Poznań University of Technology

Faculty of Control, Robotics
and Electrical Engineering

Institute of Robotics and Machine Intelligence

Division of Control and Industrial Electronics

DC speed control using IR sensor

Microprocessor Systems

Project report

Wojtek Piersiala, 144651
wojtek.piersiala@student.put.poznan.pl

Bhargav Malasani Nagaraj, 139591
bhargav.malasaninagaraj@student.put.poznan.pl

Ram Ram, 146902
ram.ram@student.put.poznan.pl

Instructor:
Adrian Wójcik, M.Sc.
Adrian.Wojcik@put.poznan.pl

21-02-2022

Contents

Introduction

The final task of Microprocessor systems laboratory consists of microprocessor-based control and measurement system for DC motor speed measuring using IR sensor with the NUCLEO-F722ZE evaluation board as a hardware platform.

Specification

The project consists of a DC motor controlled by PWM via H-Bridge coupled with an IR sensor to measure the speed of the motor. The microcontroller is interfaced with the sensor using GPIO pins on the evaluation board, the communication between PC and the MCU is established using the serial port interface, it is also used to deploy the user interface to read and write the measurement and reference values. The user interface is established using an LCD to read the values of the measurement and reference. The control range achieved for the system is from 10Hz to 29Hz(600 RPM to 1,740 RPM) with the dead zone from 0Hz to 9 Hz(the motor will not rotate in this range) The following figure Fig.1 presents the block diagram of the general principle of the operation of the system.

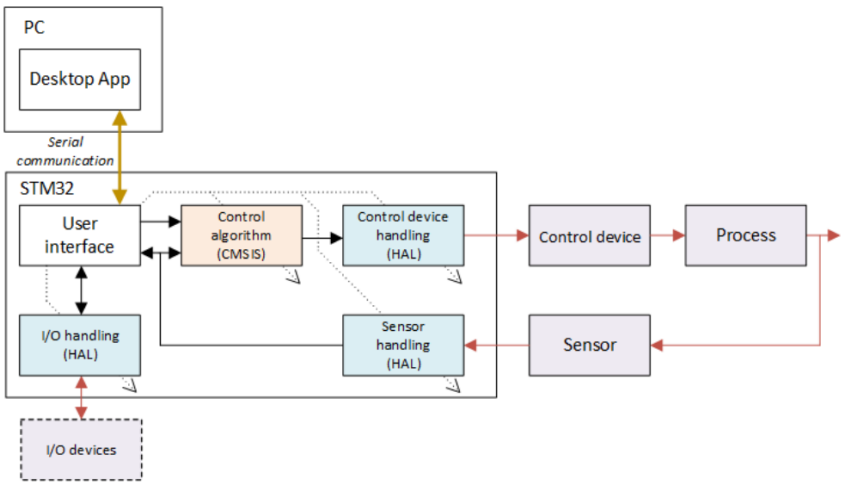

Fig. 1. Block Diagram of operation of the system

Implementation

2.1 Hardware

1 Control Device and Process

The device we are controlling is a DC motor that is connected to the MCU via H-Bridge. The DC motor can be controlled using Pulse Width Modulation by changing the duty cycle to control the speed of rotations. The H-bridge adds the possibility of controlling the direction of rotation by changing the polarity of the DC motor.

2 Sensor

The sensor module is made up of a set of proximity sensors coupled with the LM393 comparator. The principle of working is that the module senses the reflective surface of the rotor which passes the sensor many times, then the comparator sends this data in form of pulses which will be used to calculate the rotation frequency.

The Hardware Circuit can found in the appendix A

2.2 Software

1 Control Algorithm and Control Device Handling

The Timer callback function is used in order to model a PID controller and calculate the duty-cycle value using modeled PID controller.The listing lst.1 shows the implementation of the PID controller and duty-cycle Calculation.
The code snippet in listing lst.2 shows the initialization of the H-bridge. The listing lst.3 shows implementation of polarity change for the DC-motor

2 Sensor Handling

The GPIO external interupt callback function is used for interfacing with the sensor. The sensor modules sends the data in the form pulses which is used to calculate frequency. The listing lst.3 shows the calculation of the frequency.

3 I/O Handling

- Input: The input handling is implemented in two different ways, one is using a serial port and the rotary encoder, the reference value for the system is set using these two ways.

  In serial interface the reference value can set using the command 'Uxx' where ('xx' is the reference value).The serial interface is also used to request data to be displayed using the command 'R:x'('x' is Y,U and E for measurement value, reference value and error value respectively). The listing lst.4 shows the implementation of serial interface.

  In Rotary encoder interface the reference value can be increased or decreased by rotating the encoder in clock-wise and anti-clock-wise direction respectively. The listing lst.5 shows the implementation of rotary encoder

- Output: The output handling is implemented in two different ways, one is using a serial port and the LCD module.

  As discussed previously the serial interface can be used to request data to be displayed, it can display measurement value, reference value and error value.The listing lst.4 shows the implementation of serial interface.

  The LCD module can display measurement value and reference value.The code snippets from the listing lst.7 shows the implementation of LCD module.

The LCD module used is connected with the I2C expander. The drivers for the I2C can be found in the appendix C[1]

2.3 Simulation

1 Simulink Model

PID (Proportional-Integral-Derivative) control
The PID controller is a discrete-time controller running at 0.08 seconds.The model of a closed loop system uses the PID Controller block. This block generates a voltage signal driving the dc motor to track desired shaft rotation speed. In addition to voltage, the dc motor subsystem takes torque disturbance as an input, allowing us to simulate how well the controller rejects disturbances. We also modeled analog sensor noise in the speed measurement.

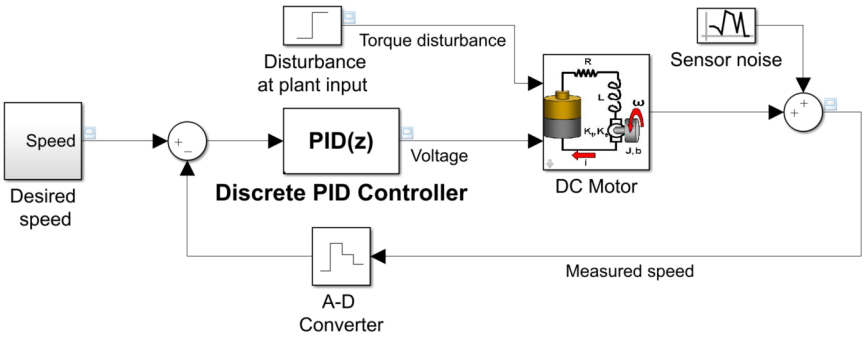

Fig. 2. Simulink Model of the System

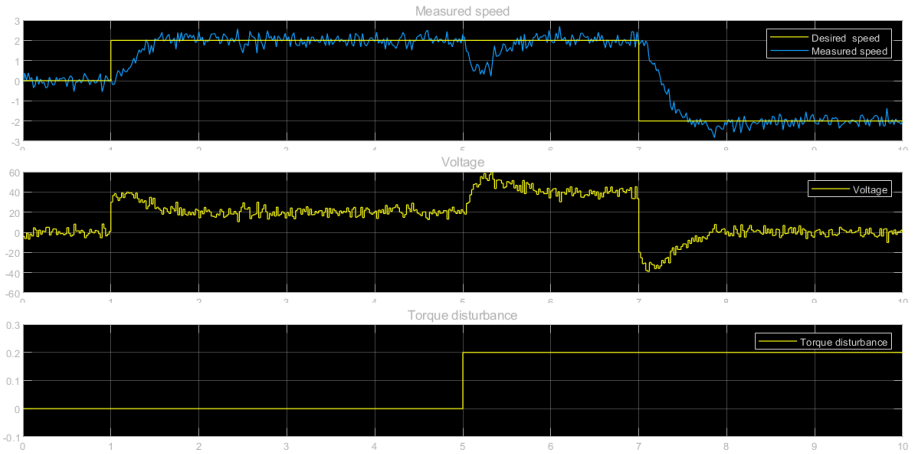

Fig. 3. Simulation Results

2 Tuning

Tuning linearizes the model at the default operating point and automatically determines PID controller gains to achieve reasonable performance and robustness based on linearized plant model. Results demonstrate good tracking with zero steady-state error, fast rise time, low overshoot, and good torque disturbance rejection (torque disturbance is modeled as a step change from 0 to 0.2 Newton-meters at 5 seconds)

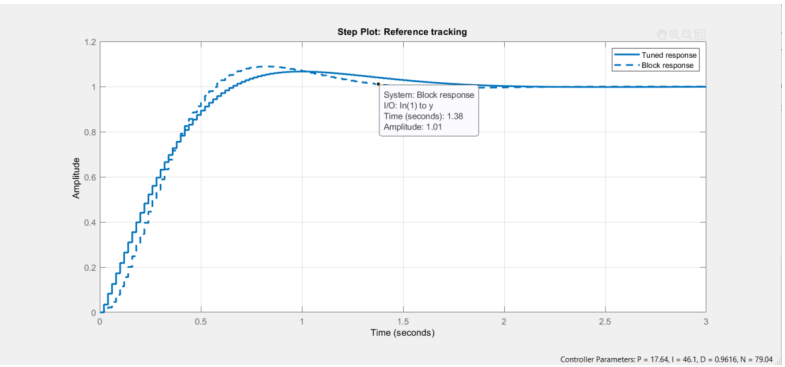

Fig. 4. Tuning the PID

Test Results

The figure fig.5 shows the output from the LCD module.

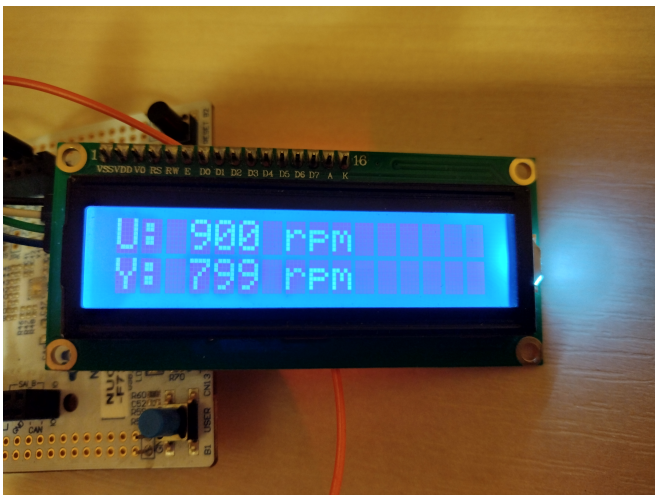

Fig. 5. LCD Module

The figure fig.6 shows the Serial Port Terminal.

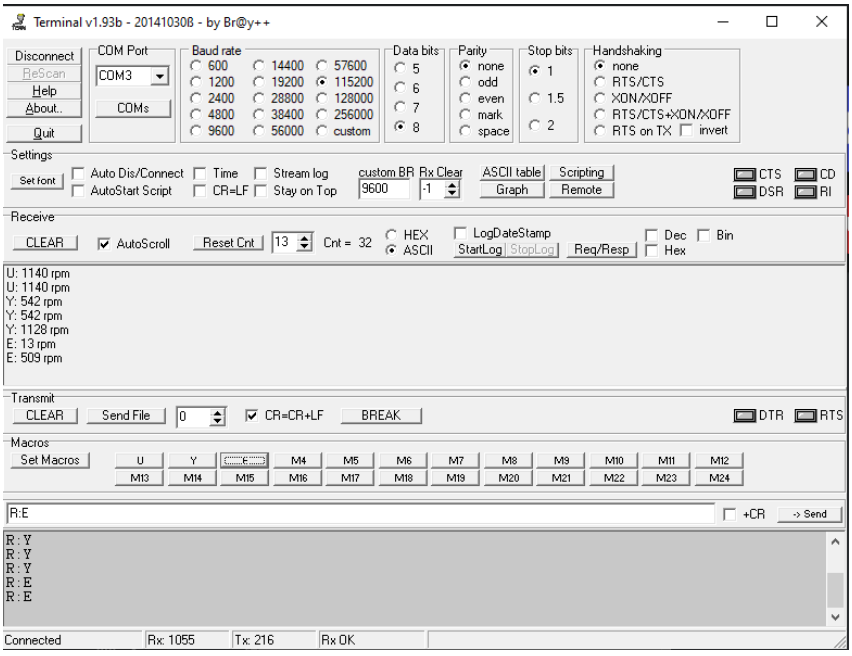

Fig. 6. Serial Port Terminal

The figures fig.7 shows the values of PID using Serial Wire Viewer.

Fig. 7. PID values

The figures fig.8 shows the values of Measurement, Reference and Error using Serial Wire Viewer.

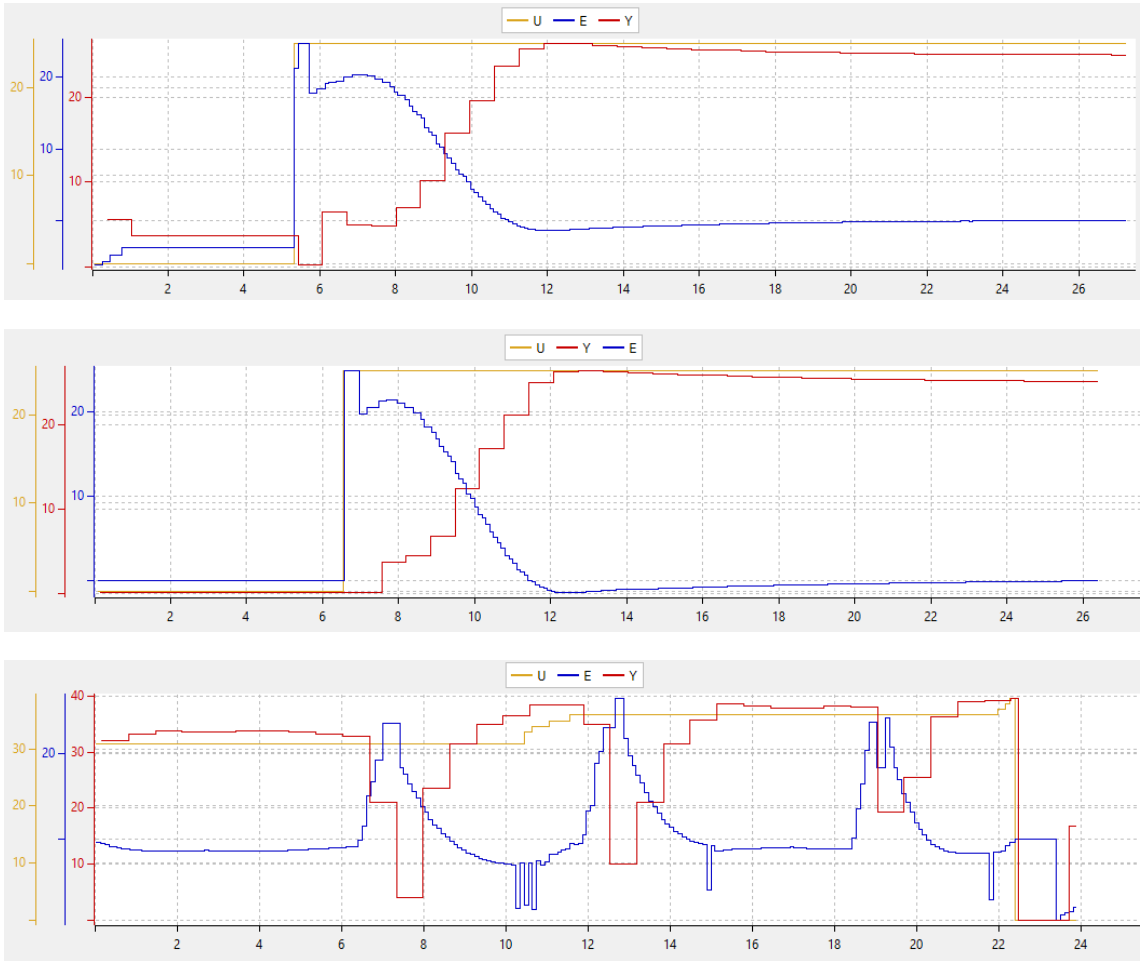

Fig. 8. Measurement, Reference and Error Values

Summary

We were able to perform all the descriptive functionality of a PWM-controlled DC motor with an IR sensor to measure the motor's speed. The system's control range from 10Hz to 29Hz was achieved. The serial port interface was used to create communication between the PC and the MCU, as well as to deploy the user interface to read and write the measurement and reference values. To read the measurement and reference data, the user interface was set up using an LCD. All the files are availble in Github[2].

Appendix A: Hardware Circuit

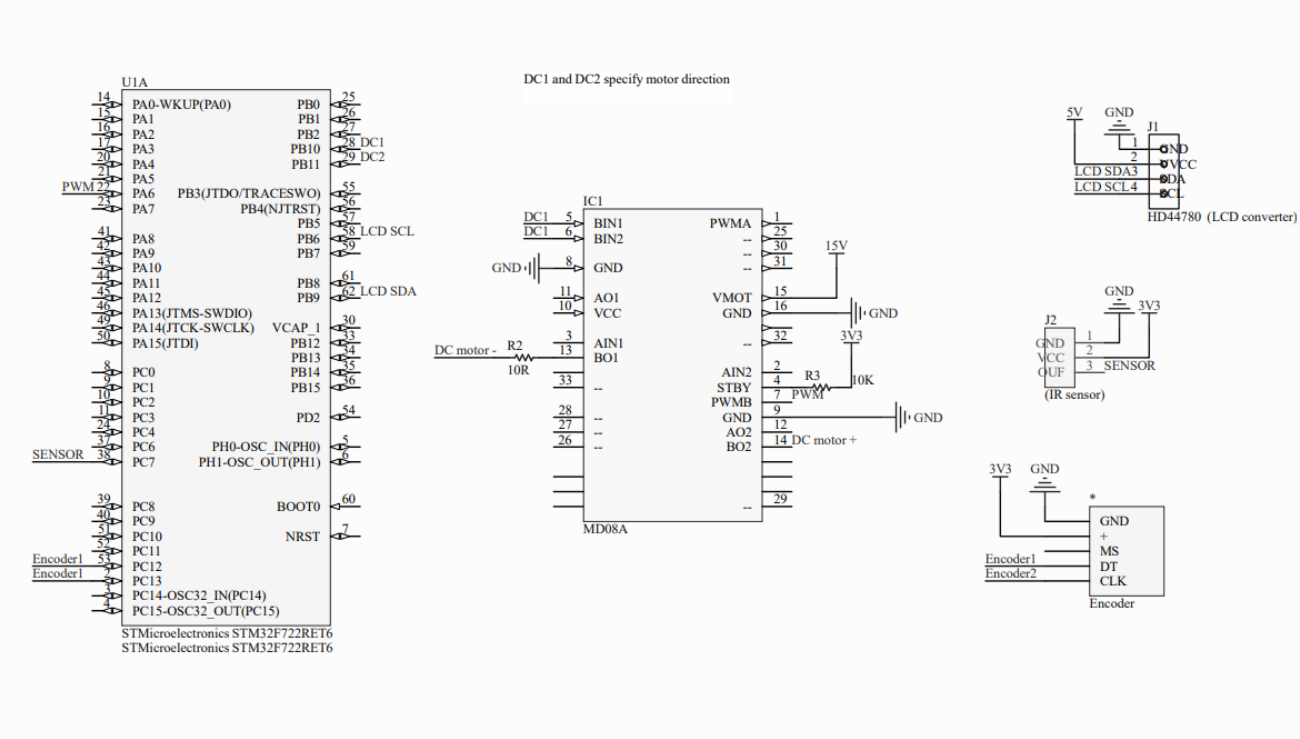

Fig. 9. Hardware Circuit

Appendix B: Source Codes

Listing 1. Timer callback function for PID implementaion

```
/*
    ********************************************************************************

 @brief  Time Callback (implements PID controll)
 @author  Wojciech Piersiala
 @param[in] htim tim handler
 @return None
 @version V1.0
 @date    18-Feb-2022

/*********************************************************************************
    */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
```

```
11.         if(htim->Instance==TIM5){
12.                 rot_freq=out_frequency;
13.                 total_time+=1/((float)sampling_freq);
14.                 Y=rot_freq;
15.                 U=U_tmp+count;
16.                 if(U<0){
17.                         U=0;
18.                         count=0;
19.                 }
20.                 E=U-Y;
21.
22.                 //I
23.                 I = prev_I+ E+prev_E;
24.                 Duty_test_I=I*k_I;
25.                 if(Duty_test_I<1){
26.                         Duty_test_I=1;
27.                 }
28.                 else
29.                         if(Duty_test_I>MAX_DUTY){
30.                                 Duty_test_I=MAX_DUTY;
31.                                 I=MAX_DUTY/k_I;
32.                         }
33.
34.                 //P
35.                 Duty_test_P=E*k_E;
36.                 if(Duty_test_P<1){
37.                         Duty_test_P=1;
38.                 }
39.                 else
40.                         if(Duty_test_P>MAX_DUTY){
41.                                 Duty_test_P=MAX_DUTY;
42.                         }
43.
44.                 //D
45.                 D = (E-prev_E);
46.                 Duty_test_D=k_D*D;
47.                 if(Duty_test_D<-900){
48.                         Duty_test_D=-900;
49.                 }
50.                 else
51.                         if(Duty_test_D>MAX_DUTY){
52.                                 Duty_test_D=MAX_DUTY;
53.                         }
54.
55.                 Duty_test=(int)(Duty_test_P+Duty_test_I+Duty_test_D);
56.
57.                 prev_E=E;
58.                 prev_I=I;
59.
60.                 if(Duty_test<1){
61.                         Duty_test=1;
62.                 }
63.                 else
64.                         if(Duty_test>MAX_DUTY){
65.                                 Duty_test=MAX_DUTY;
66.                         }
67.
68.                 __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,Duty_test);
69.         }
70.
71. }
```

Listing 2. H-bridge initialization

```
// H bridge
        HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,50);
        HAL_GPIO_WritePin(DC1_GPIO_Port, DC1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(DC2_GPIO_Port, DC2_Pin, GPIO_PIN_SET);
```

Listing 3. GPIO callback for frequency calculation and direction control of DC motor

```
/*
    ********************************************************************************

/* @brief  GPIO external interupr callback (Implements interaface with IR sensor
    and polarity change using USER_Btn)
/* @author  Wojciech Piersiala
/* @param[in] GPIO_Pin GPIO Pin handler
/* @return None
/* @version V1.0
/* @date    17-Feb-2022

/*******************************************************************************
    */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
        if(GPIO_Pin == USER_Btn_Pin){
                HAL_GPIO_TogglePin(DC1_GPIO_Port, DC1_Pin);
                HAL_GPIO_TogglePin(DC2_GPIO_Port, DC2_Pin);

        }
        if(GPIO_Pin == Sensor_Pin){
                timer_val=__HAL_TIM_GET_COUNTER(&htim2);

                if(timer_val>15000){
                        time_elapsed=timer_val;
                }

                frequency = 1/(float)time_elapsed*1000000;
                __HAL_TIM_SET_COUNTER(&htim2,0);

                if(frequency>2){
                        ratio_frequency=(frequency/old_frequency);
                        old_frequency = frequency;
                }

                if(ratio_frequency<1.3){
                        out_frequency = frequency;
                }
        }
}
```

Listing 4. UART callback for serial interface

```
/*
    ********************************************************************************

 @brief   UART Callback(Serail communication interface to read and write the
     values)
 @author  Bhargav Malasani,Wojciech Piersiala
 @param[in] huart huart handler
 @return None
 @version V2.0
 @date    20-Feb-2022

/*******************************************************************************
    */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
```

```
        if(huart->Instance == USART3){
                HAL_UART_Receive_IT(&huart3, buff, 3);
                char inx0 =buff[0];

                switch(inx0){
                case ('S'):{
                        int val0, val1, new_duty;
                        val0 = (int)(buff[1]-'0');
                        val1 = (int)(buff[2]-'0');
                        new_duty=(val0*10+val1)*100;
                        __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1,new_duty);

                        break;
                }
                case ('U'):{
                        int val0, val1, new_duty;
                        val0 = (int)(buff[1]-'0');
                        val1 = (int)(buff[2]-'0');
                        U_tmp=val0*10+val1;
                        count=0;   // count test
                        break;
                }
                case ('R'):{
                        char USARTdisp3[17];
                        int len2;
                        if(buff[2]=='Y'){

                        len2=sprintf(USARTdisp3, "Y:␣%d␣rpm␣\n\r",(int)(Y*60));}

                        if(buff[2]=='U'){

                                                len2=sprintf(USARTdisp3, "U:␣%d␣
                                                    rpm␣\n\r",(int)(U*60));}

                        if(buff[2]=='E'){

                                                len2=sprintf(USARTdisp3, "E:␣%d␣
                                                    rpm␣\n\r",(int)((E)*60));}

                        HAL_UART_Transmit(&huart3, &USARTdisp3, len2, 100);
                        break;
                }
        }
}
}
```

Listing 5. Timer IC callback for Rotary encoder

```
/*
    ********************************************************************************

 @brief  Timer IC callback(implements interface with rotary encoder)
 @author  Wojciech Piersiala
 @param[in] htim TIM handler
 @return None/* @version V1.0
 @date    17-Feb-2022

 ********************************************************************************
    */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim){
        encoder_counter=__HAL_TIM_GET_COUNTER(htim);
        count=((int16_t)encoder_counter)/4;


```

```
14.         if (count>30){
15.                 count=30;
16.                 encoder_counter=(uint32_t)(4*count);
17.                 __HAL_TIM_SET_COUNTER(htim,encoder_counter);
18.         }else if(count<-30){
19.                 count=-30;
20.                 encoder_counter=(0xFFFFFFFF+4*count);
21.                 __HAL_TIM_SET_COUNTER(htim,encoder_counter);
22.
23.         }
24.
25.
26. }
```

Listing 6. LCD implementation

```
01. //LCD initialization
02.
03.         disp.addr = (0x3F << 1);
04.         disp.bl = true;
05.         lcd_init(&disp);
06.
07. // Display using LCD
08.         while (1)
09.         {
10.
11.                 sprintf(LCDdisplay1, "U:␣%d␣rpm",(int)(U*60));
12.                 sprintf(LCDdisplay2, "Y:␣%d␣rpm",(int)(Y*60));
13.
14.                 sprintf((char *)disp.f_line, LCDdisplay1);
15.                 sprintf((char *)disp.s_line, LCDdisplay2);
16.
17.                 lcd_display(&disp);
18.
19.
20.
21.                 HAL_Delay(500);
22.                 /* USER CODE END WHILE */
23.
24.                 /* USER CODE BEGIN 3 */
25.         }
```

Appendix C: Drivers

Listing 7. $lcd_i2c.h$

```
01. /* The MIT License
02.  *
03.  * Copyright (c) 2020 Piotr Duba
04.  *
05.  * Permission is hereby granted, free of charge, to any person obtaining a copy
06.  * of this software and associated documentation files (the "Software"), to deal
07.  * in the Software without restriction, including without limitation the rights
08.  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
09.  * copies of the Software, and to permit persons to whom the Software is
10.  * furnished to do so, subject to the following conditions:
11.  *
12.  * The above copyright notice and this permission notice shall be included in all
13.  * copies or substantial portions of the Software.
14.  *
15.  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16.  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
```

```
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */
#ifndef INC_LCD_I2C_H_
#define INC_LCD_I2C_H_

#include <stdbool.h>
#include <stdint.h>

/*
 *          PCF8574 <-> HD44780
 *
 * I2C I/O    P7 P6 P5 P4 P3 P2 P1 P0
 * LCD        D7 D6 D5 D4 BL EN RW RS
 *
 * */

#define HI2C_DEF hi2c1

#define RS_PIN 0x01
#define RW_PIN 0x02
#define EN_PIN 0x04
#define BL_PIN 0x08

#define INIT_8_BIT_MODE 0x30
#define INIT_4_BIT_MODE 0x02

#define CLEAR_LCD       0x01

#define UNDERLINE_OFF_BLINK_OFF         0x0C
#define UNDERLINE_OFF_BLINK_ON          0x0D
#define UNDERLINE_ON_BLINK_OFF          0x0E
#define UNDERLINE_ON_BLINK_ON           0x0F

#define FIRST_CHAR_LINE_1       0x80
#define FIRST_CHAR_LINE_2       0xC0

struct lcd_disp {
        uint8_t addr;
        char f_line[17];
        char s_line[17];
        bool bl;
};

void lcd_init(struct lcd_disp * lcd);
void lcd_write(uint8_t addr, uint8_t data, uint8_t xpin);
void lcd_display(struct lcd_disp * lcd);
void lcd_clear(struct lcd_disp * lcd);

#endif /* INC_LCD_I2C_H_ */
```

Listing 8. lcd_i2c.c

```
/* The MIT License
 *
 * Copyright (c) 2020 Piotr Duba
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 */

#include "lcd_i2c.h"
//#include "stm32f3xx_hal.h"
#include "i2c.h"



void lcd_init(struct lcd_disp * lcd)
{
        uint8_t xpin = 0;
        /* set backlight */
        if(lcd->bl)
        {
                xpin = BL_PIN;
        }

        /* init sequence */
        HAL_Delay(40);
        lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);
        HAL_Delay(5);
        lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);
        HAL_Delay(1);
        lcd_write(lcd->addr, INIT_8_BIT_MODE, xpin);

        /* set 4-bit mode */
        lcd_write(lcd->addr, INIT_4_BIT_MODE, xpin);

        /* set cursor mode */
        lcd_write(lcd->addr, UNDERLINE_OFF_BLINK_OFF, xpin);

        /* clear */
        lcd_clear(lcd);

}

void lcd_write(uint8_t addr, uint8_t data, uint8_t xpin)
{
        uint8_t tx_data[4];

        /* split data */
        tx_data[0] = (data & 0xF0) | EN_PIN | xpin;
        tx_data[1] = (data & 0xF0) | xpin;
        tx_data[2] = (data << 4) | EN_PIN | xpin;
        tx_data[3] = (data << 4) | xpin;

        /* send data via i2c */
        HAL_I2C_Master_Transmit(&HI2C_DEF, addr, tx_data, 4, 100);

        HAL_Delay(5);
```

```
72. }
73.
74. void lcd_display(struct lcd_disp * lcd)
75. {
76.         uint8_t xpin = 0, i = 0;
77.
78.         /* set backlight */
79.         if(lcd->bl)
80.         {
81.                 xpin = BL_PIN;
82.         }
83.
84.         lcd_clear(lcd);
85.
86.         /* send first line data */
87.         lcd_write(lcd->addr, FIRST_CHAR_LINE_1, xpin);
88.         while(lcd->f_line[i])
89.         {
90.                 lcd_write(lcd->addr, lcd->f_line[i], (xpin | RS_PIN));
91.                 i++;
92.         }
93.
94.         /* send second line data */
95.         i = 0;
96.         lcd_write(lcd->addr, FIRST_CHAR_LINE_2, xpin);
97.         while(lcd->s_line[i])
98.         {
99.                 lcd_write(lcd->addr, lcd->s_line[i], (xpin | RS_PIN));
100.                i++;
101.        }
102. }
103.
104. void lcd_clear(struct lcd_disp * lcd)
105. {
106.        uint8_t xpin = 0;
107.
108.        /* set backlight */
109.        if(lcd->bl)
110.        {
111.                xpin = BL_PIN;
112.        }
113.
114.        /* clear display */
115.        lcd_write(lcd->addr, CLEAR_LCD, xpin);
116. }
```

References

1. *I2C driver*. Available also from: https://github.com/ptrre/kurs-stm32-e16-lcd.
2. *GitHub Repository*. Available also from: https://github.com/BhargavMN/Control_system_722_1.git.