Logger przesyłu danych przez RS232

1. Cele projektu

Projekt zakłada zaprojektowanie, wykonanie oraz zaprogramowanie układu opartego o mikrokontroler NXP ARM LPC1765. Logger ma umożliwiać transfer danych pomiędzy dwoma urządzeniami przy jednoczesnym przechwytywaniu i zapisywaniu przesyłanych komunikatów. Odebrane dane powinny być zapisywane na kartę micro SD.

2. Schemat elektryczny i projekt płytki

Wszystkie etapy projektowe płytki zostały wykonane przy użyciu programu Eagle 6.3.0 Light. Do dokumentu dołączony jest schemat elektryczny płytki.

3. Opis wybranych elementów

3.1. Bootloader i programowanie

Aby móc zaprogramować pamięć Flash mikrokontrolera konieczne jest uruchomienie go w trybie ISP (In-System programming). LPC17xx posiadają do tego celu pin P2[10], który po otrzymaniu poziomu niskiego w momencie doprowadzenia zasilania włącza tryb ISP. Poziom niski na pinie jest dostarczony poprzez jego zwarcie za pomocą zworki do masy. Do wgrania programu służy interfejs UART0, który jest podłączony do portów RS232.

3.2. UART

UART (Universal Asynchronous Receiver and Transmitter) to interfejs służący do asynchronicznej komunikacji poprzez port szeregowy. Dodatkowo jest wyposażony w bufor do gromadzenia informacji przy szybkich transmisjach. Logger wykorzystuje interfejs UART0 oraz UART3 do odbierania danych z linii TXD oraz RXD. Każdy odczytany znak jest zapisywany do aktualnego bufora. Jeśli bufor jest pełny następuje zmiana bufora zapisu. Inicjalizowane jest przerwanie EINT0_IRQHandler, w której następuje zapis zapełnionego bufora do pliku. W tym czasie kolejne dane nie są tracone, w przypadku przyjścia przerwania UART0 lub UART3, gdyż są zapisywane do drugiego bufora. Program posługuje się dwoma buforami, które są kolejno ustawiane jako bufor do zapisu danych.

Dane są rozróżniane w następujący sposób: Każda z linii ma swoje oznaczenie, są to kolejno TXD oraz RXD. Ponadto odczytane dane są dodatkowo oznaczane stemplem czasowym. Stempel czasowy to czas, który upłynął od momentu uruchomienia urządzenia, liczony w sekundach. Za odpowiednią inkrementację stempla czasowego odpowiedzialny jest timer.

Ustawienia UART to: 9600 Baud rate, 8 bitów na bajt, bez kontroli parzystości i jeden bit stopu.

3.3. Pozostałe linie RS 232

Wszystkie sygnały na pozostałych liniach RS 232 są przechwytywane i informacja o nich jest także zapisywana w buforze. Do obsługi zdarzeń na poszczególnych liniach wykorzystywana jest funkcja obsługi przerwań `EINT3_IRQHandler()`, gdyż linie te są podłączone do Portu 0, więc każde zdarzenie na wyżej wymienionych liniach powoduje wywołanie właśnie tej funkcji.

3.4. Karta SD

Slot karty micro SD jest podłączony do interfejsu SPI. Działa on na zasadzie Master-Slave. Warto podkreślić, że komunikacja w obie strony dla tego interfejsu przebiega jednocześnie tj. 1 bit jest wysyłany i w tym samym momencie 1 bit jest odbierany. Dodatkowo slot karty micro SD jest zabezpieczony układem przeciw przepięciowym, dzięki czemu układ jest chroniony przed wyładowaniami elektrostatycznymi i skokami napięć przy wkładaniu/wyjmowaniu karty ze slotu przy doprowadzonym zasilaniu.

3.5. PWM

W projekcie został także wykorzystany moduł PWM. Za jego pomocą wyświetlane są odpowiednie wartości na poszczególnych diodach LED. Dzięki wykorzystaniu tego modułu możemy decydować o długości wyświetlania stanu wysokiego na poszczególnych diodach, co pozwala uzyskać efekt płynnego „gaszenia" się diody. Został on także wykorzystany do ustawienia priorytetów poszczególnych kolorów. Poniżej lista oznaczeń wg priorytetu (kolor czerwony ma najwyższy priorytet):

- CZERWONY – oznacza wystąpienie błędu
- NIEBIESKI – oznacza prawidłowe zakończenie pracy urządzenia
- ZIELONY – oznacza gotowość urządzenia do pracy (prawidłowa inicjalizacja)
- ŻÓŁTY – oznacza transmisję danych z linii RXD lub TPD
- FIOLETOWY – oznacza stan zapisu danych do pliku
- LEKKONIEBIESKI – oznacza dane z linii stanowych RS 232
- BIAŁY – oznacza stan neutralny

3.6. Zasilanie

Logger posiada możliwość zasilania z mini USB oraz z niezależnego wtyku zasilającego. Ze względu na niedostępność użytego w projekcie wtyku zrealizowane jest wyłącznie zasilanie poprzez USB.

4. Program

Program został napisany w środowisku CooCox CoIDE. Do obsługi karty SD używana jest biblioteka FatFs do pobrania ze strony http://elm-chan.org/fsw/ff/00index_e.html. Reszta bibliotek jest dostępna na stronie http://www.brc-electronics.nl/librarys.

Kod programy prezentuje się w sposób następujący:

```
#include <stdio.h>
#include <string.h>
#include "lpc17xx_uart.h"
#include "cmsis/core_cm3.h"
#include "gpio.h"
#include "timer.h"
#include "SD/ff.h"
#include "SD/init.h"

#define PWM_PRESCALE 100 //100 cykli PCLK = TC + 1 (1 us)
#define BUF_SIZE 1000
#define RS_SIZE 8
#define TIME_SIZE 25
#define MAX_LED 5000

typedef enum {
    TXD = 0,
    RXD,
    RI,
    DTR,
    CTS,
    RTS,
    DSR,
    DCD,
    NONE
} dataType;

char source[8][RS_SIZE] = {
        {"\r\n[TXD]"},
        {"\r\n[RXD]"},
        {"\r\n[RI ]"},
        {"\r\n[DTR]"},
        {"\r\n[CTS]"},
        {"\r\n[RTS]"},
        {"\r\n[DSR]"},
        {"\r\n[DCD]"}};

// 0-blue, 1-red, 2-green
int leds[7][3] = {
        {MAX_LED, 0, MAX_LED}, //red - priorytet 0 - blad
        {0, MAX_LED, MAX_LED}, //blue - priorytet 1 - koniec pracy
        {MAX_LED, MAX_LED, 0}, //green - priorytet 2 - gotowy do pracy
        {MAX_LED, 0, 0},       //yellow - priorytet 3 - dane z uart (RXD, TXD)
        {0, 0, MAX_LED},       //purple - priorytet 4 - zapis do pliku
        {0, MAX_LED, 0},       //light blue - priorytet 5 - inne stany RS232
        {0, 0, 0}};            //white - priorytet 6 - neutralny stan

dataType type = NONE;

char time_buffer[TIME_SIZE] = "[+0]";

char buffA[BUF_SIZE + RS_SIZE + TIME_SIZE];
char buffB[BUF_SIZE + RS_SIZE + TIME_SIZE];

char *buff_write = buffA;
char *buff_save = buffB;
```

```
uint32_t buffI = 0;
uint32_t buffS = 0;

//timer
unsigned long long int current_time = 0;
uint32_t time_buf_size = 5; //liczba znakow do zapisu z time_buffer
uint8_t time_changed = 0;
uint8_t end_work = 0;

//Zmienne SD
FRESULT errf = FR_OK;
UINT bytes_written;
FATFS fatfs[1];
FIL log_file;

//leds
uint8_t is_new_led_priority = 0;
uint8_t new_led_priority = 0;
uint8_t cur_led_priority = 6;
uint32_t blue = MAX_LED;
uint32_t red = MAX_LED;
uint32_t green = MAX_LED;

void LED_init(void);
void BUTTON_END_init(void);
void BUTTONS_RS_init(void);
void BUFFER_SAVE_init(void);
void SD_init(void);
void TIMER_init(void);
void UART0_init(void);
void UART3_init(void);
void PWM1_init(void);

int main (void) {
    LED_init();
    init_in_main();
    PWM1_init();

    BUTTON_END_init();
    BUFFER_SAVE_init();

    SD_init();
    TIMER_init();

    UART0_init();
    UART3_init();
    BUTTONS_RS_init();

    while(1)
        ;

    return 0;
}

void PWM1_IRQHandler (void) {
    if (is_new_led_priority) {
        is_new_led_priority = 0;
        if (new_led_priority <= cur_led_priority || (blue >= MAX_LED && red >=
MAX_LED && green >= MAX_LED)) {
```

```
            cur_led_priority = new_led_priority;
            blue = leds[cur_led_priority][0];
            red = leds[cur_led_priority][1];
            green = leds[cur_led_priority][2];
        }
    }

    if ((LPC_PWM1->IR & 0x01) == 0x01) { //sprawdz, czy to przerwanie z MR0
        LPC_PWM1->MR4 = blue;
        LPC_PWM1->MR5 = red;
        LPC_PWM1->MR6 = green;
        LPC_PWM1->LER = (1 << 4) | (1 << 5) | (1 << 6);

        blue = blue < MAX_LED ? blue + 25 : MAX_LED;
        red = red < MAX_LED ? red + 25 : MAX_LED;
        green = green < MAX_LED ? green + 25 : MAX_LED;
    }

    LPC_PWM1->IR = 0xff; // wyczysc flagi przerwan
    return;
}

void UART0_IRQHandler(void) {
    uint8_t inter_val, lsr_val;
    uint8_t dummy;

    inter_val = LPC_UART0->IIR;
    inter_val &= 0x07; //bity 1-3 zawieraja rodzaj przerwania
    //linia statusu
    if (inter_val == UART_IIR_INTID_RLS) {
        lsr_val = LPC_UART0->LSR;
        if (lsr_val & (UART_LSR_OE | UART_LSR_PE | UART_LSR_FE | UART_LSR_BI |
UART_LSR_RXFE | UART_LSR_RDR)) {
            dummy = LPC_UART0->RBR; //odczyt wartosci, aby wyczyscic flagi
            return;
        }
    }

    //linia danych
    else if (inter_val == UART_IIR_INTID_RDA) {
        is_new_led_priority = 1;
        new_led_priority = 3;
        if (type != TXD || time_changed) {
            type = TXD;
            memcpy(buff_write+buffI, source[type], RS_SIZE-1);
            buffI += RS_SIZE-1;
            memcpy(buff_write+buffI, time_buffer, time_buf_size);
            time_changed = 0;
            buffI += time_buf_size-1;
        }
        char znak = LPC_UART0->RBR;
        if (znak != '\n') {
            buff_write[buffI++] = znak;
            if (znak == '\r') {
                buff_write[buffI++] = '\n';
            }
        }
    }
```

```
    //thre - transmit holding register empty
    else if (inter_val == UART_IIR_INTID_THRE) {
        lsr_val = LPC_UART0->LSR;
        return;
    }

    if (buffI >= BUF_SIZE-1) {
        buffS = buffI;
        char *tmp = buff_write;
        buff_write = buff_save;
        buff_save = tmp;
        buffI = 0;
        NVIC->STIR = EINT0_IRQn;
    }

    return;
}

void UART3_IRQHandler(void) {
    uint8_t inter_val, lsr_val;
    uint8_t dummy;

    inter_val = LPC_UART3->IIR;
    inter_val &= 0x07; //bity 1-3 zawieraja rodzaj przerwania
    //linia statusu
    if (inter_val == UART_IIR_INTID_RLS) {
        lsr_val = LPC_UART3->LSR;
        // Check for errors
        if (lsr_val & (UART_LSR_OE | UART_LSR_PE | UART_LSR_FE | UART_LSR_BI |
UART_LSR_RXFE | UART_LSR_RDR)) {
            dummy = LPC_UART3->RBR; //odczyt wartosci, aby wyczyscic flagi
            return;
        }
    }

    //linia danych
    else if (inter_val == UART_IIR_INTID_RDA) {
        is_new_led_priority = 1;
        new_led_priority = 3;
        if (type != RXD || time_changed) {
            type = RXD;
            memcpy(buff_write+buffI, source[type], RS_SIZE-1);
            buffI += RS_SIZE-1;
            memcpy(buff_write+buffI, time_buffer, time_buf_size);
            time_changed = 0;
            buffI += time_buf_size-1;
        }
        char znak = LPC_UART3->RBR;
        if (znak != '\n') {
            buff_write[buffI++] = znak;
            if (znak == '\r') {
                buff_write[buffI++] = '\n';
            }
        }
    }

    //thre - transmit holding register empty
    else if (inter_val == UART_IIR_INTID_THRE) {
        lsr_val = LPC_UART3->LSR;
```

```
        return;
    }

    if (buffI >= BUF_SIZE-1) {
        buffS = buffI;
        char *tmp = buff_write;
        buff_write = buff_save;
        buff_save = tmp;
        buffI = 0;
        NVIC->STIR = EINT0_IRQn;
    }

   return;
}

void TIMER0_IRQHandler(void) {
    if ((LPC_TIM0->IR & 0x01) == 0x01) {
        current_time++;
        int cx = snprintf (time_buffer, 25, "[+%lld]", current_time);
        time_buf_size = cx + 1;
        time_changed = 1;
    }

    LPC_TIM0->IR |= 1 << 0; // wyczysc przerwania
}

void EINT0_IRQHandler(void) {
    is_new_led_priority = 1;
    new_led_priority = 4;
    if (buffS > 0) {
        errf = f_write(&log_file, buff_save, buffS, &bytes_written);
        if (errf != FR_OK) {
            is_new_led_priority = 1;
            new_led_priority = 0;
        }
    }

    NVIC_ClearPendingIRQ(EINT0_IRQn);
}

void EINT3_IRQHandler(void) {
    if (end_work) {
        GPIOClearInterrupt();
        return;
    }

    if (GPIOCheckInterrupts(2, 10, FALLING)) {
        NVIC_DisableIRQ(TIMER0_IRQn);
        NVIC_DisableIRQ(UART0_IRQn);
        NVIC_DisableIRQ(UART3_IRQn);
        NVIC_DisableIRQ(EINT0_IRQn);
        delayMs(0, 100); // Odczekanie 100ms na ustabilizowanie się napięć

        if (GPIOGetValue(2, 10) == 0) {
            if (buffI > 0) {
                errf = f_write(&log_file, buff_write, buffI, &bytes_written);
                if (errf != FR_OK) {
                    is_new_led_priority = 1;
                    new_led_priority = 0;
```

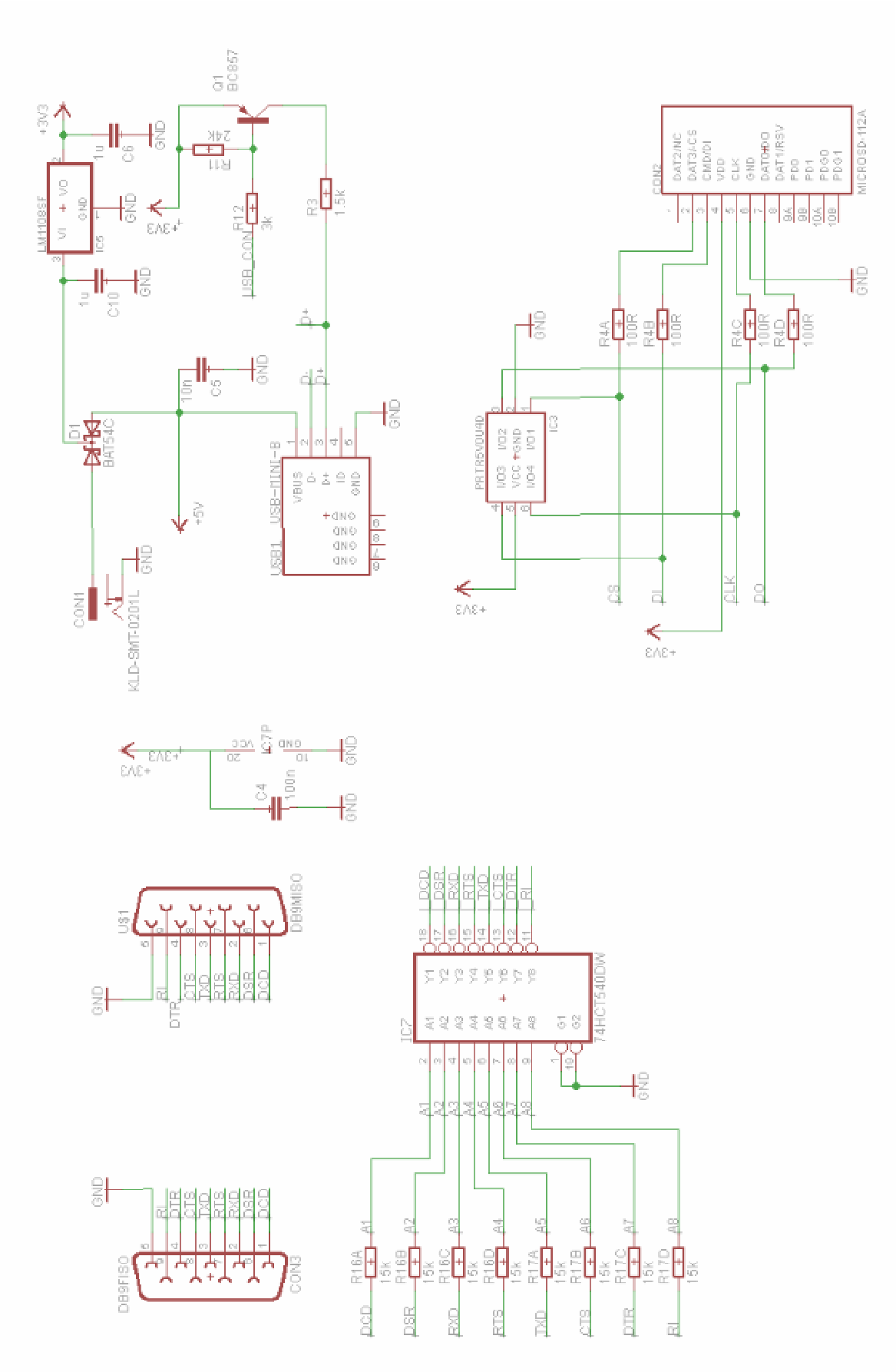
```
            }
        }
        buffI = 0;
        errf = f_close(&log_file);
        if (errf != FR_OK) {
            is_new_led_priority = 1;
            new_led_priority = 0;
        }
        is_new_led_priority = 1;
        new_led_priority = 1;
    }

    // Deinicjalizacja UART0
    UART_DeInit((LPC_UART_TypeDef *)LPC_UART0);
    UART_DeInit((LPC_UART_TypeDef *)LPC_UART3);

    end_work = 1;
    GPIOClearInterrupt();
    return;
}

is_new_led_priority = 1;
new_led_priority = 5;
dataType tmp = type;

if (GPIOCheckInterrupts(0, 4, FALLING)) {
    if (type != TXD && type != RXD) {
        type = RI;
    }
} else if (GPIOCheckInterrupts(0, 5, FALLING)) {
    type = DTR;
} else if (GPIOCheckInterrupts(0, 6, FALLING)) {
    type = CTS;
} else if (GPIOCheckInterrupts(0, 7, FALLING)) {
    type = RTS;
} else if (GPIOCheckInterrupts(0, 8, FALLING)) {
    type = DSR;
} else if (GPIOCheckInterrupts(0, 9, FALLING)) {
    type = DCD;
}

if (tmp != type || time_changed) {
    memcpy(buff_write+buffI, source[type], RS_SIZE-1);
    buffI += RS_SIZE-1;
    memcpy(buff_write+buffI, time_buffer, time_buf_size);
    time_changed = 0;
    buffI += time_buf_size-1;

    if (buffI >= BUF_SIZE-1) {
        buffS = buffI;
        char *tmp = buff_write;
        buff_write = buff_save;
        buff_save = tmp;
        buffI = 0;
        NVIC->STIR = EINT0_IRQn;
    }
}

GPIOClearInterrupt();
```

```
}

void PWM1_init (void) {
    LPC_SC->PCONP |= (1 << 6); // upewnij sie, ze PWM jest wlaczony (domyslnie
jest)
    LPC_SC->PCLKSEL0 |= (1 << 12); // PCLK_PWM1 = CCLK (100MHz)
    LPC_PWM1->TCR = 2; // reset PWM

    LPC_PINCON->PINSEL3 |= (1 << 15); // BLUE wyjscie dla Pin1.23
    LPC_PINCON->PINSEL3 |= (1 << 17); // RED wyjscie dla Pin1.24
    LPC_PINCON->PINSEL3 |= (1 << 21); // GREEN wyjscie dla Pin1.26

    LPC_PWM1->PR = PWM_PRESCALE-1; // jedno tykniecie TC co 1us
    LPC_PWM1->MR0 = MAX_LED; // 1 przerwanie co 5ms
    LPC_PWM1->MCR |= (1 << 1) | (1 << 0); // wyzeruj timer i wywolaj przerwanie,
gdy MR0 bedzie rowne PWM1->TC

    LPC_PWM1->MR4 = LPC_PWM1->MR5 = LPC_PWM1->MR6 = MAX_LED;

    // zaktualizuj MR0, MR4, MR5 oraz MR6
    LPC_PWM1->LER = (1 << 0) | (1 << 4) | (1 << 5) | (1 << 6);

    // skonfiguruj kanal 4, 5 i 6 jako pojedyncze zbocze PWM
    LPC_PWM1->PCR |= (0 << 4) | (0 << 5) | (0 << 6);

    // udostepnij wyjscie kanalu 4, 5 i 6 PWM
    LPC_PWM1->PCR |= (1 << 12) | (1 << 13) | (1 << 14);

    LPC_PWM1->TCR = (1 << 3) | (1 << 0); // wlacz tryb PWM oraz odliczanie

    NVIC_EnableIRQ(PWM1_IRQn);
}

void TIMER_init() {
    NVIC_DisableIRQ(TIMER0_IRQn);
    LPC_TIM0->TCR = 0;
    LPC_TIM0->IR = 0x1<<0;
    LPC_SC->PCONP |= 1 << 1; //wlacz Timer 0
    LPC_SC->PCLKSEL0 |= 1 << 3; // zegar dla timera = CCLK/2
    LPC_TIM0->MR0 = 50000000; // 1 * 50MHz = 50000000 (1 sec)
    LPC_TIM0->MCR = 3; // przerwanie, gdy MR0 = TC
    LPC_TIM0->TC = 0;  // zero Timer Counter value
    LPC_TIM0->PC = 0;  // zero Prescale Counter
    LPC_TIM0->PR = 0;  // zero Prescale Register

    NVIC_EnableIRQ(TIMER0_IRQn);
    LPC_TIM0->TCR = 1; // Start timer
}

void LED_init(void) {
    // Ustawienie sterowania diodą
    GPIOSetDir(1, 23, OUTPUT);
    GPIOSetDir(1, 24, OUTPUT);
    GPIOSetDir(1, 26, OUTPUT);

    // wylacz diode
    GPIOSetValue(1, 23, HIGH);
    GPIOSetValue(1, 24, HIGH);
    GPIOSetValue(1, 26, HIGH);
```

```
}

void UART0_init() {
    UART0_Init(9600);

    uint32_t reg_val;
    reg_val = LPC_UART0->LSR; //odczyt, aby wyczyscic linie statusu
    while ((LPC_UART0->LSR & (UART_LSR_THRE|UART_LSR_TEMT)) !=
(UART_LSR_THRE|UART_LSR_TEMT));
    while (LPC_UART0->LSR & UART_LSR_RDR) {
        reg_val = LPC_UART0->RBR; //odczyt smieci z linii danych
    }
}

void UART3_init() {
    UART3_Init(9600);

    uint32_t reg_val;
    reg_val = LPC_UART3->LSR; //odczyt, aby wyczyscic linie statusu
    while ((LPC_UART3->LSR & (UART_LSR_THRE|UART_LSR_TEMT)) !=
(UART_LSR_THRE|UART_LSR_TEMT));
    while (LPC_UART3->LSR & UART_LSR_RDR) {
        reg_val = LPC_UART3->RBR; //odczyt smieci z linii danych
    }

    //inicjalizacja zakonczona powodzeniem
    is_new_led_priority = 1;
    new_led_priority = 2;

    //wlacz UART'y i udostepnij przerwania
    LPC_UART0->IER = UART_IER_RBRINT_EN | UART_IER_RLSINT_EN;
    LPC_UART3->IER = UART_IER_RBRINT_EN | UART_IER_RLSINT_EN;

    NVIC_EnableIRQ(UART0_IRQn);
    NVIC_EnableIRQ(UART3_IRQn);
}

void BUTTON_END_init (void) {
    GPIOSetDir(2, 10, INPUT); // Ustawienie P2[10] do działania jako guzik
    GPIOSetPull(2, 10, PULLUP);
    GPIOSetInterrupt(2, 10, FALLING); // Przerwanie na zbocze opadające pinu
P2[10]
}

void BUTTONS_RS_init (void) {
    int8_t i;
    //ustaw piny P0.4, P0.5, P0.6, P0.7, P0.8, P0.9
    for (i=4; i<=9; i++) {
        GPIOSetDir(0, i, INPUT);
        GPIOSetPull(0, i , PULLUP);
        GPIOSetInterrupt(0, i, FALLING);
    }
}

void BUFFER_SAVE_init (void) {
    NVIC_EnableIRQ(EINT0_IRQn);
}

void SD_init(void) {
```

```
    int8_t atempts = 0;
    DSTATUS errd = FR_OK;
    do {
        errd = disk_initialize(0);
        if (errd == FR_OK) {
            errf = f_mount(0, &fatfs[0]);
            errf |= f_open(&log_file, "sd_log.txt", 10);
        }

        atempts++;
    } while (errd != FR_OK && atempts < 100);

    if (atempts >= 100 || errd != FR_OK || errd != FR_OK) {
        is_new_led_priority = 1;
        new_led_priority = 0;
    }
}
```

Dodatek A: Schemat elektryczny



Logger RS232