Referência do Arquivo main.c

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

Definições e Macros

```
#define F_CPU   16000000UL
```

Funções

```
    void adc_init (void)
uint16_t adc_read (uint8_t canal)
    void timer0_init ()
    void timer2_init ()
    void recebeuDisparo ()
    void moverTanque (uint8_t comando)
     int main (void)
         ISR (TIMER2_OVF_vect)
```

Variáveis

```
           RH_ASK rf_driver
 volatile uint8_t overflow_count = 0
          uint8_t leds [] = {2,3,4}
              int vidas = 3
          uint8_t vivo = 1
 volatile uint8_t adc_flag = 0
volatile uint16_t adc_value = 0
              int valor = 0
```

Definições e macros

◆ F_CPU

```
#define F_CPU   16000000UL
```

Funções

◆ adc_init()

```
void adc_init ( void )
```

```
21 {
22     ADMUX |= (1 << REFS0);
23     ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // prescaler 128
24 }
```

◆ adc_read()

uint16_t adc_read (uint8_t canal)

```
27  {
28      ADMUX = (ADMUX & 0XF0) | (canal & 0x0F);
29      ADCSRA |= (1 << ADSC);
30      while (ADCSRA & (1 << ADSC));
31      return ADC;
32  }
```

◆ ISR()

ISR (TIMER2_OVF_vect)

```
245 {
246     overflow_count++;
247     if (overflow_count >= 61)
248     {
249         PORTB ^= (1 << PB3); // Pisca o Laser
250         overflow_count = 0;
251     }
252 }
```

◆ main()

int main (void)

```
193 {
194     //CI L293D: Este CI é responsável pelo controle dos motores do projeto
195     DDRD |= (1<<5)|(1<<6); // PWM 1,2 e PWM 3,4 , respectivamente
196     DDRD |= (1<<7); //PIN 3A
197     DDRB |= (1<<0); //PIN 4A
198     DDRB |= (1<<1); //PIN 2A
199     DDRB |= (1<<2); //PIN 1A
200
201     //Os pinos PB2 e PD7 são responsáveis por girar o tanque para FRENTE
202     //Os pinos PB1 e PD0 são responsáveis por girar o tanque para TRÁS
203
204     //Laser
205     DDRB |= (1<<3);
206
207     //LEDs para vida
208     DDRC |= (1<<4)|(1<<3)|(1<<2);
209
210     //LDR
211     DDRC &= ~(1 << PC5);
212
213     //Ascende os LEDs
214     PORTC |= (1<<4)|(1<<3)|(1<<2);
215
216     timer0_init();
217     timer2_init();
218     adc_init();
219     rf_driver.init(); //Inicia a comunicação entre o controle e o Atmega328 via rádio
220
221     while(vivo == 1)
222     {
223         valor = adc_read(5);
224         //Verifica o valor de LDR; se a condição for verdadeira, chama a função recebeuDispar
    o();
225         if (valor > 1000)
226         {
227             recebeuDisparo();
228         }
229
230         // Verifica comandos recebidos pelo receptor 433Mhz
231         uint8_t buf[2]; //buff[0] é o ID do sinal e buff[1] o ID da posição do joystick no co
    ntrole
232         uint8_t buflen = sizeof(buf);
233         if(rf_driver.recv(buf, &buflen))
234         {
235             //Se a varíavel de identificação corresponder, então é o sinal correto
236             if(buf[0] == 0xA5)
237             {
238             moverTanque(buf[1]);
239             }
240         }
241     }
242 }
```

◆ moverTanque()

void moverTanque (uint8_t comando)

```
{
    //Os pinos que controlam a polaridade do motor da ESQUERDA são: PD7 e PB0
    //Os pinos que controlam a polaridade do motor da DIREITA são: PB1 e PB2
    switch(comando)
    {
        case 1: // frente
            OCR0A = 255; //PD6
            OCR0B = 255; //PD5
            //Encerra o envio de tensão para os pinos PB0 e PB1
            PORTB &= ~(1<<0); //Polaridade para trás
            PORTB &= ~(1<<1); //Polaridade para trás
            //Envia tensão para os pinos PD7 e PB2
            PORTD |= (1<<7);// Polaridade para frente
            PORTB |= (1<<2);// Polaridade para frente
            break;
        case 2: // ré
            OCR0A = 255;
            OCR0B = 255;
            //Encerra o envio de tensão para os pinos PD7 e PB2
            PORTD &= ~(1<<7);// Polaridade para frente
            PORTB &= ~(1<<2);// Polaridade para frente
            //Envia tensão para os pinos PD7 e PB2
            PORTB |= (1<<0);  //Polaridade para trás
            PORTB |= (1<<1); //Polaridade para trás
            break;
        case 3: // esquerda
            //Envia tensão somente para o PB2
            OCR0A = 0;
            OCR0B = 255;
            PORTD &= ~(1<<7);// Polaridade para frente
            PORTB &= ~(1<<0); //Polaridade para trás
            PORTB &= ~(1<<1); //Polaridade para trás
            PORTB |= (1<<2);// Polaridade para frente
            break;
        case 4: // direita
            //Envia tensão somente para o PD7
            OCR0A = 255;
            OCR0B = 0;
            PORTB &= ~(1<<1); //Polaridade para trás
            PORTB &= ~(1<<0); //Polaridade para trás
            PORTB &= ~(1<<2);// Polaridade para frente
            PORTD |= (1<<7); // Polaridade para frente
            break;
        case 5: // diagonal sup esq
            //Encerra o envio de tensão para os pinos PB0 e PB1
            OCR0A = 127;
            OCR0B = 255;
            PORTB &= ~(1<<0); //Polaridade para trás
            PORTB &= ~(1<<1); //Polaridade para trás
            //Envia tensão para os pinos PD7 e PB2
            PORTD |= (1<<7); // Polaridade para frente
            PORTB |= (1<<2); // Polaridade para frente
            break;
        case 6: // diagonal sup dir
            OCR0A = 255;
            OCR0B = 127;
            //Encerra o envio de tensão para os pinos PB0 e PB1
            PORTB &= ~(1<<0); //Polaridade para trás
            PORTB &= ~(1<<1); //Polaridade para trás
            //Envia tensão para os pinos PD7 e PB2
            PORTD |= (1<<7); // Polaridade para frente
            PORTB |= (1<<2); // Polaridade para frente
            break;
        case 7: // diagonal inf esq
            OCR0A = 127;
            OCR0B = 255;
            //Encerra o envio de tensão para os pinos PD7 e PB2
            PORTD &= ~(1<<7); // Polaridade para frente zerada
            PORTB &= ~(1<<2); // Polaridade para frente zerada
            //Envia tensão para os pinos PB0 e PB1;
            PORTB |= (1<<0); //Polaridade para trás
            PORTB |= (1<<1); //Polaridade para trás
            break;
        case 8: // diagonal inf dir
            OCR0A = 255;
            OCR0B = 127;
            //Encerra o envio de tensão para os pinos PD7 e PB2
            PORTD &= ~(1<<7);
            PORTB &= ~(1<<2);
            //Envia tensão para os pinos PB0 e PB1;
            PORTB |= (1<<0);
            PORTB |= (1<<1);
            break;
    }
}
```

◆ recebeuDisparo()

```
void recebeuDisparo ( )
```

```
                    {
    vidas--;
    if(vidas!=0)
    {
        vivo = 0;
        PORTC &= ~(1 << leds[vidas]);
        //Realiza o 180º graus
        OCR0A = 255;
        OCR0B = 255;
        PORTB &= ~(1<<2);
        PORTB &= ~(1<<0);
        PORTD |= (1<<7);
        PORTB |= (1<<1);
        _delay_ms(2000); // delay até realizar o giro de 180º
        //Encerra o giro, desliga os motores e permanece inativo
        PORTB &= ~(1<<3);
        TIMSK2 &= ~(1 << TOIE2);
        OCR0A = 0;
        OCR0B = 0;
        PORTB &= ~(1<<2);
        PORTB &= ~(1<<1);
        PORTB &= ~(1<<0);
        PORTD &= ~(1<<7);
        _delay_ms(3000);
        vivo = 1;
        TIMSK2 |= (1 << TOIE2);
    }
    else
    {
        //Todas as vidas foram encerradas, logo não pode funcionar mais
        vivo = 0;
        PORTB &= ~(1<<3);
        TIMSK2 &= ~(1 << TOIE2);
        PORTC &= ~(1 << leds[vidas]); //Desliga o último LED restante
        //Realiza o 180º graus
        OCR0A = 255;
        OCR0B = 255;
        PORTB &= ~(1<<2);
        PORTB &= ~(1<<0);
        PORTD |= (1<<7);
        PORTB |= (1<<1);
        _delay_ms(2000); // delay até realizar o giro de 180º
        //Encerra o giro, desliga os motores e permanece inativo
        PORTB &= ~(1<<3);
        TIMSK2 &= ~(1 << TOIE2);
        OCR0A = 0;
        OCR0B = 0;
        PORTB &= ~(1<<2);
        PORTB &= ~(1<<1);
        PORTB &= ~(1<<0);
        PORTD &= ~(1<<7);
    }
}
```

◆ timer0_init()

```
void timer0_init ( )
```

```
{
    TCCR0A = (1 << WGM01) | (1 << WGM00);
    TCCR0A |= (1 << COM0A1) | (1 << COM0B1);
    TCCR0B = (1 << CS01) | (1 << CS00);
}
```

◆ timer2_init()

```
void timer2_init ( )
```

```
{
    TCCR2A = 0x00;
    TCCR2B = (1 << CS22) | (1 << CS21) | (1 << CS20);
    TIMSK2 = (1 << TOIE2);
    sei();
}
```

Variáveis

◆ adc_flag

```
volatile uint8_t adc_flag = 0
```

◆ adc_value

```
volatile uint16_t adc_value = 0
```

◆ leds

```
uint8_t leds[] = {2,3,4}
```

```
    13   {2,3,4}; // Array para identificação dos LEDs em DDRC
```

◆ overflow_count

```
volatile uint8_t overflow_count = 0
```

◆ rf_driver

```
RH_ASK rf_driver
```

◆ valor

```
int valor = 0
```

◆ vidas

```
int vidas = 3
```

◆ vivo

```
uint8_t vivo = 1
```