UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Faculdade de Engenharia Elétrica

João Paulo Cândido Nascimento

Thiago Fernandes Pereira de Freitas

Projeto Final de Sistemas Embarcados: Implementação de um controlador de tempo de execução de tarefas em um sistema qualquer

Uberlândia
2017

Universidade Federal de Uberlândia

Faculdade de Engenharia Elétrica

Relatório Final

Relatório final do projeto da disciplina de Sistemas Embarcados da Faculdade de Engenharia Elétrica da UFU.

João Paulo Cândido Nascimento – 11421ECP004 – joaopcandido@ufu.br

Thiago Fernandes P. de Freitas – 11421ECP009 – thiagofp@ufu.br

Professor: Fábio Vincenzi

Uberlândia

Julho de 2017

Resumo

Este trabalho visa a implementação de um dispositivo cujo intuito é controlar o tempo de execução de uma determinada tarefa, em um sistema qualquer. Desse modo, para implementar o controlador em questão e mostrar seu funcionamento, será feito o acionamento de um motor DC.

O projeto será implementado com o microcontrolador Arduino, e contará com a utilização de um motor DC, um módulo bluetooth HC-05, dois LEDs, dois displays de 7 segmentos acompanhados de decodificadores, e alguns resistores. Serão utilizados também conceitos de programação de embarcados em C, aprendidos na disciplina de Sistemas Embarcados.

Sumário

1. Introdução

Com a evolução da tecnologia, sistemas automatizados de execução de tarefas têm surgido a cada dia. Diversos são os exemplos que podem ser citados de equipamentos de automação que já estão presentes no nosso dia-a-dia, desde simples caixas automáticos de banco ou terminais de autoatendimento em cinemas, até processos de produção inteiramente automatizados em uma fábrica de veículo.

Não importa qual seja a situação, a tecnologia tem nos auxiliado a cumprir tarefas de maneira mais precisa, eficiente e produtiva. Dessa forma, o presente projeto tem o intuito de implementar um controlador de tempo de execução de tarefas em um sistema qualquer, de modo a facilitar essa execução.

O controlador de tempo em questão será implementado com o microcontrolador Arduino, utilizando de comunicação bluetooth para realizar tal controle. A ideia do projeto é que o usuário, tendo em mãos um dispositivo Android com a aplicação do projeto instalada, possa controlar por quanto tempo deseja acionar um determinado sistema, enviando essa informação via comunicação bluetooth, bem como ter a possibilidade de interromper esse processo a qualquer momento durante sua execução.

Para exemplificar uma situação em que o controlador de tempo possa ser aplicado, suponhamos um sistema de automação residencial, no qual o usuário deseja ter controle de quando irrigar seu jardim. Por meio do controlador, ele poderia, no aplicativo, determinar por quanto tempo gostaria que o sistema de irrigação fosse acionado e, após transcorrido o tempo escolhido, ele seria desligado. Suponhamos agora que o usuário tenha colocado um tempo de 10 minutos, porém, com 5 minutos de execução, ele já esteja satisfeito com o resultado. Com esse controlador, ele poderia interromper o sistema de irrigação.

Além disso, o controlador de tempo pode ser utilizado também, por exemplo, em um processo industrial. Suponhamos uma situação em que se deseja verificar a eficiência de uma esteira de um processo produtivo, em uma etapa de empacotamento. Com esse controlador, o usuário poderia acionar a esteira por um tempo desejado, dez minutos por exemplo, e, de acordo com a quantidade de caixas que foram empacotadas neste intervalo de tempo específico, é possível avaliar a eficiência de tal processo.

2. Desenvolvimento do Controlador

2.1. Lista de Materiais Utilizados

Para a implementação do controlador de tempo de execução de tarefas, serão utilizados os seguintes materiais:

- 1 Arduino UNO ATMEGA 328;
- 1 módulo bluetooth RS232 HC-05;
- 2 displays de 7 segmentos do tipo anodo comum;
- 2 decodificadores BCD para displays de 7 segmentos SN74LS47;
- 8 resistores de 330 Ω;
- 4 resistores de 470 Ω;
- 2 LEDs de alto brilho (1 vermelho e 1 verde);
- 1 potenciômetro de 10 kΩ;
- 1 motor DC 5V.

2.2. Diagrama Esquemático do Hardware

Os materiais listados no item anterior foram organizados para a montagem do hardware do controlador, da seguinte forma:

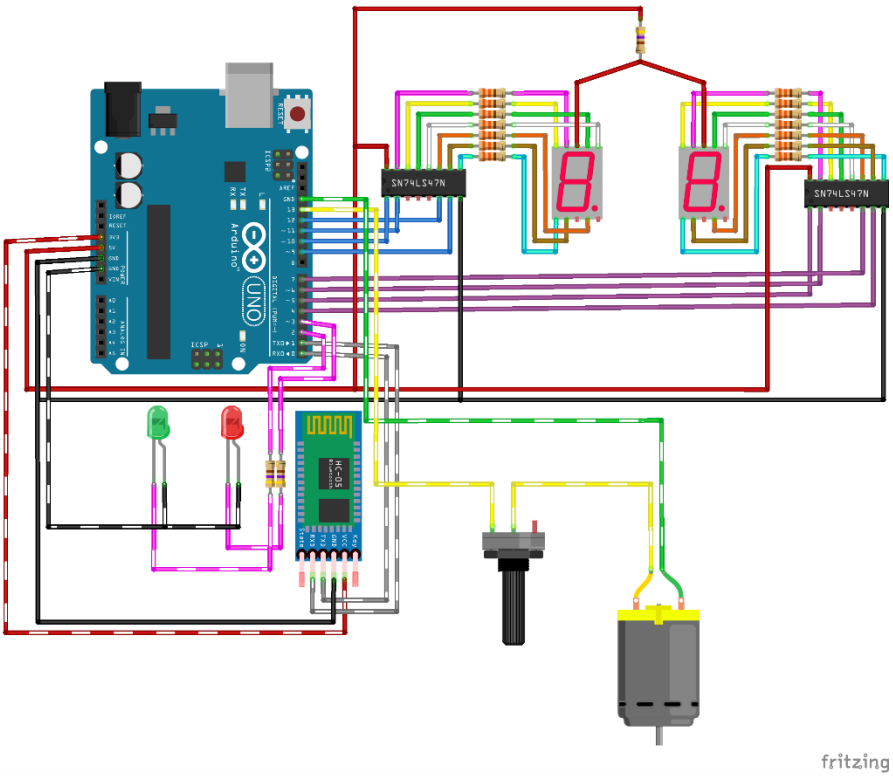

Figura 1 – Diagrama Esquemático do Hardware

2.3. Texto explicativo do Hardware

A base deste projeto é o microcontrolador Arduino. Nele, foi acoplado o módulo bluetooth, utilizando as portas digitais 0 e 1 (Rx e Tx, respectivamente), de modo que a primeira é utilizada para recebimento de dados e a segunda para transmissão. Dessa maneira, o Arduino recebe o comando do aplicativo Android por meio da porta Rx.

Nas portas digitais 2 e 3, foram ligados dois LEDs, um verde e um vermelho, respectivamente, juntamente com resistores de 470 Ω (para fins de proteção), com a finalidade de indicar o estado em que o controlador se encontra: no caso em que está disponível para receber uma instrução do usuário, o LED verde encontra-se aceso, enquanto o vermelho encontra-se apagado. Por outro lado, se o LED vermelho está aceso e o verde apagado, isso significa que o controlador está ocupado executando uma tarefa e, para que uma nova tarefa seja executada, é preciso interromper o processo em execução. Todo esse tráfego de instruções e interrupções é feito via bluetooth, por meio de uma aplicação Android.

Os displays foram ligados nas demais portas digitais, de modo que o display da direita foi conectado nas portas de 4 a 7, e o da esquerda nas portas de 9 a 12. Assim, os displays são acionados pelo Arduino de acordo com o tempo de execução requerido via bluetooth, de tal forma que o dígito é enviado para o decodificador em base binária de 4 bits, e o mesmo converte para decimal, exibindo o resultado no display. Entre as portas dos decodificadores e dos displays, foram colocados resistores de 330 Ω, e, na alimentação dos displays, resistores de 470 Ω, a fim de reduzir a intensidade de corrente.

Por último, a porta digital 13 é destinada ao acionamento do sistema que se deseja controlar. Nesse projeto em questão, o motor DC de 5V, ligado juntamente com um potenciômetro, a fim de controlar a frequência de giro do motor.

2.4. Implementação do Software no Arduino

Segue código implementado, com comentários explicativos:

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <util/delay.h>

#define SREG_GLOBAL_INT_ENABLE 7

uint16_t u16_count = 0, u16_cMin = 0;
char c_interrupt, c_measure;
bool b_avaliable = true;

//MACRO de Timer, executando a cada 1 segundo (configurado no
Init_Config())
ISR(TIMER1_COMPA_vect)
{
  //O sistema é acionado com base no valor armazenado na variável
c_measure, recebido via Bluetooth no loop principal
  switch (c_measure)
  {
    case 's': //caso o usuário tenha escolhido segundos:

      //Verificando a contagem do tempo requerido pelo usuário:
      if (u16_count == 0) //Se zero,
      {
        PORTB &= ~(1 << PORTB5); //Desliga o motor, acoplado na porta
13 do arduíno
      }

      //Exibe nos Displays de 7 segmentos o tempo restante:
      DisplayDezena(u16_count / 10);
      DisplayUnidade(u16_count % 10);

      if (u16_count > 0) //Se a contagem ainda for maior que zero,
      {
        PORTB |= (1 << PORTB5); //o motor continua sendo acionado,
        u16_count--; //E o tempo decresce de 1 a cada segundo (já que
a MACRO executa esse código a cada segundo)
      }
      break;

    case 'm': //caso o usuário tenha escolhido minutos:

      //Verificando a contagem do tempo requerido pelo usuário:
      if (u16_count == 0) //Se zero,
      {
        PORTB &= ~(1 << PORTB5); //Desliga o motor, acoplado na porta
13 do arduíno
      }

      u16_cMin++; //Incrementa o contador de 1 minuto a cada execução
da MACRO

      //Exibe quantos minutos ainda faltam:
      DisplayDezena(u16_count / 10);
      DisplayUnidade(u16_count % 10);

      if (u16_count > 0)//Se a contagem ainda for maior que zero,
```

```
      {
        PORTB |= (1 << PORTB5); //o motor continua sendo acionado e,
        if (u16_cMin == 60)// se o contador de 1 minuto chega em 60
(após 60 segundos),
        {
          u16_cMin = 0; //esse contador é zerado e, então,
          u16_count--; //o tempo restante é decrementado de 1.
        }
      }
      break;

    default: //caso default para quando a instrução da unidade de
medida ainda não foi escolhida.
      DisplayDezena(0);
      DisplayUnidade(0);
      break;
  }
}


void Init_Config()
{
  /* -------- Configurações do Monitor Serial -------- */

  UBRR0H = 0;
  UBRR0L = 103;

  /* -------- Port Set -------- */

  DDRB |= (1 << DDB4) | (1 << DDB3) | (1 << DDB2) | (1 << DDB1);
//Display Esquerda
  DDRD |= (1 << DDD7) | (1 << DDD6) | (1 << DDD5) | (1 << DDD4);
//Display Direita

  DDRD |= (1 << DDD3) | (1 << DDD2); //LEDs
  DDRB |= (1 << DDB5); //Motor

  /* -------- Configurações do TIMER -------- */

  TIMSK1 &= ~((1 << ICIE1) | (1 << OCIE1B) | (1 << OCIE1A) | (1 <<
TOIE1));
  TIMSK1 |= (1 << OCIE1A);
  TCCR1B |= (1 << WGM12);
  TCCR1B |= (1 << CS12) | (1 << CS10); //Prescaler de 1024
  OCR1A = 15624;  //Frequência do Registrador, com base no prescaler
escolhido de 1024, para que a MACRO de timer execute a cada 1 segundo

  /* -------- Configurações do BLUETOOTH -------- */

  UCSR0A &= ~(1 << U2X0);  /* _BV(U2X0) = (1 << U2X0) */
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00); /* 8-bit data  */
  UCSR0B = (1 << RXEN0) | (1 << TXEN0);   /* Enable Rc_x and Tc_x */
}

char Receive_Data(void) {

  while ( !( UCSR0A & (1 << RXC0)) ) { // Função para receber dados do
bluetooth

  }
  return UDR0;
```

```
}

void Transmit_Data( char data )
{
  while ( !( UCSR0A & (1 << UDRE0)) );

  UDR0 = data;
}

uint16_t ConvertC(char c_chart) { //Converte uma variável do tipo char
para int
  uint16_t u16_num;
  u16_num = c_chart - '0';
  return u16_num;
}

char ConvertI(uint16_t u16_intg) { //Converte umma variável do tipo
uint16_t para char
  char c_charct;
  c_charct = u16_intg + '0';
  return c_charct;
}

uint16_t CriaInt(uint16_t u16_n1, uint16_t u16_n2) { //faz a criação
de um número de dois dígitos, unindo o inteiro da dezena e da unidade
  uint16_t u16_n3;
  u16_n3 = (u16_n1 * 10) + u16_n2;
  return u16_n3;
}

int main(void)
{
  Init_Config();

  while (1)
  {
    SREG |= (1 << SREG_GLOBAL_INT_ENABLE);

    char c_x;
    uint16_t i = 0;
    uint16_t u16_unid, u16_dezn, u16_number;
    char c_receive_number[2];
    char c_strDez, c_strUn;

    /*
      O Switch é uma máquina de estados que analisa a disponibilidade
do controlador, ou seja, se está disponível ou não para receber novas
instruções de execução.
    */

    switch (b_avaliable)
    {

      /*
        Caso o sistema esteja disponível, o LED Verde é aceso e o LED
Vermelho apagado, e o controlador fica aguardando a primeira instrução
do usuário.
      */

      case true:
        PORTD |= (1 << PORTD2); //Acende LED Verde
```

```
        PORTD &= ~(1 << PORTD3); //Apaga LED Vermelho
        c_measure = Receive_Data(); //Controlador pronto para receber
a primeira instrução (escolha da unidade de tempo de execução: minutos
ou segundos)
        b_avaliable = false; //Após recebida, o controlador entra no
estado indisponível
        break;

      /*
        No estado indisponível, o LED Vermelho é aceso e o LED Verde
apagado, e começa a análise dos dados recebidos.
      */

      case false:
        PORTD |= (1 << PORTD3); //Acende LED Vermelho
        PORTD &= ~(1 << PORTD2); //Apaga LED Verde

        /*
          O While abaixo é responsável por receber os dados do App e
colocar tais dados em um vetor de caracteres.
        */

        while (i < 2)
        {
          c_x = Receive_Data(); // nesta parte, a variável c_x recebe
o dado do tipo char do App
          c_receive_number[i] = c_x; // o vetor de char recebe, em sua
primeira posição esse X
          i++; // incrementa a posição
        }

        /*
          O trecho abaixo trabalha com a conversão de char para int
        */

        c_strDez = c_receive_number[0]; // c_strDez recebe a primeira
posição do vetor
        c_strUn = c_receive_number[1]; // c_strUn recebe a segunda
posição do vetor

        // ConvertC é uma função criada para Converter char para
uint16_t utilizando a tabela ASCII
        u16_unid = ConvertC(c_strUn);

        u16_dezn = ConvertC(c_strDez);

        // Criauint16_t é uma função que, recebe dois inteiros,
referentes à dezena e à unidade do mesmo e faz a junção deles em um
inteiro só.
        u16_number = CriaInt(u16_dezn, u16_unid);

        //A variável u16_count recebe, então, o número criado a partir
da instrução enviada pelo usuário, referente a quanto tempo o sistema
deve permanecer em atividade.
        u16_count = u16_number;

        c_interrupt = Receive_Data();
        /*
          Após o envio de uma instrução, o controlador fica aguardando
um comando de interrupção,
```

```
          mesmo depois de transcorrido todo o tempo de atividade
requerido pelo usuário.
          Dessa forma, o sistema só estará pronto para ser utilizado
novamente após receber a ordem
          do usuário via bluetooth
        */
        u16_count = 0; //Depois de recebido o comando de interrupção,
a variável u16_count é zerada.

        b_avaliable = true; //termina a análise e torna o bluetooth
disponível novamente
        break;
    }
  }
}


/* As funções abaixo foram criadas para acender os segmentos de cada
Display referentes à dezena e à unidade de um número.
  Como os decodificadores utilizados trabalham com números binários,
os 'cases' do 'switch' foram construídos para enviar
  corretamente esses números para o decoder, de modo que:
  Para o Display da Unidade:
    Bit:   A3   A2   A1   A0
    Port:  D7   D6   D5   D4
  Para o Display da Dezena:
    Bit:   A3   A2   A1   A0
    Port:  B4   B3   B2   B1
  (Obs: o número 9, por exemplo, em binário, é 1001. Assim: A3 = 1; A2
= 0; A1 = 0; A0 = 1)
*/
void DisplayUnidade(uint16_t u16_cont) {
  switch (u16_cont)
  {
    case 0:
      PORTD &= ~((1 << PORTD4) | (1 << PORTD5) | (1 << PORTD6) | (1 <<
PORTD7));
      break;
    case 1:
      PORTD &= ~((1 << PORTD5) | (1 << PORTD6) | (1 << PORTD7));
      PORTD |= (1 << PORTD4);
      break;
    case 2:
      PORTD &= ~((1 << PORTD4) | (1 << PORTD6) | (1 << PORTD7));
      PORTD |= (1 << PORTD5);
      break;
    case 3:
      PORTD &= ~((1 << PORTD6) | (1 << PORTD7));
      PORTD |= (1 << PORTD4) | (1 << PORTD5);
      break;
    case 4:
      PORTD &= ~((1 << PORTD4) | (1 << PORTD5) | (1 << PORTD7));
      PORTD |= (1 << PORTD6);
      break;
    case 5:
      PORTD &= ~((1 << PORTD5) | (1 << PORTD7));
      PORTD |= (1 << PORTD4) | (1 << PORTD6);
      break;
    case 6:
      PORTD &= ~((1 << PORTD4) | (1 << PORTD7));
      PORTD |= (1 << PORTD5) | (1 << PORTD6);
```

```
      break;
    case 7:
      PORTD &= ~(1 << PORTD7);
      PORTD |= (1 << PORTD4) | (1 << PORTD5) | (1 << PORTD6);
      break;
    case 8:
      PORTD &= ~((1 << PORTD4) | (1 << PORTD5) | (1 << PORTD6));
      PORTD |= (1 << PORTD7);
      break;
    case 9:
      PORTD &= ~((1 << PORTD5) | (1 << PORTD6));
      PORTD |= (1 << PORTD4) | (1 << PORTD7);
      break;
    default:
      break;
  }
}

void DisplayDezena(uint16_t u16_cont) {
  switch (u16_cont)
  {
    case 0:
      PORTB &= ~((1 << PORTB1) | (1 << PORTB2) | (1 << PORTB3) | (1 <<
PORTB4));
      break;
    case 1:
      PORTB &= ~((1 << PORTB2) | (1 << PORTB3) | (1 << PORTB4));
      PORTB |= (1 << PORTB1);
      break;
    case 2:
      PORTB &= ~((1 << PORTB1) | (1 << PORTB3) | (1 << PORTB4));
      PORTB |= (1 << PORTB2);
      break;
    case 3:
      PORTB &= ~((1 << PORTB3) | (1 << PORTB4));
      PORTB |= (1 << PORTB1) | (1 << PORTB2);
      break;
    case 4:
      PORTB &= ~((1 << PORTB1) | (1 << PORTB2) | (1 << PORTB4));
      PORTB |= (1 << PORTB3);
      break;
    case 5:
      PORTB &= ~((1 << PORTB2) | (1 << PORTB4));
      PORTB |= (1 << PORTB1) | (1 << PORTB3);
      break;
    case 6:
      PORTB &= ~((1 << PORTB1) | (1 << PORTB4));
      PORTB |= (1 << PORTB2) | (1 << PORTB3);
      break;
    case 7:
      PORTB &= ~(1 << PORTB4);
      PORTB |= (1 << PORTB1) | (1 << PORTB2) | (1 << PORTB3);
      break;
    case 8:
      PORTB &= ~((1 << PORTB1) | (1 << PORTB2) | (1 << PORTB3));
      PORTB |= (1 << PORTB4);
      break;
    case 9:
      PORTB &= ~((1 << PORTB2) | (1 << PORTB3));
      PORTB |= (1 << PORTB1) | (1 << PORTB4);
      break;
```

```
    default:
      break;
  }
}
```

2.5. Implementação do Aplicativo Android

A aplicação foi desenvolvida pelo Android Studio. Ela basicamente envia a quantidade de tempo que o usuário digita para o módulo bluetooth após este usuário escolher a opção de minutos ou segundos.

Primeiramente, o usuário deve fazer o pareamento do celular com o módulo bluetooth.

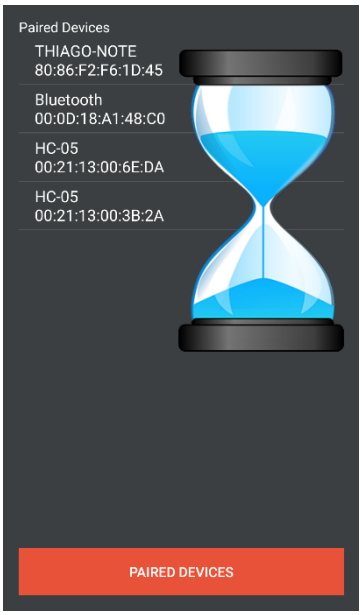

Figura 2 – Pareamento Celular com Módulo Bluetooth

Após isso, deve-se escolher a função de minutos ou segundos.

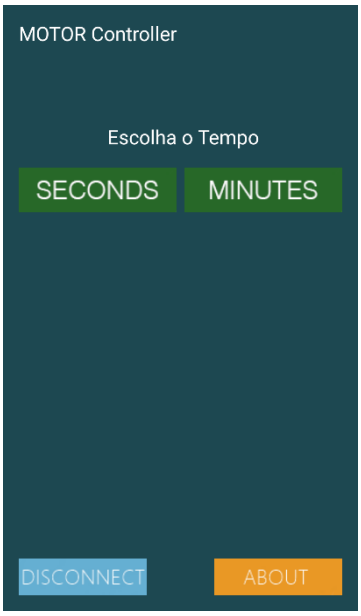

Figura 2 – Escolha da opção: Segundos ou Minutos

Escolhida a função, a tela é atualizada, aparece a opção para usuário digitar a quantidade de tempo desejada e enviar este tempo para o módulo. Como utilizamos 2 displays de 7 segmentos, forçamos o usuário a digitar um número de apenas dois dígitos.

Figura 3 – Inserir tempo desejado

Depois de enviado o tempo para o Arduino (clicando no botão Timer), a opção de interrupção aparece. A qualquer momento que o usuário desejar, ao clicar em “Stop”, os displays são zerados e aparece novamente a tela de escolha de minutos ou segundos.

Figura 4 – Monitoramento já iniciado, opção de parada disponível

3. Cronograma das Etapas do Projeto

O desenvolvimento deste projeto foi dividido em VI etapas, listadas a seguir:

I. Listagem e compra dos materiais utilizados no projeto;
II. Entendimento da linguagem ANSI-C e de registradores do Arduino;
III. Estudo e compreensão do funcionamento de um display de 7 seguimentos;
IV. Estudo e compreensão da implementação de comunicação UART via Bluetooth utilizando o módulo HC-05;
V. Estudo e compreensão do algoritmo de timer via registradores do Arduino;
VI. Integração dos conhecimentos adquiridos e montagem final do projeto.

As etapas foram desenvolvidas da seguinte forma:

Atividade	Mês (2017)		
	Mai	Jun	Jul
I	■		
II	■	■	
III		■	
IV		■	
V		■	
VI			■

Tabela 1 – Cronograma das Etapas do Projeto

4. Resultados Experimentais

Após a implementação do projeto, foram feitos testes para avaliar o bom funcionamento do controlador.

O primeiro teste realizado foi com o ciclo de instruções do controlador operando em segundos. Desse modo, foi escolhido, na tela do aplicativo, o botão “segundos”, fazendo com que o estado “ocupado” fosse iniciado. Em seguida, foi inserido o valor de 40. Assim, o controlador acionou o motor DC e ligou o display de 7 segmentos, com valor inicial de 40. A cada segundo, o número exibido era decrementado de 1 e, quando chegou em 0, o motor foi desligado. Foi enviado, então, pelo aplicativo, um comando, indicando que a tarefa foi concluída, e o controlador voltou ao estado de “disponível”.

De modo semelhante, foi realizado também o teste com minutos. O tempo de acionamento escolhido foi de 2 minutos e, como esperado, o número exibido no

display, inicialmente 2, decrementava a cada minuto. Ao zerar, o motor foi desligado, e o sistema voltou a estar disponível.

Por último, foram realizados testes de interrupção, de modo que, antes do tempo pré-determinado, foi enviado um comando, pelo aplicativo, para que o sistema desligasse naquele momento. Como esperado, o motor foi desligado, o display zerado, e o sistema retornou novamente ao estado “disponível”.

5. Referências Bibliográficas

[1] RUSSELL, David J. Introduction to embedded systems: using ANSI C and the arduino development environment. **Synthesis Lectures on Digital Circuits and Systems**, v. 5, n. 1, p. 1-275, 2010.

[2] EVANS, Martin; NOBLE, Joshua J.; HOCHENBAUM, Jordan. **Arduino in action**. Manning, 2013.

[3] PAHUJA, Ritika; KUMAR, Narender. Android Mobile Phone Controlled Bluetooth Robot Using 8051 Microcontroller. **International Journal of Scientific Engineering and Research**, v. 2, n. 7, 2014.

[4] PIYARE, R.; TAZIL, M. Bluetooth based home automation system using Android phones. In: **IEEE 15TH International symposium on consumer electronics (ISCE)**. 2011. p. 14-17.