

# Licznik Geigera-Müllera

Marek Morawski, Alina Svekla

Czerwiec 2023

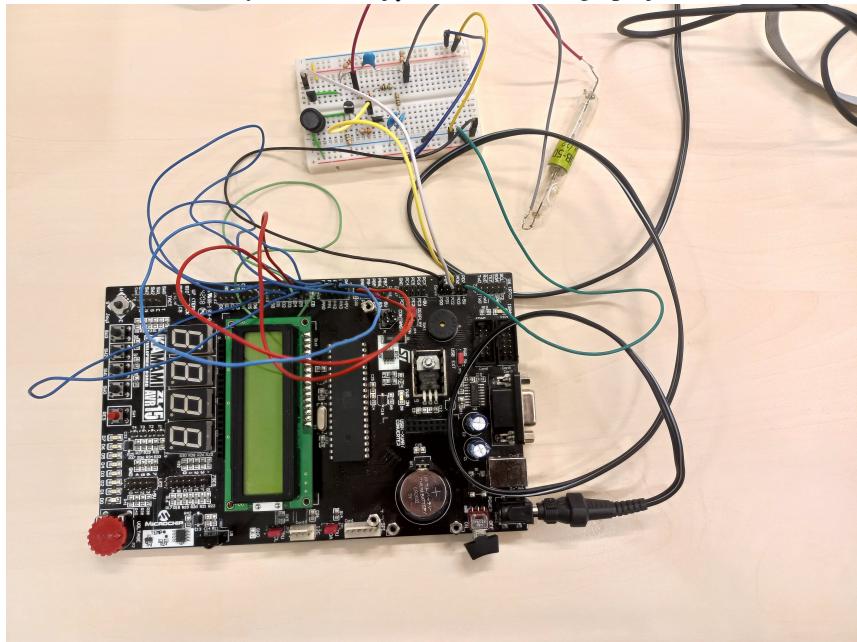
## 1 Cel projektu

Celem projektu było stworzenie licznika Geigera-Müllera do rejestracji i pomiaru intensywności promieniowania jonizującego. Gdy do tuby Geigera-Müllera wpadnie cząstka promieniowania, dochodzi do jonizacji zawartego w tej tubie gazu (najczęściej jest to argon). Drucik wewnętrz tuby jest elektrodą dodatnią, a obudowa jest elektrodą ujemną. Elektrody w tubie są spolaryzowane wysokim napięciem stałym, zwykle około 400V (w naszym układzie napięcie jest zwiększone dzięki przetwornicy zrealizowanej za pomocą odpowiednich elementów elektronicznych). Elektrony docierające do anody powodują wyładowanie lawinowe, co skutkuje impulsem elektrycznym. Licznik rejestruje te impulsy, każdy następny wykryty impuls zwiększa wartość liczbową pokazywaną na wyświetlaczu i powoduje mignięcie diody. Im silniejsze jest źródło promieniowania jonizującego, na którego działanie wystawiamy licznik, tym większa częstotliwość zmian wartości na wyświetlaczu.

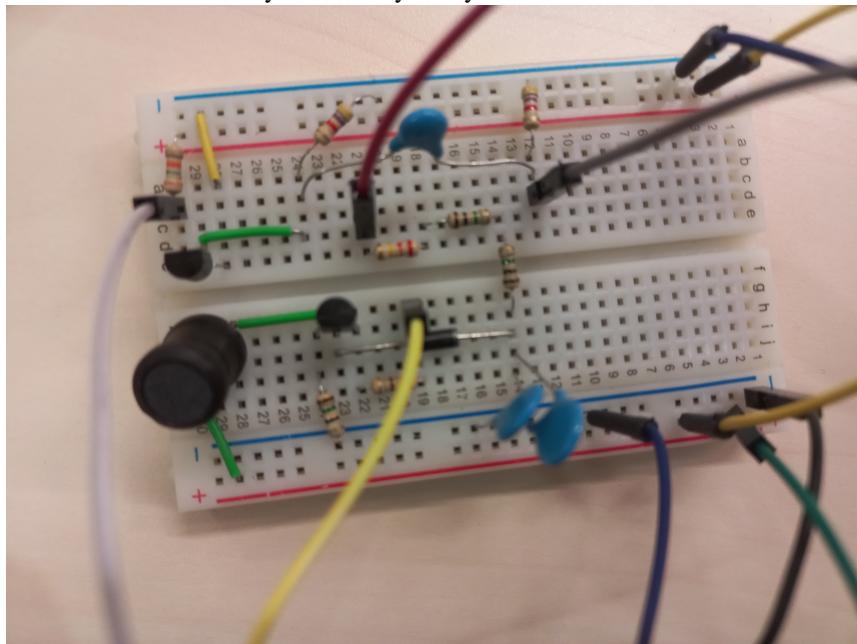
## 2 Schemat projektu i zdjęcia

Film prezentujący działanie licznika w pracowni radiologicznej na Wydziale Fizyki Uniwersytetu Warszawskiego: <https://youtu.be/aq1e0eS5P0Q>

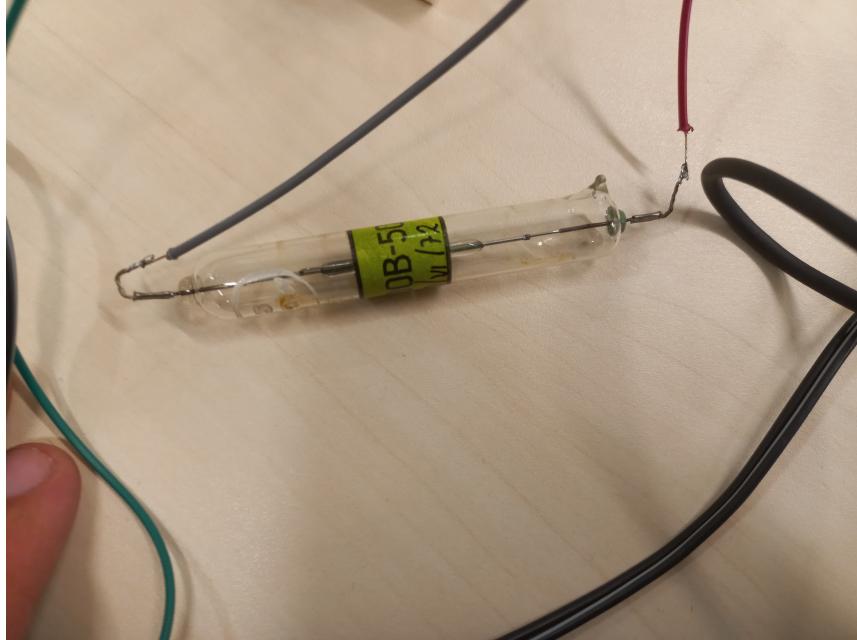
Rysunek 1: Zdjęcie zrealizowanego projektu



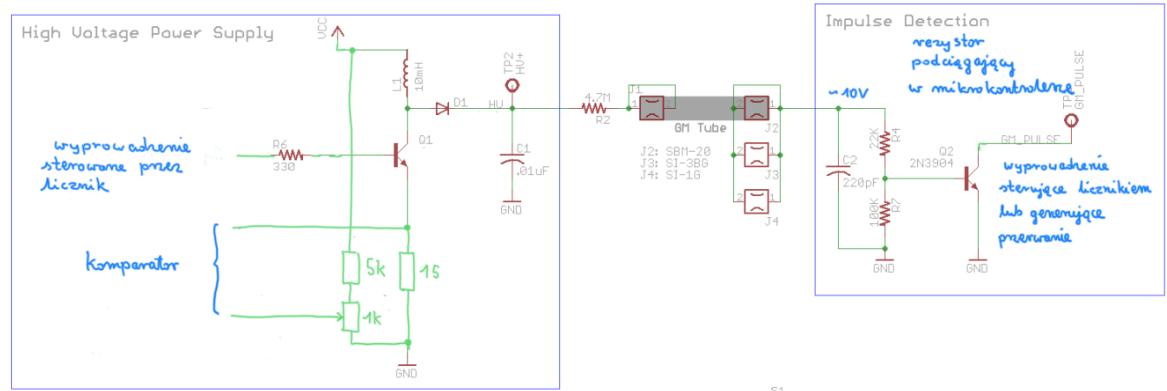
Rysunek 2: Płytkę stykową z elementami



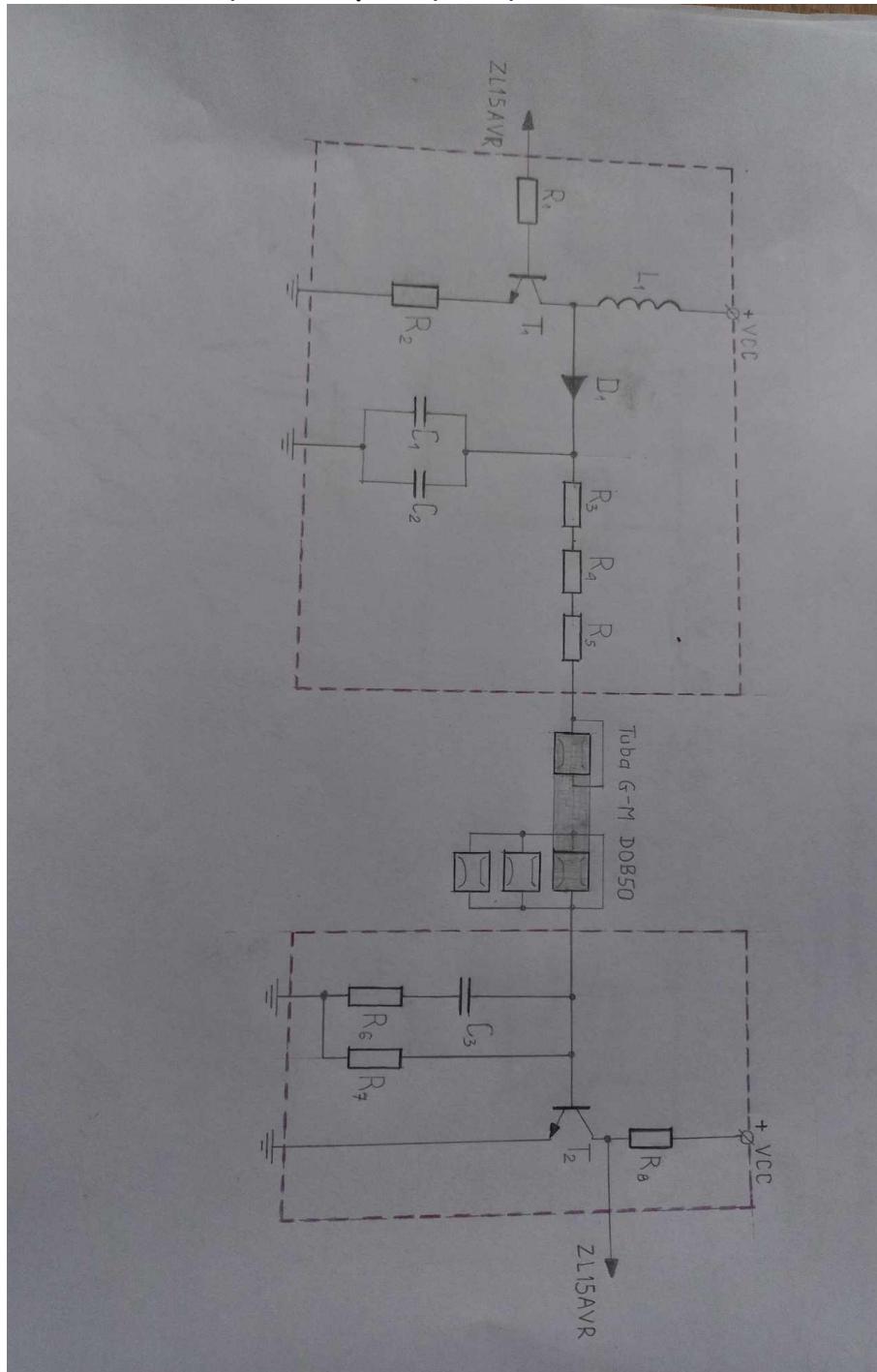
Rysunek 3: Tuba Geigera-Müllera typu DOB-50



Rysunek 4: Schemat, na którym wzorowaliśmy się, tworząc projekt - ostatecznie uległ on pewnym modyfikacjom



Rysunek 5: Ręcznie rysowany schemat układu



### 3 Wykaz komponentów zastosowanych w projekcie

- 1 x Tuba Geigera-Müllera typu DOB-50
- 1 x ZL15AVR - zestaw uruchomieniowy dla mikrokontrolerów AVR
- 1 x Wyświetlacz LCD 1602
- 1 x Płytki stykowa, na której zamontowano niżej wymienione elementy:
  - $L_1$  - Cewka o indukcyjności 10mH
  - $T_1$  - Tranzystor STBV32
  - $T_2$  - Tranzystor 2N3904
  - $D_1$  - Dioda BA159
  - $C_1, C_2, C_3$  - Kondensator 220 pF
  - $R_1$  - Rezystor 330  $\Omega$
  - $R_2$  - Rezystor 15  $\Omega$
  - $R_3, R_4$  - Rezystor 1 M $\Omega$
  - $R_5$  - Rezystor 220 k $\Omega$
  - $R_6, R_7$  - Rezystor 4,7 k $\Omega$
  - $R_8$  - Rezystor 36 k $\Omega$

### 4 Kod programu

#### 4.1 Sygnał PWM sterujący przetwornicą napięcia

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>

//-----
class Counter1PWM
{
public:
    Counter1PWM()
    {
        // Wyjście OC1A.
    }
}
```

```

DDRD = 1 << PD5;

// Częstotliwość sygnału PWM = 3 kHz.
ICR1 = int(F_CPU/3000-1+0.5);
// Czas trwania stanu wysokiego.
set(16 * 170); // t_1 = 170 us.

TCCR1A =
    1 << COM1A1 | // Clear OC1A on compare match, set at bottom.
    1 << WGM11; // Fast PWM 0..ICR1.

TCCR1B =
    1 << WGM13 | 1 << WGM12 | // Fast PWM 0..ICR1.
    1 << CS10; // f = F_CPU.

}

void set(uint16_t v)
{
    OCR1A = v;
}
};

//-----
volatile int n = 0;

ISR(INT0_vect)
{
    ++n;
    PORTA ^= 1 << PA0;
    _delay_ms(1);
    PORTA &= ~(1 << PA0);
}

//-----
int main()
{
    Counter1PWM pwm;
    MCUCR |= 1 << ISC00; // Kada zmiana napięcia generuje przerwanie.
    GICR |= 1 << INT0; // Uaktywniamy przerwanie.
    DDRA = 1 << PA0;
    sei(); // Włączamy obsługę przerwa .
    while (true)
    {
        printf("%d\n", n);
        _delay_ms(100);
    }
    return 0;
}

//-----

```

## 4.2 stdout do RS232 i na wyświetlaczu tekstowym LCD w układzie ZL15AVR

```
#include <avr/io.h>
#include <stdio.h>
#include "lcd.h"

//+++++
class LCDText : private LCD<1, 1>
{
private:
    uint8_t x, y;
    // Ponownie definiuję operacje zapisu i odczytu. Tym razem czekam na
    // gotowość wyświetlacza.
    void wr(uint8_t rs, uint8_t data)
    {
        wait();
        LCD<1, 1>::wr(rs, data);
    }
    //-----
    uint8_t rd(uint8_t rs)
    {
        wait();
        return LCD<1, 1>::rd(rs);
    }
    //-----
    // Ustawiam wewnętrzny wskaźnik tak, że odpowiada miejscu o współrzędnych
    // (x,y).
    void at(uint8_t x, uint8_t y)
    {
        wr(0, 0b10000000 | ((y << 6) + x));
    }
public:
    //-----
    LCDText() : x(0), y(0)
    {
        // Przygotowanie wyświetlacza do pracy.
        // 4-bit data line, 2 lines
        wr(0, 0b00101000);
        // clear display
        wr(0, 0b00000001);
        // return home
        wr(0, 0b00000010);
        // ac auto increase, no display shift
        wr(0, 0b00000110);
        // display on, cursor off, blinking off
        wr(0, 0b00001100);
    }
private:
```

```

//-----
// Odczytanie kodu znaku z miejsca (x,y).
uint8_t get_at(uint8_t x, uint8_t y)
{
    at(x, y);
    return rd(1);
}
//-----
// Zapisanie znaku w miejscu (x,y).
void put_at(uint8_t x, uint8_t y, uint8_t c)
{
    at(x, y);
    wr(1, c);
}
public:
//-----
void put(uint8_t c)
{
    if (c=='\r')
    {
        for (; y<2; ++y)
        {
            for (; x<16; ++x)
            {
                if (get_at(x, y)!=' ')
                    put_at(x, y, ' ');
            }
            x = 0;
        }
        x = y = 0;
    }
    else if (c>=' ')
    {
        put_at(x, y, c);
        if (++x==16)
        {
            x = 0;
            if (++y==2)
                y = 0;
        }
    }
}
};

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#define UDR0
#define UDR UDR0
#define UCSRA UCSRA0
#define USCRB USCRB0
#define UBRRL UBRROL

```

```

#define UBRRH UBRR0H
#define UDRE UDRE0
#define U2X U2X0
#define TXEN TXENO
#endif // UDRO

//+++++
class Usart
{
public:
    Usart()
    {
        // 115200 bps
        UBRRH = (F_CPU / 8 / 115200 - 1) / 256;
        UBRRH = (F_CPU / 8 / 115200 - 1) % 256;
        UCSRA |= 1 << U2X;
        // Aktywuję nadajnik i odbiornik.
        UCSRB = 1 << TXEN;
    }
//-----
inline void put(uint8_t c)
{
    // Czekam na zakończenie trwającej transmisji znaku.
    while (!(UCSRA & 1 << UDRE));
    // Zlecam wysłanie znaku.
    UDR = c;
}
};

//+++++
class StdOut
{
private:
    static FILE f;
    static StdOut* instance;
    LCDText lcd_text;
    Usart usart;
//-----
static int put(char c, FILE*)
{
    if (c=='\n')
        put('\r', NULL);
    if (instance)
    {
        instance->lcd_text.put(c);
        instance->usart.put(c);
    }
    return 0;
}
public:
//-----

```

```
StdOut()
{
    instance = this;
    f.put = put;
    f.flags = _FDEV_SETUP_WRITE;
    stdout = &f;
}
};

FILE StdOut::f;
StdOut* StdOut::instance;
StdOut lcdtext_usart_stdout;
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

---