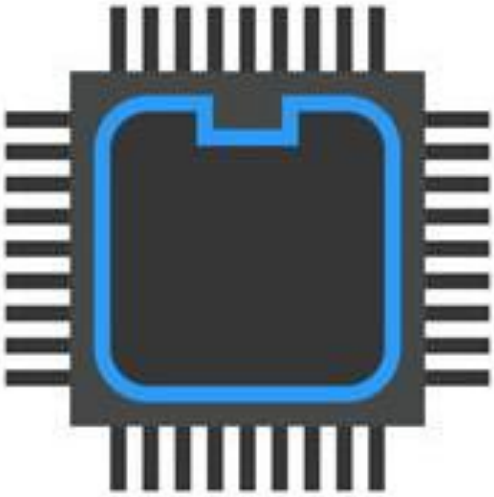
Microprocessadores

Projeto Final



Sensor de Estacionamento

2ºL_EEC-EC-01 Grupo 5 12/07/2021

Bernardo Milheiro – 190250008

Mateus Araújo – 190250057

Índice

Introdução

Devido ao avanço tecnológico atual, é previsível que os objetos que utilizamos no nosso dia-a-dia se tornem mais inteligentes, mais eficientes ou até mesmo, que tenham uma utilização mais facilitada do ponto de vista do utilizador. Um exemplo deste avanço tecnológico é o sensor de estacionamento, o qual tornou mais facilitada uma prática bastante comum quando utilizamos um automóvel. Através da utilização de um microcontrolador da família Atmega (Atmega328P), de LEDs e de um Buzzer, desenvolvemos então, um sensor de estacionamento que informe o utilizador da sua proximidade a um obstáculo.

Descrição técnica

Planeamento do Projeto

De forma a desenvolver o sistema foram necessários os seguintes componentes:

- Microcontrolador Atmega328P (Arduino UNO).
- Sensor ultrassónico de distância, HC-SR04.
- LEDs Vermelho, Verde e Amarelo.
- Resistências de 220 Ohm.
- Buzzer Ativo.

Em baixo, é apresentado os esquemas das ligações e do circuito desenvolvido:

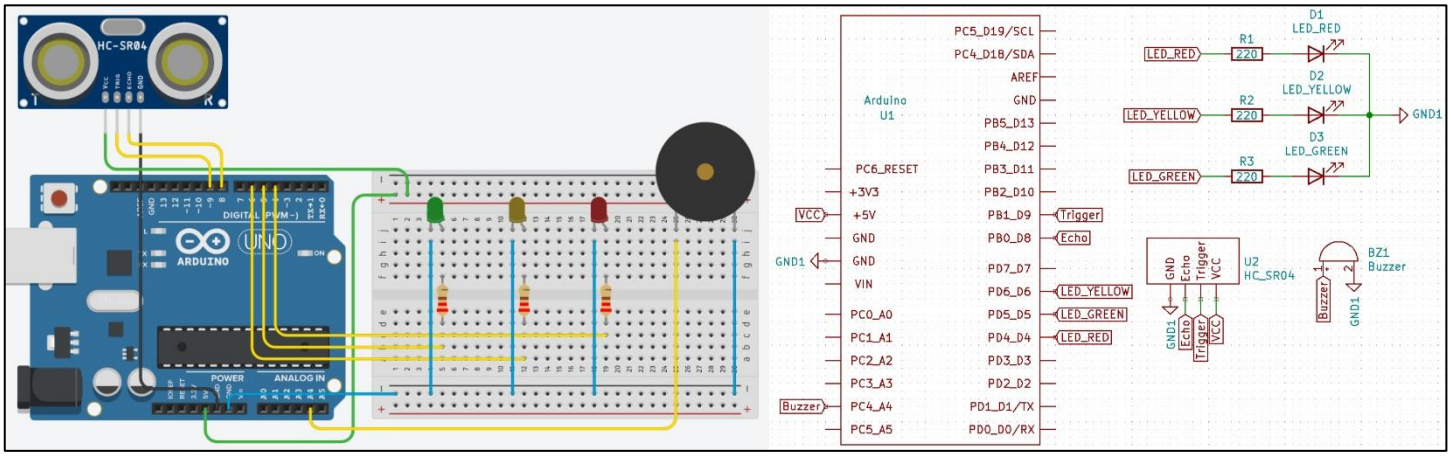


Figura 1 - Esquema do Hardware e Esquema Elétrico

Para que seja considerado que o **Sensor de Estacionamento** está a funcionar nas condições devidas, os LEDs devem demonstrar a proximidade ao obstáculo, alternando entre eles e o Buzzer deve apitar com mais frequência quando o veículo se aproxima do obstáculo e deixar de apitar quando existe uma distância de segurança. O LED Verde deve acender quando existe bastante distância entre o veículo e o obstáculo. O LED Vermelho, pelo contrário, deve acender quando o veículo e o obstáculo estão bastante próximos. O LED Amarelo deve acender quando existe uma certa proximidade, mas ainda sem perigo de choque.

O código foi então desenvolvido e organizado por tópicos/funcionalidades de forma a puder ser feita uma análise detalhada e eficiente.

Definições e Bibliotecas ***Código 1***

Para começar, foram incluídas as bibliotecas necessárias ao funcionamento do sistema e as funções BETWEEN e char2int():

```
#define F_CPU 16000000UL                    // 16 MHz clock speed
#include <avr/io.h>                         // Carrega a biblioteca que permite utilizar as funções de input e output
#include <avr/interrupt.h>                  // Carrega a biblioteca que permite utilizar as funções de interrupção
#include <stdint.h>                         // Carrega a biblioteca que permite utilizar inteiros com larguras especificadas
#include <stdlib.h>                         // Carrega a biblioteca que permite utilizar inteiros com larguras especificadas
```

Figura 2 - Código: Bibliotecas

```
// Operação de "Entre"
#define BETWEEN(value, min, max) (value < max && value > min)

// Função necessária para transformar um array de char numa variavel inteira
int char2int (char *array, size_t n){
    int number = 0;                                 // Inteiro que vai retornar
    int mult = 1;                                   // Multiplicador
    while (n--){                                    // por cada caracter no vetor
        // Numero igual à conversao para inteiro multiplicado por mult, que vai definir a posição do numero
        number += (array[n] - '0') * mult;
        mult *= 10;
    }
    return number;
}
```

Figura 3 - Código: BETWEEN e char2int()

A função **BETWEEN** recebe o valor que pretendemos saber se está no intervalo e faz uma simples operação AND para saber se o valor é maior que o valor mínimo e menor que o valor máximo do intervalo recebido.

A função ***char2int*** recebe o vetor de caracteres e o tamanho do vetor em formato de **unsigned int**, visto que estamos a trabalhar com um vetor que recebe esse tipo de dados. Esta função iguala a variável que vamos retornar, à soma das conversões (ASCII) para inteiro, multiplicado pela variável mult. Simplificando com um exemplo, se tivermos vetor = [3,2,1] e n = 3, a função fará o seguinte:

number = (vetor[3] – ‘0’)*mult number = (‘1’ – ‘0’)*1 = 1

number = (vetor[2] – ‘0’)*mult number = (‘2’ – ‘0’)*10 = 20

number = (vetor[1] – ‘0’)*mult number = (‘3’ – ‘0’)*100 = 300

Logo, number += (array[n] - '0') * mult, será igual a 321.

Porta Série ***Código 2***

Começou-se por desenvolver também o código necessário para a utilização da **Porta Série** do Atmega328P, de forma a observar através do software CoolTerm, a distância que o Sensor HC-SR04 está a medir. Como pode ser observado no código desenvolvido, foi definida a Taxa de transmissão e o Baud Rate adequado, tal como, a variável data e tecla, para, em seguida, desenvolver as funções necessárias à sua inicialização, à transmissão e receção de dados e ainda à transmissão de uma string.

A função ***USART_Init*** recebe a Taxa de Transmissão pretendida e ativa o transmissor e o recetor da USART, inicializando a 1 os bits TXEN0 e RXEN0 no registo UCSRnB. Quando o transmissor e o recetor são ativados, a USART passa a controlar as operações que ocorrem nos pinos TxDn e RxDn.

```
void USART_Init (unsigned int ubrr){                    // Função para Inicialização da USART
    UBRR0 = ubrr;                                       // Ajusta a Taxa de Transmissão
    UCSR0B = (1<<RXEN0)|(1<<TXEN0);                     // Ativa o transmissor e o recetor
}
```

Figura 4 - Código: USART_Init()

A transmissão de dados (***USART_Transmit***) é iniciada quando, após verificado que o registo de transmissão está vazio, colocamos os dados a transmitir no buffer de transmissão, mais precisamente, escrevendo no registo UDRn. A receção de dados (***USART_Receive***) é iniciada quando o Recetor deteta um start bit válido e guarda esses dados até detetar um stop bit, transmitindo-os, em seguida, pelo buffer e escrevendo no registo UDRn.

```
void USART_Transmit (char data){                    // Função de Transmissão de Dados
    while(!(UCSR0A & (1<<UDRE0)));                  // Operação while para garantir que o registo de transmissão está vazio
    UDR0 = (data++);                                // Serve para colocar os dados e enviar para o USART
}

char USART_Receive (void){                          // Função de Receção de Dados
    while(!(UCSR0A & (1<<RXC0)));                   // Enquanto não detetar o stop bit
    return UDR0;                                    // Lê o registo e retorna
}
```

Figura 5 - Código: USART_Transmit() e USART_Receive()

A função ***enviaString*** recebe uma string e através dos mecanismos de transmissão da USART envia, caracter a caracter, a string recebida.

```
void enviaString(unsigned char string[]){           // Função para enviar uma string
    unsigned int i=0;                               // Variavel de posição no string
    while(string[i] != 0){                          // Enquanto houver caracteres
        USART_Transmit(string[i]);                  // Envia os caracteres
        i++;                                        // Incrementa i
    }
}
```

Figura 6 - Código: enviaString()

Através da função acima, serão enviadas as seguintes strings:

```
char *menu = "Iniciar Sensor de Estacionamento?\n\nSim - Pressionar a tecla 's'.\nNão - Pressionar a tecla 'n'.\n\nOpção: ";
char *nao = "\nBoa Viagem!\n";
```

Figura 7 - Código: Menus

Para concluir o código necessário para a **Porta Série**, foi desenvolvida a rotina de interrupção da USART.

```
ISR(USART_RX_vect){                                 // Interrupção da USART
    data = UDR0;                                    // Dá a variavel data o valor do registo UDR0
}
```

Figura 8 - Código: Interrupção USART

LEDs

Código 3

Para configurar os **LEDs**, foi apenas necessário definir as máscaras que permitem aceder aos pinos associados aos **LEDs.** O **LED Vermelho** está ligado ao pino digital 4 (PIND4), o **LED Verde** está ligado ao pino digital 5 (PIND5) e o **LED Amarelo** está ligado ao pino digital 6 (PIND6).

```
// Define as máscaras que permitem aceder aos pinos associados aos LEDs
#define LED_RED     0x10
#define LED_GREEN   0x20
#define LED_YELLOW  0x40
```

Figura 9 - Código: LEDs

Timers

Código 4

Em relação aos **Timers**, foi necessário criar a função ***timer1***. Este Timer é um temporizador/contador de 16 bits, permitindo uma temporização precisa das ações do programa e ainda a geração de formas de onda e medição temporal dos sinais. Consultando o datasheet do Atmega328P, no modo normal, a configuração dos registos será a seguinte:

TCCR1A = 0x00, pois WGM13:0 = 0000, COM1A1:0 = 0 e COM1B1:0 = 0;

TCCR1B = 0x03, pois WGM13:0 = 0000, CS12:0 = 011 (para relógio interno com divisor por 64), ICNC1 = 0 e ICES1 = 0;

TCCR1C = 0x00, pois FOC1A = 0 e FOC1B = 0; TIMSK1 = 0x01, pois TOIE1 = 1 para ativação da interrupção por overflow do Timer1 os outros bits a 0 porque não vamos utilizar as outras interrupções;

TIMSK1 = 0x01, pois TOIE1 = 1 para ativação da interrupção por overflow do Timer1. Restantes bits a 0 porque não iremos utilizar as outras interrupções:

Tendo por base uma temporização única de 1ms e de forma a gerar as ondas quadradas pretendidas, programou-se o registo de contagem do **Timer1** (**TCNT1**). Calculou-se o valor inicial de forma a que decorra 1ms entre o momento inicial e o momento em que há overflow:

$$f_{osc} = 16MHz \quad Divisor = 64 \quad f_{contador} = \frac{f_{osc}}{Divisor} = \frac{1}{T} = 0.25MHz$$

$$T = 4\mu s \quad t = 1ms \quad Nº\ de\ Contagens = \frac{t}{T} = 250$$

$$Valor\ Inicial\ de\ Contagem = 2^{16} - 250 = 65286$$

No Atmega328P, todas as **interrupções** são mascaráveis, ou seja, existe um bit individual para ativação/desativação de cada **interrupção** e existe ainda um bit de ativação/desativação global das interrupções (Global Interrupt Enable). A função que efetua a ativação global das interrupções é a função ***sei()*** que coloca o bit de ativação/desativação global das interrupções a 1 (I = 1), ativando as interrupções cujos bits de ativação individual estiverem a 1.

Por fim, desenvolvemos a rotina de serviço à **interrupção** associada à ocorrência de overflow no **Timer1**, a partir da informação de que o endereço dessa rotina será guardado em forma de vetor. Utilizando os valores acima obtidos, a rotina de serviço à interrupção será então executada, de 1 em 1ms.

```
void timer1(void){      // Função Timer1 para uso das interrupções
    TCCR1A = 0x00;
    // WGM13:0 = 0000 para modo normal;
    // COM1A1:0 = 0 e COM1B1:0 = 0 no modo normal

    TCCR1B = 0x03;
    // WGM13:0 = 0000 para modo normal;
    // CS12:0 = 011 para relógio interno com divisor por 64;
    // ICNC1 = 0 e ICES1 = 0 no modo normal

    TCCR1C = 0x00;
    // FOC1A = 0 e FOC1B = 0 no modo normal

    TIMSK1 = (1 << TOIE1);
    // Ativa as interrupções do Timer1

    TCNT1 = 65286;
    // Valor inicial de contagem = 2^16-250 = 65286

    sei();
    // Ativação Interrupções Globais
}

volatile unsigned char cont_ovf1 = 0;              // Inicializa a variavel de contagem de Overflows

ISR(TIMER1_OVF_vect){                              // Interrupção Timer1
    TCNT1 = 65286;                                 // Valor inicial de contagem
    cont_ovf1++;                                   // Incrementa o contador à passagem de 1ms
}
```

Figura 10 - Código: Timer1

Semelhante ao Timer1, foi criada a função ***timer0***. Sendo este um contador de 8 bits, o código desenvolvido foi o seguinte:

```
void timer0(void) {
    TCCR0A = (0<<WGM00);                           // Modo Normal
    TCCR0B = (0b011 << CS00);                      // Prescaler de 64
    TCNT0 = 256-250;                               // Valor inicial de contagem
    TIMSK0 = (1<<TOIE0);                           // Ativação da Interrupção do timer0
    sei();                                         // Ativação Interrupções Globais
}

volatile unsigned char cont_ovf0 = 0;              // Inicializa a variavel de contagem de Overflows

ISR(TIMER0_OVF_vect){                              // Interrupção Timer0
    TCNT0 = 6;                                     // Valor inicial de contagem
    cont_ovf0++;                                   // Incrementa o contador à passagem de 1ms
}
```

Figura 11 - Código: Timer0

Sensor HC-SR04 ***Código 5***

O **sensor ultrassónico HC-SR04** é baseado no princípio do SONAR e do RADAR, os quais são utilizados para determinar a distância a um objeto. O sensor ultrassónico gera ondas de som de alta frequência (ultrassom) e no momento em que essas ondas atingem um objeto, as mesmas refletem como eco e são detetadas pelo recetor do sensor. Ao obter o tempo que o eco demora a retornar ao recetor, podemos calcular a que distancia o sensor está do objeto.

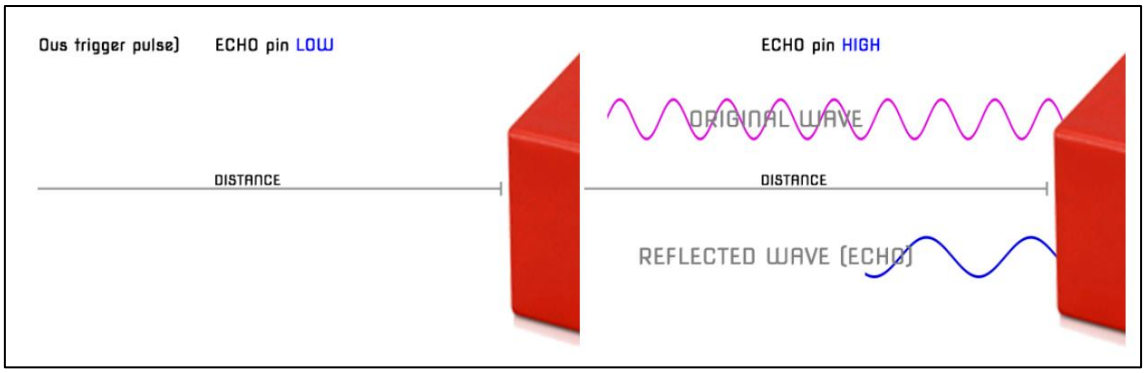


Figura 12 - Representação do funcionamento do Sensor

No **sensor ultrassónico HC-SR04**, o pulso *trigger* gera uma onda ultrassónica com frequência de 40kHz. Após gerar a onda, o pino *Echo* fica *High* e mantem o seu estado *High* até que deixe de receber ondas provenientes do eco, logo, o tempo em que o pino Echo está a *High* será a soma do tempo que a onda demorou a percorrer a distância até ao objeto com o tempo que demorou a voltar ao sensor. Com esta informação, podemos deduzir a fórmula de calculo da distância ao objeto:

Distancia = Velocidade do Som (340m/s) * (Tempo pino *Echo* (*High*) / 2)

Em relação ao código, foram desenvolvidas algumas funções de forma a garantir o seu correto funcionamento. Inicialmente, foi inicializada a variável **distance**, um vetor de **char** que vai conter a distância que será transmitida através da USART e foram definidas as máscaras dos pinos associados ao *Trigger* e ao *Echo* do sensor. Em seguida, foi desenvolvida a função ***trigger*** que tem como objetivo acionar o *Trigger* do Sensor e permite alterar a frequência com que o mesmo envia uma onda ultrassónica.

```
volatile int16_t distance;                      // Inicializa a variavel de distancia
char vetor[8];                                  // Inicializa o vetor de char que conterá a distancia a ser enviada (USART)

// Define as máscaras que permitem aceder aos pinos associados ao Sensor
#define SR_Trigger          PINB1
#define SR_Echo             PINB0
#define ECHO                (1<<SR_Echo)

void trigger(){                                 // Função que ativa o trigger
    PORTB |= 1<<SR_Trigger;                     // Ativa o Trigger
    while (cont_ovf0 != 60);                    // Aguarda até que o nº de Overflows seja 60 (60ms)
    PORTB &= ~(1<<SR_Trigger);                  // Desativa o Trigger
}
```

Figura 13 - Código: Variáveis, Mascaras e trigger()

A função ***HCSR04_Init***, tem como propósito fazer as configurações iniciais do sensor. Primeiramente, coloca o registo de contagem a 0, aciona o *Trigger* através da função referida anteriormente e, em seguida, como referido na explicação anterior, o sistema espera até que o *Echo* do sensor rececione informação para, em seguida, iniciar a contagem no **Timer1** e interromper a mesma quando o *Echo* deixar de receber ondas ultrassónicas. A distância foi obtida através dos seguintes cálculos:

$$TCNT1 = \frac{t}{4\mu s} \Leftrightarrow t = TCNT1 \cdot (4 \cdot 10^{-6}) \qquad Distance\ (mm) = 340000 \cdot \frac{t}{2}$$

$$Distance\ (mm) = 340000 \cdot \frac{TCNT1 \cdot (4 \cdot 10^{-6})}{2} = TCNT1 \cdot 0.68$$

Essa mesma distância foi depois convertida para um vetor através da função ***dtostrf()***:

dtostrf(valor a converter, tamanho, nº casas decimais, vetor onde se vai armazenar)

```
void HCSR04_Init(){                          // Função que inicializa o Sensor HC-SR04
    TCNT1 = 0;                               // Coloca os registos de contagem a 0
    trigger();                               // Aciona o Trigger do Sensor
    while((PINB & ECHO) == 0);               // Enquanto o Echo não estiver a receber ondas
    TCCR1B = (1<<CS10);                      // Inicia o Timer1
    while((PINB & ECHO) != 0);               // Enquanto o Echo estiver a receber ondas
    TCCR1B = 0;                              // Interrompe o Timer1
    distance = TCNT1*0.68;                   // Variavel que guarda a distancia
    dtostrf(distance, 3, 0, vetor);          // Transforma a variavel distance num vetor

}
```

Figura 14 - Código: HCSR04_Init

A função ***enviaDistancia***, recebe o nº de caracteres a transmitir e transmite a distância em forma de vetor, **char** a **char**, através da função de transmissão da USART.

```
void enviaDistancia(char numChar){           // Função que envia a distancia para o USART
    uint8_t i;                               // Variavel para posição no vetor
    char mm[4] = " mm.";                     // Unidade da distancia
    char distancia[13] = "\n\nDistancia:  ";  // Distancia:
    for(i = 0; i<sizeof(distancia); i++){    // Percorre as posições dos chars do vetor
        USART_Transmit(distancia[i]);        // Transmite a string char a char
    }
    for(i = 0; i<numChar; i++){              // Percorre as posições dos chars do vetor
        USART_Transmit(vetor[i]);            // Transmite a distancia contida no vetor
    }
    for(i = 0; i<sizeof(mm); i++){           // Percorre as posições dos chars do vetor
        USART_Transmit(mm[i]);               // Transmite a string char a char
    }
}
```

Figura 15 - Código: enviaDistancia

Buzzer

Código 6

Como alternativa aos LEDs, o **Buzzer** também informa o condutor da proximidade ao obstáculo através do som que produz com mais frequência consoante essa distância. O **Buzzer** que foi utilizado é um Buzzer ativo que apenas precisa alimentação para produzir o som.

Relativamente ao código do Buzzer, foi necessário criar uma função para que ele produzisse o som, ***buzzer***, uma função que recebesse o intervalo de tempo pretendido entre "apitos", ***buzzerIntervalo***, e as funções que permitem alterar a frequência com que o Buzzer "apita", associadas a cada LED, ***buzzerVermelho, buzzerAmarelo***.

Após criada a variável que representa o nº de Overflows desejado, **nmr_ovf**, a função ***buzzer*** começa por iniciar o **Timer0**, liga o **Buzzer** que está conectado no PORTC e enquanto o contador de Overflows não for igual a 100, ou seja, enquanto não passarem 100ms, o Buzzer mantem-se ligado, caso contrário, desliga-se.

A função ***buzzerIntervalo*** recebe a variável **nmr_ovf**, ou seja, o número de overflows pretendido, traduzindo-se para nº de milissegundos pretendidos e "corre" a função anterior ***buzze*** no intervalo pretendido.

As funções ***buzzerVermelho, buzzerAmarelo*** diferem apenas na frequência com que é executada a função ***buzzer***, utilizando a função ***buzzerIntervalo*** para escolher o intervalo de tempo pretendido.

```
volatile unsigned char nmr_ovf;                    // Inicializa a variavel de nº de Overflows desejado

void buzzer(){                                     // Função que vai produzir o som do Buzzer
    TCNT0 = 0;                                     // Reset no Contador
    PORTC = 0xff;                                  // Liga o buzzer conectado no PORTC
    while(cont_ovf0 != 100);                       // Aguarda até que o nº de Overflows seja 100
    PORTC = 0x00;                                  // Desliga o buzzer conectado no PORTC
}

volatile unsigned char buzzerIntervalo(nmr_ovf){   // Função que recebe o intervalo de tempo pretendido entre "apitos"
    TCNT0 = 0;                                     // Reset no Contador
    while(cont_ovf0 != nmr_ovf);                   // Aguarda até que o nº de Overflows seja igual ao nº de Overflows pretendido
    buzzer();                                      // Função que vai produzir o som do Buzzer
}

void buzzerVermelho(){                             // Função do Comportamento do Buzzer relativamente ao LED Vermelho
    // Buzzer apita 3 vezes em intervalos de 10ms
    buzzerIntervalo(10);
    buzzerIntervalo(10);
    buzzerIntervalo(10);
}

void buzzerAmarelo(){                              // Função do Comportamento do Buzzer relativamente ao LED Amarelo
    // Buzzer apita 2 vezes em intervalos de 50ms
    buzzerIntervalo(50);
    buzzerIntervalo(50);
}
```

Figura 16 - Código: Buzzer

Sensor de Estacionamento ***Código 7***

Criadas todas as funções necessárias ao funcionamento do **Sensor de Estacionamento**, falta desenhar o seu comportamento.

Começou-se por criar a variável inteira que irá conter a distância medida e se a distancia for maior que 0 (de forma a prevenir pequenos erros do sensor HC-SR04), deve enviar a distância que está a medir através da função ***enviaDistancia*** e em seguida, se a distancia medida estiver entre 191 e 260, os LEDs Vermelho e Amarelo desligam-se e liga-se o LED Verde. Se a distancia estiver entre 100 e 190mm, os LEDs Vermelho e Verde desligam-se, liga-se o LED Amarelo e o Buzzer comporta-se conforme a função ***buzzerAmarelo*****.** Por fim, se a distancia for menor que 100 ou maior do que 330mm, os LEDs Amarelo e Verde desligam-se, liga-se o LED Vermelho e o Buzzer comporta-se conforme a função ***buzzerVermelho*****.** Caso o Sensor não detete obstáculos, os LEDs desligam-se.

Por ser um **Sensor de Estacionamento** e por ser apenas uma pequena representação do mesmo, só é necessário medir distâncias pequenas, apesar do Sensor HC-SR04 ter capacidade de medir distâncias maiores.

```
//=============================================================================//
//                          SENSOR DE ESTACIONAMENTO                           //
//=============================================================================//
int distancia;                                  // Variavel inteira de distancia

void sensorEstacionamento(){
    if (distancia > 0)                          // Se a distancia for maior que 0
    {
        enviaDistancia(3);
        if (BETWEEN(distancia,191,260))         // Se a distancia for entre 191 e 260mm
        {
            PORTD &= ~(LED_RED|LED_YELLOW);     // Desliga os LEDs Vermelho e Amarelo
            PORTD |= LED_GREEN;                 // Liga o LED Verde
        }
        if (BETWEEN(distancia,100,190))         // Se a distancia for entre 100 e 190mm
        {
            PORTD &= ~(LED_RED|LED_GREEN);      // Desliga os LEDs Vermelho e Verde
            PORTD |= LED_YELLOW;                // Liga o LED Amarelo
            buzzerAmarelo();                    // Comportamento do Buzzer relativo ao LED Amarelo
        }
        if (distancia < 100 || distancia > 330) // Se a distancia for menor que 100mm ou maior que 300mm
        {
            PORTD &= ~(LED_YELLOW|LED_GREEN);   // Desliga os LEDs Amarelo e Verde
            PORTD |= LED_RED;                   // Liga o LED Vermelho
            buzzerVermelho();                   // Comportamento do Buzzer relativo ao LED Vermelho
        }
        }else {                                 // Caso a distancia não esteja no intervalo pretendido
        PORTD &= ~(LED_RED|LED_YELLOW|LED_GREEN); // Desliga todos os LEDs
    }
}
```

Figura 17 - Código: Sensor de Estacionamento

Main

Código 8

Para concluir, o ***main()*** será o local onde serão executadas as funções desenvolvidas anteriormente. Para uma melhor organização, o ***main()*** tem também a mesma ordem de tópicos que o restante código.

Começa-se por inicializar a rotina de configuração da **USART** e, em seguida, coloca-se os **LEDs** como output no DDRD, inicializa-se o **Timer1** e o **Timer0**, coloca-se o pino *Trigger* como *Input* e o PORTC onde o **Buzzer** se situa como *Output*.

Entrando agora no *loop* ***while(1)***, é enviado um menu em forma de string, para que, posteriormente, seja iniciado o sistema no software CoolTerm e é dado à variável tecla o valor da tecla pressionada através da utilização da função ***USART_Receive()***. Caso se tenha pressionado a tecla 's', o sistema fará 15 medições de distância, iniciando a função ***HCSR04_Init()*** e transformando o vetor onde a distância foi anteriormente guardada numa variável inteira, a variável **int distancia**, a qual será utilizada quando a função ***sensorEstacionamento()*** for, em seguida, iniciada. Após as 15 medições, será mostrado, de novo, o menu inicial. Caso a tecla pressionada tenha sido 'n', é enviada uma string correspondente ao desligar do sistema.

```
//=============================================================================//
//                                    MAIN                                     //
//=============================================================================//
int main(){
    USART_Init(MYUBRR);                                 // Inicialização da rotina de configuração da USART0

    DDRD = 0x70;                                        // Coloca os LEDs como Output

    timer0();                                           // Inicia o Timer0
    timer1();                                           // Inicia o Timer1

    DDRB = (1<<SR_Trigger);                             // Coloca o pino Trigger como Input

    DDRC = 0xff;                                        // Configura a PORTC como Output

    while(1){
        enviaString(menu);                              // Envia para o USART o menu
        tecla = USART_Receive();                        // A USART recebe a tecla pressionada e dá esse valor a tecla
        switch(tecla){
            case 's':                                   // Caso seja pressionada a tecla s
                for (int i=0; i<15; i++){               // Faz 15 medições
                    HCSR04_Init();                      // Função que inicializa o Sensor HC-SR04
                    distancia = char2int(vetor,3);      // Inicializa a variavel distancia com o valor inteiro de vetor
                    sensorEstacionamento();             // Inicia o comportamento do Sensor de Estacionamento
                }
            case 'n':                                   // Caso seja pressionada a tecla n
            enviaString(nao);                           // Envia para a USART a mensagem nao
            break;
        }
    }
}
```

Figura 18 - Código: Main

Terminando, testou-se o sistema com o código implementado e confirmou-se o seu correto funcionamento.

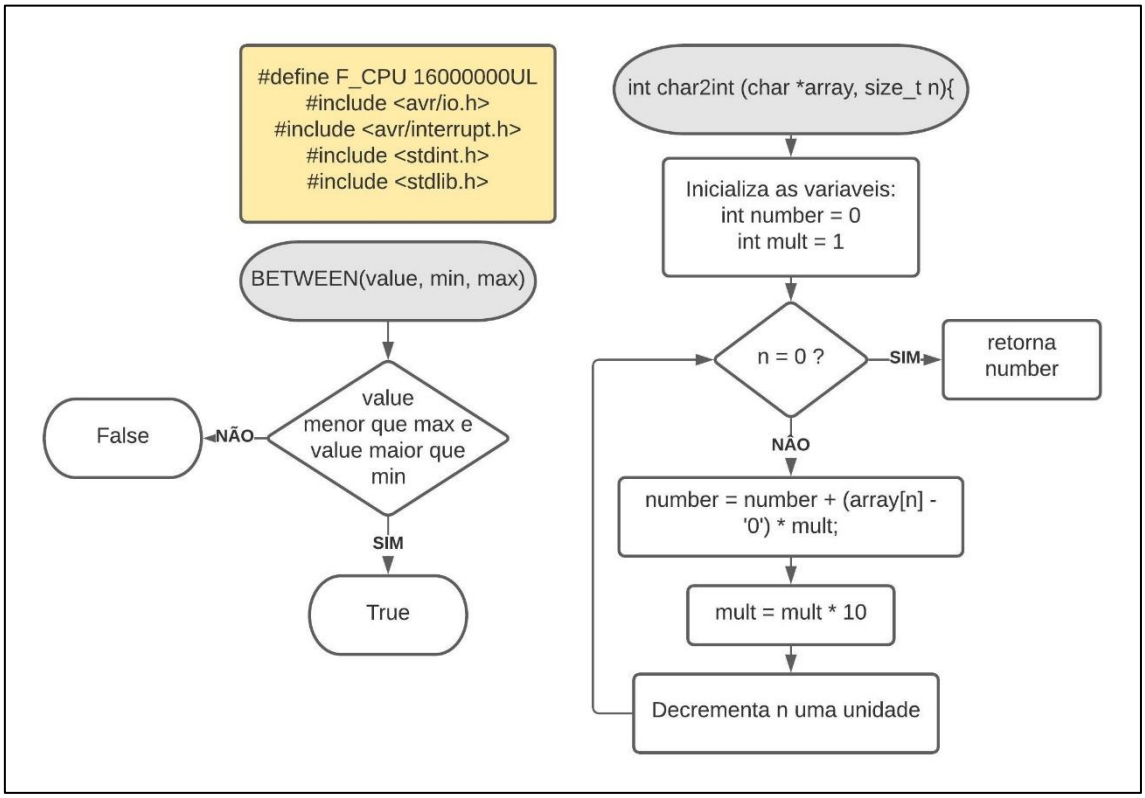
Todo o código e o fluxograma estão representados nas páginas seguintes.

Conclusão

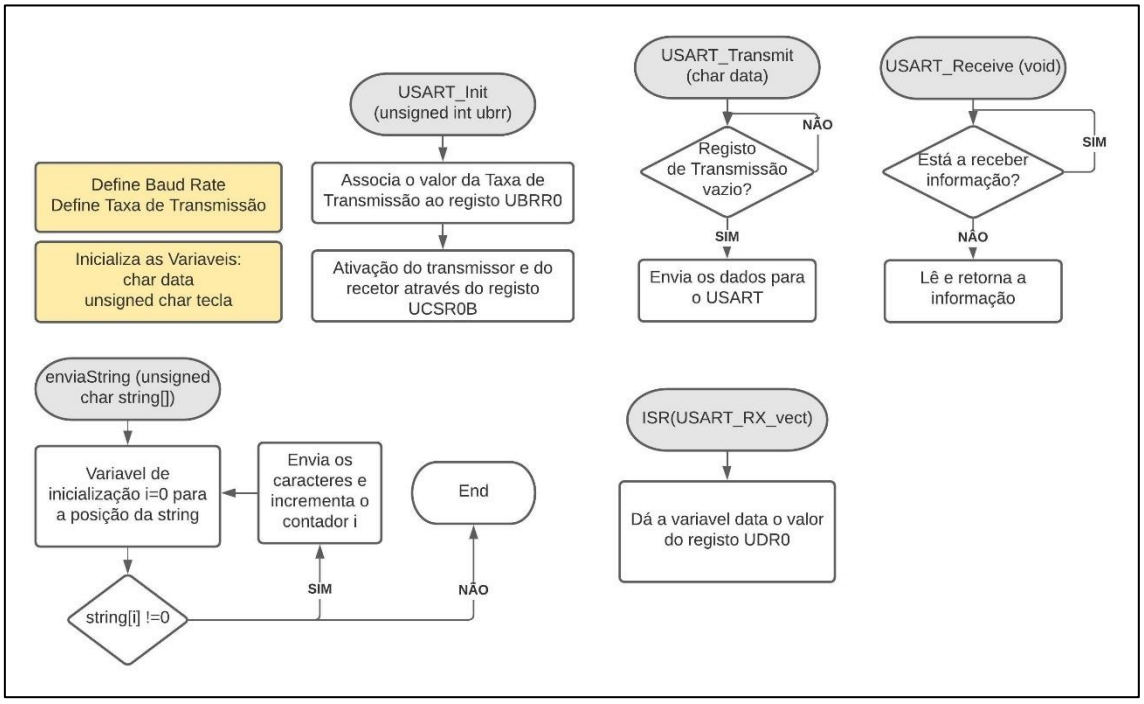
Concluindo, a utilização do Sensor HC-SR04 permitiu desenvolver um Sensor de Estacionamento capaz de simular, mesmo que em ponto pequeno, o estacionamento ou aproximação de um veículo ao obstáculo. Relativamente à velocidade e a precisão do sensor são, inevitavelmente, pontos a melhorar, mas que ainda assim permitiram, no conjunto de todo o projeto, perceber o funcionamento de um sensor de estacionamento comum que seja aplicado por fabricantes de automóveis.

Fluxogramas

Definições e Bibliotecas

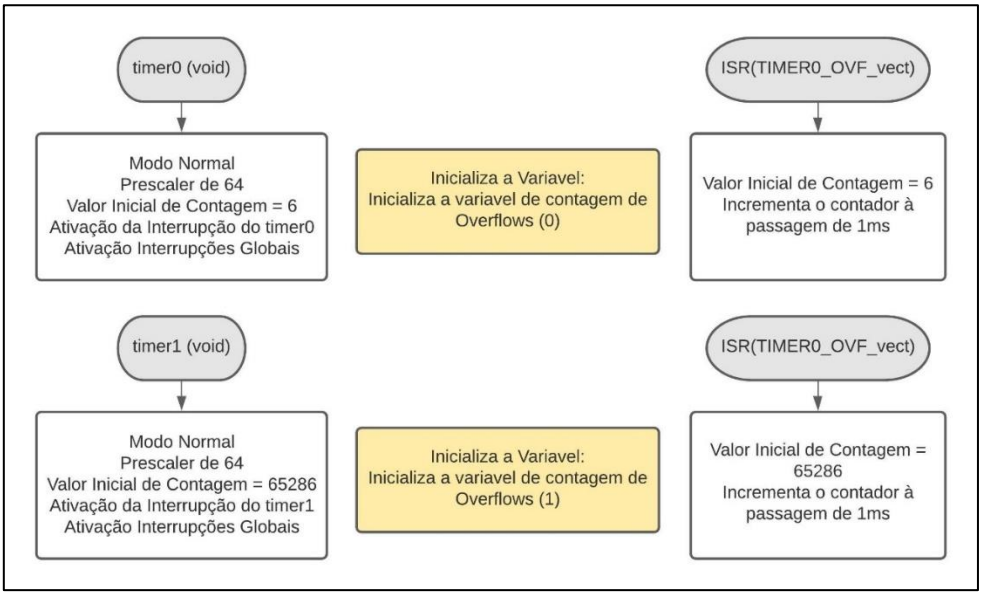


Porta Série

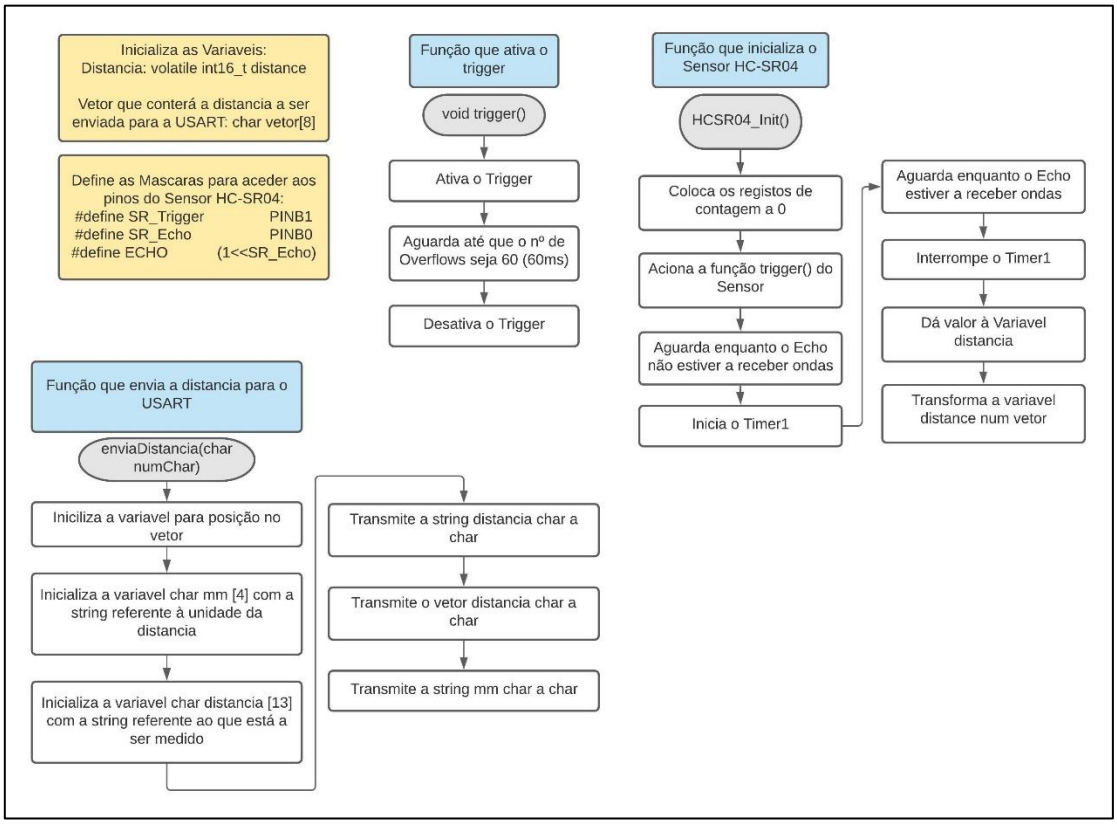


LEDs

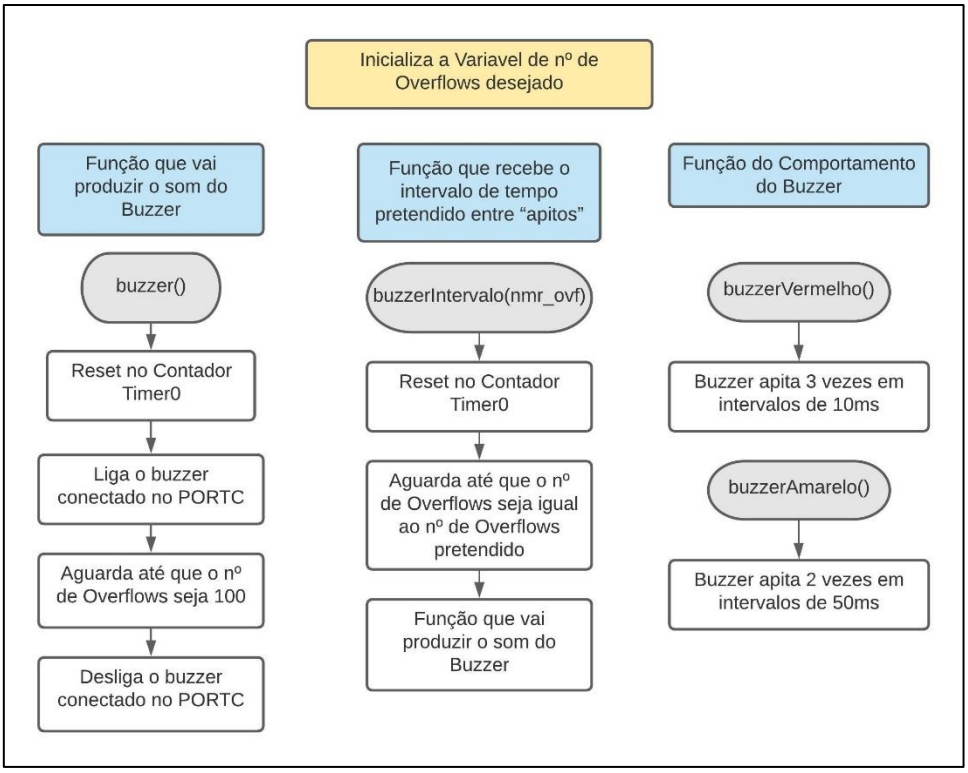
Timers



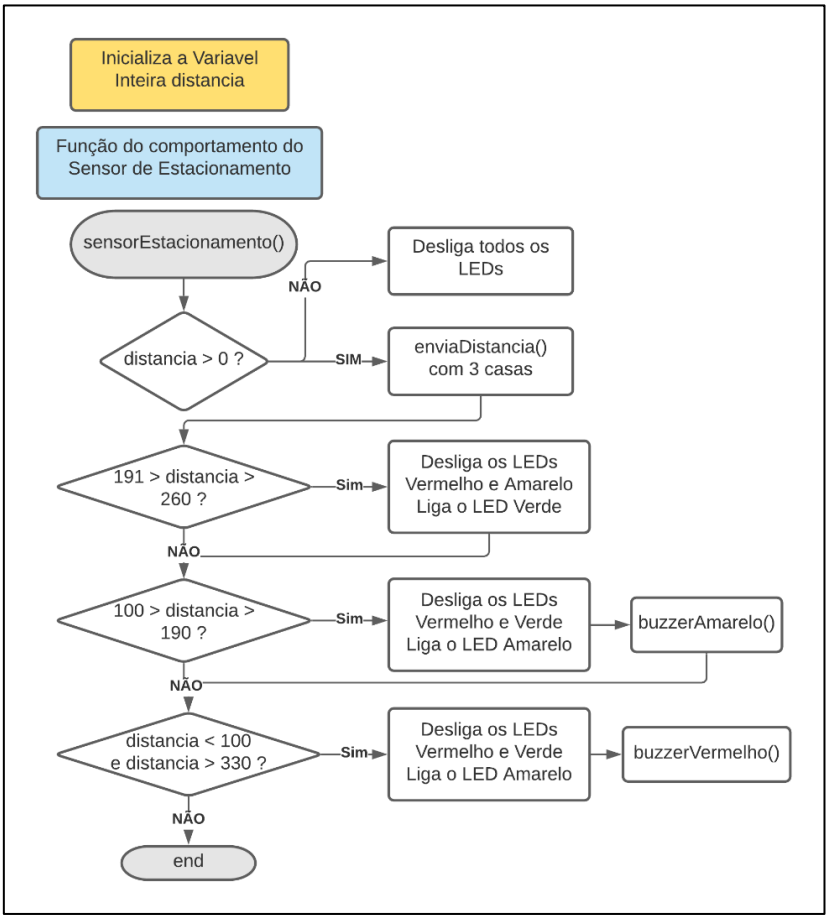
Sensor HC-SR04



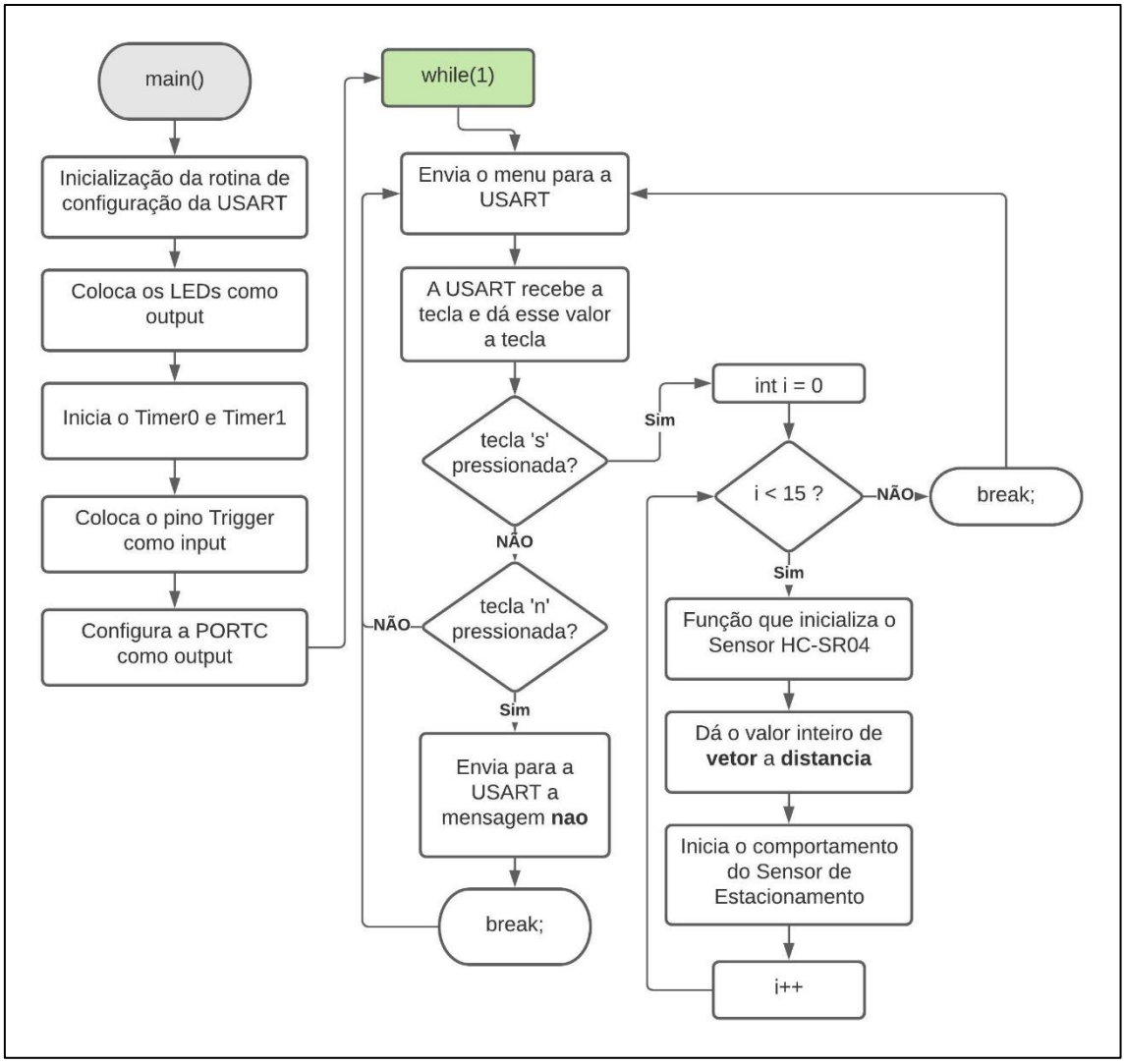
Buzzer



Sensor de Estacionamento



Main



Anexo

Código 1 – Definições e Bibliotecas

```
//=========================================================================//
//                                                                         //
//                                SUMÁRIO                                  //
//                                                                         //
//                  Definições e Bibliotecas                               //
//                  Porta Serie (USART)                                    //
//                  LEDs                                                   //
//                  Timers                                                 //
//                  Sensor HC-SR04                                         //
//                  Buzzer                                                 //
//                  Sensor de Estacionamento                               //
//                                                                         //
//=========================================================================//

// 16 MHz clock speed
#define F_CPU 16000000UL
// Carrega a biblioteca que permite utilizar as funções de input e output
#include <avr/io.h>
```

```
// Carrega a biblioteca que permite utilizar as funções de interrupção
#include <avr/interrupt.h>
// Carrega a biblioteca que permite utilizar inteiros com larguras especificadas
#include <stdint.h>
// Carrega a biblioteca que permite utilizar inteiros com larguras especificadas
#include <stdlib.h>

// Operação de "Entre"
#define BETWEEN(value, min, max) (value < max && value > min)

// Função necessária para transformar um array de char numa variavel inteira
int char2int (char *array, size_t n){
	// Inteiro que vai retornar
	int number = 0;
	// Multiplicador
	int mult = 1;
	// por cada caracter no vetor
	while (n--){
		// Numero igual à conversão para inteiro multiplicado por mult, que
		vai definir a posição do numero
		number += (array[n] - '0') * mult;
		mult *= 10;
	}
	return number;
}
```

Código 2 – Porta Série

```
//============================================================================ //
//                                PORTA SERIE                                   //
//============================================================================ //
// Define os parâmetros necessários para a utilização da Porta Serie
#define BAUD 9600                                   // Baud Rate
#define MYUBRR F_CPU/16/BAUD-1                      // Taxa de Transmissão
char data;                                          // Inicializa variável data
unsigned char tecla;                                // Inicializa a variável tecla

// Funções necessárias à utilização do USART
void USART_Init (unsigned int ubrr){     // Função para Inicialização da USART
	UBRR0 = ubrr;                        // Ajusta a Taxa de Transmissão
	UCSR0B = (1<<RXEN0)|(1<<TXEN0);      // Ativa o transmissor e o recetor
}
void USART_Transmit (char data){         // Função de Transmissão de Dados
	while(!(UCSR0A & (1<<UDRE0)));       // Operação while para garantir que o
                                         registo de transmissão está vazio
	UDR0 = (data++);                     // Serve para colocar os dados e enviar
                                         para o USART
}
char USART_Receive (void){               // Função de Receção de Dados
	while(!(UCSR0A & (1<<RXC0)));        // Enquanto não detetar o stop bit
	return UDR0;                         // Lê o registo e retorna
}
void enviaString(unsigned char string[]){// Função para enviar uma string
	unsigned int i=0;                    // Variável de posição no string
	while(string[i] != 0){               // Enquanto houver caracteres
		USART_Transmit(string[i]); // Envia os caracteres
		i++;                             // Incrementa i
	}
}
```

```
ISR(USART_RX_vect){                 // Interrupção da USART
       data = UDR0;                 // Dá a variável data o valor do registo UDR0
}

char *menu = "Iniciar Sensor de Estacionamento?\n\nSim - Pressionar a tecla
's'.\nNão - Pressionar a tecla 'n'.\n\nOpção: ";
char *nao = "\nBoa Viagem!\n";
```

Código 3 – LEDs

```
//=========================================================================== //
//                                LED's                                      //
//=========================================================================== //
// Define as máscaras que permitem aceder aos pinos associados aos LEDs
#define LED_RED             0x10
#define LED_GREEN           0x20
#define LED_YELLOW          0x40
```

Código 4 – Timers

```
//=========================================================================== //
//                                TIMERS                                     //
//=========================================================================== //
void timer0(void) {
       TCCR0A = (0<<WGM00);                 // Modo Normal
       TCCR0B = (0b011 << CS00);            // Prescaler de 64
       TCNT0 = 256-250;                     // Valor inicial de contagem
       TIMSK0 = (1<<TOIE0);                 // Ativação da Interrupção do timer0
       sei();                               // Ativação Interrupções Globais
}
volatile unsigned char cont_ovf0 = 0;       // Inicializa a variável de contagem de
                                            Overflows

ISR(TIMER0_OVF_vect){               // Interrupção Timer0
       TCNT0 = 6;                   // Valor inicial de contagem
       cont_ovf0++;                 // Incrementa o contador à passagem de 1ms
}
void timer1(void){          // Função Timer1 para uso das interrupções
       TCCR1A = 0x00;
       // WGM13:0 = 0000 para modo normal;
       // COM1A1:0 = 0 e COM1B1:0 = 0 no modo normal

       TCCR1B = 0x03;
       // WGM13:0 = 0000 para modo normal;
       // CS12:0 = 011 para relógio interno com divisor por 64;
       // ICNC1 = 0 e ICES1 = 0 no modo normal

       TCCR1C = 0x00;
       // FOC1A = 0 e FOC1B = 0 no modo normal

       TIMSK1 = (1 << TOIE1);
       // Ativa as interrupções do Timer1

       TCNT1 = 65286;
       // Valor inicial de contagem = 2^16-250 = 65286

       sei();
       // Ativação Interrupções Globais
}

volatile unsigned char cont_ovf1 = 0;       // Inicializa a variável de contagem de
                                            Overflows
```

```
ISR(TIMER1_OVF_vect){              // Interrupção Timer1
      TCNT1 = 65286;               // Valor inicial de contagem
      cont_ovf1++;                 // Incrementa o contador à passagem de 1ms
}
```

Código 5 – Sensor HC-SR04

```
//====================================================================== //
//                          SENSOR HC-SR04                              //
//====================================================================== //
volatile int16_t distance;         // Inicializa a variável de distancia
char vetor[8];                     // Inicializa o vetor de char que conterá a
                                   distancia a ser enviada (USART)

// Define as máscaras que permitem aceder aos pinos associados ao Sensor
#define SR_Trigger                 PINB1
#define SR_Echo                    PINB0
#define ECHO                       (1<<SR_Echo)

void trigger(){                    // Função que ativa o trigger
      PORTB |= 1<<SR_Trigger;      // Ativa o Trigger
      while (cont_ovf0 != 60);     // Aguarda até que o nº de Overflows seja 60
                                   (60ms)
      PORTB &= ~(1<<SR_Trigger); // Desativa o Trigger
}

void HCSR04_Init(){                // Função que inicializa o Sensor HC-SR04
      TCNT1 = 0;                   // Coloca os registos de contagem a 0
      trigger();                   // Aciona o Trigger do Sensor
      while((PINB & ECHO) == 0); // Enquanto o Echo não estiver a receber ondas
      TCCR1B = (1<<CS10);          // Inicia o Timer1
      while((PINB & ECHO) != 0); // Enquanto o Echo estiver a receber ondas
      TCCR1B = 0;                  // Interrompe o Timer1
      distance = TCNT1*0.68;       // Variável que guarda a distancia
      dtostrf(distance, 3, 0, vetor);// Transforma a variável distance num vetor

}
void enviaDistancia(char numChar){// Função que envia a distância para o USART
      uint8_t i;                   // Variável para posição no vetor
      char mm[4] = " mm.";         // Unidade da distancia
      char distancia[13] = "\n\nDistancia:  "; // Distancia:
      for(i = 0; i<sizeof(distancia); i++){    // Percorre as posições dos chars
                                               do vetor
            USART_Transmit(distancia[i]);      // Transmite a string char a char
      }
      for(i = 0; i<numChar; i++){              // Percorre as posições dos chars
                                               do vetor
            USART_Transmit(vetor[i]);          // Transmite a distancia contida
                                               no vetor
      }
      for(i = 0; i<sizeof(mm); i++){           // Percorre as posições dos chars
                                               do vetor
            USART_Transmit(mm[i]);             // Transmite a string char a char
      }
}
```

Código 6 – Buzzer

```
//====================================================================== //
//                              BUZZER                                  //
//====================================================================== //
volatile unsigned char nmr_ovf;    // Inicializa a variável de nº de Overflows
                                   desejado
```

```
void buzzer(){                    // Função que vai produzir o som do Buzzer
	TCNT0 = 0;                    // Reset no Contador
	PORTC = 0xff;                 // Liga o buzzer conectado no PORTC

	while(cont_ovf0 != 100);      // Aguarda até que o nº de Overflows seja 100
	PORTC = 0x00;                 // Desliga o buzzer conectado no PORTC
}

// Função que recebe o intervalo de tempo pretendido entre “apitos”
volatile unsigned char buzzerIntervalo(nmr_ovf){
	TCNT0 = 0;                          // Reset no Contador
	while(cont_ovf0 != nmr_ovf);        // Aguarda até que o nº de Overflows
                                        seja igual ao nº de Overflows pretendido
	buzzer();                           // Função que vai produzir o som do
                                        Buzzer
}

void buzzerVermelho(){      // Função do Comportamento do Buzzer relativamente ao
                            LED Vermelho
	// Buzzer apita 3 vezes em intervalos de 10ms
	buzzerIntervalo(10);
	buzzerIntervalo(10);
	buzzerIntervalo(10);
}

void buzzerAmarelo(){       // Função do Comportamento do Buzzer relativamente ao
                            LED Amarelo
	// Buzzer apita 2 vezes em intervalos de 50ms
	buzzerIntervalo(50);
	buzzerIntervalo(50);
}
```

Código 7 – Sensor de Estacionamento

```
//=========================================================================== //
//                          SENSOR DE ESTACIONAMENTO                         //
//=========================================================================== //
int distancia;                    // Variável inteira de distância

void sensorEstacionamento(){
	if (distancia > 0)            // Se a distancia for maior que 0
	{
		enviaDistancia(3);  // Envia a distancia para a USART
		// Se a distancia for entre 191 e 260mm
		if (BETWEEN(distancia,191,260))
		{
			PORTD &= ~(LED_RED|LED_YELLOW);   // Desliga os LEDs Vermelho
                                                 e Amarelo
			PORTD |= LED_GREEN;               // Liga o LED Verde
		}
		// Se a distancia for entre 100 e 190mm
		if (BETWEEN(distancia,100,190))
		{
			PORTD &= ~(LED_RED|LED_GREEN);    // Desliga os LEDs Vermelho
                                                 e Verde
			PORTD |= LED_YELLOW;              // Liga o LED Amarelo
			buzzerAmarelo();     // Comportamento do Buzzer relativo ao
                                    LED Amarelo
		}
		// Se a distancia for menor que 100mm ou maior que 300mm
		if (distancia < 100 || distancia > 330)
		{
			PORTD &= ~(LED_YELLOW|LED_GREEN); // Desliga os LEDs Amarelo
                                                 e Verde
```

```
                PORTD |= LED_RED;                // Liga o LED Vermelho
                buzzerVermelho();    // Comportamento do Buzzer relativo ao
                                     LED Vermelho
            }
            }else {       // Caso a distância não esteja no intervalo pretendido
            PORTD &= ~(LED_RED|LED_YELLOW|LED_GREEN);       // Desliga todos os
                                                            LEDs
        }
}
```

Código 8 – Main

```
//=========================================================================== //
//                              MAIN                                          //
//=========================================================================== //
int main(){
        USART_Init(MYUBRR); // Inicialização da rotina de configuração da USART0

        DDRD = 0x70;        // Coloca os LEDs como Output

        timer0();           // Inicia o Timer0
        timer1();           // Inicia o Timer1

        DDRB = (1<<SR_Trigger);     // Coloca o pino Trigger como Input

        DDRC = 0xff;                // Configura a PORTC como Output

        while(1){
                enviaString(menu);          // Envia para o USART o menu
                tecla = USART_Receive();    // A USART recebe a tecla pressionada e
                                            dá esse valor a tecla

                switch(tecla){
                        case 's':           // Caso seja pressionada a tecla s
                                for (int i=0; i<15; i++){  // Faz 15 medições
                                        HCSR04_Init();      // Função que inicializa o
                                                            Sensor HC-SR04
                                        distancia = char2int(vetor,3);
        // Inicializa a variável distancia com o valor inteiro de vetor
                                        sensorEstacionamento();
        // Inicia o comportamento do Sensor de Estacionamento
                                }
                        break;
                        case 'n':           // Caso seja pressionada a tecla n
                        enviaString(nao);   // Envia para a USART a mensagem nao
                        break;
                }
        }
}
```