

# 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 <b>adc_init</b> (void)
uint16_t <b>adc_read</b> (uint8_t canal)
void <b>timer0_init</b> ()
void <b>timer2_init</b> ()
void <b>recebeuDisparo</b> ()
void <b>moverTanque</b> (uint8_t comando)
int <b>main</b> (void)
<b>ISR</b> (TIMER2_OVF_vect)

## Variáveis

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

## Definições e macros

◆ F\_CPU

#define F\_CPU 16000000UL

## Funções

◆ adc\_init()

void adc\_init (void )

21

22

23

24

{

ADMUX |= (1 << REFS0);

ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // prescaler 128

}

◆ adc\_read()

▼ Definições e Macros	F_CPU
▼ Funções	adc_init
	adc_read
	timer0_init
	timer2_init
	recebeuDisparo
	moverTanque
	main
	ISR
▼ Variáveis	rf_driver
	overflow_count
	leds
	vidas
	vivo
	adc_flag
	adc_value
	valor
▼ Definições e macros	F_CPU
▼ Funções	adc_init
	adc_read
	ISR
	main
	moverTanque
	recebeuDisparo
	timer0_init
	timer2_init
▼ Variáveis	adc_flag
	adc_value
	leds
	overflow_count
	rf_driver
	valor
	vidas
	vivo

```
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
225         o();
226         if (valor > 1000)
227         {
228             recebeuDisparo();
229         }
230
231         // Verifica comandos recebidos pelo receptor 433Mhz
232         uint8_t buf[2]; //buff[0] é o ID do sinal e buff[1] o ID da posição do joystick no co
233         ntrole
234         uint8_t buflen = sizeof(buf);
235         if(rf_driver.recv(buf, &buflen))
236         {
237             //Se a variável de identificação corresponder, então é o sinal correto
238             if(buf[0] == 0xA5)
239             {
240                 moverTanque(buf[1]);
241             }
242         }
243     }
244 }
```

◆ moverTanque()

```
void moverTanque ( uint8_t comando )
```

```
104 {
105     //Os pinos que controlam a polaridade do motor da ESQUERDA são: PD7 e PB0
106     //Os pinos que controlam a polaridade do motor da DIREITA são: PB1 e PB2
107     switch(comando)
108     {
109         case 1: // frente
110             OCR0A = 255; //PD6
111             OCR0B = 255; //PD5
112             //Encerra o envio de tensão para os pinos PB0 e PB1
113             PORTB &= ~(1<<0); //Polaridade para trás
114             PORTB &= ~(1<<1); //Polaridade para trás
115             //Envia tensão para os pinos PD7 e PB2
116             PORTD |= (1<<7); // Polaridade para frente
117             PORTB |= (1<<2); // Polaridade para frente
118             break;
119         case 2: // ré
120             OCR0A = 255;
121             OCR0B = 255;
122             //Encerra o envio de tensão para os pinos PD7 e PB2
123             PORTD &= ~(1<<7); // Polaridade para frente
124             PORTB &= ~(1<<2); // Polaridade para frente
125             //Envia tensão para os pinos PD7 e PB2
126             PORTB |= (1<<0); //Polaridade para trás
127             PORTB |= (1<<1); //Polaridade para trás
128             break;
129         case 3: // esquerda
130             //Envia tensão somente para o PB2
131             OCR0A = 0;
132             OCR0B = 255;
133             PORTD &= ~(1<<7); // Polaridade para frente
134             PORTB &= ~(1<<0); //Polaridade para trás
135             PORTB &= ~(1<<1); //Polaridade para trás
136             PORTB |= (1<<2); // Polaridade para frente
137             break;
138         case 4: // direita
139             //Envia tensão somente para o PD7
140             OCR0A = 255;
141             OCR0B = 0;
142             PORTB &= ~(1<<1); //Polaridade para trás
143             PORTB &= ~(1<<0); //Polaridade para trás
144             PORTB &= ~(1<<2); // Polaridade para frente
145             PORTD |= (1<<7); // Polaridade para frente
146             break;
147         case 5: // diagonal sup esq
148             //Encerra o envio de tensão para os pinos PB0 e PB1
149             OCR0A = 127;
150             OCR0B = 255;
151             PORTB &= ~(1<<0); //Polaridade para trás
152             PORTB &= ~(1<<1); //Polaridade para trás
153             //Envia tensão para os pinos PD7 e PB2
154             PORTD |= (1<<7); // Polaridade para frente
155             PORTB |= (1<<2); // Polaridade para frente
156             break;
157         case 6: // diagonal sup dir
158             OCR0A = 255;
159             OCR0B = 127;
160             //Encerra o envio de tensão para os pinos PB0 e PB1
161             PORTB &= ~(1<<0); //Polaridade para trás
162             PORTB &= ~(1<<1); //Polaridade para trás
163             //Envia tensão para os pinos PD7 e PB2
164             PORTD |= (1<<7); // Polaridade para frente
165             PORTB |= (1<<2); // Polaridade para frente
166             break;
167         case 7: // diagonal inf esq
168             OCR0A = 127;
169             OCR0B = 255;
170             //Encerra o envio de tensão para os pinos PD7 e PB2
171             PORTD &= ~(1<<7); // Polaridade para frente zerada
172             PORTB &= ~(1<<2); // Polaridade para frente zerada
173             //Envia tensão para os pinos PB0 e PB1;
174             PORTB |= (1<<0); //Polaridade para trás
175             PORTB |= (1<<1); //Polaridade para trás
176             break;
177         case 8: // diagonal inf dir
178             OCR0A = 255;
179             OCR0B = 127;
180             //Encerra o envio de tensão para os pinos PD7 e PB2
181             PORTD &= ~(1<<7);
182             PORTB &= ~(1<<2);
183             //Envia tensão para os pinos PB0 e PB1;
184             PORTB |= (1<<0);
185             PORTB |= (1<<1);
186             break;
187     }
188 }
```

◆ recebeuDisparo()

```
void recebeuDisparo ( )
```

```
49         {
50     vidas--;
51     if(vidas!=0)
52     {
53         vivo = 0;
54         PORTC &= ~(1 << leds[vidas]);
55         //Realiza o 180º graus
56         OCR0A = 255;
57         OCR0B = 255;
58         PORTB &= ~(1<<2);
59         PORTB &= ~(1<<0);
60         PORTD |= (1<<7);
61         PORTB |= (1<<1);
62         _delay_ms(2000); // delay até realizar o giro de 180º
63         //Encerra o giro, desliga os motores e permanece inativo
64         PORTB &= ~(1<<3);
65         TIMSK2 &= ~(1 << TOIE2);
66         OCR0A = 0;
67         OCR0B = 0;
68         PORTB &= ~(1<<2);
69         PORTB &= ~(1<<1);
70         PORTB &= ~(1<<0);
71         PORTD &= ~(1<<7);
72         _delay_ms(3000);
73         vivo = 1;
74         TIMSK2 |= (1 << TOIE2);
75     }
76     else
77     {
78         //Todas as vidas foram encerradas, logo não pode funcionar mais
79         vivo = 0;
80         PORTB &= ~(1<<3);
81         TIMSK2 &= ~(1 << TOIE2);
82         PORTC &= ~(1 << leds[vidas]); //Desliga o último LED restante
83         //Realiza o 180º graus
84         OCR0A = 255;
85         OCR0B = 255;
86         PORTB &= ~(1<<2);
87         PORTB &= ~(1<<0);
88         PORTD |= (1<<7);
89         PORTB |= (1<<1);
90         _delay_ms(2000); // delay até realizar o giro de 180º
91         //Encerra o giro, desliga os motores e permanece inativo
92         PORTB &= ~(1<<3);
93         TIMSK2 &= ~(1 << TOIE2);
94         OCR0A = 0;
95         OCR0B = 0;
96         PORTB &= ~(1<<2);
97         PORTB &= ~(1<<1);
98         PORTB &= ~(1<<0);
99         PORTD &= ~(1<<7);
100     }
101 }
```

◆ timer0\_init()

```
void timer0_init ( )
```

```
35 {
36     TCCR0A = (1 << WGM01) | (1 << WGM00);
37     TCCR0A |= (1 << COM0A1) | (1 << COM0B1);
38     TCCR0B = (1 << CS01) | (1 << CS00);
39 }
```

◆ timer2\_init()

```
void timer2_init ( )
```

```
42 {
43     TCCR2A = 0x00;
44     TCCR2B = (1 << CS22) | (1 << CS21) | (1 << CS20);
45     TIMSK2 = (1 << TOIE2);
46     sei();
47 }
```

◆ 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
```