Исходные коды для программирования робота «*RoboChuck*»

Приложение 1. Исходный код прошивки модуля управления

Инициализация параметров процессора и компилятора.

```
#include <mega128.h>
#include <stdio.h>
#include <delay.h>
#include "led_states.h"


#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define boolean char
#define true 1
#define false 0
#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
```

Работа с *USART1*

```
#define RX_BUFFER_SIZE1 8
char rx_buffer1[RX_BUFFER_SIZE1];

#if RX_BUFFER_SIZE1<256
unsigned char rx_wr_index1,rx_rd_index1,rx_counter1;
#else
unsigned int rx_wr_index1,rx_rd_index1,rx_counter1;
#endif

//   Этот флаг устанавливается при переполнении входящего буфера
USART1
bit rx_buffer_overflow1;

// USART1 прерывание приемника
interrupt [USART1_RXC] void usart1_rx_isr(void)
{
     char status,data;
     status=UCSR1A;
```

```
     data=UDR1;
     if    ((status    &    (FRAMING_ERROR    |    PARITY_ERROR    |
DATA_OVERRUN))==0)
     {
          rx_buffer1[rx_wr_index1]=data;
          if (++rx_wr_index1 == RX_BUFFER_SIZE1) rx_wr_index1=0;
          if (++rx_counter1 == RX_BUFFER_SIZE1)
          {
               rx_counter1=0;
               rx_buffer_overflow1=1;
          };
     };
}

// Достаем символ из буфера приемника USART1
#pragma used+
char getchar1(void)
{
     char data;
     while (rx_counter1==0);
     data=rx_buffer1[rx_rd_index1];
     if (++rx_rd_index1 == RX_BUFFER_SIZE1) rx_rd_index1=0;
#asm("cli")
     --rx_counter1;
#asm("sei")
     return data;
}
#pragma used-
// USART1 буфер передатчика
#define TX_BUFFER_SIZE1 8
char tx_buffer1[TX_BUFFER_SIZE1];

#if TX_BUFFER_SIZE1<256
unsigned char tx_wr_index1,tx_rd_index1,tx_counter1;
#else
unsigned int tx_wr_index1,tx_rd_index1,tx_counter1;
#endif
```

```
// USART1 прерывание по передаче

interrupt [USART1_TXC] void usart1_tx_isr(void)
{
 if (tx_counter1)
     {
          --tx_counter1;
          UDR1=tx_buffer1[tx_rd_index1];
          if (++tx_rd_index1 == TX_BUFFER_SIZE1) tx_rd_index1=0;
     };
}
```

```
// Пишем символ в буфер USART1 передатчика

#pragma used+
void putchar1(char c)
{
     while (tx_counter1 == TX_BUFFER_SIZE1);
#asm("cli")
 if (tx_counter1 || ((UCSR1A & DATA_REGISTER_EMPTY)==0))
     {
          tx_buffer1[tx_wr_index1]=c;
          if (++tx_wr_index1 == TX_BUFFER_SIZE1) tx_wr_index1=0;
          ++tx_counter1;
     }
     else
          UDR1=c;
#asm("sei")
}
#pragma used-
```

Работа с USART0

```
#define RX_BUFFER_SIZE0 8
char rx_buffer0[RX_BUFFER_SIZE0];

#if RX_BUFFER_SIZE0<256
unsigned char rx_wr_index0,rx_rd_index0,rx_counter0;
#else
unsigned int rx_wr_index0,rx_rd_index0,rx_counter0;
#endif

bit rx_buffer_overflow0;

interrupt [USART0_RXC] void usart0_rx_isr(void)
{
     char status,data;
     status=UCSR0A;
     data=UDR0;
     if    ((status    &    (FRAMING_ERROR    |    PARITY_ERROR    |
DATA_OVERRUN))==0)
     {
          rx_buffer0[rx_wr_index0]=data;
          if (++rx_wr_index0 == RX_BUFFER_SIZE0) rx_wr_index0=0;
          if (++rx_counter0 == RX_BUFFER_SIZE0)
          {
               rx_counter0=0;
               rx_buffer_overflow0=1;
          };
     };
}

#ifndef _DEBUG_TERMINAL_IO_
#define _ALTERNATE_GETCHAR_
#pragma used+
```

```
char getchar(void)
{
     char data;
     while (rx_counter0==0);
     data=rx_buffer0[rx_rd_index0];
     if (++rx_rd_index0 == RX_BUFFER_SIZE0) rx_rd_index0=0;
#asm("cli")
     --rx_counter0;
#asm("sei")
     return data;
}
#pragma used-
#endif

#define TX_BUFFER_SIZE0 8
char tx_buffer0[TX_BUFFER_SIZE0];

#if TX_BUFFER_SIZE0<256
unsigned char tx_wr_index0,tx_rd_index0,tx_counter0;
#else
unsigned int tx_wr_index0,tx_rd_index0,tx_counter0;
#endif

interrupt [USART0_TXC] void usart0_tx_isr(void)
{
     if (tx_counter0)
     {
          --tx_counter0;
          UDR0=tx_buffer0[tx_rd_index0];
          if (++tx_rd_index0 == TX_BUFFER_SIZE0) tx_rd_index0=0;
     };
}

#ifndef _DEBUG_TERMINAL_IO_
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
     while (tx_counter0 == TX_BUFFER_SIZE0);
#asm("cli")
     if (tx_counter0 || ((UCSR0A & DATA_REGISTER_EMPTY)==0))
     {
          tx_buffer0[tx_wr_index0]=c;
          if (++tx_wr_index0 == TX_BUFFER_SIZE0) tx_wr_index0=0;
          ++tx_counter0;
     }
     else
          UDR0=c;
#asm("sei")
}
#pragma used-
#endif

#include <stdio.h>
```

Работа с АЦП

```
#define ADC_VREF_TYPE 0x00

// Прерывание АЦП
interrupt [ADC_INT] void adc_isr(void)
{
     // Read the AD conversion result
     // adc_data=ADCW;
}

// Читаем данные от АЦП по каналу номер adc_input
// Эти данные являются величиной освещенности под соответвующим датчиком отражения

unsigned int read_adc(unsigned char adc_input)
{
     unsigned int result = 0;
     char i = 0;
     ADMUX = adc_input ;
     for (i = 0; i < 16 ; i++){
          ADCSRA |= 0x40;
          while ((ADCSRA & 0x40));
          result += ADCW;
     }
     result >>= 4;
     return result;
}
```

Инициализируем параметры процессора

```
void init()
{
 // Инициализация портов ввода-вывода

 // Инициализируем порт A
 // 0 - ввод, 1 - вывод. Порт восьмибитный соответственно его
 // инициализация суть один байт.Он обозначается как DDR(название порта).Все остальные порты инициализируются аналогично

 // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T State1=T State0=T
     PORTA=0x00;
     DDRA=0x00;
```

```
     // Port B initialization
     //  Func7=In  Func6=In  Func5=In  Func4=In  Func3=In  Func2=In
Func1=In Func0=In
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T
State1=T State0=T
     PORTB=0x00;
     DDRB=0x00;

     // Port C initialization
     //  Func7=In  Func6=In  Func5=In  Func4=In  Func3=In  Func2=In
Func1=In Func0=In
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T
State1=T State0=T
     PORTC=0x00;
     DDRC=0x00;

     // Port D initialization
     //  Func7=In  Func6=In  Func5=In  Func4=In  Func3=In  Func2=In
Func1=In Func0=In
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T
State1=T State0=T
     PORTD=0x00;
     DDRD=0x00;

     // Port E initialization
     //  Func7=In  Func6=In  Func5=In  Func4=In  Func3=In  Func2=In
Func1=In Func0=In
     //  State7=T  State6=T  State5=T  State4=T  State3=T  State2=T
State1=T State0=T
     PORTE=0x00;
     DDRE=0x00;

     // Port F initialization
     //  Func7=In  Func6=In  Func5=In  Func4=Out  Func3=In  Func2=Out
Func1=In Func0=Out
     //  State7=T  State6=T  State5=T  State4=0  State3=T  State2=0
State1=T State0=0
     PORTF=0x14;
     DDRF=0x15;

     // Port G initialization
     // Func4=In Func3=In Func2=In Func1=In Func0=In
     // State4=T State3=T State2=T State1=T State0=T
     PORTG=0x00;
     DDRG=0x00;

     // Timer/Counter 0 initialization
     // Clock source: System Clock
     // Clock value: Timer 0 Stopped
     // Mode: Normal top=FFh
     // OC0 output: Disconnected
     ASSR=0x00;
     TCCR0=0x00;
     TCNT0=0x00;
     OCR0=0x00;
```

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter two initialization
// Clock source: System Clock
// Clock value: Timer two Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter three initialization
// Clock source: System Clock
// Clock value: Timer three Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer three Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
TCNT3L=0x00;
```

```
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x00;

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud rate: 115200
UCSR0A=0x00;
UCSR0B=0xD8;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x03;
// USART1 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART1 Receiver: On
// USART1 Transmitter: On
// USART1 Mode: Asynchronous
// USART1 Baud rate: 38400
UCSR1A=0x00;
UCSR1B=0xD8;
UCSR1C=0x06;
UBRR1H=0x00;
UBRR1L=0x0B;


// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
      // ADC initialization
      // ADC Clock frequency: 1000.000 kHz
      // ADC Voltage Reference: AREF pin
      ADMUX=ADC_VREF_TYPE & 0xff;
      ADCSRA=0x8B;

}
```

Определяем глобальные переменные

```
// Число от 0 до 1024, характеризующее освещенность датчика 1.
unsigned int led_1;

// Булевская величина – есть под датчиком линия или ее нет.
char led_1_state;

unsigned int led_2;
char led_2_state;
unsigned int led_3;
char led_3_state;

// Характеризует состояния датчиков в целом. Идентична переменным
led_x_state
// с x = 1..3

char sensor_state = 0;

// Состояние автомата A1
char y1 = 0;

// Направление движения
char direction = 0;

// Состояние датчиков на предыдущей итерации
char prev_state = 255;

// Переменная, отвечающая за последний отреагировавший боковой
сенсор
// 1 – левый, 0 – правый
signed char last_matched_sensor = 1;
// Уровень срабатывания датчика
unsigned int switch_on_level = 40;
// Состояние автомата A0
char y0 = 0;

// Флаг необходимости перечитывать АЦП
boolean need_to_read = 1;
```

```
// Документацию функций автоматов смотри выше
boolean x1_0()
{
     return sensor_state == st_led_idle;
}
boolean x1_1()
{
     return sensor_state == st_led_left;
}
boolean x1_2()
{
     return sensor_state == st_led_left_center;
}
boolean x1_3()
{
     return sensor_state == st_led_center;
}
boolean x1_4()
{
     return sensor_state == st_led_right;
}
boolean x1_5()
{
     return sensor_state == st_led_right_center;
}
boolean x1_6()
{
     return sensor_state == st_led_all;
}

// Эта функция обеспечивает чтение данных с датчиков отражения
находящихся под роботом
// С начала читаем данные с первого датчика
// 16 раз читаем данные с включенным и выключенным подсветочным
светодиодом.
// Результатом измерения будет их разность, деленная на 16.
// В зависимости от ее отношения с величиной switch_on_level будет
установлено
// Есть под датчиком линия или нет?

void read_state()
{
     // Включили подсветку
     PORTF &= 0xfe;
     led_1 = read_adc(1);

     // Выключили подсветку
     PORTF |= 0x01;
     led_1 = read_adc(1) - led_1;
     led_1_state = (led_1 < switch_on_level) ? 1 : 0;
```

```
	// Включили подсветку
	PORTF &= 0xfb;
	led_2 = read_adc(3);

	// Выключили подсветку
	PORTF |= 0x04;
	led_2 = read_adc(1) - led_2;
	led_2_state = (led_2 < switch_on_level) ? 1 : 0;

	// Включили подсветку
	PORTF &= 0xef;
	led_3 = read_adc(5);

	// Выключили подсветку
	PORTF |= 0x10;
	led_3 = read_adc(1) - led_3;
	led_3_state = (led_3 < switch_on_level) ? 1 : 0;
	sensor_state = led_1_state + led_2_state * 2 + led_3_state*4;
}
```

Задаем направления движения

```
void move_forward()
{
	direction = 0;
}

void turn_left()
{
	direction = 2;
}

void turn_right()
{
	direction = 3;
}

void set_last_matched_left()
{
	last_matched_sensor = -1;
}

void set_last_matched_right()
{
	last_matched_sensor = 1;
}

void stop()
{
	putchar1('s');
	delay_us(5);
}
```

```
// В этой функции происходит общение с модулем движения. В
//  зависимости  от  значения  переменной  direction  посылаются
различные сигналы. Протокол общения можно найти выше

void start()
{
     switch (direction)
     {
     case 0 :
          putchar1('f');
          delay_us(250);
          putchar1(0x02);
          break;
     case 1 :
          putchar1('b');
          delay_us(250);
          putchar1(0x02);
          break;
     case 2 :
          putchar1('1');
          delay_us(250);
          putchar1(0x02);
          delay_us(250);
          putchar1('2');
          delay_us(250);
          putchar1(0x83);
          break;
     case 3 :
          putchar1('2');
          delay_us(250);
          putchar1(0x02);
          delay_us(250);
          putchar1('1');
          delay_us(250);
          putchar1(0x83);
          break;
     }
     delay_us(250);
}


void z1_1()
{
     move_forward();
}

void z1_2() {
     turn_left();
}

void z1_3() {
     turn_right();
}
```

```
void z1_4()
{
     set_last_matched_left();
}

void z1_5()
{
     set_last_matched_right();
}

void z1_6() {
     switch (last_matched_sensor)
     {
     case -1 :
          {
               turn_left();
               break;
          }
     case 1 :
          {
               turn_right();
               break;
          }
     }
}
```

Реализация переходов в автомате 1

```
void y1_0()
{
     if (x1_3()||x1_6())
     {
          y1 = 1;
     } else if (x1_2())
     {
          y1 = 2;
     } else if (x1_5())
     {
          y1 = 3;
     } else if (x1_1())
     {
          z1_2();
          y1 = 4;
     } else if (x1_4())
     {
          y1 = 6;
          z1_3();
     }
}

//Under center
void y1_1()
{
     if (x1_0())
```

```
	{
		z1_6();
		y1 = 5;
	} else if (x1_2())
	{
		y1 = 2;
	}else if (x1_5())
	{
		y1 = 3;
	}
}

//Under left and center
void y1_2()
{
	z1_4();
	if (x1_0()){
		z1_2();
		y1 = 5;
	} else if (x1_1())
	{
		z1_2();
		y1 = 4;

	} else if (x1_6()||x1_3()||x1_5())
	{
		y1 = 1;
	}
}

//Under right and center
void y1_3()
{
	z1_5();
	if (x1_0()){
		z1_3();
		y1 = 5;
	} else if (x1_4())
	{
		z1_3();
		y1 = 6;
	} else if (x1_6()||x1_3()||x1_2())
	{
		y1 = 1;
	}
}

//Under left
void y1_4()
{
	z1_4();
	if (x1_0())
	{
		y1 = 5;
	} else if (x1_6()||x1_2())
```

```
     {
          y1 = 2;
          z1_1();
     }
}

//Idle, no sensors
void y1_5()
{

     if (x1_1())
     {
          z1_2();
          y1 = 4;
     }else if (x1_2())
     {
          z1_4();
          z1_1();
          y1 = 1;
     }else if (x1_5())
     {
          z1_5();
          z1_2();
          y1 = 1;
     }else if (x1_6()||x1_3()){
          z1_1();
          y1 = 1;
     }else if (x1_4()){
          z1_3();
          y1 = 6;
     }else if (sensor_state == st_led_left_right){
          z1_1();
     }
}

//Under right
void y1_6()
{
     z1_5();
     if (x1_0())
     {
          y1 = 5;
     } else if (x1_6()||x1_5())
     {
          z1_1();
          y1 = 2;
     }
}
```

```
// В буфере приемника есть данные
boolean x_0_1()
{
     char result = 0;
#asm("cli");
     result = (rx_counter0 != 0);
#asm("sei");
     return result;
}

//Находящиеся в буфере приемника данные – это команда поменять состояние
boolean x_0_2()
{
     boolean result = false;
#asm("cli");
     result = (rx_buffer0[rx_rd_index0] == 'q');
#asm("sei");
     return result;
}
//Функция отвечает за отправку данных с сенсоров пользователю.
//Сначала данные читаются из АЦП. После этого они посылаются пользователю
void send_sensor_states()
{
     read_state();
     putchar((char)(led_1 >> 8));
     delay_us(70);
     putchar((char)((led_1 << 8)>>8));
     delay_us(70);
     putchar((char)(led_2 >> 8));
     delay_us(70);
     putchar((char)((led_2 << 8)>>8));
     delay_us(70);
     putchar((char)(led_3 >> 8));
     delay_us(70);
     putchar((char)((led_3 << 8)>>8));
     delay_us(70);
     need_to_read = 0;
}
```



```
void z_0_0()
{
     stop();
}

void z_0_1()
{
```

```
     move_forward();
     start();
}

void z_0_2()
{
     char cmd = getchar();
     char cmd2;
     switch (cmd) {
       case '1' :
            {
                 putchar1(cmd);
                 cmd2 = getchar();
                 delay_us(100);
                 putchar1(cmd2);
                 delay_us(250);
                 break;
            }
       case '2' :
            {
                 putchar1(cmd);
                 cmd2 = getchar();
                 delay_us(100);
                 putchar1(cmd2);
                 delay_us(250);
                 break;
            }
       case 's' :
            {
                 stop();
                 break;
            }
       case 'f' :
            {
                 putchar1(cmd);
                 cmd2 = getchar();
                 delay_us(100);
                 putchar1(cmd2);
                 delay_us(250);
                 break;
            }
       case 'b' :
            {
                 putchar1(cmd);
                 cmd2 = getchar();
                 delay_us(100);
                 putchar1(cmd2);
                 delay_us(250);
                 break;
            }
       case 'r' :
            {
                 putchar1(cmd);
                 cmd2 = getchar();
                 delay_us(100);
```

```
                putchar1(cmd2);
                delay_us(250);
                break;
            }
        case 'l' :
            {
                putchar1(cmd);
                cmd2 = getchar();
                delay_us(100);
                putchar1(cmd2);
                delay_us(250);
                break;
            }
        case 'd' :
            {
                send_sensor_states();
                break;
            }
        case 'c' :
            {
                putchar(y1);
                delay_us(70);
                break;
            }
        case 'm' :
            {
                putchar(y0);
                delay_us(70);
                break;
            }
        case 'w' :
            {
                delay_us(70);
                cmd2 = getchar();
                switch_on_level = (signed int) cmd2;

            }
    };
}

//Change state of main statemachine
void z_0_3()
{
    char cmd = getchar();
    delay_us(70);
    prev_state = 255;
    cmd = getchar();
    delay_us(70);
    switch (cmd)
    {
    case '0' :
        {
            z_0_0();
            y0 = 0;
            break;
```

```
            }
      case '1' :
            {
                  z_0_1();
                  y0 = 1;
                  y1 = 0;
                  break;
            }
      }
}

//Return information in autonomy state
void z_0_4()
{
      char cmd = getchar();
      switch (cmd)
      {
      case 'd' :
            {
                  send_sensor_states();
                  break;
            }
      case 'c' :
            {
                  putchar(y1);
                  delay_us(70);
                  break;
            }
      case 'm' :
            {
                  putchar(y0);
                  delay_us(70);
                  break;
            }
      }
}
```

Запускаем автомат 1

```
void a1()
{

      if (need_to_read != 0)
      {
            read_state();
      } else
      {
            need_to_read = 1;
      }
      if (y1 != prev_state)
      {
            prev_state = y1;
```

```
            start();
        }
        switch (y1)
        {
        case 0 :
            {
                y1_0();
                break;
            }
        case 1 :
            {
                y1_1();
                break;
            }
        case 2 :
            {
                y1_2();
                break;
            }
        case 3 :
            {
                y1_3();
                break;
            }
        case 4 :
            {
                y1_4();
                break;
            }
        case 5 :
            {
                y1_5();
                break;
            }
        case 6 :
            {
                y1_6();
                break;
            }
        };
}
```

Состояния автомата 0

```
void y0_1()
{
    //message recivied
    if (x_0_1())
    {
        //Message to change mode has the biggest priority
        if (x_0_2())
        {
            //Change mode and do correspondng influence
```

```
                z_0_3();
            } else
            {
                //Other messages just do something
                z_0_4();
                a1();
            }
        } else
        {
            a1();
        }

}

void y0_0()
{
        //Message recivied
        if (x_0_1())
        {
            //Message to change mode has the biggest priority
            if (x_0_2())
            {
                //Change mode and do correspondng influence
                z_0_3();
            } else
            {
                //Other messages just do something
                z_0_2();
            }
        }
}
```

Запуск автомата 0

```
void a0()
{
        switch (y0)
        {
        case 0 :
            {     y0_0();
            break;
            }
        case 1 :
            {
                y0_1();
                break;
            }

        }
}
```

```
    void main(void)
    {
         init();
         // Global enable interrupts
         #asm("sei")
         while (1)
         {
              a0();
         }
    }
```

Приложение 2. Исходный код прошивки модуля движения

Инициализируем компилятор и процессор

```
#include <90s2313.h>
#include <delay.h>

#define RXB8 1
#define TXB8 0
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
```

Работаем с USART0

```
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on UART Receiver buffer overflow
bit rx_buffer_overflow;

// UART Receiver interrupt service routine
interrupt [UART_RXC] void uart_rx_isr(void)
{
     char status,data;
     status=USR;
     data=UDR;
```

```
     if ((status & (FRAMING_ERROR | DATA_OVERRUN))==0)
     {
          rx_buffer[rx_wr_index]=data;
          if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
          if (++rx_counter == RX_BUFFER_SIZE)
          {
               rx_counter=0;
               rx_buffer_overflow=1;
          };
     };
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the UART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
     char data;
     while (rx_counter==0);
     data=rx_buffer[rx_rd_index];
     if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
     --rx_counter;
#asm("sei")
     return data;
}
#pragma used-
#endif

// UART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE<256
unsigned char tx_wr_index,tx_rd_index,tx_counter;
#else
unsigned int tx_wr_index,tx_rd_index,tx_counter;
#endif

// UART Transmitter interrupt service routine
interrupt [UART_TXC] void uart_tx_isr(void)
{
     if (tx_counter)
     {
          --tx_counter;
          UDR=tx_buffer[tx_rd_index];
          if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
     };
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the UART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
```

```
void putchar(char c)
{
     while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
     if (tx_counter || ((USR & DATA_REGISTER_EMPTY)==0))
     {
          tx_buffer[tx_wr_index]=c;
          if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
          ++tx_counter;
     }
     else
          UDR=c;
#asm("sei")
}
#pragma used-
#endif
```

Инициализируем направления вращения моторов

```
// Standard Input/Output functions

#include <stdio.h>
// Left and Right motors speed
unsigned char left;
unsigned char right;
unsigned char leftLevel = 0x00;
unsigned char rightLevel = 0x00;
```

Обрабатываем прерывание по таймеру

```
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
     // Reinitialize Timer 0 value
     TCNT0=0xD0;

     right++;
     if ((right <= rightLevel)&&(rightLevel > 0))
     {
          //open port
          PORTB.3 = 0;
     }
     else
     {
          PORTB.3 = 1;
          //close port
     }
     left++;
     if ((left <= leftLevel)&&(leftLevel > 0))
     {
```

```
            //open port
            PORTB.2 = 0;
      }
      else
      {
            //close port
            PORTB.2 = 1;
      }
}
```

Инициализируем переменные программы

```
void init()
{
      //init speed
      left = 0x00;
      right = 0x00;
      //init levels
      leftLevel = 0x00;
      rightLevel = 0x00;

      // Input/Output Ports initialization
      // Port B initialization
      // Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=Out
Func1=In Func0=In
      // State7=T State6=T State5=T State4=T State3=1 State2=1 State1=T
State0=T
      PORTB=0x0C;
      DDRB=0x0C;

      // Port D initialization
      // Func6=Out Func5=Out Func4=In Func3=In Func2=In Func1=In
Func0=In
      // State6=1 State5=1 State4=T State3=T State2=T State1=T State0=T
      PORTD=0x40;
      DDRD=0x60;

      // Timer/Counter 0 initialization
      // Clock source: System Clock
      // Clock value: 1000.000 kHz
      TCCR0=0x02;
      TCNT0=0x00;

      // Timer/Counter 1 initialization
      // Clock source: System Clock
      // Clock value: 1000.000 kHz
      // Mode: Normal top=FFFFh
      // OC1 output: Discon.
      // Noise Canceler: Off
      // Input Capture on Falling Edge
      // Timer 1 Overflow Interrupt: On
      // Input Capture Interrupt: Off
```

```
      // Compare Match Interrupt: Off

      TCCR1A=0x00;
      TCCR1B=0x00;
      TCNT1H=0x00;
      TCNT1L=0x00;
      OCR1H=0x00;
      OCR1L=0x00;
      // External Interrupt(s) initialization
      // INT0: Off
      // INT1: Off
      GIMSK=0x00;
      MCUCR=0x00;

      // Timer(s)/Counter(s) Interrupt(s) initialization
      TIMSK=0x82;

      // UART initialization
      // Communication Parameters: 8 Data, 1 Stop, No Parity
      // UART Receiver: On
      // UART Transmitter: On
      // UART Baud rate: 56000
      UCR=0xD8;
      UBRR=0x0C;

      // Analog Comparator initialization
      // Analog Comparator: Off
      // Analog Comparator Input Capture by Timer/Counter 1: Off
      ACSR=0x80;
}
```

Эта функция реализует протокол общения с модулем движения описанный в документации

```
void main(void)
{
      char cmd;
      init();

      // Global enable interrupts
      #asm("sei")

      while (1)
      {
            cmd = getchar();
            switch (cmd) {
                  //control the left "gusenitsa"
             case '1' :
                   {
                        cmd = getchar();
                       #asm("cli");
                        //get the direction bit
                        PORTD.5 = cmd & 0x80;
```

```
                    //clear the direction bit
                    cmd = cmd & ~0x80;
                    //set speed to left
                    if (cmd == 0x07) {
                         leftLevel = 0xff;
                    } else {
                         leftLevel = cmd * 36;
                    }
                    #asm("sei");
                    break;
               }
               //control right
          case '2' :
               {
                    cmd = getchar();
                    #asm("cli");
                    t1_state = 0;
                    PORTD.6 = cmd & 0x80;
                    //clear the direction bit
                    cmd = cmd & ~0x80;
                    //set speed to left
                    if (cmd == 0x07) {
                         rightLevel = 0xff;
                    } else {
                         rightLevel = cmd * 36;
                    }
#asm("sei");
                    break;
               }
               //stop
          case 'S' :
          case 's' :
               {
                    #asm("cli");
                    rightLevel = 0;
                    leftLevel = 0;
                    #asm("sei");
                    break;
               }
               //move forward
          case 'F' :
          case 'f' :
               {
                    cmd = getchar();
                    #asm("cli");
                    t1_state = 0;
                    PORTD.6 = 1;
                    PORTD.5 = 1;
                    if (cmd == 0x07) {
                         leftLevel = 0xff;
                         rightLevel = 0xff;
                    } else {
                         leftLevel = cmd * 36;
                         rightLevel = cmd * 36;
                    }
```

```
            #asm("sei");
            break;
        }
        //move backward
    case 'b' :
    case 'B' :
        {
            cmd = getchar();
            #asm("cli");
            t1_state = 0;
            PORTD.6 = 0;
            PORTD.5 = 0;
            if (cmd == 0x07) {
                leftLevel = 0xff;
                rightLevel = 0xff;
            } else {
                leftLevel = cmd * 36;
                rightLevel = cmd * 36;
            }
           #asm("sei");
            break;
        }
        //turn left relative to the center
    case 'L' :
    case 'l' :
        {
            cmd = getchar();
            #asm("cli");
            t1_state = 0;
            PORTD.6 = 1;
            PORTD.5 = 0;
            if (cmd == 0x07) {
                leftLevel = 0xff;
                rightLevel = 0xff;
            } else {
                leftLevel = cmd * 36;
                rightLevel = cmd * 36;
            }
            #asm("sei");
            break;
        }
        //turn right relative to the center
    case 'R' :
    case 'r' :
        {
            cmd = getchar();
            #asm("cli")
            t1_state = 0;
            PORTD.6 = 0;
            PORTD.5 = 1;
            if (cmd == 0x07) {
                leftLevel = 0xff;
                rightLevel = 0xff;
            } else {
                leftLevel = cmd * 36;
```

```
                    rightLevel = cmd * 36;
                }
               #asm("sei");
                break;
            }
       default :
            {
                break;
            }
      }
};}
```