LAMPIRAN II

Listing Program

LISTING PROGRAM MIKROKONTROLER MASTER

```
#define waktu_sampling          10      //satuan ms
#define time_sampling  (65535-43200*waktu_sampling/1000)

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <compat/twi.h>

#include "SPI.h"
#include "lcd.h"

volatile float Kp=1, Ki=1, Kd=1, PID;
volatile float lpwm, rpwm, MAXPWM=100;
volatile float error, set_point=7, del_error, last_error, sigma_error;
volatile int sensor;
volatile int dirka,dirki,pwmka,pwmki,j=0,sampling_cnt=1;
volatile int puluhribu,ratusan,puluhan,satuan,temp;

void PID_Line_Following();
void InitUART(void);
void TransmitByte( unsigned char data );
void tampil(int dat);

int main (void)
{
        _delay_ms(2500);
        i2c_init();
        //SPI_MasterInit_int();
        DDRC = (1<<4)|(1<<5)|(1<<6)|(1<<7);
        DDRD = (1<<4)|(1<<5)|(1<<6)|(1<<7);
        lcd_init();
        lcd_clrscr();
        DDRA = 255;
        _delay_ms(7500);
        InitUART();
        TCCR0 = ( (1<<CS01) | (1<<CS00) );
        TIMSK = 0x04;
        TCCR1B = ( (1<<CS12) | (0<<CS10) );
        TCNT1 = time_sampling;
        sei();
        while (1)
        {
                lcd_gotoxy(0,1);
                tampil(sensor);
        }
        return 0;
}

ISR(TIMER1_OVF_vect){
        TCNT1 = time_sampling;
```

```
/*
   count=0;
   DDRB  |= _BV(1);//ARAH=OUT;
   PORTB |= _BV(1);//pulsa=1;
   _delay_us(5);
   PORTB &= ~_BV(1);//pulsa=0;
   // port as input
   DDRB  &= ~_BV(1);//ARAH=INP;
   // with pull-up
   PORTB |= _BV(1);//pulsa=1;
   while (bit_is_clear(PINB,1)) {};
   while (bit_is_set(PINB,1))
   {
   count++;
   }
   waktu=(count*0.09012906482); // KRISTAL = 11,059200Mhz
   jarak=((waktu*0.3495)/2);
   jarak=jarak*1.3;      //Kalibrasi
         if(jarak<=150)
               PORTB |= _BV(0);//bel=1;
         else
               PORTB &= ~_BV(0);//bel=0;
         if(jarak<=100)
               MAXPWM = 0;
         else if(jarak<=200){
               MAXPWM = (jarak-100)*10;
               if(MAXPWM>200)
                     MAXPWM=200;
         }
         else
               MAXPWM = 200;
*/
/*
         PV = jarak;
         error = (float)(PV-100);
   P = Kp * error;

   I = (Ki * (error + last_error)) * 0.2;

   D = (Kd * (error - last_error)) / 0.2;

   last_error = error;

   MV = P + I + D;

   speed_temp = MV;

   if(speed_temp<-255)  //speed_temp=speed sementara
         speed=-255;
         else if(speed_temp>255)
               speed=255;
   else
```

```
        speed=speed_temp;

        if(speed>=0){
                dirka=1;
                dirki=1;
        }
        else{
                dirka=0;
                dirki=0;
                speed=-speed;
        }
        pwmka=6*speed;
        pwmki=pwmka;
*/
/*
        if(jarak>=100)
                scanBlackLine2();
        else{

                dirka=0;
                dirki=0;
                speed=150-jarak;

        pwmka=6*speed;
        pwmki=pwmka;

        }
*/
        //SENSOR
        //**********************I2C**********
        i2c_start();
        check_status(TW_START);
        i2c_write(0xD0+TW_READ);
        check_status(TW_MR_SLA_ACK);
        sensor = i2c_readNak();
        check_status(TW_MR_DATA_NACK);
        i2c_stop();
        //**********************SPI*************
/*
        PORTA=0b10100000;
        SPI_MasterTransmit_int(0xAA);
        SPI_MasterTransmit_int(0xff);
        sensor   = SPDR;
*/
        j++;
        if(j>=sampling_cnt){
                PID_Line_Following();
                j=0;
        }

        //MOTOR KANAN
        //**********************I2C**********
```

```
        i2c_start();
        check_status(TW_START);
        i2c_write(0xD2+TW_WRITE);
        check_status(TW_MT_SLA_ACK);
        if(dirka)
                i2c_write(0x0f);//maju
        else
                i2c_write(0xf0);//mundur
        check_status(TW_MT_DATA_ACK);
        i2c_write(pwmka%255);//lsb
        check_status(TW_MT_DATA_ACK);
        i2c_write(pwmka/255);//msb
        check_status(TW_MT_DATA_ACK);
        i2c_stop();
        //*********************SPI*************
/*
        PORTA=0b01100000;
        SPI_MasterTransmit_int(255);
        SPI_MasterTransmit_int(255);
        if(dirka)
                SPI_MasterTransmit_int(0x0f);//maju
        else
                SPI_MasterTransmit_int(0xf0);//mundur
        SPI_MasterTransmit_int(pwmka%255);//lsb
        SPI_MasterTransmit_int(pwmka/255);//msb
*/
        //MOTOR KIRI
        //************************I2C***********
        i2c_start();
        check_status(TW_START);
        i2c_write(0xD4+TW_WRITE);
        check_status(TW_MT_SLA_ACK);
        if(dirki)
                i2c_write(0x0f);//maju
        else
                i2c_write(0xf0);//mundur
        check_status(TW_MT_DATA_ACK);
        i2c_write(pwmki%255);//lsb
        check_status(TW_MT_DATA_ACK);
        i2c_write(pwmki/255);//msb
        check_status(TW_MT_DATA_ACK);
        i2c_stop();
        //*********************SPI*************
/*
        PORTA=0b11000000;
        SPI_MasterTransmit_int(255);
        SPI_MasterTransmit_int(255);
        if(dirki)
                SPI_MasterTransmit_int(0x0f);
        else
                SPI_MasterTransmit_int(0xf0);
        SPI_MasterTransmit_int(pwmki%255);//lsb
```

```
        SPI_MasterTransmit_int(pwmki/255);//msb
*/
        temp = sensor;
        puluhribu = temp/1000 +48;
        ratusan   = (temp%1000)/100 +48;
        puluhan   = (temp%100)/10 +48;
        satuan    = (temp%10) +48;
        TransmitByte(puluhribu);
        TransmitByte(ratusan);
        TransmitByte(puluhan);
        TransmitByte(satuan);
        TransmitByte(0x0D);
}

void PID_Line_Following(){
        error = set_point - sensor;
        del_error = error - last_error;
        sigma_error += error;

        PID = error*Kp + del_error*Ki + sigma_error*Kd;

        rpwm = MAXPWM + PID;
        lpwm = MAXPWM - PID;

        if (lpwm < 0) lpwm = 0;
        if (lpwm > 255) lpwm = 255;
        if (rpwm < 0) rpwm = 0;
        if (rpwm > 255) rpwm = 255;

        pwmki = 6*lpwm;
        pwmka = 6*rpwm;
        dirki=1;
        dirka=1;

        last_error = error;
}

void InitUART(void){
        UCSRA=0x00;
        UCSRB=0x98;
        UCSRC=0x86;
        UBRRH=0x00;
        UBRRL=11;
}

void TransmitByte( volatile unsigned char data ){
        while ( !(UCSRA & (1<<UDRE)) );       // Wait for empty transmit buffer
        UDR = data;// Start transmittion
}

void tampil(int dat){
   int data;
```

```
    data = dat / 10000;
    data+=0x30;
    lcd_putch(data);

        dat%=10000;
    data = dat / 1000;
    data+=0x30;
    lcd_putch(data);

    dat%=1000;
    data = dat / 100;
    data+=0x30;
    lcd_putch(data);

    dat%=100;
    data = dat / 10;
    data+=0x30;
    lcd_putch(data);

    dat%=10;
    data = dat + 0x30;
    lcd_putch(data);
}
```

LISTING PROGRAM MIKROKONTROLER SLAVE MOTOR

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "SPI.h"
#include "twi_slave.h"

volatile int data_spi_rx;
volatile int data_spi_tx;
volatile uint8_t i, j=0,direction,k=0,terima=0,first=0;
volatile int rpm_actual[10],rpm_set_point[10],header[2];
volatile int PWM;

volatile unsigned char TWI_buf[4];     // Transceiver buffer. Set the size in the header file
volatile unsigned char TWI_msgSize  = 0;            // Number of bytes to be transmitted.
volatile unsigned char TWI_state    = 0xF8;  // State byte. Default set to
TWI_NO_STATE.

void setMotorSpeed(int speed);

int main (void)
{
        _delay_ms(5000);
        //SPI_SlaveInit();
```

```
	TWAR = 0xD3;// D3=kanan, D5=kiri
	TWCR = 0x45;
	sei();
	TCCR1A=0xA1;
	TCCR1B=0x0C;
	DDRB |= _BV(0);
	DDRB |= _BV(1);
	while (1)
	{
		PWM = (255*rpm_set_point[2]+rpm_set_point[1])/6;
		if(rpm_set_point[0]==0x0F)
			setMotorSpeed(PWM );
		else if(rpm_set_point[0]==0xF0)
			setMotorSpeed(-PWM);
	}
}
/*
ISR(SPI_STC_vect){
	data_spi_rx=SPDR;
	if(data_spi_rx==255)
		k++;
	else
		k=0;

	if(k>=2){
		j=0;
		terima=1;
	}
	else if(terima){
		rpm_set_point[j] = data_spi_rx;
		j++;
		if(j>=3)
			terima=0;
	}

	SPDR=rpm_actual[j];//data_sensor[j];
}
*/
ISR(TWI_vect){
static unsigned char TWI_bufPtr;

switch (TWSR){
case TWI_STX_ADR_ACK:// Own SLA+R has been received; ACK has been returned
TWI_bufPtr   = 0;// Set buffer pointer to first data location
case TWI_STX_DATA_ACK:// Data byte in TWDR has been transmitted; ACK has
been received
TWDR = rpm_actual[TWI_bufPtr++];
TWCR = (1<<TWEN)|  // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|  // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|
(0<<TWWC);
break;
```

```
case TWI_STX_DATA_NACK:  // Data byte in TWDR has been transmitted; NACK has
been received.
// I.e. this could be the end of the transmission.
if (TWI_bufPtr == TWI_msgSize) // Have we transceived all expected data?
{
//TWI_statusReg.lastTransOK = TRUE;// Set status bits to completed successfully.
}
else // Master has sent a NACK before all data where sent.
{
TWI_state = TWSR; // Store TWI State as errormessage.
}

TWCR = (1<<TWEN)|  // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)| // Keep interrupt enabled and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Answer on next address match
(0<<TWWC);
break;
case TWI_SRX_ADR_ACK: // Own SLA+W has been received ACK has been returned
TWI_bufPtr   = 0;  // Set buffer pointer to first data location

// Reset the TWI Interupt to wait for a new event.
TWCR = (1<<TWEN)|   // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|   // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Expect ACK on this transmission
(0<<TWWC);


break;
case TWI_SRX_ADR_DATA_ACK: // Previously addressed with own SLA+W;
case TWI_SRX_GEN_DATA_ACK: // Previously addressed with general call;
rpm_set_point[TWI_bufPtr++]    = TWDR;
//      TWI_statusReg.lastTransOK = TRUE;     // Set flag transmission successfull.
// Reset the TWI Interupt to wait for a new event.
TWCR = (1<<TWEN)|   // TWI Interface enabled
(1<<TWIE)|(1<<TWINT)|    // Enable TWI Interupt and clear the flag to send byte
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Send ACK after next reception
(0<<TWWC);  //
break;
case TWI_SRX_STOP_RESTART:       // A STOP condition
// Enter not addressed mode and listen to address match
TWCR = (1<<TWEN)|          // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)|      // Enable interrupt and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Wait for new address match
(0<<TWWC);

break;
case TWI_SRX_ADR_DATA_NACK: // Previously addressed with own SLA+W;
case TWI_SRX_GEN_DATA_NACK:  // Previously addressed with general call;
case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been
transmitted
//    case TWI_NO_STATE   // No relevant state information available;
case TWI_BUS_ERROR:   // Bus error due to an illegal START or STOP condition
```

```
TWI_state = TWSR;   //Store TWI State as errormessage,
TWCR =   (1<<TWSTO)|(1<<TWINT);  //Recover from TWI_BUS_ERROR,
break;
default:
TWI_state = TWSR;    // Store TWI State as errormessage
TWCR = (1<<TWEN)|   // Enable TWI-interface and release TWI pins
(1<<TWIE)|(1<<TWINT)|   // Keep interrupt enabled and clear the flag
(1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|// Acknowledge on any new requests.
(0<<TWWC);
}
}

void setMotorSpeed(int speed){

  unsigned char reverse = 0;

  if (speed < 0){
    speed = -speed;    // make speed a positive quantity
    reverse = 1;    // preserve the direction
  }
  if(speed>10)
  speed-=10;
  else
  speed=0;

  if (speed > 0xFF)    // 0xFF = 255
    speed = 0xFF;

  if (reverse){
    PORTB &= ~_BV(0);
    OCR1A = speed;
  }
  else{    // forward
    PORTB |= _BV(0);
    OCR1A = 255 - speed;
  }

}
```