Vyšší odborná škola a Střední průmyslová škola elektrotechnická
Božetěchova 3, Olomouc
Laboratoře elektrotechnických měření

SAMOSTATNÝ PROJEKT MIT

Název úlohy	Číslo úlohy
MĚŘIČ VZDÁLENOSTI	MIT-01

Zadání

1. Ultrazvukový snímač měří vzdálenost, která se vypisuje na alfanumerický display a na UART. LED ring signalizuje když je objekt blízko. Enkodérem nastavujeme minimální vzdálenost, kterou považuje za hodnotu kdy už je objekt blízko. Hodnota se ukazuje na displayi.

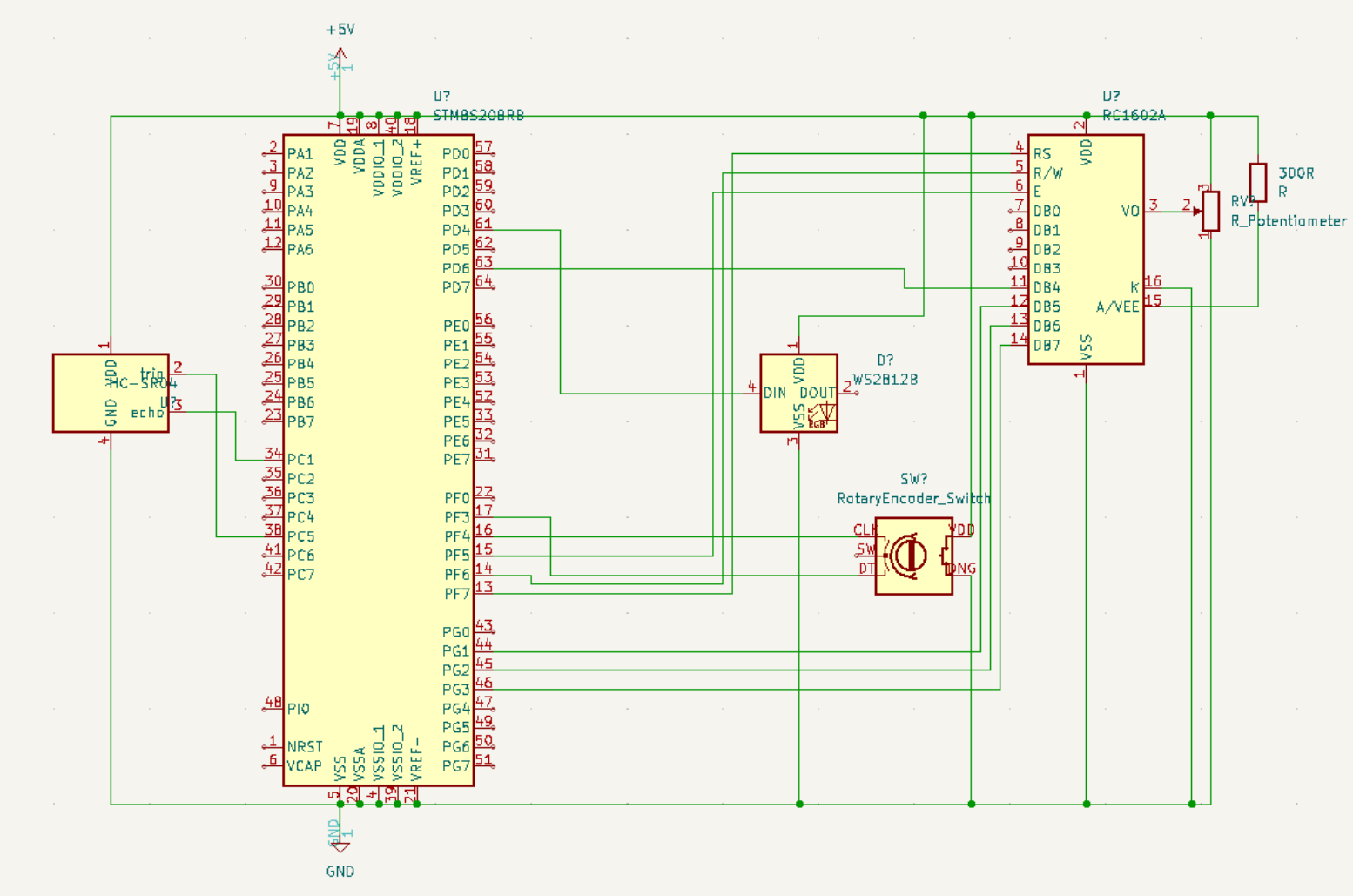
Použitý software: Visual Studio Code
KiCad

Doba vypracování: 12 hodin

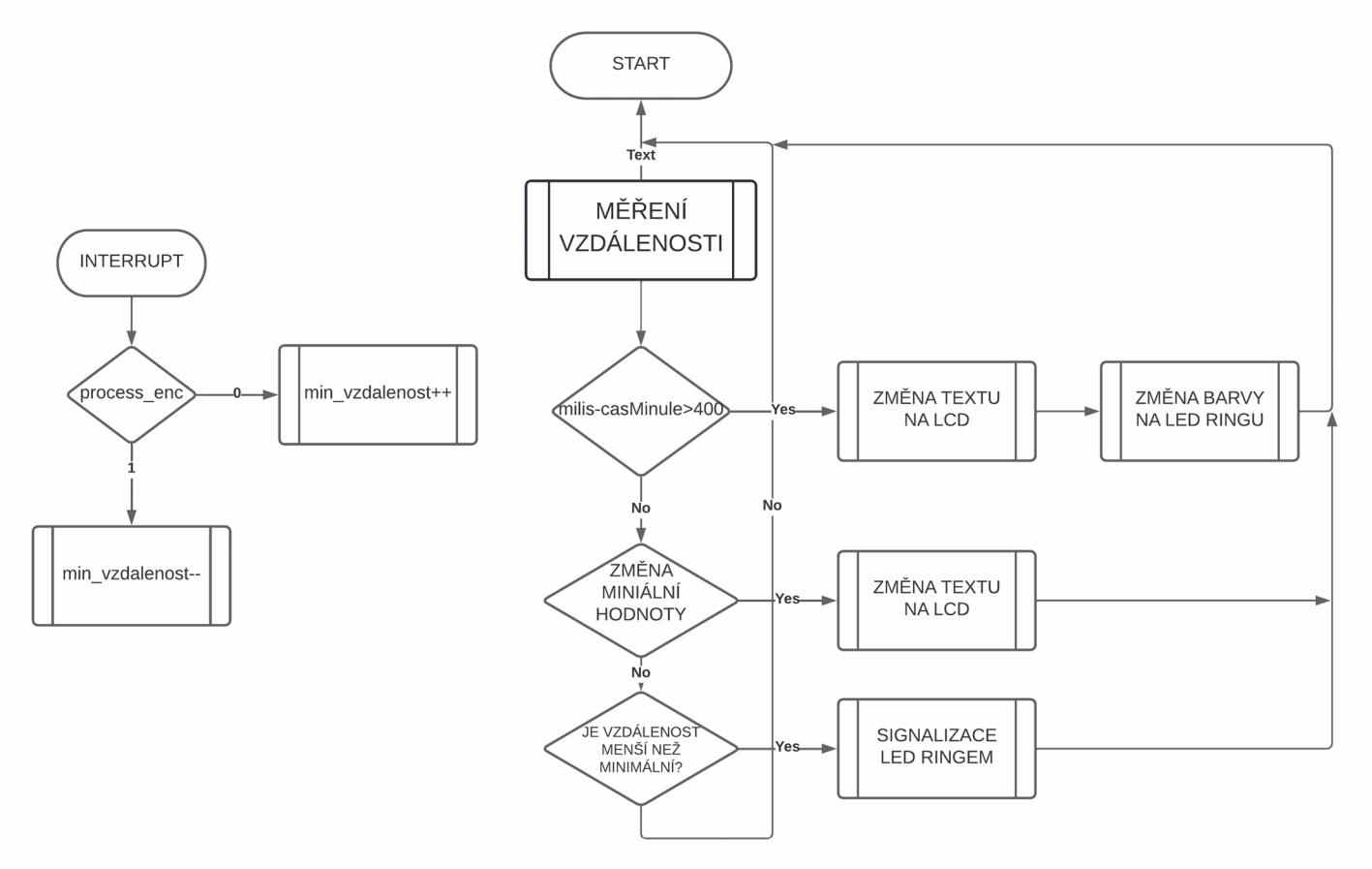
Poř. č.	Příjmení a jméno	Třída	Školní rok
31	VOJTĚCHOVSKÝ Michal	4A	2021/22

Datum vypracování	Počet listů
23.02.2022	9

SCHÉMA ZAPOJENÍ:



VÝVOJOVÝ DIAGRAM:



SLOVNÍ POPIS FUNKCE:

Při samotném startu je minimální vzdálenost nastavena na 15cm. Pomocí ultrazvukového snímače měříme vzdálenost od zařízení. Naměřenou vzdálenost převádíme na barevné spektrum a barvu posíláme na LED ring. Pokud je vzdálenost menší než minimální nastavená vzdálenost signalizujeme blikáním LED ringu. Minimální hodnotu nastavujeme enkodérem. Hodnota se vypisuje jak na LCD display tak po na UART.

ZÁVĚR

Zařízení může sloužit jako měření vzdálenosti či přítomnosti například v garáži při parkování auta. Kde vidíme vzdálenost na LED ringu bez potřebného čtení čísel. Malá nevýhoda ale je, že při nerovném povrchu a nestabilní vzdálenosti světlo různě problikuje. Tento problém by šlo do budoucna ošetřit lepším algoritmem.

MAIN.C

```
#include "stm8s.h"
#include "milis.h"
#include "delay.h"
#include <stdio.h>
#include "spse_stm8.h"
#include "stm8_hd44780.h"
#include "stm8s_adc2.h"
#include "uart1.h"

#define MASURMENT_PERON 444    // maximální celkový čas měření (ms)
#define CLK_PORT GPIOF
#define CLK_PIN  GPIO_PIN_4
#define DT_PORT GPIOF
#define DT_PIN  GPIO_PIN_3
//neopixel data PC6
#define L_PULSE 6 // 6*1/16MHz = 6*62.5 = 375ns (~400ns)
#define H_PULSE 12 // 12*1/16MHz = 12*62.5 = 750ns (~800ns)
#define PULSE_LEN 2 // délka spouštěcího (trigger) pulzu pro ultrazvuk
#define MEASURMENT_PERIOD 100 // perioda měření ultrazvukem (měla by být víc jak
(maximální_dosah*2)/rychlost_zvuku)

int16_t capture; // tady bude aktuální výsledek měření (času)
uint8_t capture_flag=0; // tady budeme indikovat že v capture je čerstvý výsledek
uint8_t colors[24*3];
uint16_t minVzdalenost = 15;
uint16_t vzdalenostMinule;
uint16_t minule=1; // pamatuje si minulý stav vstupu A (nutné k detekování sestupné hrany)
  // pokud je na vstupu A hodnota 0 a minule byla hodnota 1 tak jsme zachytili sestupnou hranu
bool zmena = FALSE;
uint32_t casMinuel;

void init_tim2(void){
  GPIO_Init(GPIOD,GPIO_PIN_4,GPIO_MODE_OUT_PP_LOW_FAST); // PD3 (TIM2_CH1) as output
  TIM2_OC1Init(TIM2_OCMODE_PWM2, TIM2_OUTPUTSTATE_ENABLE,1, TIM2_OCPOLARITY_HIGH); // One-Pulse
configuration (with CCR1/delay=1)
 TIM2_TimeBaseInit(TIM2_PRESCALER_1, L_PULSE); // Selecting prescaler (Period/ARR value will be
set to relevant value later)
 TIM2_SelectOnePulseMode(TIM2_OPMODE_SINGLE); // Selecting One Pulse Mode
}

void setup(void)
{
  CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);      // taktovani MCU na 16MHz

  init_milis();
  init_uart1();

   // na pinech/vstupech ADC_IN2 (PB4) a ADC_IN3 (PB5) vypneme vstupní buffer
  ADC2_SchmittTriggerConfig(ADC2_SCHMITTTRIG_CHANNEL4, DISABLE);
  ADC2_SchmittTriggerConfig(ADC2_SCHMITTTRIG_CHANNEL5, DISABLE);
```

```
  // při inicializaci volíme frekvenci AD převodníku mezi 1-4MHz při 3.3V
  // mezi 1-6MHz při 5V napájení
  // nastavíme clock pro ADC (16MHz / 4 = 4MHz)
  ADC2_PrescalerConfig(ADC2_PRESSEL_FCPU_D4);

  // volíme zarovnání výsledku (typicky vpravo, jen vyjmečně je výhodné vlevo)
  ADC2_AlignConfig(ADC2_ALIGN_RIGHT);

  // nasatvíme multiplexer na některý ze vstupních kanálů
  ADC2_Select_Channel(ADC2_CHANNEL_4);
  // rozběhneme AD převodník
  ADC2_Cmd(ENABLE);
  // počkáme než se AD převodník rozběhne (~7us)
  ADC2_Startup_Wait();
}

#define L_PATTERN 0b01110000    // 3x125ns (8MHZ SPI)
#define H_PATTERN 0b01111100    // 5x125ns (8MHZ SPI), first and last bit must be zero (to
remain MOSI in Low between frames/bits)
// takes array of LED_number * 3 bytes (RGB per LED)

void neopixel(uint8_t * data, uint16_t length)
{
  uint8_t mask;
  disableInterrupts(); // can be omitted if interrupts do not take more then about ~25us
  while(length){          // for all bytes from input array
    length--;
    mask=0b10000000;    // for all bits in byte
    while(mask){
      while(TIM2->CR1 & TIM2_CR1_CEN); // wait until timer stops (wait if transmitting last
bit)
      if(mask & data[length]){ // send pulse with coresponding length ("L" od "H")
        TIM2->ARRL = H_PULSE; // set pulse width for "H" bit
      }else{
        TIM2->ARRL = L_PULSE; // set pulse width for "L" bit
      }
      TIM2->CR1 |= TIM2_CR1_CEN; // Start timer (start single pulse generation)
      mask = mask >> 1;
    }
  }
  enableInterrupts();
}

void my_delay_ms(uint16_t ms) {
  uint16_t  i;
  for (i=0; i<ms; i = i+1){
    _delay_us(250);
    _delay_us(248);
    _delay_us(250);
    _delay_us(250);
  }
}
```

```
void process_enc(void){

  if(GPIO_ReadInputPin(CLK_PORT,CLK_PIN) == RESET && minule==1){
    minule = 0; // nyní je pin v log.0
    // pøeèteme stav vstupu B
    if(GPIO_ReadInputPin(DT_PORT,DT_PIN) == RESET){
      // log.0 na vstupu B (krok jedním smìrem)
      vzdalenostMinule = minVzdalenost;
      if(minVzdalenost < 400){
        minVzdalenost++;
      }

    }else{
      // log.1 na vstupu B (krok druhým smìrem)
      vzdalenostMinule = minVzdalenost;
      if(minVzdalenost > 10){
        minVzdalenost--;
      }
    }
  }
  if(GPIO_ReadInputPin(CLK_PORT,CLK_PIN) != RESET){minule = 1;} // pokud je vstup A v log.1
}

INTERRUPT_HANDLER(TIM3_UPD_OVF_BRK_IRQHandler, 15)
{
   TIM3_ClearITPendingBit(TIM3_IT_UPDATE); // vyeistit vlajku (nutné vždy)
   process_enc(); // zkontrolovat stav pinu enkodéru
}
void init_enc(void){
  // enkodéry jsou jen spínaèe, takže vstupy volíme ve stejném režimu jako pro tlaèítka
GPIO_Init(CLK_PORT,CLK_PIN,GPIO_MODE_IN_PU_NO_IT);  // vstup, s vnitøním pullup rezistorem
GPIO_Init(DT_PORT,DT_PIN,GPIO_MODE_IN_PU_NO_IT);
}

void init_tim3(void){
TIM3_TimeBaseInit(TIM3_PRESCALER_16,1999); // perioda pøeteèení/update 2ms
TIM3_ITConfig(TIM3_IT_UPDATE, ENABLE); // povolit pøerušení
TIM3_Cmd(ENABLE); // spustit timer
}


void clearAll(){
  uint8_t  i = 0;
  while(i<24){
    uint8_t index = i*3;
    colors[index] = 0;
    colors[index+1] = 0;
    colors[index+2] = 0;
    i = i+1;
  }
  neopixel(colors, sizeof(colors));
}
```

```
uint8_t stage=0; // stavový automat
uint16_t time=0; // pro časování pomocí milis
void process_measurment(void){
  switch(stage){
  case 0: // čekáme než uplyne  MEASURMENT_PERIOD abychom odstartovali měření
    if(milis()-time > MEASURMENT_PERIOD){
      time = milis();
      GPIO_WriteHigh(GPIOC,GPIO_PIN_5); // zahájíme trigger pulz
      stage = 1; // a bdueme čekat až uplyne čas trigger pulzu
    }
    break;
  case 1: // čekáme než uplyne PULSE_LEN (generuje trigger pulse)
    if(milis()-time > PULSE_LEN){
      GPIO_WriteLow(GPIOC,GPIO_PIN_5); // ukončíme trigger pulz
      stage = 2; // a přejdeme do fáze kdy očekáváme echo
    }
    break;
  case 2: // čekáme jestli dostaneme odezvu (čekáme na echo)
    if(TIM1_GetFlagStatus(TIM1_FLAG_CC2) != RESET){ // hlídáme zda timer hlásí změření pulzu
      capture = TIM1_GetCapture2(); // uložíme výsledek měření
      capture = capture * 0.034/2;
      printf("%d\r\n", capture);
      capture_flag=1; // dáme vědět zbytku programu že máme nový platný výsledek
      stage = 0; // a začneme znovu od začátku
    }else if(milis()-time > MEASURMENT_PERIOD){ // pokud timer nezachytil pulz po dlouhou dobu,
tak echo nepřijde
      stage = 0; // a začneme znovu od začátku
    }
    break;
  default: // pokud se cokoli pokazí
  stage = 0; // začneme znovu od začátku
  }
}

void init_tim1(void){
GPIO_Init(GPIOC, GPIO_PIN_1, GPIO_MODE_IN_FL_NO_IT); // PC1 (TIM1_CH1) jako vstup
TIM1_TimeBaseInit(15,TIM1_COUNTERMODE_UP,0xffff,0); // timer necháme volně běžet (do
maximálního stropu) s časovou základnou 1MHz (1us)
// Konfigurujeme parametry capture kanálu 1 - komplikované, nelze popsat v krátkém komentáři
TIM1_ICInit(TIM1_CHANNEL_1,TIM1_ICPOLARITY_RISING,TIM1_ICSELECTION_DIRECTTI,TIM1_ICPSC_DIV1,0);
// Konfigurujeme parametry capture kanálu 2 - komplikované, nelze popsat v krátkém komentáři
TIM1_ICInit(TIM1_CHANNEL_2,TIM1_ICPOLARITY_FALLING,TIM1_ICSELECTION_INDIRECTTI,TIM1_ICPSC_DIV1,
0);
TIM1_SelectInputTrigger(TIM1_TS_TI1FP1); // Zdroj signálu pro Clock/Trigger controller
TIM1_SelectSlaveMode(TIM1_SLAVEMODE_RESET); // Clock/Trigger má po příchodu signálu provést
RESET timeru
TIM1_ClearFlag(TIM1_FLAG_CC2); // pro jistotu vyčistíme vlajku signalizující záchyt a změření
echo pulzu
TIM1_Cmd(ENABLE); // spustíme timer ať běží na pozadí
}
char text[32] = "";
```

```
void LCD_print(void){
  sprintf(text,"hranice=%3ucm", minVzdalenost);
  lcd_clear();
  lcd_puts(text);
  sprintf(text,"vzdalenost=%3ucm", capture);
  lcd_gotoxy(0,1);
  lcd_puts(text);
}
uint8_t green (uint16_t x){
  x = x%256;
  if(x<42){
    return x*50/42;
  }
  if(x<128){
    return 50;
  }
  if(x<170){
    return 50-((x-128)*50/42);
  }
  return 0;
}

uint8_t red (uint16_t x){
  return green(x-170);
}

uint8_t blue (uint16_t x){
  return green(x-85);
}

void fillAll(uint8_t x) {

  uint8_t  i = 0;
  while(i<24){
    uint8_t index = i*3;
    colors[index] = blue(x);
    colors[index+1] = red(x);
    colors[index+2] = green(x);
    i = i+1;
  }

  neopixel(colors, sizeof(colors));
}

void tooClose(void){
    fillAll(255);
    my_delay_ms(100);
    clearAll();
    my_delay_ms(100);
  }
int main(void)
{
  uint8_t captureColor;
```

```
  vzdalenostMinule = minVzdalenost;
  CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1); // 16MHz from internal RC

  init_enc();        // inicializace vstupu enkodéru
  init_tim3();// spustí tim3 s poerušením každé 2ms
  init_tim1(); // nastavit a spustit timer
  init_tim2();
  setup();
  lcd_init();     // init GPIOS and init lcd to 4bit mode
  LCD_print();
  GPIO_Init(GPIOC, GPIO_PIN_5, GPIO_MODE_OUT_PP_LOW_SLOW); // výstup - "trigger pulz pro
ultrazvuk"
  casMinuel = milis();
  while (TRUE) {
    process_measurment();
    if( milis() - casMinuel >= 400){
      LCD_print();
      milis();
      if(capture>255){
      captureColor = 255;
      }
      else {
        captureColor = capture;
      }
      fillAll(captureColor);
      casMinuel = milis();
      }

    if(minVzdalenost != vzdalenostMinule){
      LCD_print();
      printf("%d\r\n", minVzdalenost);
      vzdalenostMinule = minVzdalenost;

    }
    if( capture < minVzdalenost){
      tooClose();
    }
  }
}
```