Manual Técnico - Assistente de Direção

Engenharia de Controle e Automação
Universidade Federal de Santa Catarina
Arquitetura e Programação de Sistemas Microcontrolados
Professor: Werner Kraus Junior

Fernando Battisti
Iago de Oliveira Silvestre
Igor Assis Rocha Yamamoto

.: developed by AUTOFIGI :.

01/07/2015

Sumário

Capítulo 1

Introdução

Neste manual serão descrito os aspectos técnicos do projeto implementado no microprocessador ATMEGA328 da ATMEL, idealizado e realizado pela equipe AUTOFIGI. Serão discutidos aspectos tais quais como:

- Funcionamento do sistema

- Estruturamento do código

- Descritivo do código

- Interação com dispositivos externos.

- Configuraçções de registradores

O projeto de assistente de direção tem em vista evitar acidentes e danos na estrutura do automóvel.

Capítulo 2

Funcionamento

Obs: Por "Sinal de alerta"(muito citado no manual) entende-se como o aviso que o sistema dá ao usuário quando o sensor se aproxima de um obstáculo, neste caso o aviso é um pulso no "Buzzer"e no o "Led vermelho".

O usuário irá interagir com o sistema através de dois botões: (1) e (2)

Na figura 2.1 pode-se ver os itens que compõem o projeto.

Segue abaixo a lista dos itens:

(1) Botão ON/OFF: Usado para ligar ou desligar o sistema.

(2) Botão de Configuração: Usado para colocar as configurações pessoais do motorista.

(3) Led Verde: Sinaliza se o sistema está ligado ou desligado.

(4) Led Vermelho: Pisca na mesma frequência do buzzer, conforme o veículo se aproxima de um obstáculo.

(5) Buzzer: É o emissor do "bip".

(6) Sensor Ultrassônico: Detecta a distância entre o carro e o obstáculo.

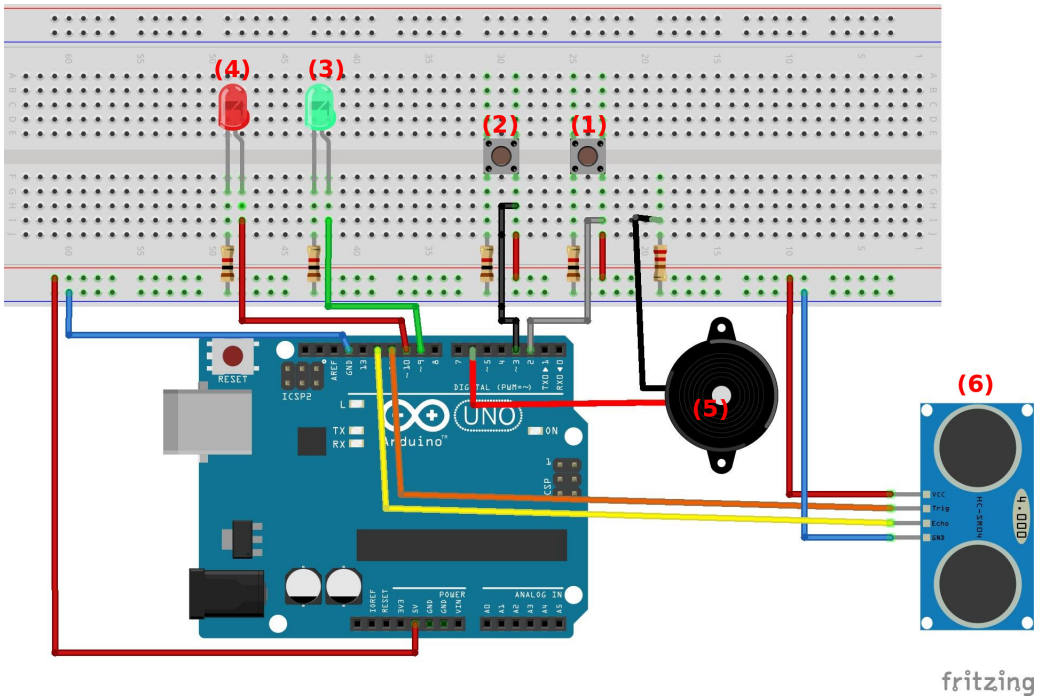

Figura 2.1:

Para facilitar o entendimento, foi feito um diagrama (Figura 2.2) mostrando os principais pontos do funcionamento.

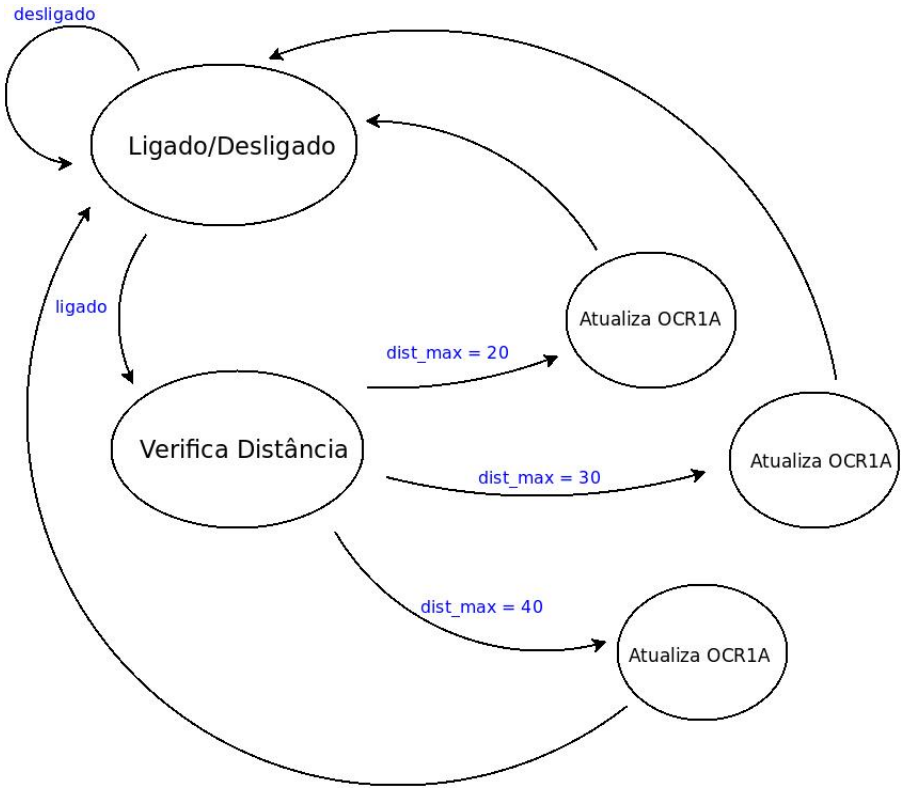

Figura 2.2:

Quando o sistema está desligado ele permanece no estado "Ligado/Desligado". Nesse estado o **TI-**

MER1 (usado para ajustar o intervalo entre os sinais de alerta) é desabilitado e a **INT1** (usada para alterar as configurações pessoais) é habilitada.

Quando o sistema está ligado, o programa fica continuamente atualizando o valor do registrador *OCR1A* (esse registrador contém o valor top, que é continuamente comparado com o valor de TCNT1). Ao alterar o valor do registrador *OCR1A*, temos diferentes intervalos de tempo para o "Sinal de alerta", podendo assim sinalizar quando o sensor está se aproximando ou afastando do obstáculo.

No projeto em questão, foi decidido que a distância máxima na qual o sistema emitiria os "Sinais de alerta"pudesse ser alterada. Essa distância máxima pode ser modificada de duas formas:

- Ao iniciar o sistema: Nesse momento o sistema pede para que o usuário escolha o seu nível de instrução, podendo este ser: Iniciante (distância máxima = 20cm), Intermediário (distância máxima = 30cm) ou Profissional (distância máxima = 40cm).

- Quando o sistema estiver no estado "Ligado/Desligado": Aqui, para modificar a distância máxima, o programador deverá pressionar o Botão (2). Ao apertar o Botao (2) é gerada a **INT1**, na qual ele poderá selecionar o nível de instrução do usuário, podendo este ser: Iniciante (distância máxima = 20cm), Intermediário (distância máxima = 30cm) ou Profissional (distância máxima = 40cm).

Para a atualização do valor do registrador *OCR1A*, dependendo do nível de instrução do motorista, o valor de "top"(variável que armazena o valor a ser colocado em OCR1A) é calculado através de funções logarítmicas diferentes. Na Figura 2.3 é mostrado um gráfico da Distância entre sensor e obstáculo X Frequência entre "Sinais de alerta".

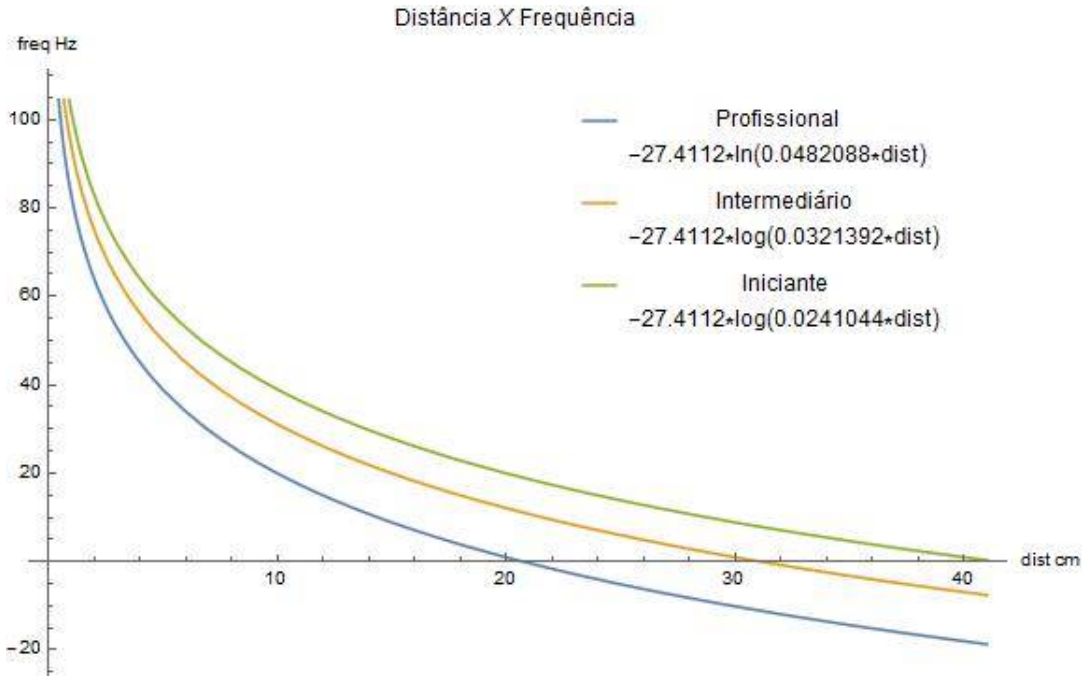

Figura 2.3:

Quando o motorista for estacionar seu veículo, ele deverá iniciar o monitoramento apertando o Botão ON/OFF (1), e assim a **INT0** (Usada para ligar ou desligar o sistema) é gerada. Então, o usuário

poderá notar:

- Um pequeno áudio de inicialização.

- Um LED de cor verde (3) aceso, indicando que o sistema está ligado.

- Ver uma mensagem na Serial dizendo:

  Bem-vindo ao seu assistente de condução!

Agora o motorista poderá os "Sinais de alerta"com a frequência variando a medida que o veículo se aproxima de um obstáculo. Além disso, o condutor poderá visualizar a distância entre o carro e o obstáculo pela Serial.

Após o carro ser estacionado, o sistema deverá ser desligado pressionando o Botão ON/OFF (1) novamente.

Capítulo 3

Estruturamento do Código Fonte

Logo no inicio do código, são encontradas alguns comentários sobre a equipe que realizou o projeto, além da data quando o projeto foi finalizado entre outras informações. Após essas informações, temos a declaração de variáveis e funções que são usadas no projeto. (Ver figura 3.1)

```
//-------------------------------------------------------------------------------------------------------//
//                                                                                                       //
// Projeto da disciplina de Arquitetura e Programacao de Sistemas Microcontrolados - DAS5332             //
//                                                                                                       //
// Professor: Werner Kraus Junior                                                                        //
//                                                                                                       //
// Equipe: Fernando Battisti                                                                             //
//         Iago de Oliveira Silvestre                                                                    //
//         Igor Assis Rocha Yamamoto                                                                     //
//                                                                                                       //
// Julho de 2015                                                                                         //
//                                                                                                       //
//-------------------------------------------------------------------------------------------------------//

#include <avr/interrupt.h>
#include "pitches.h"

const byte TONE_PIN = 6;                    // Buzzer no pino 6
const byte ECHO_PIN = 12;                   // Echo do sensor ultrassonico no pino 12
const byte TRIGGER_PIN = 11;                // Trigger do sensor ultrassonico no pino 11
const byte LED_PISCA = 10;                  // Led que pisca de acordo com a frequencia de 'Sinais de alerta' no pino 10
const byte LED_ON  = 9;                     // Led que sinaliza ON/OFF no pino 9
const unsigned int CLK_TIMER = 15625;
int max_distance = 40;                      // Declaracao e inicializacao da variavel max_distance
byte on = LOW;                              // Declaracao e inicializacao da variavel on
int beep = NOTE_C5;                         // Declaracao e inicializacao da variavel beep
unsigned int top = 15625;                   // Declaracao e inicializacao da variavel top
void sing();                                // Declaracao da funcao sing do tipo void
void configInt0();                          // Declaracao da funcao configInt0 do tipo void
void configInt1();                          // Declaracao da funcao configInt1 do tipo void
void configTimer1();                        // Declaracao da funcao configTimer1 do tipo void
void configpersonal();                      // Declaracao da funcao configpersonal do tipo void
void buzz(int targetPin, long frequency, long len); // Declaracao da funcao buzz do tipo void
float dist_measure();                       // Declaracao da funcao dist_measure do tipo float
```

Figura 3.1:

Após as declarações iniciais, encontramos duas funções que devem estar obrigatoriamente presente em qualquer código Arduino (ver Figura 3.2), são elas:

- void setup(): É executada somente uma vez quando a placa é energizada ou cada vez que a placa é "resetada". Nela configuramos a **USART0**, **INT0**, **INT1** e **TIMER1**.

- void loop(): Essa função se repete continuamente durante a execução do programa. É nela que atualizamos o valor da variável "top"(variável que terá seu valor atribuido a OCR1A).

```
void setup() {
}
void loop() {
}
```

Figura 3.2:

Depois das funções obrigatórias, encontramos 3 funções que são usadas para configuração das interrupções (ver Figura 3.3), são elas:

- void configInt0(): Essa função configura os registradores utilizados na **INT0**. Aqui foram configurados os registradores *EICRA* e *EIMSK*.

- void configInt1(): Essa função configura os registradores utilizados na **INT1**. Aqui também foram configurados os registradores *EICRA* e *EIMSK*.

- void configTimer1(): Essa função configura os registradores utilizados no **TIMER1**. Aqui foram configurados os registradores *TCCR1A*, *TCCR1B*, *TIMKS1* e *OCR1A*.

```
void configInt0() {                    // Funcao configInt0(): Configura a interrupcao 0
}
void configInt1() {                    // Funcao configInt1(): Configura a interrupcao 1
}
void configTimer1() {                  // Funcao configTimer1(): Configura o timer 1
}
```

Figura 3.3:

Em seguida configuramos as funções que serão chamadas quando as interrupções ocorrerem. Isso é feito com o auxílio da macro ISR(), que recebe como parâmetro o vetor de interrupção que desejamos configurar (ver Figura 3.4). No nosso caso, devemos configurar 3 vetores, são eles:

- INT0_vect: Vetor da Interrupção Externa 0.

- INT1_vect: Vetor da Interrupção Externa 1.

- TIMER1_COMPA_vect: Vetor da Interrupção por Comparação do Timer 1.

```
ISR(INT0_vect) {                        //Tratador da INT0

}

ISR(INT1_vect) {                        // Tratador da INT1

}

ISR(TIMER1_COMPA_vect) {                // Tratador do TIMER.

}
```

Figura 3.4:

Por fim, temos as 3 últimas funções do projeto, uma relacionada as configurações pessoais do usuário e as outras duas relacionadas a emissão de efeitos sonoros (Ver figura 3.5). São elas:

- void configpersonal(): Seleciona o nível de instrução do condutor e retorna a distância máxima que o sistema começa a emitir os sinais de alerta.

- void buzz(int targetPin, long frequency, long len): Emite um efeito sonoro. Essa função tem três pararâmetros: O pino a ser escrito, a frequência, e o tempo de duração do efeito.

- void sing(): Emite uma melodia.

```
void configpersonal() {       // Funcao configpersonal(): Seleciona o nivel de instrucao do condutor e retorna
                              // a distancia maxima que o sistema comeca a emitir os "Sinais de alerta"
}

void buzz(int targetPin, long frequency, long len) {   // Funcao buzz(): Usada para emitir um efeit
                                                       //                 um efeito sonoro
}

void sing(){                                           // Funcao sing(): Usada para emitir uma melodia

}
```

Figura 3.5:

Capítulo 4

Descritiva do Código Fonte

Para uma melhor compreensão do código-fonte, algumas informações adicionais merecem destaque:

-> No projeto foram usadas 3 interrupções, são elas:

- **Interrupção Externa 0**: Responsável por ligar ou desligar o sistema através do Botão ON/OFF (1). Está configurada no modo RISING (engatilha quando o Botão ON/OFF (1) vai de LOW para HIGH).

- **Interrupção Externa 1**: Responsável pela edição das configurações do sistema (nível de instrução do motorista). Está configurada no modo RISING (engatilha quando o Botão de Configuração (2) vai de LOW para HIGH).

- Interrupção por Comparação do **Timer 1**: Responsável pelos intervalos de tempo entre os "Sinais de alerta". Está configurada no modo CTC (Clear Timer on Compare) e engatilha sempre que o valor do registrador TCNT1 atingir o valor contido em OCR1A.

-> Para a comunicação serial foi usada a **USART** (Universal Synchronous and Asynchronous serial Receiver and Transmitter). A **USART** foi configurada para fazer transmissão e recepção com 8 bits de dados com um boudrate de 9600 bits por segundo.

Obs: *Para uma explicação mais detalhada consulte a secção* ***Registradores Configurados no ATMEGA328.***

-> Para a emissão das notas musicais, o programa usa um header file chamado *pitches.h*. No header file estão definidas as notas musicais.

Segue abaixo o código-fonte implementado para a realização do projeto.

```
//-------------------------------------------------------------------------------------------------------------//
//                                                                                                             //
// Projeto da disciplina de Arquitetura e Programacao de Sistemas Microcontrolados - DAS5332                   //
//                                                                                                             //
// Professor: Werner Kraus Junior                                                                              //
//                                                                                                             //
// Equipe: Fernando Battisti                                                                                   //
//         Iago de Oliveira Silvestre                                                                          //
//         Igor Assis Rocha Yamamoto                                                                           //
//                                                                                                             //
// Julho de 2015                                                                                               //
//                                                                                                             //
//-------------------------------------------------------------------------------------------------------------//

#include <avr/interrupt.h>
#include "pitches.h"

const byte TONE_PIN = 6;                    // Buzzer no pino 6
const byte ECHO_PIN = 12;                   // Echo do sensor ultrassonico no pino 12
const byte TRIGGER_PIN = 11;                // Trigger do sensor ultrassonico no pino 11
const byte LED_PISCA = 10;                  // Led que pisca de acordo com a frequencia de 'Sinais de alerta' no pino 10
const byte LED_ON  = 9;                     // Led que sinaliza ON/OFF no pino 9
const unsigned int CLK_TIMER = 15625;
int max_distance = 40;                      // Declaracao e inicializacao da variavel max_distance
byte on = LOW;                              // Declaracao e inicializacao da variavel on
int beep = NOTE_C5;                         // Declaracao e inicializacao da variavel beep
unsigned int top = 15625;                   // Declaracao e inicializacao da variavel top
void sing();                                // Declaracao da funcao sing do tipo void
void configInt0();                          // Declaracao da funcao configInt0 do tipo void
void configInt1();                          // Declaracao da funcao configInt1 do tipo void
void configTimer1();                        // Declaracao da funcao configTimer1 do tipo void
void configpersonal();                      // Declaracao da funcao configpersonal do tipo void
void buzz(int targetPin, long frequency, long len); // Declaracao da funcao buzz do tipo void
float dist_measure();                       // Declaracao da funcao dist_measure do tipo float



void setup() {
  PRR = 0;                                  // Sair do modo de baixo consumo
  UBRR0 = 103;                              // Usado para configurar o boudrate
  UCSR0A = 0;                               // Limpa UCSR0A
  UCSR0B = (1 << TXEN0) | (1 << RXEN0);     // Int TX desabilitada, circuito TX habilitado
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);   // 8 bits, sem paridade
  pinMode(TRIGGER_PIN,OUTPUT);              // Pino com Trigger sera saida
  pinMode(ECHO_PIN,INPUT);                  // Pino com Echo sera entrada
  pinMode(LED_ON, OUTPUT);                  // Pino com Led sera saida
  pinMode(LED_PISCA, OUTPUT);               // Pino com Led sera saida
  pinMode(TONE_PIN, OUTPUT);                // Pino com Buzzer sera saida
  sei();                                    // Halibita as configuracoes globais
  configInt0();
  configInt1();
  configTimer1();
  configpersonal();
}



void loop() {
  digitalWrite(LED_ON, on);                 // Acende ou apaga o Led de ON/OFF
  delay(50);
  if (on) {                                 // Se o sistema estiver ligado...
    EIMSK &= (0xFF ^ (1 << INT1));          // Desabilita a interrupcao 1
    float dist = dist_measure();
    if (dist != -1) {                       // Verifica se a distancia esta na faixa selecionada
      TIMSK1 = (1 << OCIE1A);
      switch(max_distance){                 // Atribui um valor a variavel top dependendo da max_distance
        case 20:                            // e da dist
          top = CLK_TIMER/(-27.4112*log(0.0482088*dist));
          break;
        case 30:
          top = CLK_TIMER/(-27.4112*log(0.0321392*dist));
          break;
        case 40:
          top = CLK_TIMER/(-27.4112*log(0.0241044*dist));
          break;
        default:
          top = CLK_TIMER/(-27.4112*log(0.0241044*dist));
          break;
      }
      if (top > 65535) top = 65535;                             // top pode ser no maximo ( 2e16 - 1 )
    }
    Serial.print("Ping: ");
    Serial.print(dist);
    Serial.println("cm");
    Serial.println(top);
  }
  else{                                                         // Se o sistema estiver desligado...
    TIMSK1 = (0 << OCIE1A);                                     // Desliga o timer com o buzzer
    EIMSK |= (1 << INT1);                                       // Habilita a interrupcao externa 1
  }
}
```

```
//-----------------------------------------------------------------------------------------------------//

void configInt0() {                     // Funcao configInt0(): Configura a interrupcao 0
  EICRA = (1 << ISC01) | (1 << ISC00);     // Registrador de configuracao do modo RISING
  EIMSK = (1 << INT0);                     // Habilita interrupao externa 0
}

void configInt1() {                     // Funcao configInt1(): Configura a interrupcao 1
  EICRA |= (1 << ISC11) | (1 << ISC10);    // Registrador de configuracao do modo RISING
  EIMSK |= (1 << INT1);                    // Habilita interrupao externa 1
}

void configTimer1() {                   // Funcao configTimer1(): Configura o timer 1
  TCCR1A = 0;                              // Limpa TCCR1A
  TCCR1B = (1 << CS12) | (1 << CS10) | (1 << WGM12); // Prescaler divide por 64 f_clk = 16 MHz/256 = 250 kHz
                                                     // Modo CTC (Clear Timer on Compare)
  TIMSK1 = (0 << OCIE1A);                  // Desabilita interrupaoo comparacao timer 1
  OCR1A = top;                             // Atribui o valor da variavel top no registrador OCR1A
}




ISR(INT0_vect) {                                    //Tratador da INT0
  if(on){
    Serial.println("Desligando...");
  }
  else{
    Serial.println("Bem-vindo ao seu assistente de conducao!");
    sing();
  }
  on = !on;
}

ISR(INT1_vect) {                                    // Tratador da INT1
  while(!( UCSR0A & (1<<UDRE0) ) );
  Serial.println("Configuracoes do sistema");
  configpersonal();                                 // Chama a funcao configpersonal()
}

ISR(TIMER1_COMPA_vect) {                            // Tratador do TIMER.
  TCCR1B = (0 << CS12) | (0 << CS11) | (0 << CS10);
  buzz(TONE_PIN, beep, 100);
  OCR1A = top;                                      // Atualiza valor de OCR1A
  TCCR1B = (1 << CS12) | (1 << CS10) | (1 << WGM12);
}




float dist_measure(){                       // Funçao dist_measure(): retorna a distancia entre o sensor e o objeto mais proximo
  float duration, distance;

  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);

  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);

  digitalWrite(TRIGGER_PIN, LOW);
  duration = pulseIn(ECHO_PIN, HIGH);

  Serial.print("duration: ");
  Serial.println(duration);

  distance = duration/58.2;                         // Calcula a distancia (em cm) baseado na velocidade do som
  if (2<= distance && distance <= ){       // A mediçao minima do sensor é 2cm e a maxima é determinada pelo usuario
    return distance;
  }
  else{
    return -1;
  }
}
```

```
void configpersonal() {                                 // Funcao configpersonal(): Seleciona o nivel de instrucao do condutor e retorna
                                                        // a distancia maxima que o sistema comeca a emitir os "Sinais de alerta"
  Serial.println("Escolha o nivel de instrucao do condutor do veiculo: ");
  Serial.println("1.   Profissional ");
  Serial.println("2.   Intermediario ");
  Serial.println("3.   Iniciante ");
  while (!( UCSR0A & (1 << RXC0) ) );                   // Espera ate que o usuario digite algo
  char ctrlUser = UDR0;                                 // Atribui o que foi digitado a ctrlUser
  switch (ctrlUser) {                                   // Seleciona uma distancia maxima dependendo
    case '1':                                           // do nivel de instrucao do condutor
      max_distance = 20;
      beep = NOTE_D4;
      break;
    case '2':
      max_distance = 30;
      beep = NOTE_D5;
      break;
    case '3':
      max_distance = 40;
      beep = NOTE_D6;
      break;
  }
  Serial.println(ctrlUser);
  Serial.print("distancia maxima: ");
  Serial.println(max_distance);
}
```

```
void buzz(int targetPin, long frequency, long len) { // Funcao buzz(): Usada para emitir
  digitalWrite(LED_PISCA,HIGH);                      //                um efeito sonoro
  long delayValue = 1000000/frequency/2;
  long numCycles = frequency * len/ 1000;
  for (long i=0; i < numCycles; i++){
    digitalWrite(targetPin,HIGH);
    delayMicroseconds(delayValue);
    digitalWrite(targetPin,LOW);
    delayMicroseconds(delayValue);
  }
  digitalWrite(LED_PISCA,LOW);
}
```

```
void sing(){                                    // Funcao sing(): Usada para emitir uma melodia
  int melody[] = {
  NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  };
 int size = sizeof(melody) / sizeof(int);
 for (int thisNote = 0; thisNote < size; thisNote++) {
   int noteDuration = 1000/12;
   buzz(TONE_PIN, melody[thisNote],noteDuration);
   int pauseBetweenNotes = noteDuration * 1.30;
   delay(pauseBetweenNotes);
   buzz(TONE_PIN, 0,noteDuration);
  }
}
```

Capítulo 5

Interação com Dispositivos Externos

Para a realização do projeto, foram usados os seguintes dispositivos externos:

- Chaves momentâneas: (1) e (2) da Figura 2.1. Quando o botão é apertado, os contatos entre os terminais de cada lado são ligados entre si.

- Leds (Verde e Vermelho): (3) e (4) da Figura 2.1. Emite uma luz quando uma pequena corrente o excita.

- Buzzer: (5) da Figura 2.1. Quando uma corrente elétrica passa por ele, ele emite um som.

- Resistores: Limita a corrente elétrica que passa pelo circuito. Para limitar mais ou menos corrente, o valor deste componente pode variar.

- Sensor Ultrassônico: (6) da Figura 2.1. É usado para medir distâncias.

Dos itens citados acima, consideramos que o sensor ultrassônico merece uma explicação mais detalhada.

O sensor ultrassônico utilizado no projeto é o HC-SR04. Ele é composto de um emissor e um receptor de ondas sonoras e trabalha com ondas de altíssima frequências, na faixa dos 40KHz.

O sinal emitido, ao colidir com algum obstáculo, é refletido de volta na difereção do sensor. Durante esse processo, o aparelho mede o tempo entre o sinal emitido e o sinal recebido. Como conhecemos a velocidade do som no ar, é possivel então calcularmos a distância entre o sensor e o obstáculo através da equação $d = \frac{v \times t}{2}$. Fizemos a divisão por 2 pois o tempo medido pelo sensor é na verdade o tempo para ir e voltar, ou seja, duas vezes a distância que queremos descobrir.

Na Figura 5.1 podemos ver um gráfico do ângulo x performace do sensor.

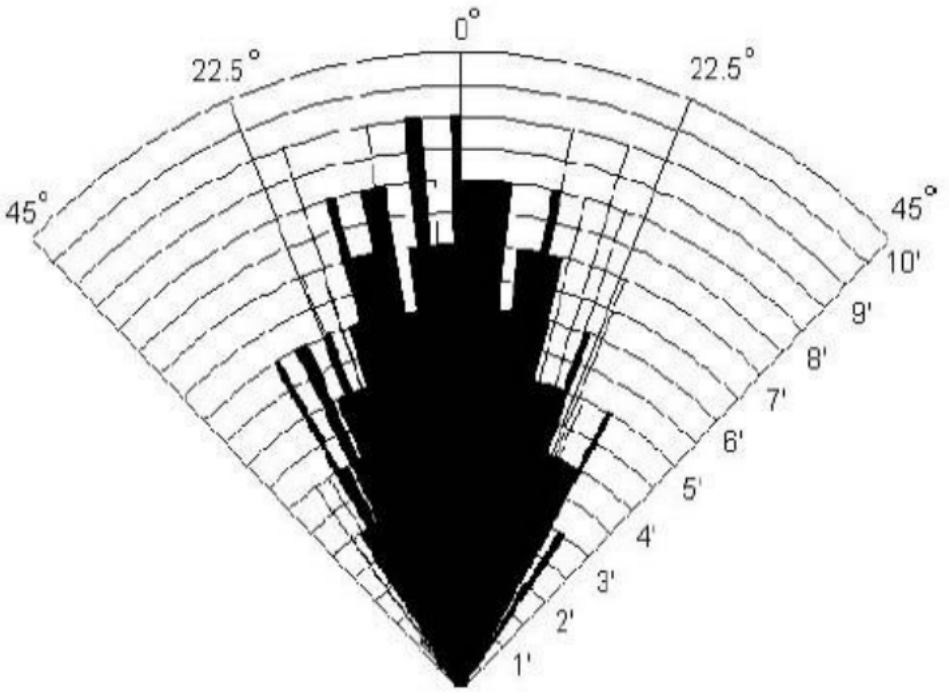

Figura 5.1: Ãngulo x Performace

Capítulo 6

Registradores Configurados no ATMEGA328

Descreveremos aqui como foi feita a configuração dos registradores utilizados no projeto.

6.1 Interrupções Externas 0 e 1

Para as interrupções externas 0 e 1, usamos os seguintes registradores:

- *EICRA* (External Interrupt Control Register A): Esse registrador é usado para configurar o modo como a **INT0** e a **INT1** irão atuar.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

As tabelas abaixo mostram as configurações dos bits com suas respectivas descrições.

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Table 12-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

No projeto, configuramos ambas interrupções no modo FALLING edge. Para isso, os bits *ISC11* e *ISC01* foram setados.

- *EIMSK* (External Interrupt Mask Register): Esse registrador é usado para habilitar as interrupções externas.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

No projeto, setamos os bits *INT0* e *INT1* para que as interrupções externas **INT0** e **INT1** fossem habilitadas.

6.2 Comunicação Serial USART0

Para a **USART0**, usamos os seguintes registradores:

- *UDR0* (USART I/O Data Register 0): É o buffer de transmissão e recebimento. No projeto, o registrador é usado tanto para escrita quanto para leitura na configuração do nível do motorista.

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- *UCSR0A* (USART Control and Status Register 0 A): Nesse registrador usamos o bit *RXC0* para indicar que o usuário selecionou uma opção na função void configpersonal().

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- *UCSR0B* (USART Control and Status Register 0 B): Esse registrador foi utilizado para habilitarmos a transmissão e recebimento de dados. Para isso foram setados os bits *TXEN0* e *RXEN0*.

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- *UCSR0C* (USART Control and Status Register 0 C): Esse registrador foi usado para que a comunicação serial aconteça de forma assíncrona, sem paridade e com um tamanho de dados recebidos e

transmitidos de 8 bits.

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

De acordo com a tabela abaixo, para um tamanho de dados de 8 bits, foi setado os bits *UCSZ01* e *UCSZ00*.

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- *UBRR0* (USART Baud Rate Registers): Esse registrador é usado para configurar o boud rate. No projeto, usamos um boud rate de 9600 bps. Para isso o valor de *UBRR0* deve ser igual a 103.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

A tabela abaixo mostra as equações para o cálculo do boud rate e do valor do registrador *UBRR0* de acordo com o modo de operação. No nosso caso, usamos a equação destacada.

Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}n+1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}n+1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}n+1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{2BAUD} - 1$

Apresentamos abaixo como chegamos nesse valor, lembrando que possuimos um clock de 16MHz.

$$UBRRn = \frac{f_{osc}}{16 \times BOUD} - 1$$

Substituindo $f_{osc} = 16MHz$ e $BOUD = 9600$, obtemos o valor a ser colocado em $UBRR0$, que é 103.

6.3 Timer 1

Para o **TIMER1**, usamos os seguintes registradores:

- *TCCR1A* (Timer/Counter1 Control Register A): No projeto, esse registrador é zerado.

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- *TCCR1B* (Timer/Counter1 Control Register B): Usamos esse registrador para configurar o modo da forma de onda gerada pelo **TIMER1** e a divisão de clock.

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

No projeto, configuramos o **TIMER1** no modo CTC. Conforme a tabela abaixo, setamos o bit *WGM12*, e assim o valor de *OCR1A* fica definido como o valor de topo (máximo valor de contagem).

Table 15-4. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Além do modo CTC, no registrador *TCCR1B* configuramos a divisão do clock. No projeto, usamos uma divisão de clock por 256, assim ficamos com um clock de 62500Hz. Para isso, conforme a tabela abaixo, setamos s bit *CS12*.

Table 15-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

- *TIMSK1* (Timer/Counter1 Interrupt Mask Register): Usamos esse registrados para habilitar ou desabilitar a interrupção do **TIMER1** por comparação. Para isso usamos o bit *OCIE1A*.

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	