PRAXE

Datum: 26.03. 2024	Střední průmyslová škola, Chomutov, Školní 50, příspěvková organizace	Třída: V4
Číslo úlohy: 6.	Světelný řádek	Jméno: Tomáš Bartoš

Zadání:

S využitím vývojového kitu AVR a periferie světelného řádku sestavte v jazyku C program zobrazující aktuální čas.

Výsledné řešení musí obsahovat následující funkcionalitu:

- Zobrazení aktuálního času s rozlišením sekund a viditelným oddělením jednotlivých časových polí např. blikající dvojtečkou.
- Možnost nastavení času a 12h/24h režimu zobrazení pomocí tlačítek připojených na volném portu MCU
- Oddělenou interpretaci aktuálního času a zobrazované informace v kódu (data vs. bitmapa na displeji) spolu s pomocnou funkcí pro ovládání jednotlivých registrů displeje v podobě zapisRegistr (registr, data)
- Podporu zobrazení alespoň čísel a dvojtečky. Celý font nemusíte počítat ručně, využijte libovolný z dostupných (nebo svých) nástrojů. Musíte však vědět, jak formát znaku vypadá (a ideálně si nepřidělávat zbytečnou práci v kódu několikerou úpravou připraveného fontu).

Teorie:

Řešení úlohy světelného řádku za využití mikroprocesoru ATmega128 vyžaduje znalost základních ovládacích mechanismů mikrokontroleru společně s podrobnou znalostí principu funkce samotného řádku.

Řádek použitý v úloze pracuje na principu datových registrů, jejichž výstupy jsou po dodání vhodného výběrového signálu polány na příslušnou matici. LED matice řádku jsou aktivní v log. 0, zatímco registry jsou aktivní v log. 1.

Řádek obsahuje celkem 7 registrů:

- Registry 0-5 = řádkové registry pro ukládání dat
- Registr 6 = sloupcový registr sloužící k výběru sloupce, kam budou zaslána data z řádkových registrů

Zápis informace na řádek pak funguje tak, že jsou předem připravená data (byte), reprezentující jeden sloupec zobrazovaného charakteru přivedena na libovolný řádkový registr, ten je krátkým pulzem (log.1, delay, log.0) zaktivován a data ze vstupu jsou přivedena na jeho výstup. Nyní je však třeba ještě definovat do jakého sloupce se mají data zapsat, a tudíž jsou na datovou sběrnici přivedena adresa sloupce k aktivaci a je vybrán sloupcový registr (odeslání stejného pulzu jako u zápisu dat na řádkový registr). LED dioda segmentu se poté aktivuje ve chvíli, kdy je její řádkový a sloupcový vodič v log. 0.

Co se samotného mikrokontroleru týče, tak ten je pro účely této úlohy potřeba jen správně nakonfigurovat přes makefile a správně pomocí něho zaovládat řádek. Co se nastavovaných hodnot Makefilu týče, tak by smě měly shodovat s těmito:

- Typ procesoru = ATmega128
- Typ programátoru = jtagmkI
- Cílové soubory ke kompilaci = main.c
- Frekvenci procesoru = 14745600Hz

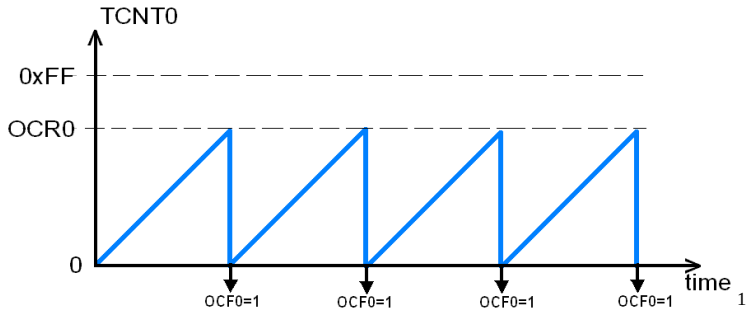
Softwarová stránka implementace řešení běžící na mikrokontroleru ATmega128 je v základu postavena na práci s těmito třemi registry:

- Registr směru = DDRx
- Datový /výstupní registr = PORTx
- Vstupní registr = PINx

Nadále je v řešení implementovaný čítač/časovač v režimu CTC, který funguje na principu nastavení maximální hodnoty tiků, po jejichž dosažení se jejich hodnota anuluje a přičítání pokračuje znovu. Způsob konfigurace pro konkrétní řešení je vypsán níže. V konfiguraci je zvolena předdělička o velikosti 1024 a maximální hodnota je vypočtena dynamicky v závislosti na frekvenci procesoru tak, aby se přerušení vykonávalo každou sekundu.

```
uint16_t compare_value = (F_CPU / (1024 * TICK_INTERVAL)) - 1; // dynamic max. value calculation
TCCR1B = (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode, prescaler 1024
TCNT1 = 0;                                         // reset timer
OCR1A = compare_value;                             // set the compare value
TIMSK |= (1 << OCIE1A);                            // enable interrupt on compare match
```

Diagram průběhu CTC čítače:



[1]

Popis programu:

Fáze inicializace:

Ihned po překladu kódu nastani integrální činnost inicializace. V průběhu této činnosti jsou deklarovány všechny globální proměnné definovány jak v hlavičkovém souboru *config.h*, tak i ty v hlavním kódu. Jmenovitě nejdůležitějšími jsou:

- Custom struktura *Clock* sloužící k ukládání aktuálního času
- Custom struktura *Button* pro jednodušší práci s tlačítky
- Boolen proměnná *editting* značící aktuální mód
- Boolen proměnná *am* značící odpolední či dopolední čas

V moment, kdy jsou inicializovány tyto globální objekty a jejich instance, počne se s vykonáváním obsahu funkce *setup*. Jako první se nastaví port dopomocných LED diod, umístěných na panelu s tlačítky, do stavu výstupu a zapíše se na něj hodnota 0xFF pro zhasnutí. Následně se pokračuje k konfigurací portů pro datové a konfigurační registry společně s portem tlačítek. Poslední inicializační činností je nastavení interního timeru do CTC režimu a povolení přerušení.

Hlavní smyčka programu:

Ihned po ukončení fáze inicializace se krom hlavní smyčky spouští i interní timer. V moment, kdy je dosaženo trigger hodnoty je spuštěna obsluha přerušení timeru (*ISR*). Ihned po vyvolání je zkontrolován stav globální proměnné editting, pokud je řádek ve stavu editu, je obsluha přerušena, aby nedocházelo k neustálému přičítání času při nastavování. V opačném případě je však k aktuálnímu stavu jednotek sekund (clock.seconds_ones) přičtena 1. Následně je řadou větvení zkontrolováno přetečení jakékoliv informace do vyššího řádu a poté se vykoná dodatečná logika buď pro 12 nebo 24 hodinový formát. U 12 hodinového formátu je kontrolován přesah hodnoty 12:59:59 na displeji, tak aby se při přičtení jedné sekundy čas automaticky přeformátoval na 1:00:00 a hodiny indikovali přepnutí z AM na PM či naopak (*am=!am*).

Vzhledem k tomu, že je funkce ISR asynchronní nijak, krom úpravy aktuálního času neovlivňuje vykonávání hlavní smyčky programu. Ta pak funguje na principu neustálého updatu hodnoty na displeji a následné kontroly všech 4 tlačítek (mode, plus, minus, edit).

Co se samotného zobrazování aktuálního času na displeji týče, tak je přímo pro tuto akci v kódu definována funkce *displayTime*. Daná funkce iteruje skrz všechny sloupce matice a postupně si z globální proměnné definující font (*MAP*) bere řádková data pro aktuální sloupec. Aby bylo možné zobrazovat speciální charaktery, je zde využito jednoduché logiky, která při získávání dat pro poslední sloupec upraví příslušná řádková data tak, aby obsahovali dvojtečku či znak pro AM/PM čas. Zápis na displej je obsluhováno metodou *writeToRegister*, které jsou data pro aktuální řádky přikládána společně s adresou příslušné matice. Důležitým detailem volání je negace přikládaných dat, ta je prováděna z důvodu, že displej pracuje s negovanou logickou. Informace zapsaná na tyto řádkové registry však k zobrazení informace nestačí, je totiž ještě nutné na dobu zobrazení zaktivovat příslušný sloupcový registr (*((0x01 << číslo_sloupce) ^ 0xFF*) a po dané době ho deaktivovat (data = 0xFF), což je opět provedeno funkcí *writeToRegister*.

[1] aboratorul 1: Întreruperi, Timere [CS Open CourseWare] (pub.ro)

Funkce *writeToRegister* pracuje na jednoduchém principu zapsání informace obdržené v parametru dat na datový port registrů řádku, společně s krátkým pulzem dat, jenž definují adresu příslušného sloupce, zapsaným na konfigurační registr.

Kontrola stavu tlačítek je prováděna pomocí funkce *buttonPressed*, která při vyvolání přečte stav portu a po aplikování masky determinuje stav tlačítka.

Pokud je stisknuto tlačítko *MODE,* je za využití datové struktury *Button* a její proměnné *counter* kontrolována doba stisku. Pokud došlo k dostatečně dlouhému stisku a řádek je v módu editování proběhne posunutí pomyslného kurzoru o 1 matici doprava a stav je indikován probliknutém indikačních LED diod. V případě, že řádek není ve stavu editace, spustí se metoda přepnutí aktuálního režimu hodin *switchClockFormat*.

Funkce *switchClockFormat* ihned po spuštění kontroluje stav aktuálního časového formátu a na základě něj převrátí aktuální formát na opačný. Následně vyhodnocuje, zda byl čas přepnut na 24 hodinový formát. Pokud ano a byl ve 12 hodinovém formátu čas nastaven na PM, je ke stávajícímu vyjádření hodin 12 a výsledný čas je uložen. V opačném případě, tedy v případě, kdy byl čas přepnut do 12 hodinového formátu, je podmínkou zkontrolováno, zdali je čas menší než maximální 12 hodinový časový údaj a na základě toho nastavena hodnota proměnné *am*. Aby se docílilo efektivního převodu tak, následuje sekce kódu mající na starost převod času nad 20:00 na PM čas (převod odečtením 12 z aktuální reprezentace hodin).

Stisk tlačítka *PLUS* je vyhodnocen pomocí stejné logiky jako tlačítko *MODE*, jediným rozdílem je, že proměnná, na které se kontroluje *counter* je nyní *plus_button.* Akce, jež je vykonána při stisku je změna aktuální vybrané hodnoty o plus 1 (funkce *editTime*) a reset aktuálně vybrané matice (funkce *resetMatrix*). Tato akce však nastane jen v moment, kdy je nastavena proměnná *editting* na *true*.

Podobně funguje tlačítko *MINUS*, ovšem zde je místo inkrementace hodnota dekrementována o 1 a *counter* proměnná je brána z proměnné *minus_button.*

Po stisku tlačítka *EDIT* je opět jako u tlačítek předchozích inkrementován *counter*, který je uchován v proměnné *edit_button*. Jakmile je dosaženo trigger hodnoty, je převrácena log. Hodnota proměnné editting (*true-> false* a *false->true*).

Rozbor Proměnných a metod:

Metody:

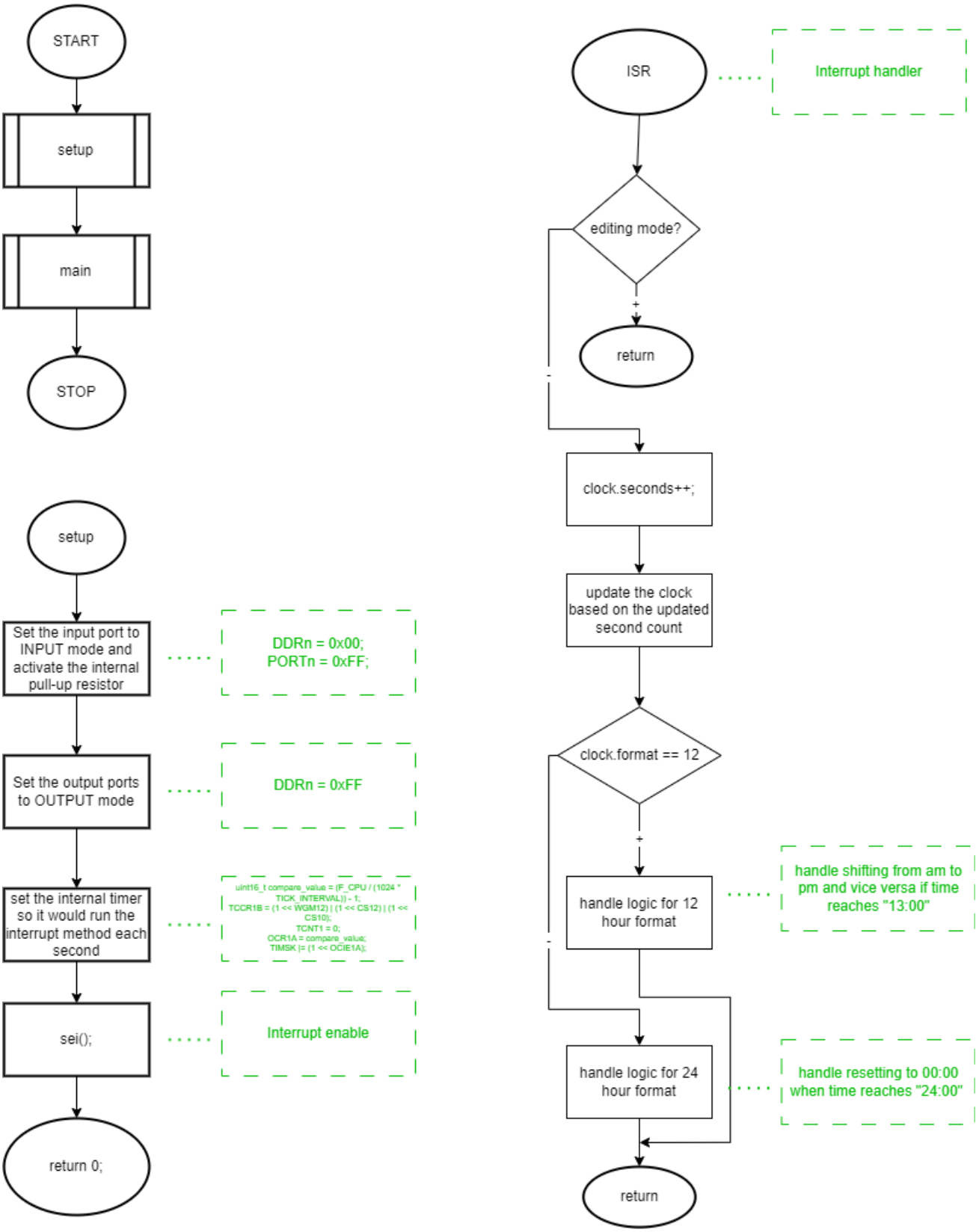
main.c			
Návratový typ	Název	Parametry	Popis
void	writeToRegister(...)	uint8_t data, int register_addr	Zapisuje data do registru podle zadané adresy.
void	resetDisplay()	-	Resetuje celý displej.
void	resetMatrix(...)	int address	Resetuje jednu matici na displeji.
void	displayTime()	-	Zobrazuje aktuální čas na displeji.
bool	buttonPressed(...)	uint8_t address, int index	Kontroluje, zda je stisknuto tlačítko.
void	editTime(...)	int delta, int matrixIndex	Upravuje čas podle zadaného rozdílu (delta) a indexu matice.
void	switchClockFormat()	-	Přepíná mezi formáty času (12h/24h).
void	setup()	-	Nastavuje komunikaci a počáteční stavy pinů.
void	main()	-	Hlavní metoda programu.
void	ISR(TIMER1_COMPA_vect)	-	Obslužná rutina přerušení pro Timer 1 pro aktualizaci hodin.

Proměnné:

main.c		
Typ	Název	Popis
Clock	clock	Instance struktury Clock, reprezentující stav hodin.
Button	mode_button	Instance tlačítka pro režim (mode).
Button	plus_button	Instance tlačítka pro zvětšení hodnoty.
Button	minus_button	Instance tlačítka pro zmenšení hodnoty.
Button	edit_button	Instance tlačítka pro úpravu.
bool	editing	Proměnná indikující stav editace.
int	edit_matrix_index	Index matice pro úpravu.
bool	am	Indikace AM či PM formátu.

config.h		
Typ	Název	Popis
Makro	MATRIX_DATA_CONF	Registr pro konfiguraci datových pinů matice.
Makro	MATRIX_CONTROL_CONF	Registr pro konfiguraci řídicích pinů matice.
Makro	MATRIX_DATA_OUT	Výstupní port pro data matice.
Makro	MATRIX_CONTROL_OUT	Výstupní port pro řízení matice.
Makro	BUTTON_PORT_CONF	Registr pro konfiguraci portu tlačítek.
Makro	BUTTON_DATA_OUT	Výstupní port pro data tlačítek.
Makro	BUTTON_DATA_IN	Vstupní port pro data tlačítek.
Makro	MATRIX_CONTROL_0	Hodnota pro ovládání prvního řádku matice.
Makro	MATRIX_CONTROL_1	Hodnota pro ovládání druhého řádku matice.
Makro	MATRIX_CONTROL_2	Hodnota pro ovládání třetího řádku matice.
Makro	MATRIX_CONTROL_3	Hodnota pro ovládání čtvrtého řádku matice.
Makro	MATRIX_CONTROL_4	Hodnota pro ovládání pátého řádku matice.
Makro	MATRIX_CONTROL_5	Hodnota pro ovládání šestého řádku matice.
Makro	MATRIX_CONTROL_8	Hodnota pro ovládání osmého řádku matice.
Makro	MATRIX_COLUMN_INDEX	Index sloupce matice.
Makro	DISPLAY_DELAY	Zpoždění pro zobrazení na displeji.
Makro	STANDARD_DELAY	Standardní zpoždění.
Makro	BTN_STANDARD_TRIGGER	Standardní spouštěcí prahová hodnota pro tlačítko.
Makro	BTN_LONG_TRIGGER	Dlouhá spouštěcí prahová hodnota pro tlačítko.
Makro	MODE_BTN	Adresa tlačítka pro režim.
Makro	PLUS_BTN	Adresa tlačítka pro zvětšení hodnoty.
Makro	MINUS_BTN	Adresa tlačítka pro zmenšení hodnoty.
Makro	EDIT_BTN	Adresa tlačítka pro úpravu.
Konstantní pole	MAP	Pole obsahující mapování číslic na jejich binární reprezentaci.
Makro	TICK_INTERVAL	Interval tiknutí pro přerušení (jedna sekunda).

Vývojový diagram:



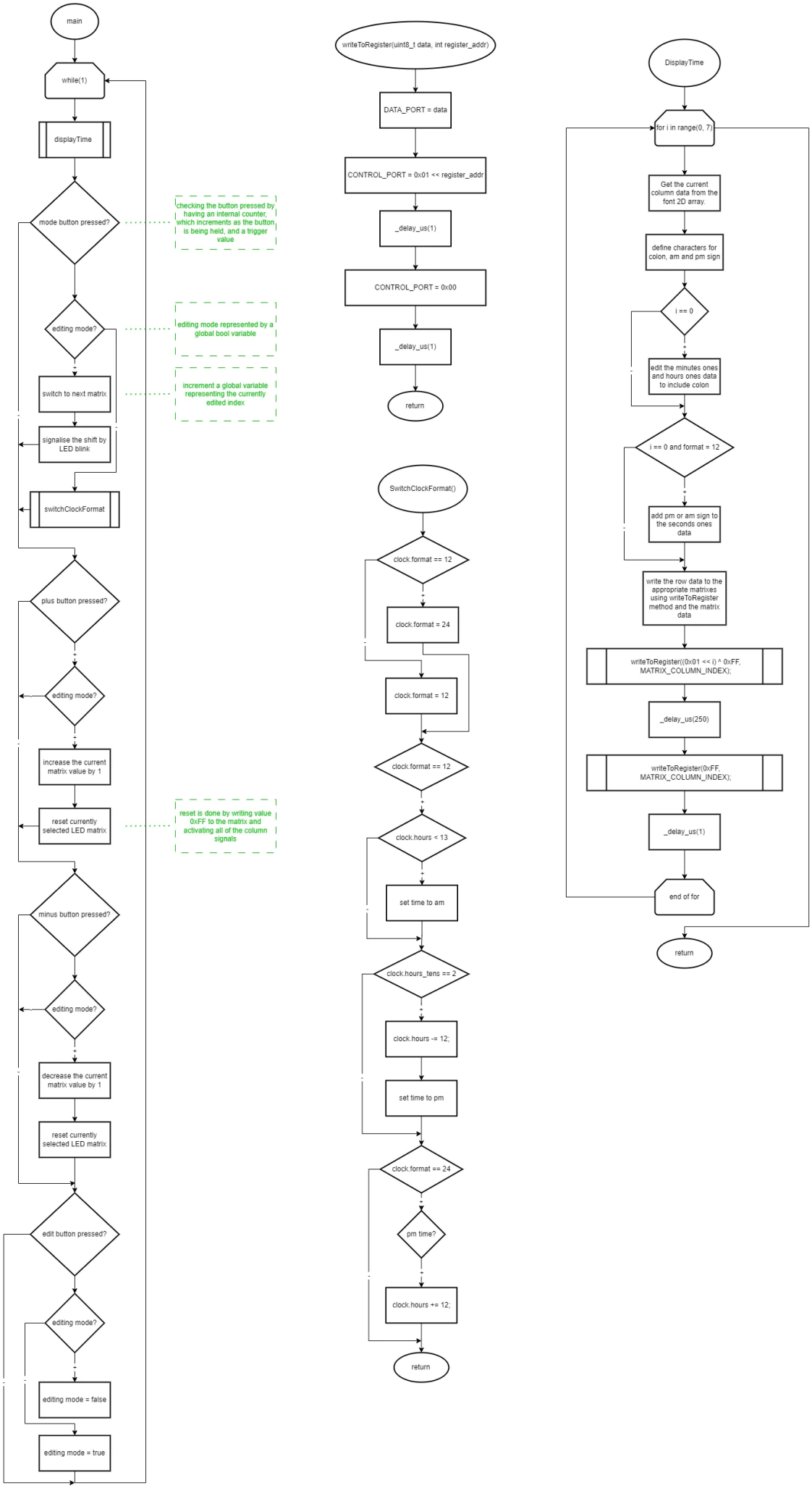
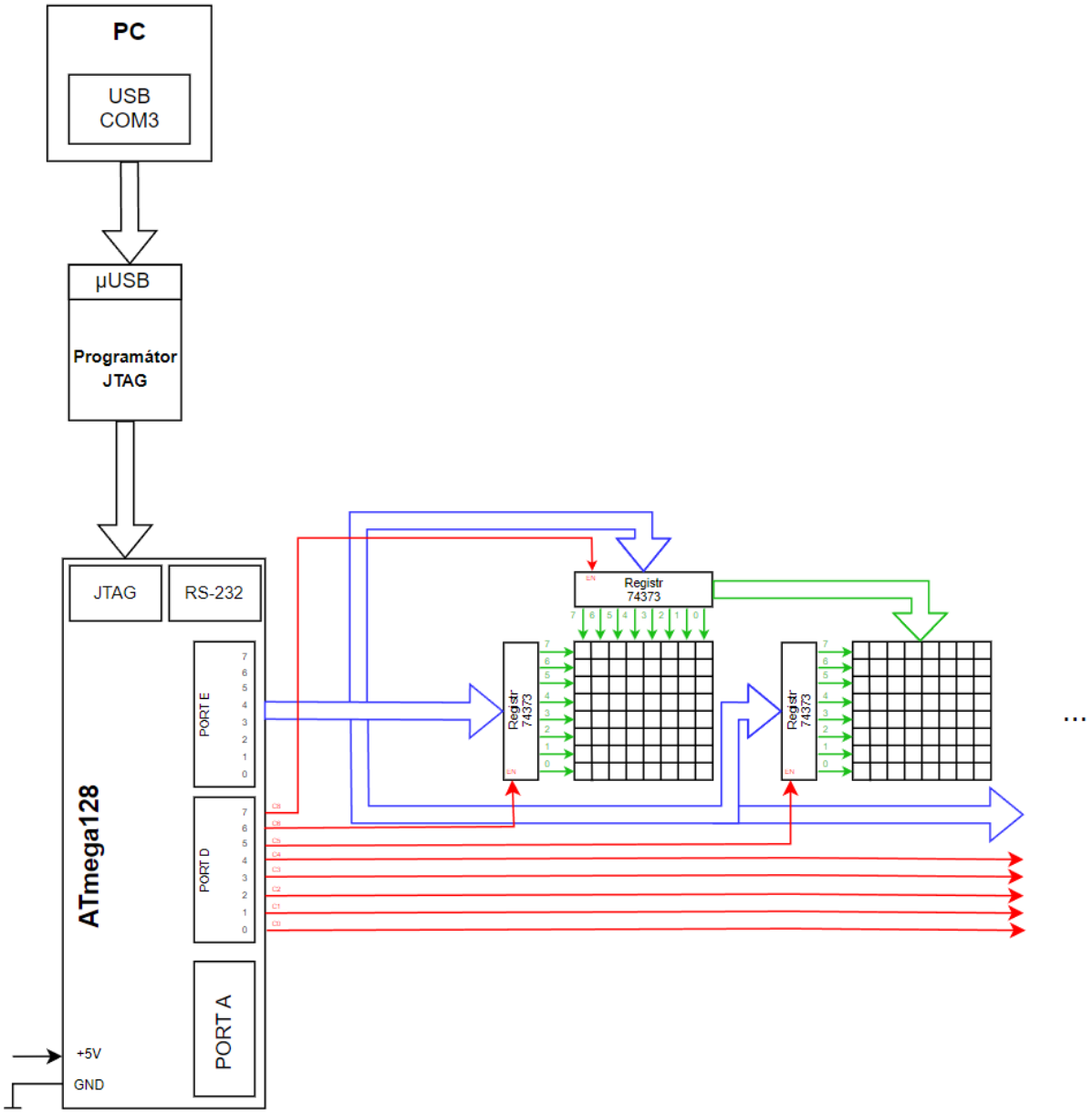

main
while(1)
displayTime
mode button pressed?
checking the button pressed by having an internal counter, which increments as the button is being held, and a trigger value
editing mode?
editing mode represented by a global bool variable
switch to next matrix
increment a global variable representing the currently edited index
signalise the shift by LED blink
switchClockFormat
plus button pressed?
editing mode?
increase the current matrix value by 1
reset currently selected LED matrix
reset is done by writing value 0xFF to the matrix and activating all of the column signals
minus button pressed?
editing mode?
decrease the current matrix value by 1
reset currently selected LED matrix
edit button pressed?
editing mode?
editing mode = false
editing mode = true
writeToRegister(uint8_t data, int register_addr)
DATA_PORT = data
CONTROL_PORT = 0x01 << register_addr
_delay_us(1)
CONTROL_PORT = 0x00
_delay_us(1)
return
SwitchClockFormat()
clock.format == 12
clock.format = 24
clock.format = 12
clock.format == 12
clock.hours < 13
set time to am
clock.hours_tens == 2
clock.hours -= 12;
set time to pm
clock.format == 24
pm time?
clock.hours += 12;
return
DisplayTime
for i in range(0, 7)
Get the current column data from the font 2D array.
define characters for colon, am and pm sign
i == 0
edit the minutes ones and hours ones data to include colon
i == 0 and format = 12
add pm or am sign to the seconds ones data
write the row data to the appropriate matrixes using writeToRegister method and the matrix data
writeToRegister((0x01 << i) ^ 0xFF, MATRIX_COLUMN_INDEX);
_delay_us(250)
writeToRegister(0xFF, MATRIX_COLUMN_INDEX);
_delay_us(1)
end of for
return

Schéma zapojení:



Komentovaný výpis program:

main.c:

```
#include "config.h"

void writeToRegister(uint8_t data, int register_addr);
void resetDisplay(void);
void resetMatrix(int addr);
void displayTime(void);
bool buttonPressed(uint8_t address, int index);
void editTime(int delta, int matrixIndex);
void switchClockFormat();

/**
 * @brief Structure to represent the clock state.
 */
typedef struct main_clock
{
    volatile int hours_tens;   /**< The tens digit of the hours. */
```

```
    volatile int hours_ones;   /**< The ones digit of the hours. */
    volatile int minutes_tens; /**< The tens digit of the minutes. */
    volatile int minutes_ones; /**< The ones digit of the minutes. */
    volatile int seconds_tens; /**< The tens digit of the seconds. */
    volatile int seconds_ones; /**< The ones digit of the seconds. */
    volatile int format;       /**< The format of the clock (12 or 24). */
} Clock;

typedef struct main_button
{
    volatile uint8_t address;
    volatile int index;
    volatile int counter;
} Button;

Clock clock = {0, 0, 0, 0, 0, 0, 24}; /**< The instance of the Clock structure. */

// button instances
Button mode_button = {MODE_BTN, 0, 0};
Button plus_button = {PLUS_BTN, 1, 0};
Button minus_button = {MINUS_BTN, 2, 0};
Button edit_button = {EDIT_BTN, 3, 0};

bool editting = false;
int edit_matrix_index = 0;
bool am = true;
int helper = -1;

/**
 * @brief Sets up the communication and initial pin states.
 */
void setup(void)
{
    // setup of the ports
    // set ports as output

    DDRA = 0xFF;
    PORTA = 0xFF;

    MATRIX_DATA_CONF = 0xFF;
    MATRIX_CONTROL_CONF = 0xFF;

    // set button port as input
    // buttons
    BUTTON_PORT_CONF = 0x00;
    BUTTON_DATA_OUT = 0xFF;

    // set the clock timer
    uint16_t compare_value = (F_CPU / (1024 * TICK_INTERVAL)) - 1; // dynamic max. value
calculation
    TCCR1B = (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode, prescaler 1024
    TCNT1 = 0;                                         // reset timer
    OCR1A = compare_value;                             // set the compare value
```

```
    TIMSK |= (1 << OCIE1A);                                 // enable interrupt on compare
match
    sei();                                                  // enable global interrupts
}

/**
 * @brief The main method of the program.
 */
void main(void)
{
    setup();

    resetDisplay();

    while (1)
    {
        // check for button press so that time and format can be adjusted -> TODO
        displayTime();

        if (buttonPressed(mode_button.address, mode_button.index))
        {
            mode_button.counter++;

            if (mode_button.counter % BTN_STANDARD_TRIGGER == 0)
            {
                if (editting)
                {
                    edit_matrix_index++;
                    if (edit_matrix_index > 5)
                    {
                        edit_matrix_index = 0;
                    }

                    PORTA = 0x00;
                    _delay_ms(250);
                    PORTA = 0xFF;
                }
                else
                {
                    switchClockFormat();
                }
                mode_button.counter = 0;
            }
        }

        if (buttonPressed(plus_button.address, plus_button.index))
        {
            plus_button.counter++;
            if (plus_button.counter % BTN_STANDARD_TRIGGER == 0)
            {
                if (editting)
                {
                    editTime(1, edit_matrix_index);
                    resetMatrix(edit_matrix_index);
```

```
                }

                plus_button.counter = 0;
            }
        }

        if (buttonPressed(minus_button.address, minus_button.index))
        {
            minus_button.counter++;
            if (minus_button.counter % BTN_STANDARD_TRIGGER == 0)
            {
                if (editting)
                {
                    editTime(-1, edit_matrix_index);
                    resetMatrix(edit_matrix_index);
                }

                minus_button.counter = 0;
            }
        }

        if (buttonPressed(edit_button.address, mode_button.index))
        {
            edit_button.counter++;
            if (edit_button.counter % BTN_STANDARD_TRIGGER == 0)
            {
                editting = !editting;
                edit_button.counter = 0;
            }
        }
    }
}

void editTime(int delta, int matrixIndex)
{

    // increment the cl
    switch (matrixIndex)
    {
    case 0:
        clock.hours_tens += delta;
        if (clock.hours_tens < 0)
            clock.hours_tens = 0;
        break;
    case 1:
        clock.hours_ones += delta;
        if (clock.hours_ones < 0)
            clock.hours_ones = 0;
        break;
    case 2:
        clock.minutes_tens += delta;
        if (clock.minutes_tens < 0)
            clock.minutes_tens = 0;
        break;
```

```
    case 3:
        clock.minutes_ones += delta;
        if (clock.minutes_ones < 0)
            clock.minutes_ones = 0;
        break;
    case 4:
        clock.seconds_tens += delta;
        if (clock.seconds_tens < 0)
            clock.seconds_tens = 0;
        break;
    case 5:
        clock.seconds_ones += delta;
        if (clock.seconds_ones < 0)
            clock.seconds_ones = 0;
        break;
    }

    if (clock.seconds_ones == 10)
    {
        clock.seconds_ones = 0;
        clock.seconds_tens++;
    }
    if (clock.seconds_tens == 6)
    {
        clock.seconds_tens = 0;
        clock.minutes_ones++;
    }
    if (clock.minutes_ones == 10)
    {
        clock.minutes_ones = 0;
        clock.minutes_tens++;
    }
    if (clock.minutes_tens == 6)
    {
        clock.minutes_tens = 0;
        clock.hours_ones++;
    }
    if (clock.hours_ones == 10)
    {
        clock.hours_ones = 0;
        clock.hours_tens++;
    }

    // Check clock format and adjust hours accordingly
    if (clock.format == 12)
    {
        if (clock.hours_tens == 1 && clock.hours_ones == 3)
        {
            // If it's currently 13 (1 PM), reset to 1 (1 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 1;
            am = !am;
        }
        else if (clock.hours_tens > 1)
```

```
        {
            // If it's currently greater than 12, reset to 1 (1 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 1;
        }
    }
    else if (clock.format == 24)
    {
        if (clock.hours_tens == 2 && clock.hours_ones == 4)
        {
            // If it's currently 24 (midnight), reset to 0 (12 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 0;
        }
        else if (clock.hours_tens > 2)
        {
            // If it's currently greater than 23, reset to 0 (12 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 0;
        }
    }
}

bool buttonPressed(uint8_t address, int index)
{
    if ((BUTTON_DATA_IN & address) >> index != 0)
    {
        // button pressed
        return false;
    }
    return true;
}

void switchClockFormat()
{
    if (clock.format == 24)
    {
        clock.format = 12;
    }
    else
    {
        clock.format = 24;
    }


    // if time format is ewual to 12h -> update the current state of the clock
    // the state of the 12h clock should also reflect if it is AM or PM -> simply add or
delete LED to some segment
    if(clock.format == 12)
    {
        int hours = (clock.hours_tens * 10 + clock.hours_ones);

        if(hours < 13){
            am = true;
```

```
        }

        // possible time = 23:00 -> shift to 11PM
        if(clock.hours_tens == 2){
            int converted = hours - 12;

            int tens = converted / 10;
            int ones = converted % 10;

            clock.hours_tens = tens;
            clock.hours_ones = ones;

            am = false;
        }
    }

    if(clock.format == 24){
        // check if time is pm or am
        if(!am){
            // time is AM -> can keep the values
            // time is PM -> need to convert to numbers > 12
            int converted = (clock.hours_tens * 10 + clock.hours_ones) + 12;

            int tens = converted / 10;
            int ones = converted % 10;

            clock.hours_tens = tens;
            clock.hours_ones = ones;
        }
    }
}

/**
 * @brief Displays the current time on the matrix.
 */
void displayTime()
{
    // function to display current state of the clock
    for (int i = 0; i < 8; i++)
    {
        // for each of the columns
        // get "row" data from the current time variables -> HH:MM:SS

        uint8_t hours_tens_data = MAP[clock.hours_tens][7 - i];
        uint8_t hours_ones_data = MAP[clock.hours_ones][7 - i];
        uint8_t minutes_ones_data = MAP[clock.minutes_ones][7 - i];
        uint8_t minutes_tens_data = MAP[clock.minutes_tens][7 - i];
        uint8_t seconds_tens_data = MAP[clock.seconds_tens][7 - i];
        uint8_t seconds_ones_data = MAP[clock.seconds_ones][7 - i];

        // set the colons to the font
        uint8_t colon = 0b00100100;
        uint8_t format_sign = 0b11000000;
        uint8_t am_sign = 0b00001100;
```

```
        uint8_t pm_sign = 0b00000011;

        // check if the current displayed row is 0, if yes add colons to the font
        if (i == 0)
        {
            minutes_ones_data |= colon;
            hours_ones_data |= colon;
        }

        if (i == 0 && clock.format == 12)
        {
            seconds_ones_data |= format_sign;

            if(am){
                seconds_ones_data |= am_sign;
            }
            else{
                seconds_ones_data |= pm_sign;
            }
        }

        // write data to appropriate registers -> data bust be negated -> display active
in log. 0
        writeToRegister(hours_tens_data ^ 0xFF, 0);
        writeToRegister(hours_ones_data ^ 0xFF, 1);
        writeToRegister(minutes_tens_data ^ 0xFF, 2);
        writeToRegister(minutes_ones_data ^ 0xFF, 3);
        writeToRegister(seconds_tens_data ^ 0xFF, 4);
        writeToRegister(seconds_ones_data ^ 0xFF, 5);

        // activate the column register
        writeToRegister((0x01 << i) ^ 0xFF, MATRIX_COLUMN_INDEX);

        // wait for the display period
        _delay_us(250);

        // reset the column register
        writeToRegister(0xFF, MATRIX_COLUMN_INDEX);
        _delay_us(1);
        helper++;
    }
}

/**
 * @brief Resets the entire display.
 */
void resetDisplay()
{
    // function to reset the whole display
    for (int i = 7; i >= 0; i--)
    {
        resetMatrix(i);
    }
}
```

```
/**
 * @brief Resets a matrix.
 *
 * @param addres The address of the matrix.
 */
void resetMatrix(int addres)
{
    // function to reset one matrix
    for (int j = 0; j < 6; j++)
    {
        writeToRegister(0xFF, j);
    }

    writeToRegister((0x01 << addres) ^ 0xFF, MATRIX_COLUMN_INDEX);
    _delay_us(1);
    writeToRegister(0xFF, MATRIX_COLUMN_INDEX);
    _delay_us(1);
}

/**
 * @brief ISR for Timer 1 for updating the clock.
 */
ISR(TIMER1_COMPA_vect)
{
    if (editting)
    {
        // if user is edditing the current state of the clock ->  do not increment timer
        return;
    }
    // increment the clock
    clock.seconds_ones++;

    if (clock.seconds_ones == 10)
    {
        clock.seconds_ones = 0;
        clock.seconds_tens++;
    }
    if (clock.seconds_tens == 6)
    {
        clock.seconds_tens = 0;
        clock.minutes_ones++;
    }
    if (clock.minutes_ones == 10)
    {
        clock.minutes_ones = 0;
        clock.minutes_tens++;
    }
    if (clock.minutes_tens == 6)
    {
        clock.minutes_tens = 0;
        clock.hours_ones++;
    }
    if (clock.hours_ones == 10)
```

```
    {
        clock.hours_ones = 0;
        clock.hours_tens++;
    }

    // Check clock format and adjust hours accordingly
    if (clock.format == 12)
    {
        if (clock.hours_tens == 1 && clock.hours_ones == 3)
        {
            // If it's currently 13 (1 PM), reset to 1 (1 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 1;
            am = !am;
        }
        else if (clock.hours_tens > 1)
        {
            // If it's currently greater than 12, reset to 1 (1 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 1;
        }
    }
    else if (clock.format == 24)
    {
        if (clock.hours_tens == 2 && clock.hours_ones == 4)
        {
            // If it's currently 24 (midnight), reset to 0 (12 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 0;
        }
        else if (clock.hours_tens > 2)
        {
            // If it's currently greater than 23, reset to 0 (12 AM)
            clock.hours_tens = 0;
            clock.hours_ones = 0;
        }
    }
}

/**
 * @brief Writes data to a register.
 *
 * @param data The data to be written.
 * @param register_addr The address of the register.
 */
void writeToRegister(uint8_t data, int register_addr)
{
    // function that writes data to register
    // convert data to inverted form
    // set data to data port
    MATRIX_DATA_OUT = data;

    // activate the appropriate register address
    MATRIX_CONTROL_OUT = (0x01 << register_addr);
```

```
    _delay_us(1);

    // deactivate the register
    MATRIX_CONTROL_OUT = 0x00;
    _delay_us(1);
}
```

config.h:

```
#ifndef CONFIG_H
#define CONFIG_H

// Include necessary libraries
#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

// include custom liraries
#include "bin_lib.h"

// global registers for DDR, PORT, PIN
#define MATRIX_DATA_CONF (DDRE)
#define MATRIX_CONTROL_CONF (DDRB)
#define MATRIX_DATA_OUT (PORTE)
#define MATRIX_CONTROL_OUT (PORTB)

#define BUTTON_PORT_CONF (DDRC)
#define BUTTON_DATA_OUT (PORTC)
#define BUTTON_DATA_IN (PINC)

// Global pin setup
#define MATRIX_CONTROL_0 0x01
#define MATRIX_CONTROL_1 0x02
#define MATRIX_CONTROL_2 0x04
#define MATRIX_CONTROL_3 0x08
#define MATRIX_CONTROL_4 0x10
#define MATRIX_CONTROL_5 0x20
#define MATRIX_CONTROL_8 0x40

#define MATRIX_COLUMN_INDEX 6

#define DISPLAY_DELAY 250
#define STANDARD_DELAY 1

#define BTN_STANDARD_TRIGGER 100
#define BTN_LONG_TRIGGER 250

// button setup
#define MODE_BTN 0x01
```

```
#define PLUS_BTN 0x02
#define MINUS_BTN 0x04
#define EDIT_BTN 0x08

// Global variable setup
// "map" between numbers and binary representation
// querying the map -> for loop (0 - 7) and the primary index is the actial number or
character to display
const unsigned char MAP[10][8] = {
    {0b00000000, 0b01111110, 0b10000001, 0b10000001, 0b10000001, 0b01111110, 0b00000000,
0b00000000}, // 0
    {0b00000000, 0b00100001, 0b01000001, 0b11111111, 0b00000001, 0b00000000, 0b00000000,
0b00000000}, // 1
    {0b00000000, 0b01000011, 0b10000101, 0b10001001, 0b10010001, 0b11100001, 0b00000000,
0b00000000}, // 2
    {0b00000000, 0b10000001, 0b10001001, 0b10001001, 0b11111111, 0b00000000, 0b00000000,
0b00000000}, // 3
    {0b00000000, 0b00001100, 0b00010100, 0b00100100, 0b11111111, 0b00000100, 0b00000000,
0b00000000}, // 4
    {0b00000000, 0b11110010, 0b10010001, 0b10010001, 0b10010001, 0b10001110, 0b00000000,
0b00000000}, // 5
    {0b00000000, 0b00111110, 0b01010001, 0b10010001, 0b10010001, 0b00001110, 0b00000000,
0b00000000}, // 6
    {0b00000000, 0b10000000, 0b10001111, 0b10010000, 0b10100000, 0b11000000, 0b00000000,
0b00000000}, // 7
    {0b00000000, 0b01101110, 0b10010001, 0b10010001, 0b10010001, 0b01101110, 0b00000000,
0b00000000}, // 8
    {0b00000000, 0b01100010, 0b10010001, 0b10010001, 0b10010010, 0b01111100, 0b00000000,
0b00000000}, // 9
};

#define TICK_INTERVAL 1 // tick after one second has passed

#endif // CONFIG_H
```

Odpovědi na otázky:

1. V rámci vysokorychlostních přenosů (např. LVDS pro zobrazovací panely) se často setkáte s tzv. symetrickým přenosem informace (differential signaling). Pokuste se stručně vysvětlit tento princip a důvody pro jeho použití.

   Princip symetrického přenosu informace spočívá v odesílání logické informace 2 vodiči, přičemž na vodič 1 je aplikováno klasické napětí reprezentující log. hodnotu, zatímco na vodiči 2 je napětí stejné velikosti s převrácenou polaritou. Výsledkem je pak přenos logické hodnoty s využitím dvojnásobného napětí než originální hodnota, což je velmi výhodné, protože vyšší napětí znamená vyšší odolnost vůči rušení a možnost odeslání signálu na vyšší vzdálenosti.[2]

2. Při použití zobrazovačů málokdy použijete přímé řízení jednotlivých bodů na úrovni hw, spíše bude displej k procesoru připojen pomocí některého ze standardních řadičů displeje. Najděte si jednoho zástupce a popište jeho komunikační protokol.

   Driver = SSD1306 (OLED display driver)

   Komunikační protokol tohoto driveru je mimo jiné I2C. Jedná se o velmi jednoduchý komunikační protokol, který ke komunikaci využívá 2 linek (SDA a SCL). Driver samotný je na tuto sběrnici připojen jako tzv. I2C

[2] https://en.wikipedia.org/wiki/Differential_signalling

slave, a tudíž je pro komunikaci třeba zvolit správně jeho adresu. Komunikace pak probíhá pomocí packetu (startbit, adresa + r/w bit, data, ack). Důležité specifikum tohoto protokolu je, že je ho datová linka ve stavu nečinnosti v log, 1, vyplývá to z konstrukce samotných slave zařízení.[3]

3. Představte si použití sériově řazených chytrých LED (např. NeoPixel) s dobou přenosu 1,25us/bit. Pokuste se odvodit nejvyšší rychlost zobrazení (překreslování) na displeji rozměru 40x40 bodů složeném z těchto LED při 8bitovém rozlišení pro každou dílčí barvu.¨

$$N = 40 \cdot 40 = 1600\ led\ diod$$

$$t_{TOT} = N \cdot rozlišení\ barev \cdot T_{přenosu\ 1\ bitu} = 1600 \cdot 8 \cdot 1{,}25 \cdot 10^{-6} = 0{,}016\ s$$

$$f_{TOT} = \frac{1}{t_{TOT}} = \frac{1}{0{,}016} = 62{,}5\ Hz$$

Z výše uvedeného výpočtu a obecné definice one-wire protokolu, běžně užívaného u sériově zapojených NeoPixel LED diod, vyplívá, že je maximální možná rychlost přenosu informace 62,5 Hz.

Závěr:

Výsledkem řešení této úlohy je plně funkční světelný řádek zobrazující aktuální čas ve dvou formátech (12 a 24 hodin) společně s rozlišením dopoledního a odpoledního času při zvolení 12hodinového formátu.

[3] https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf