RC servos tester project – description

Control signal:

Control RC micro servo using PWM signal

Period: 20ms
Frequenzy: 50Hz

Maximum high state time: 2.5ms
Minimum high state time: 0.5ms

Equipment:

STM32F072 discovery board

7 segments 4 digits display module

RC plane micro servo

Resistors 470R, potentiometer 10k,

Breadboard, 2pcs

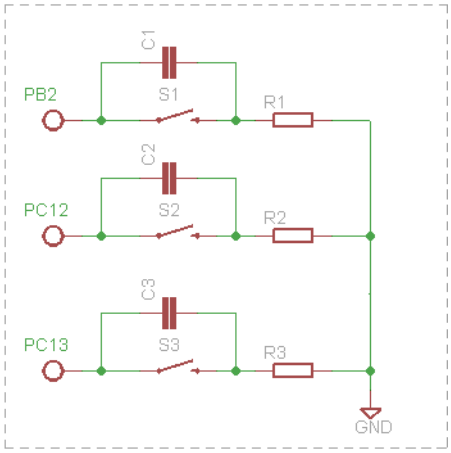
Jumper wires

Capacitors, 100nF

Connections:

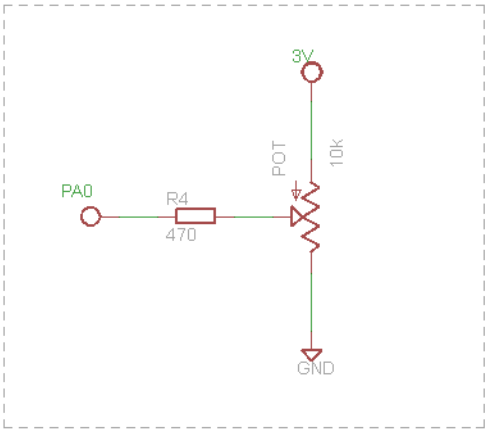
Buttons:

Microcontroller pin – resistor(470)-button-GND
PB2 – change mode
PC12 - subtraction 50us from high state time
PC13 – addition 50us to high state time



<u>Potentiometer 10k:</u>

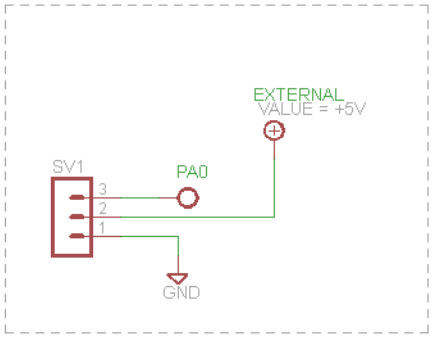
VCC 3V (from board) – PC3 – GND



<u>Servo:</u>

PA0 – signal

external voltage 5V



Display module:

VCC – external voltage 5V

GND – GND supply voltage

DA – VCC

DB – PC10

CP – PC11

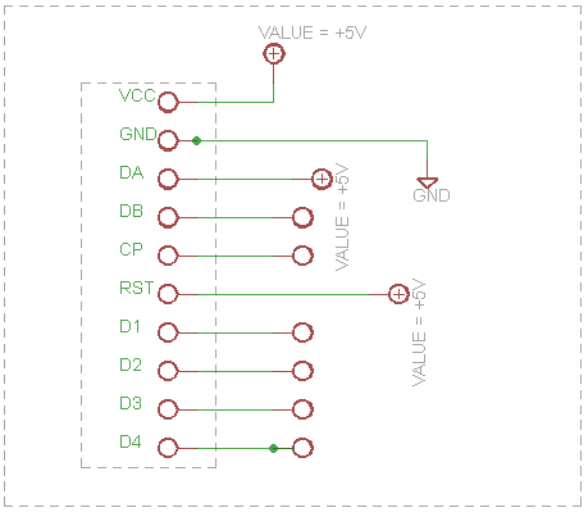
RST – VCC

D1 – PB3

D2 – PB4

D3 – PB5

D4 – PB6


VALUE = +5V
VCC
GND
DA
DB
CP
RST
D1
D2
D3
D4
GND
VALUE = +5V
VALUE = +5V

Modes (functionality):

Mode1: manual control potentiometer

Mode2: manual control joystick

Mode3: automatic control – maximum servo swing (left and right)

Mode4: set neutrum position (1500us high state time)

Mode5: manual control buttons

PWM high state displays on LED 7 segment, 4 digits on display module

Program description

Project has five modules. Program handles: servo signal PWM, display module, buttons, joystick and led lights. It uses led lights to show, which mode is active.

main.c

- main file program
- initialization button, servo signal, led display signal and leds:

```
init_btn();
init_servo();
init_led7s(5);
init_led();
init_joy();
```

- analyzing mode device by switch statement
- *wait_us(int time)* – waiting time in program, using timer TIM2. Time in microseconds

```
while(TIM6->CNT != czas){};
```

- *wait_ms(int time)* – using wait_us() to achieve wait in miliseconds

```
for( ; t < time; t++){
        wait_us(1000);
}
```

- variables:
    - *number* – number value to display, indicates high state time in microseconds
    - *TIM2->CCR1* register controls PWM signal servo
    - *pwm_btn* – controls signal setting by buttons
    - *mode* – indicates active mode

- potentiometer control, high state level calculation, 8bits resolution:

```
pwm1 = 8 * ADC1->DR + 500;
TIM2->CCR1 = T_PWM - pwm1;
```

- joystick control, 8 bits resolution

```
pwm1 = 8 * ADC1->DR + 500; //dokladnosc 8 bitowa
TIM2->CCR1 = T_PWM – pwm1;
```

- *automatically control calculation:*

```
TIM2->CCR1   = T_PWM - pwm2;
dir – direction servo moving
```

in each loop run, increase or decrease pwm2 value – dependent on direction movement

```
if(dir == 0){
        pwm2 += 25;
        if(pwm2 >= MAX_PWM)
                dir = 1;
        }else{
                pwm2 -= 25;
                 if( pwm2 <= MIN_PWM )
                        dir = 0;
        }
```

- neutrum – setting 1.5ms high state time

```
number = NEUTRUM;
TIM2->CCR1 = T_PWM - NEUTRUM;
```

- buttons control – setting value by pwm_btn (change in buttons interrupts)

led_7s_4d.c led_7s_4d.h

- multiplexing digits in display: 7 segments, 4 digits (FYQ-3641B common anode)
- setting core frequency: 48Mhz, multiplexing frequency: 200Hz
  timer settings:

  TIM_PSC *48*

  TIM_ARR *5000*

- using shift register 74hc164n to send data 7 segments display
- number – extern global variable, indicates number to display

- init_7s(int ar) – initialization display module.
  ar indicates switch time between display digits

- SPI communication protocol: microcontroller – shift register connection

SPI control – writes value to shift register in serial way:
value represents specific digit in display – switch on or switch off each segments

```
const uint8_t digits[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};
```

if data is 1 – set control value, 0 – reset
set seven bits in shift register by DATA (PC10) pin
change state on CLK (PC11) pin, rising edge, cause sending atomic value (one bit)

```
void spi_send(int cyfra) {
        int i = 0;
        int znak = cyfry[cyfra];
        for(i = 7; i > -1; i--)
        {
                clk_low;
                if((znak >> i) & 0x01) SPI_PORT->ODR |= DATA;
                else  { SPI_PORT -> ODR &= ~DATA; }
                clk_high;
        }

}
```

- After send full data it switches on particular digit

```
switch_off;
spi_send(tmp1);
GPIOB->BSRR |= GPIO_BSRR_BS_3;  //switch on digit number 3
```

- Pace multiplexing sets in interrupts – *TIM7_IRQHandler* on counter overflow: counting to ARR value

- Change active digit in multiplexing way in switch statement
  each digit is activates in interrupt – bad idea, too long operations, I don't have another idea

```
void TIM7_IRQHandler(void){
        ++d;
        if(d > 4) d = 1;
        num_disp4();
        TIM7->SR = 0;
```

```
}
```

- In num_disp4() function is switch statement – activates particular digit and sets value

```
switch(d){
        case 1:
                switch_off;
                spi_send(tmp1);
                GPIOB->BSRR |= GPIO_BSRR_BS_3;
        break;
        case 2:
          switch_off;
                spi_send(tmp2);
                GPIOB->BSRR |= GPIO_BSRR_BS_4;
        break;
        case 3:
          switch_off;
                spi_send(tmp3);
                GPIOB->BSRR |= GPIO_BSRR_BS_5;
        break;
        case 4:
          switch_off;
                spi_send(tmp4);
                GPIOB->BSRR |= GPIO_BSRR_BS_6;
        break;
        default:
                //d = 0;
        break;
}
```

- modulo operation calculates value on each digit

```
tmp1 = number/1000;
tmp2 = (number/100) % 10;
tmp3 = (number / 10) % 10;
tmp4 = number % 10;
```

led.c led.h

- Indicates mode working
- leds are build in stm32discovery board: PC6, PC7, PC8, PC9
- init_led() – initialize leds: output state pins
- switch on, switch off leds functions
- change in main.c switch statement

btn.c btn.h

- buttons tact switch handling

- extern int mode – in header file, indicates program mode
- extern int pwm_btn – indicates values control signal by buttons
- handling interrupt change state on pin – activate button, rising edge
  changing pwm_btn signal in interrupt

```
void EXTI4_15_IRQHandler(void){
case EXTI_PR_PR13:    //increments signal value PC13 button
...
break;
case EXTI_PR_PR12:    //decrements signal value PC12 button
...
break;
case EXTI_PR_PR9:     //latch joystick value by button joystick, PB9
...
break;

}
```

- choice mode button:

```
void EXTI2_3_IRQHandler(void){
        ++mode;
        EXTI->PR = EXTI_PR_PR2;
}
```

servo.c servo.h

- servo control signal
- using PWM mode 2 in timer TIM2: channel 1 is inactive as long as CNT < CCR1, else active. It counts maximum to ARR value (setting time time base – 20ms)
- ADC signal handling – control by potentiometer or joystick.

  ADC1->DR - value represents measure voltage
  data resolution – 8 bits

Developing plans

- Current consumption measure
- Using LCD display: hd44780 or touch screen TFT
- Controlling servo by mobile phone. Using Bluetooth modules
- Creating standalone device supporting by stm32 microcontrollers. Probably stm32f051
- Controlling by touch buttons

Bad solution in project:

Sending and setting number in interrupt function handling – too long
another display solves problem

Button bad quality – one press sometimes control multiple time

appendix A: full code

main.c

```
#include <stm32f0xx.h>

#include "led_7s_4d.h"

#include "servo.h"

#include "btn.h"

#include "led.h"



void wait_us(int czas);

void wait_ms(int time);

int pwm1;

int pwm2 = MIN_PWM;

uint8_t dir = 0;



int main(void){

        init_btn();

        init_servo();

        init_led7s(5);

        init_led();



while(1){

        leds_off;
```

```
switch(mode){
        case 0: //manual by potentiometer
                led1_on;
                stop_adc_joy();
                //conf_adc();
                conf_adc();

                while( !(ADC1->ISR & ADC_ISR_EOC) ){}
                pwm1 = 8 * ADC1->DR + 500;
                TIM2->CCR1 = T_PWM - pwm1;
                number = pwm1;
        break;


        case 1: //manual - joystick
                led1_on;
                stop_adc();
                conf_adc_joy();

                if( !latch )
                {
                        while( !(ADC1->ISR & ADC_ISR_EOC) ){}
                        pwm1 = 8 * ADC1->DR + 500;
                        TIM2->CCR1 = T_PWM - pwm1;
                        number = pwm1;
                }
        break;
```

```
                case 2: //neutrum
                        led3_on;
                        number = NEUTRUM;
                        TIM2->CCR1 = T_PWM - NEUTRUM;
                break;

                case 3: //buttons control
                        led4_on;
                        TIM2->CCR1 = T_PWM - pwm_btn;
                        number = pwm_btn;
                break;
                default:
                        mode = 0;
                break;

        }
}

}

void wait_us(int czas){
        RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
        TIM6->CR1 |= TIM_CR1_ARPE;
        TIM6->PSC |= 48;
        TIM6->ARR |= 1000;
        TIM6->CR1 |= TIM_CR1_CEN;
        while(TIM6->CNT != czas){};
```

```
        TIM6->CR1 &= ~TIM_CR1_CEN;
        TIM6->CNT  =0x00;
}

void wait_ms(int time){
    int t = 0;
    for( ; t < time; t++){
        wait_us(1000);
    }
}
```

led_7s_4d.h

```
#ifndef LED_7S_4D_H_
#define LED_7S_4D_H_

#include <stm32f0xx.h>
#define DSB ( 1 << 10 )
#define CP ( 1 << 11 )

//spi programowy
#define SPI_PORT GPIOC
#define DATA (1 << 10)
#define CLK (1 << 11)

//konfiguracja timera TIM7
#define TIM_PSC 48 //48MHz / 48
```

```
#define switch_off      GPIOB->BRR |= GPIO_BRR_BR_3 | GPIO_BRR_BR_4 | GPIO_BRR_BR_5 |
                        GPIO_BRR_BR_6

#define clk_low GPIOC->BRR |= CLK

#define clk_high GPIOC->BSRR |= CLK

extern volatile int number;

//funkcje


/*
        inicjalizacja wyswietlacza led


        param ar - szybkosc przelaczania do nastepnej cyfry w ms

        pod warunkiem pracy timera z czestotliwoscia 1MHz
*/
void init_led7s(int ar);

/*
void spi_send(int cyfra);

void clear_led(void);

void tim7_conf(void);

void num_disp4(void);

*/

#endif /* LED4_H_ */
```

---

led_7s_4d.c

```
 #include "led_7s_4d.h"

const uint8_t cyfry[10] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80, 0x90};


int j;

volatile int d = 0;
```

```
int bb = 0;

int abc = 0;

int arr;

volatile int number = 0;


uint8_t tmp1, tmp2, tmp3, tmp4;


void clear_led(void);

void tim7_conf(void);


void init_led7s(int ar){

	arr = 1000 * ar;

	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

	GPIOC->MODER |= GPIO_MODER_MODER10_0 | GPIO_MODER_MODER11_0;


	RCC->AHBENR |= RCC_AHBENR_GPIOBEN;

	GPIOB->MODER |= GPIO_MODER_MODER3_0 | GPIO_MODER_MODER4_0 |
GPIO_MODER_MODER5_0 | GPIO_MODER_MODER6_0;

	GPIOB->BRR |= GPIO_BRR_BR_3 | GPIO_BRR_BR_4 | GPIO_BRR_BR_5 | GPIO_BRR_BR_6;


	clear_led();

	tim7_conf();

}




void clear_led(void){

	uint8_t tmp = 0;
```

```
        GPIOC->BSRR |= DSB;
        for( tmp = 0; tmp < 9; ++tmp ){
                GPIOC->BRR |= CP;
                GPIOC->BSRR |= CP;
        }
}


/*tim7 - odpowiedzialny za multipleksowanie, przerwanie,
        zdarzeniem jest przepelnienie rejestru liczacego
        ( fcpu/PSC ) * ARR
*/

void tim7_conf(void){

        RCC->APB1ENR |= RCC_APB1ENR_TIM7EN;
        TIM7->CR1 &= ~TIM_CR1_ARPE;
        TIM7->PSC = TIM_PSC;
        TIM7->ARR = arr;
        TIM7->DIER |= TIM_DIER_UIE;
        TIM7->CR1 |= TIM_CR1_CEN;
        NVIC_EnableIRQ(TIM7_IRQn);

}

/*
        wyslanie cyfry na wyswietlacz
*/
```

```
	void spi_send(int cyfra) {
		int i = 0;
		int znak = cyfry[cyfra];
		for(i = 7; i > -1; i--)
		{ clk_low;
			if((znak >> i) & 0x01) SPI_PORT->ODR |= DATA;
			else  { SPI_PORT -> ODR &= ~DATA; }
			clk_high;
		}

	}

/*
	wyswietlenie liczby na wyswietlaczu czterocyfrowym LED
*/
void num_disp4(){
	if( number > 9999){

	}else{
	  tmp1 = number/1000;
	  tmp2 = (number/100) % 10;
	  tmp3 = (number / 10) % 10;
	  tmp4 = number % 10;

		switch(d){
			//pierwsza cyfra
			case 1:
				switch_off;
```

```
                spi_send(tmp1);
                GPIOB->BSRR |= GPIO_BSRR_BS_3;
            break;
            case 2:
             switch_off;
                spi_send(tmp2);
                GPIOB->BSRR |= GPIO_BSRR_BS_4;
            break;
            case 3:
             switch_off;
                spi_send(tmp3);
                GPIOB->BSRR |= GPIO_BSRR_BS_5;
            break;
            case 4:
             switch_off;
                spi_send(tmp4);
                GPIOB->BSRR |= GPIO_BSRR_BS_6;
            break;
            default:
                //d = 0;
            break;
        }
  }
}
```

```
//przerwanie odpowiedzialne za szybkosc przelaczania cyfr - multipleksowanie


void TIM7_IRQHandler(void){
        ++d;
        if(d > 4) d = 1;
        num_disp4();
        TIM7->SR = 0;
}
```

---

led.h

```
/*
        obsluga diod led na makiecie stm32discovery
        podlaczenie:
        LED1    PC6
        LED2    PC7
        LED3    PC8
        LED4    PC9
*/


#ifndef LED_H_
#define LED_H_


#include <stm32f0xx.h>


#define LED1 (1 << 6)
#define LED2 (1 << 8)
#define LED3 (1 << 7)
#define LED4 (1 << 9)
#define LEDS ( LED1 | LED2 | LED3 | LED4 )
```

```
#define leds_off GPIOC->BRR |= LEDS
#define led1_on GPIOC->BSRR |= LED1
#define led2_on GPIOC->BSRR |= LED2
#define led3_on GPIOC->BSRR |= LED3
#define led4_on GPIOC->BSRR |= LED4

#define led1_off GPIOC->BRR |= LED1
#define led2_off GPIOC->BRR |= LED2
#define led3_off GPIOC->BRR |= LED3
#define led4_off GPIOC->BRR |= LED4

void init_led(void);

#endif
```

---

led.c

```
#include "led.h"

void init_led(void){
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
	GPIOC->MODER |= GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
			GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0;
}
```

---

btn.h

```
	/*
		obsluga przyciskow
```

```
        mode - opcja wyboru trybu pracy serwa - zmienna definiowana i wykorzystywana w
main

        pwm_btn - zmiana reczna poprzez przycisk sterowania serwa


        pinout:

        zwieranie do GND

        PC12    zmiana pwm_btn

        PC13    zmiana pwm_btn


        PB2 zmiana mode

*/


#ifndef BTN_H_

#define BTN_H_


#include <stm32f0xx.h>


#define STEP_BTN 50


extern uint8_t latch;

extern int mode;

extern int pwm_btn;


void init_btn(void);

#endif
```

---

btn.c

```
#include "btn.h"
```

```
/*
	konfiguracja przyciskow do obslugi serwa
*/

int mode = 0;
int pwm_btn = 500;

void init_btn(void){
		//interrupt PA4 BTN
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
	GPIOA->PUPDR |= GPIO_PUPDR_PUPDR4_0;//pullup
	NVIC_EnableIRQ(EXTI4_15_IRQn);
	EXTI->IMR |= EXTI_IMR_MR4;
	EXTI->FTSR |= EXTI_FTSR_TR4;

	//BTN conf: PB2, input, pullup
	RCC->APB2ENR |=  RCC_APB2ENR_SYSCFGCOMPEN;
	RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPDR2_0;//pullup
	SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PB;
	NVIC_EnableIRQ(EXTI2_3_IRQn);
	EXTI->IMR |= EXTI_IMR_MR2;
	EXTI->FTSR |= EXTI_FTSR_TR2;
	//EXTI->RTSR = EXTI_RTSR_TR2;

		//button pc12 pc13 przerwanie
```

```
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGCOMPEN;
	GPIOC->PUPDR |= GPIO_PUPDR_PUPDR12_0 | GPIO_PUPDR_PUPDR13_0;//pullup
	NVIC_EnableIRQ(EXTI4_15_IRQn);
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI12_PC | SYSCFG_EXTICR4_EXTI13_PC;
	EXTI->IMR |= EXTI_IMR_MR12 | EXTI_IMR_MR13;
	EXTI->FTSR |= EXTI_FTSR_TR12 | EXTI_FTSR_TR13;
}




//-------------------------------------------------------------
//reakcja na zmiane opcji za pomoca przycisku
void EXTI2_3_IRQHandler(void){
	++mode;
	EXTI->PR = EXTI_PR_PR2;
}

void EXTI4_15_IRQHandler(void){
	switch(EXTI->PR){
		/*case EXTI_PR_PR4:
			EXTI->PR = EXTI_PR_PR4;
			print_signal();
		break;
		*/

		case EXTI_PR_PR13:
```

```
                EXTI->PR = EXTI_PR_PR13;
                pwm_btn += STEP_BTN;
                if( pwm_btn > 2500 )
                        pwm_btn = 2500;
        break;
        case EXTI_PR_PR12:
                EXTI->PR = EXTI_PR_PR12;
                pwm_btn -= STEP_BTN;
                if( pwm_btn < 500 )
                        pwm_btn = 500;
        break;

        case EXTI_PR_PR9:       //joystick button
                //latch ^= 0x01;
                while(++counter){}
                latch ^= 0x01;
                EXTI->PR = EXTI_PR_PR9;
        break;
    }
}
```

joystick.h

```
#ifndef JOYSTICK_H_
#define JOYSTICK_H_
#include <stm32f0xx.h>

//extern uint8_t latch;
void conf_adc_joy(void);
void stop_adc_joy(void);
void init_joy(void);
```

```
#endif
```

joystick.c

```
#include "joystick.h"


/*
	PB9 interrupt
	zatrzaskiwanie wartosci poprzez przycisk PB9
*/
void init_joy(void)
{

	RCC->APB2ENR |=  RCC_APB2ENR_SYSCFGCOMPEN;
	RCC->AHBENR |= RCC_AHBENR_GPIOBEN;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPDR9_0;//pullup
	SYSCFG->EXTICR[2] |= SYSCFG_EXTICR3_EXTI9_PB;
	NVIC_EnableIRQ(EXTI4_15_IRQn);
	EXTI->IMR |= EXTI_IMR_MR9;
	EXTI->FTSR |= EXTI_FTSR_TR9;
}


/* ADC conf: PC0 ADC_IN_10 additional function
	konfiguracja potencjometru - napiecie mierzone i obliczane pod adc
*/
 void conf_adc_joy(void){
	RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
	GPIOC->MODER |= GPIO_MODER_MODER0;
	ADC1->CHSELR |= ADC_CHSELR_CHSEL10;
	ADC1->CFGR1 |= ADC_CFGR1_CONT;
	ADC1->CFGR1 |= ADC_CFGR1_RES_1;    //res 8 BITOWA
```

```
	ADC1->CR |= ADC_CR_ADEN;
	while( !(ADC1->ISR & ADC_ISR_ADRDY));
	ADC1->CR |= ADC_CR_ADSTART;
}

void stop_adc_joy(void)
{
	ADC1->CR |= ADC_CR_ADSTP;

}
```

---

servo.h

```
	/*sterowanie serwem

		potencjometr PC3 ADC_IN_13

		sygnaB wyjsiowy PA0 TIM2_CH2

	*/

	#ifndef SERVO_H_
	#define SERVO_H_

	#include <stm32f0xx.h>
	//number = 34;

	#define TIM_PSC 48 // dla fcpu = 48MHz / 48
	#define MAX_PWM 2500
	#define MIN_PWM 500
	#define NEUTRUM 1500
```

```
#define T_PWM 20000


void init_servo(void);


#endif
```

---

servo.c

```
#include "servo.h"

void conf_adc(void);



/*
	PA0 - TIM2_CH1
*/
void init_servo(void){
	//pwm out: PA0
	RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
	GPIOA->MODER |= GPIO_MODER_MODER0_1; //AF
	GPIOA->AFR[0] |= (1 << 1); //AF2

	//TIM2 conf: pwm, arr 20ms, f 1Mhz
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
	TIM2->PSC  = TIM_PSC; //f tim2: 1Mhz
	TIM2->ARR = 20000; //20ms - okres sygnaBu
	TIM2->CR1 &= ~TIM_CR1_ARPE;
```

```
	TIM2->CCER |= TIM_CCER_CC1E;//?????? gdzie ze output, bit CC1P

	TIM2->CCMR1 |= TIM_CCMR1_OC1M_0 | TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; //111 stan L, stan H - PWM MODE 2

	TIM2->CCMR1 |= TIM_CCMR1_OC1PE; //?

	TIM2->CR1 |= TIM_CR1_CEN;


	conf_adc();

}


/* ADC conf: PC3 ADC_IN_13 additional function

	konfiguracja potencjometru - napiecie mierzone i obliczane pod adc

*/

void conf_adc(void){

	RCC->APB2ENR |= RCC_APB2ENR_ADCEN;

	RCC->AHBENR |= RCC_AHBENR_GPIOCEN;

	GPIOC->MODER |= GPIO_MODER_MODER3;

	ADC1->CHSELR |= ADC_CHSELR_CHSEL13;

	ADC1->CFGR1 |= ADC_CFGR1_CONT;

	ADC1->CR |= ADC_CR_ADEN;

	while( !(ADC1->ISR & ADC_ISR_ADRDY));

	ADC1->CR |= ADC_CR_ADSTART;

}
```

Appendix B: display module schemat