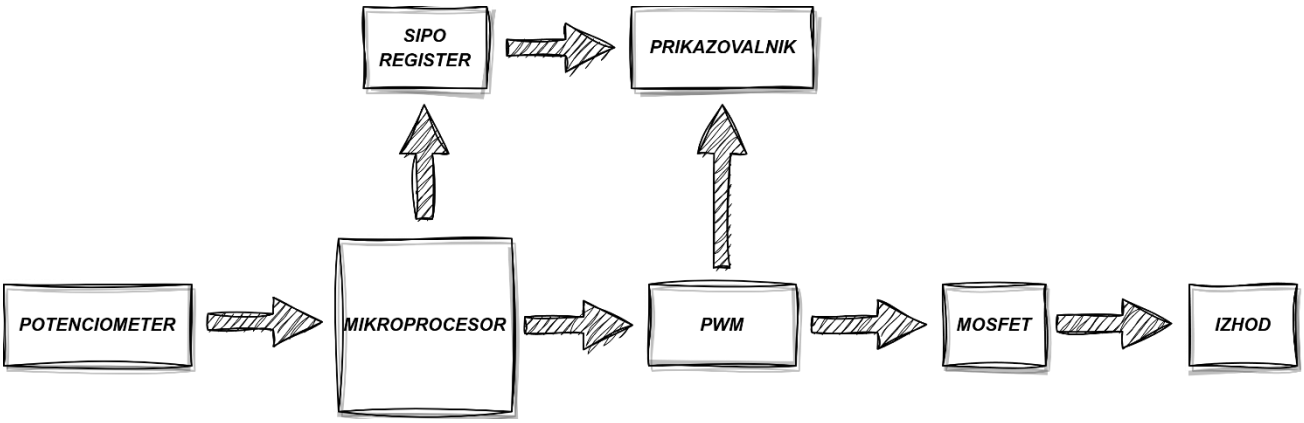


Figure 1 – Circuit block diagram

From the diagram, we see that the circuit is controlled via a potentiometer. It's connected to the microcontroller, which handles processing and control. The processor generates two PWM outputs: one connected to a power transistor (MOSFET) for LED dimming (since the MCU can’t source enough power alone), and the second for controlling the display brightness.

However, PWM alone doesn't control the display. The microcontroller sends data to the display via a SIPO register, which expands serial data into 8 parallel outputs — useful when the MCU has limited pins.

3 Altium designer

Altium Designer is a powerful PCB design tool that allows us to create the schematic, layout the PCB, and generate manufacturing files. In this section, I'll go through the key steps of designing the PCB using Altium Designer: creating the schematic, designing the PCB layout, generating the Bill of Materials (BOM), and exporting the Gerber files.

3.1 Schematics

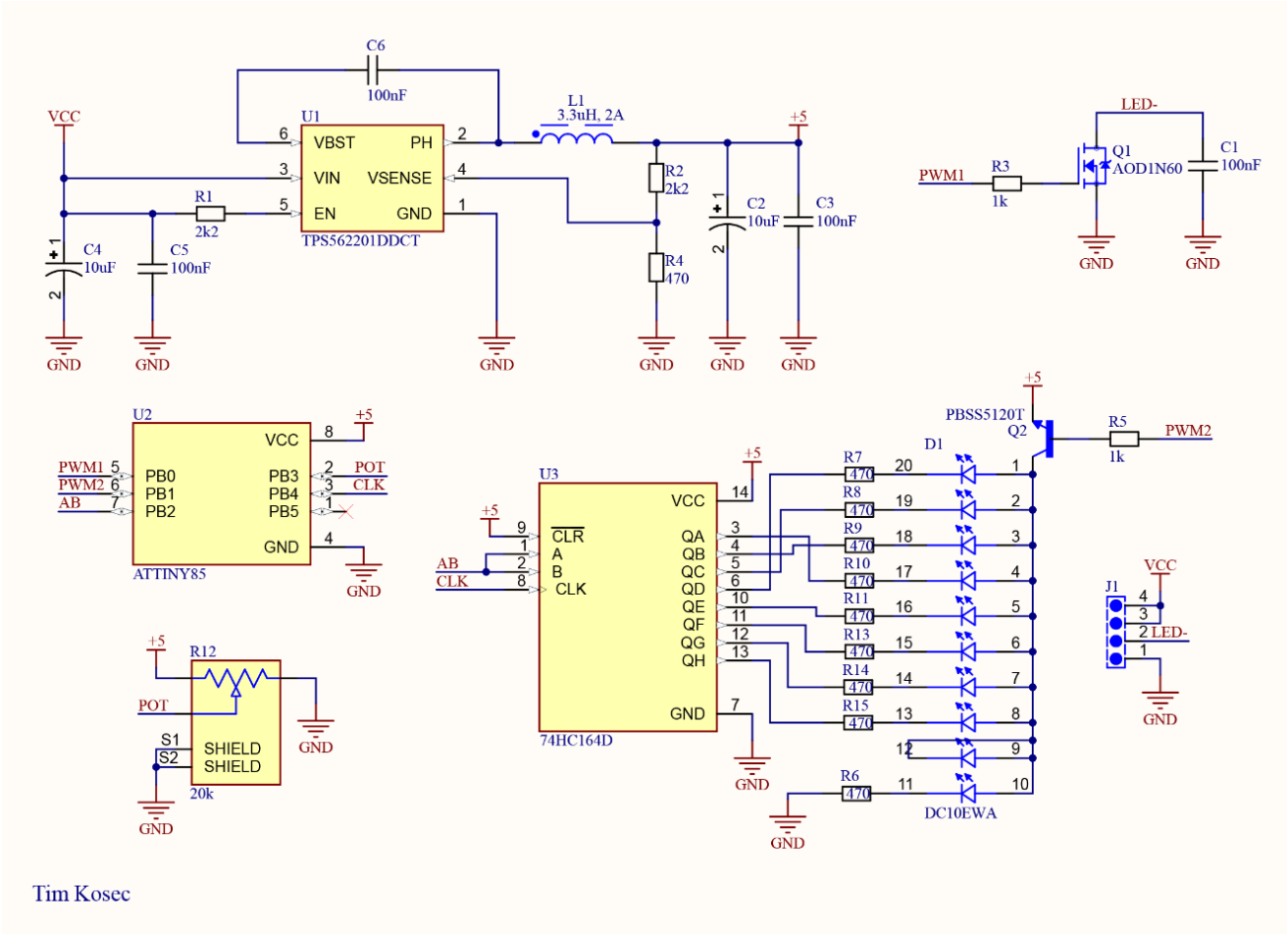
The schematic is the first step in the design process. It represents the electrical connections between components in the circuit and shows how the signal flows. For this project, I started by drawing the components and their connections.

The components in the schematic include:

- **ATtiny85**: A microcontroller for controlling the dimmer and the display.
- **MOSFET**: A transistor used to drive the LED strip by switching it on and off rapidly, dimming the light.
- **SIPO register**: This component is used to expand the available output pins of the microcontroller to control the LED display.
- **Potentiometer**: The input device for adjusting the brightness level.
- **LED bar graph**: Displays the brightness level.

To ensure correct component selection, I used the **component library** in Altium Designer, which provides pre-existing models for most commonly used parts. If necessary, custom components can be added to the library.

After placing the components in the schematic, I connected them according to the block diagram. The potentiometer was connected to an ADC pin of the ATtiny85 for reading the brightness level. The MOSFET's gate was controlled by the microcontroller's PWM output to adjust the LED brightness. The SIPO register was used to control the LED bar graph's segments in parallel.



Slika 1 Shema vezja z izbranimi vrednostmi elementov

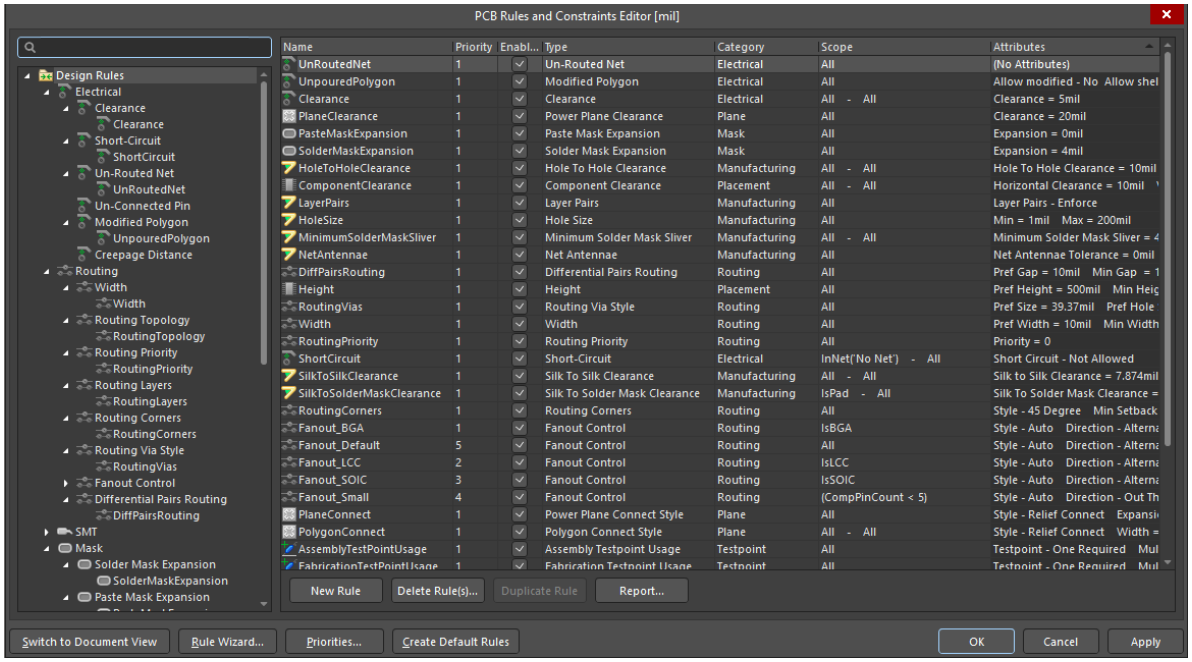
The schematic included all components and connections with their proper values, ensuring everything was wired correctly.

3.2 PCB

Once the schematic was completed and reviewed for correctness, it was time to move on to the PCB design. In Altium Designer, I created a new PCB layout and imported the schematic components to begin placing them on the board.

The main steps in designing the PCB were:

1. **Board Size**: I determined the dimensions of the PCB based on the available space in the project and the size of the components.
2. **Component Placement**: I placed the components on the PCB layout. The goal was to minimize the distance between components that would be frequently connected, such as the microcontroller and the MOSFET, to reduce the complexity of routing.
3. **Routing**: Once the components were placed, I routed the electrical connections. Altium's autorouter was used to generate the initial traces, but I manually adjusted them to optimize for size and signal integrity. The traces connecting high-current components, like the MOSFET, were made thicker to handle the increased current.
4. **Ground Plane**: A ground plane was used to reduce electromagnetic interference (EMI) and ensure stable signal operation.



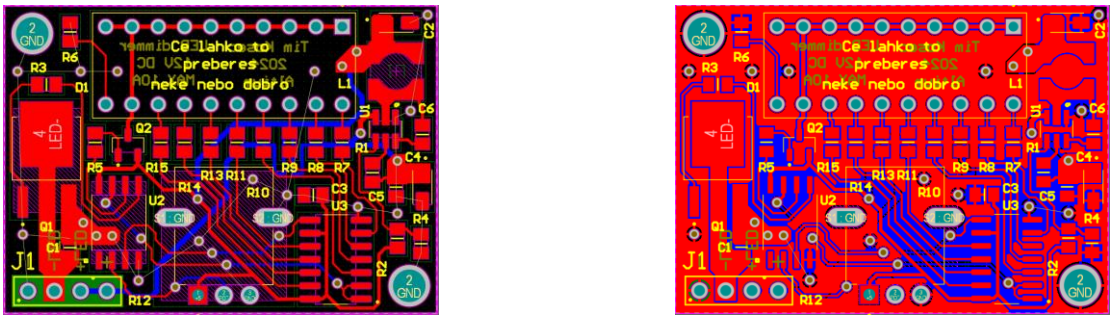
Slika 2 Zavihek s pravili dizajniranja tiskanine

In this step, I defined the design rules in Altium Designer, such as trace width, clearance, and via sizes. The software automatically flagged any violations, which helped avoid issues during manufacturing.

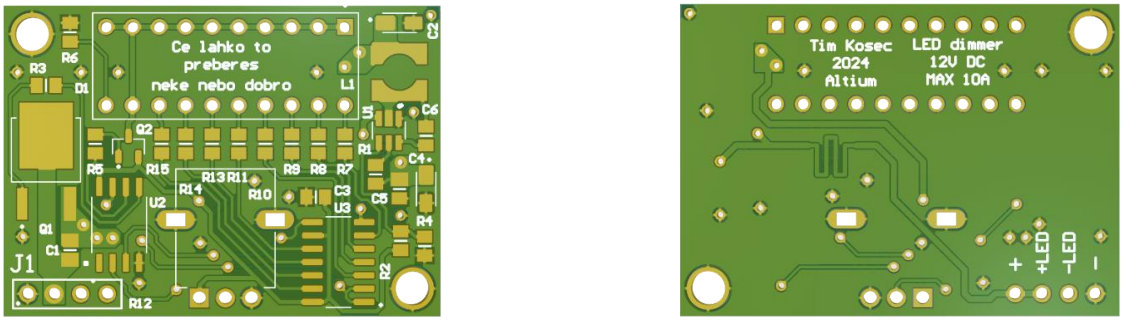
Once the layout was complete, I reviewed the design to ensure it met all the electrical and mechanical constraints before moving on to the final step.

Here's a 2D view of the PCB after the layout was completed, showing the component placement and routing.

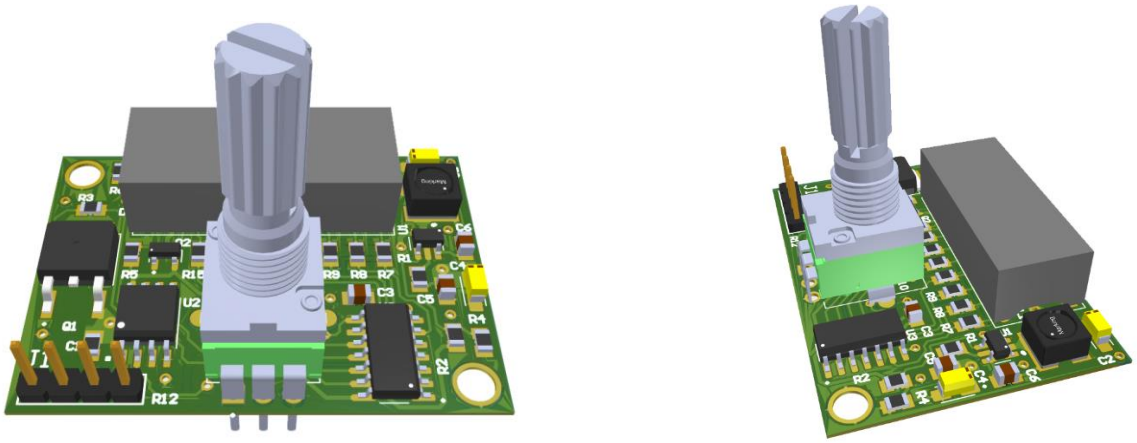
The 3D view helps visualize the finished board, including components and the physical layout. It also helps check for component interference and fit within the available space.



Slika 3 Tiskano vezje v urejevalniku



Slika 4 2D pogled tiskanega vezja



Slika 5 3D pogled tiskanega vezja

3.3 Bill of Materials

The Bill of Materials (BOM) is an important document that lists all the components required to assemble the PCB. Altium Designer automatically generates the BOM from the schematic. It includes information like part numbers, quantities, values, and manufacturers.

The BOM for this project included parts like resistors, capacitors, the ATtiny85 microcontroller, MOSFETs, SIPO registers, and the LED bar graph. It helped ensure that I ordered the correct components.

Comment	Designator	Footprint	Quantity	Manufacturer	PartNumber
100nF	C1, C3, C5, C6	CAPC2012X140N	4	FARNELL	3013476
10uF	C2, C4	CAPPM3216X180N	2	FARNELL	2333010
DC10EWA	D1	DIP750W50P254L2540H825Q20	1	FARNELL	2290323
6130XX11121	J1	61300411121	1	FARNELL	3049569
3.3uH, 2A	L1	WE-HEPC_5030	1	FARNELL	2288159
AOD1N60	Q1	TO228P1003X240-3N	1	FARNELL	2454077
PBSS5120T	Q2	SOT95P230X110-3N	1	FARNELL	8736391
2k2	R1, R2	CAPC2012X140N	2	FARNELL	2447623
1k	R3, R5	CAPC2012X140N	2	FARNELL	1652937
470	R4, R6, R7, R8, R9, R10, R11, R13, R14, R15	CAPC2012X140N	10	FARNELL	9237445
20k	R12	TRIM_P0915N-QC20BR5K	1	FARNELL	1191742
TPS562201DDCT	U1	SOT95P280X110-6N	1	FARNELL	3121817
ATTINY85	U2	SOIC127P798X216-8N	1	FARNELL	1455164
74HC164D	U3	SOIC127P600X175-14N	1	FARNELL	1085337

Slika 6 Seznam komponent (BOM)

3.4 Gerber

The final step in the PCB design process is generating the Gerber files. Gerber files are the industry-standard format for PCB manufacturing. These files describe the copper layers, solder mask, silkscreen, and other layers of the PCB.

Altium Designer generates the Gerber files directly from the PCB layout, which can then be sent to a manufacturer like JLCPCB for fabrication.

4 Assembly Process

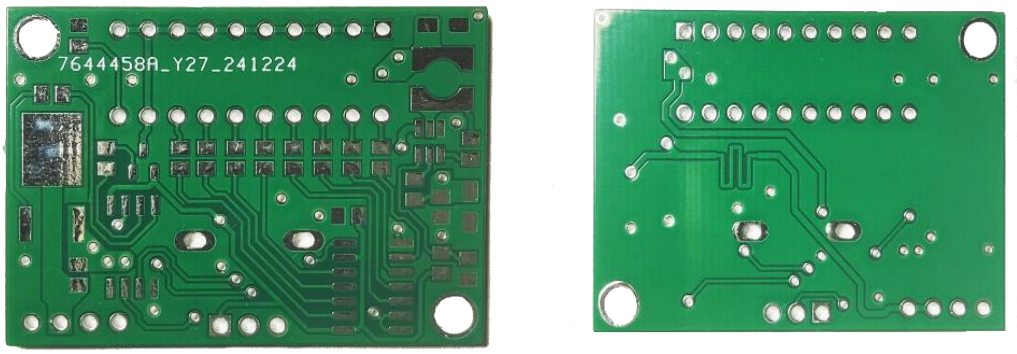
Once the PCB design is finalized and the Gerber files are generated, the next step is the assembly process. This involves soldering the components onto the PCB and verifying that the board functions as intended. Here's a breakdown of the assembly steps:

4.1 Component Sourcing and preperation

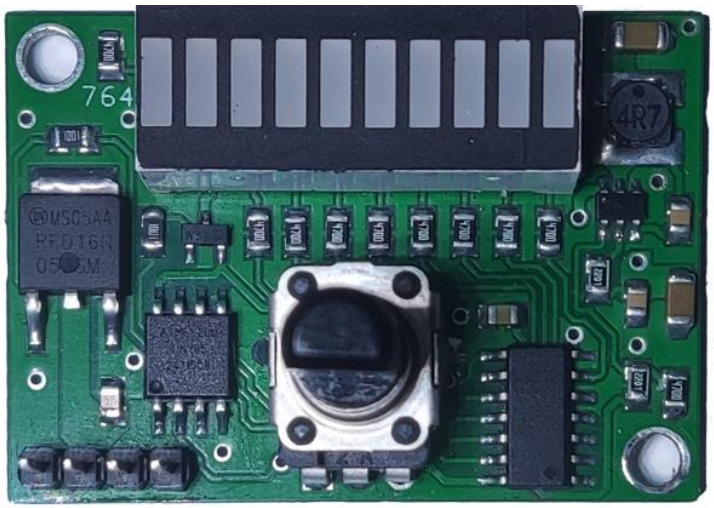
Before beginning the assembly, I ordered the components required for the PCB. I used the Bill of Materials (BOM) generated in Altium Designer to order the components from suppliers. The primary components included:

- **ATtiny85 microcontroller**
- **MOSFET transistors**
- **SIPO shift registers**
- **Potentiometer**
- **LED bar graph**
- **Resistors, capacitors, and other passive components**

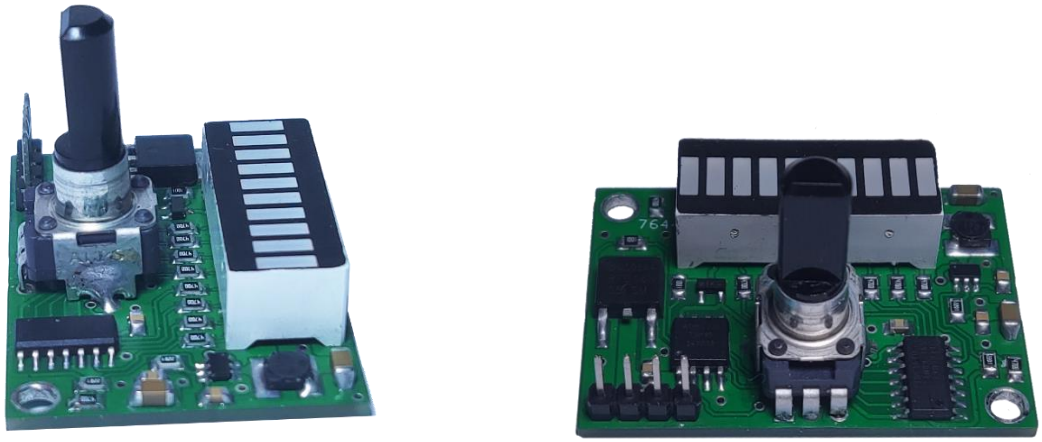
After receiving the components, I double-checked their part numbers, values, and packaging to ensure everything matched the BOM.



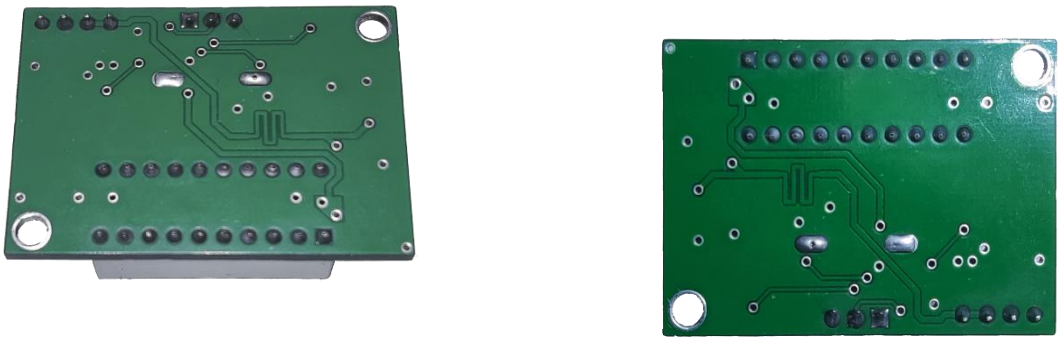
Slika 7 Tiskano vezje brez elementov



Slika 8 Končano tiskano vezje



Slika 9 Stranski pogled končanega tiskanega vezja



Slika 10 Pogled končanega tiskanega vezja od spodaj

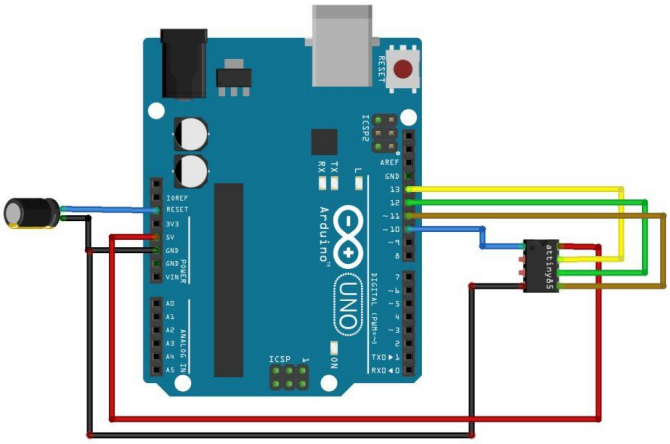
4.2 Firmware Programming

With the hardware assembled, it was time to program the ATtiny85 microcontroller. I used the **AVR ISP programmer** to upload the firmware to the microcontroller.

The firmware was developed in **Arduino IDE** and was written to handle the following functions:

1. **Reading the Potentiometer**: The analog input from the potentiometer was read and converted to a digital value.
2. **Controlling the MOSFET**: Based on the potentiometer reading, the microcontroller generated a PWM signal to control the MOSFET, adjusting the brightness of the LED strip.
3. **Controlling the LED Bar Graph**: The microcontroller sent data to the SIPO register to light up the appropriate segments of the LED bar graph based on the potentiometer value.

Once the firmware was uploaded, I tested the functionality of the microcontroller using a simple debugging method, such as blinking the onboard LED.



Slika 11 Priključitev procesorja za nalaganje programa

Slika 12 Signal na izhodu vezja

5 Final Assembly and enclosure

After confirming that the PCB was fully functional, the next step was to place it inside an enclosure to protect the electronics and ensure that the final product was aesthetically pleasing and user-friendly. The enclosure provides a durable housing that prevents damage to the components and provides a professional finish to the product. Here's how I approached the final assembly and enclosure process:

7 Firmware

```
const int CLK = 4;
const int SDT = 2;
const int POT = A3; //3
const int OUT = 0;
const int DIM = 1;

int poten = 0;
int old_poten = 0;
int timer = 0;
int OUTval = 0;
int OUTval2 = 0;
int i = 0;
int k = 0;
int r = 0;

int PWM[9] = {0, 8, 16, 32, 64, 90, 140, 200, 255};  //PWM vrednost posamezne stopnje

int dispVal = 0;

void setup() {
  pinMode(CLK, OUTPUT);
  pinMode(SDT, OUTPUT);
  pinMode(OUT, OUTPUT);
  pinMode(DIM, OUTPUT);
  pinMode(POT, INPUT);

  digitalWrite(CLK, LOW);
  digitalWrite(SDT, LOW);
  digitalWrite(OUT, LOW);
  digitalWrite(DIM, LOW);

  //Serial.begin(115200);

  dispVal = 0;
  tabela();
  digitalWrite(DIM, HIGH);
}

void tabela(){
switch(dispVal){

case 0:                 //87651234
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B11111111);
  digitalWrite(DIM, LOW);
break;

case 1:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B01111111);
  digitalWrite(DIM, LOW);
break;

case 2:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00111111);
  digitalWrite(DIM, LOW);
break;

case 3:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00011111);
  digitalWrite(DIM, LOW);
break;

case 4:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00001111);
  digitalWrite(DIM, LOW);
break;

case 5:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00001110);
  digitalWrite(DIM, LOW);
break;

case 6:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00001100);
  digitalWrite(DIM, LOW);
break;

case 7:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00001000);
  digitalWrite(DIM, LOW);
break;

case 8:
  digitalWrite(DIM, HIGH);
  shiftOut(SDT, CLK, MSBFIRST, B00000000);
  digitalWrite(DIM, LOW);
break;
}}
```

```
uint16_t pridobiUrejenoVrednost(uint16_t vhodnaVrednost)
{
  const uint16_t odstopanje = 20;
  const uint16_t steviloIzhodnihStanj = 8;
  const uint16_t stanja[steviloIzhodnihStanj + 1] = {0, 130,
222, 348, 474, 600, 726, 842, 948};
  const  uint16_t zacetnoStanje = 0;
  static uint16_t trenutnoStanje = zacetnoStanje;
  uint16_t k = stanja[trenutnoStanje];

  if (trenutnoStanje > 0) {
    k -= odstopanje;
  }
  uint16_t j = stanja[trenutnoStanje + 1];

  if (trenutnoStanje < steviloIzhodnihStanj){
    j += odstopanje;
  }

  if (vhodnaVrednost < k || vhodnaVrednost > j) {
    uint16_t i;

    for (i = 0 ; i < steviloIzhodnihStanj ; i++) {
      if (vhodnaVrednost >= stanja[i] && vhodnaVrednost <=
stanja[i + 1]) {
        break;
      }
    }

    trenutnoStanje = i;
  }

  return trenutnoStanje;
}
void loop() {

poten = pridobiUrejenoVrednost(map(analogRead(POT), 0,
1024, 1024, 0));

if(poten != old_poten){
  old_poten = poten;
  dispVal = poten;
  tabela();
  timer = 400;
  digitalWrite(DIM, LOW);
}

timer--;
if(timer<=1){
timer=1;
analogWrite(DIM, 230);
}

OUTval = PWM[poten];

if(OUTval != OUTval2){
  if(OUTval < OUTval2){
    r = 1;
    k = OUTval;
  }
  if(OUTval > OUTval2){
    r = 2;
    k = OUTval;
  }
  if(OUTval == OUTval2){
    r = 0;
    i = OUTval;
  }
  OUTval2 = OUTval;
}


if(r == 1){
  if(i > k){
    i--;
    analogWrite(OUT, i);
  }
}

if(r == 2){

  if(i < k){
    i++;
    analogWrite(OUT, i);
  }
}
delayMicroseconds(1000);
}
```