Universidade Federal de Goiás
Regional Catalão
Unidade Acadêmica Especial de Biotecnologia
Curso de Bacharelado em Ciências da Computação

# Arquitetura de Computadores

**Marco Túlio Macedo Rodrigues, Pablo Vinicius da Silva, Vitor do Valde Bernardo**

Catalão – GO

**Marco Túlio Macedo Rodrigues, Pablo Vinicius da Silva, Vitor do Valde Bernardo**

# Arquitetura de Computadores

Monografia apresentada ao Curso de Bacharelado em Ciências da Computação da Universidade Federal de Goiás – Regional Catalão, como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação. *VERSÃO REVISADA*

Orientadora: Prof. Dr. Tércio Alberto dos Santos Filho

Catalão – GO

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

, Marco Túlio Macedo Rodrigues, Pablo Vinicius da Silva, Vitor do Valde Bernardo

Arquitetura de Computadores [manuscrito] / Marco Túlio Macedo Rodrigues, Pablo Vinicius da Silva, Vitor do Valde Bernardo . –

.

41 p.: il.

Orientadora: Prof. Dr. Tércio Alberto dos Santos Filho

Monografia (Graduação) – Universidade Federal de Goiás, Unidade Acadêmica Especial de Biotecnologia, Ciências da Computação,

.

Bibliografia.

1. Arquitetura. 2. Computadores. 3. Microcontroladores. 4. PWP. 5. SPI. I. Filho, Tércio Alberto dos Santos, orient. II. Título.

CDU 004

**Marco Túlio Macedo Rodrigues, Pablo Vinicius da Silva, Vitor do Valde Bernardo**

# Arquitetura de Computadores

Monografia apresentada ao curso de Bacharelado em Ciências da Computação da Universidade Federal de Goiás – Regional Catalão.

Trabalho aprovado em 06 de de

.

**Tércio Alberto dos Santos Filho**
Orientadora

Catalão – GO

*Este trabalho é dedicado a todas as pessoas que,*
*Soham em aprender mais sobre eletrôncia.*
*Em especial, aos graduando em ciências exatas*
*De quaisquer universidades, particualres e públicas.*

# AGRADECIMENTOS

Os agradecimentos principais são direcionados à Deus que até aqui nós sustentou e nos ajudou a prevalecer.

Agradecimentos especiais são direcionados ao professor Dr. Tércio Alberto Santos Filho que sempre se dispôs à ajudar os alunos da disciplina e nunca mediu esforços para sanar as dúvidas de seus alunos, seja fora ou dentro de sala. Deixamos aqui nossos mais sinceros agradecimentos.

*“As invenções são, sobretudo,*
*o resultado de um trabalho de teimoso.”*
*(Santos Dumont)*

# RESUMO

NOME PARA REFERÊNCIA. **Arquitetura de Computadores**.

. 41 p. Monografia (Graduação) – Unidade Acadêmica Especial de Biotecnologia, Universidade Federal de Goiás – Regional Catalão, Catalão – GO.

Este trabalho é desenvolvido para a disciplina de Arquitetura de Computadores pela Universidade Federal de Goiás. O trabalho visa no desenvolvimento de 3 projetos, onde cada projeto visa trazer conhecimentos em diferentes da área disciplina. Estaremos abordando diferentes temas: PWM, Sleep Mode, Comunicação SPI.

**Palavras-chave:** Arquitetura, Computadores, Microcontroladores, PWP, SPI.

# ABSTRACT

NOME PARA REFERÊNCIA. **Arquitetura de Computadores**.

. 41 p. Monografia (Graduação) – Unidade Acadêmica Especial de Biotecnologia, Universidade Federal de Goiás – Regional Catalão, Catalão – GO.

This work is developed for a discipline of Computer Architecture by the Federal University of Goiás. The work aims at the development of 3 projects, where each project aims to bring knowledge in different disciplines area. We will be addressing different topics: PWM, Sleep Mode, SPI Communication.

**Keywords:** Architecture, Computer, Microcontrollers, PWP, SPI .

# LISTA DE ILUSTRAÇÕES

# LISTA DE CÓDIGOS-FONTE

# SUMÁRIO

CAPÍTULO

1

# INTRODUÇÃO

Neste trabalho serão apresentados alguns projetos que foram construídos sobre a linguagem C, utilizando de algumas funções específicas para uso em microcontroladores.

São projetos simples com o objetivo de simular o uso de protocolos e funções específicos, como por exemplo MLP (Modulação por lagura de pulso), uso de CPU em modo sleep mode e uso de interrupções, padrão Serial Peripheral Interface (SPI).

O principal objetivo dos projetos é proporcionar a experiencia do aluno trabalhar com os registradores direto para realização dos projetos. 3 temas básicos serão discutidos: Modulação por largura de puslo (MLP), Modos de economizar a energia em circuito utilizando Sleep Mode, Formas de dois dispositivos se comunicarem utilizando o padrão Serial Peripheral Interface (SPI).

CAPÍTULO

2

# PROEJETO 1: FADING LED

Desenvolva um projeto que através de um potenciômetro aumente ou diminua a frequência de uma determina saída X utilizando PWM.

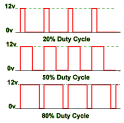
## 2.1 O que é PWM ?

A técnologia Pulse Width Modulation (PWM) ou no português Modulação de Largura de Pulso permite que os microcontroladores atenuem as luzes, controle a velocidade de motores e geram tensões análogicas.Isso é feito alterando o comprimento do pulso, permitidno assim a saída ser controlada.

Neste projeto iremos controlar a intesidade de brilho de um LED utilizando a saída (PINB0)) PWM do microncontralador: atmega8.

O pulso ocorre em uma frequência regular, neste caso na freqüência de modulação. Chamamos de Ciclo de trabalho (Dutty Cicle) a razão entre o tamanho do pulso pelo periodo de tempo. Logo, quanto maior é o tempo de trabalho maior também será a saída. Porntato, quando alteramos essa saída que alimenta o LED, alteramos a intensidade do seu brilho também.

Figura 1 – Duty Cicle




Fonte: avrprojects (2017).

## 2.2 PWM no atmega8

Todos os projetos aqui desenvolvidos serão utilizando o microcontrolador: ATMega8. Ele pode ser usado para gerar sinais PWM. Um microcontrolador como o ATMega8 tem três canais de hardware PWM a bordo do chip. Para o desenvolvimento deste trabalho estaremos utilizando o sinal PWM que está no pino PORTBO.

O PWM de hardware pode ser programado configurando os registradores do temporizador. O ATMega8 tem três registradores de temporizador que você precisa definir para programar o PWM:

1. O registrador TCCR1A deverá colocar o temporizador no modo PWM.
2. Os registradores TCNT1H e TCNT1Lsão usados para ajustar a freqüência de modulação
3. O registrador OCR1A é usado para ajustar o ciclo de trabalho.

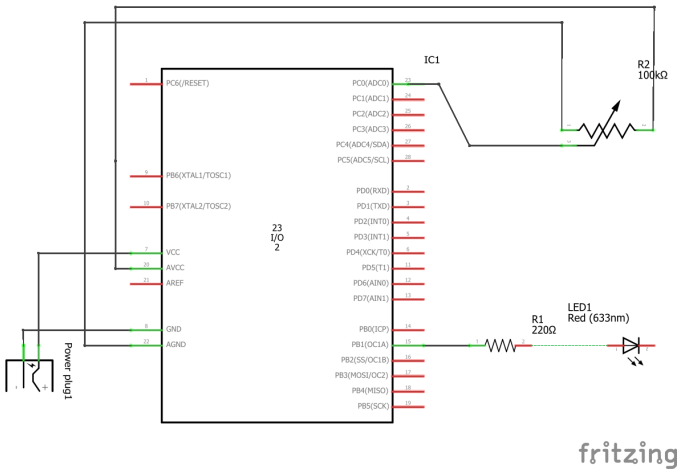
## 2.3 Esquemático e Montagem

O circuito consiste em ligar um microcontrolador ATMega8, onde que iremos ligar um LED conectador no pino PORTB0, através de um resistor de 220 ohm.

Pretendemos variar o brilho do LED através dos valores fornecidos pelo potênciometro conectado no pino PORTC0. Os valores serão lidos pela porta análogica do ATMega8, que neste caso se encontra no PORTC0.

Veja o esquematico da montagem na figura 2:

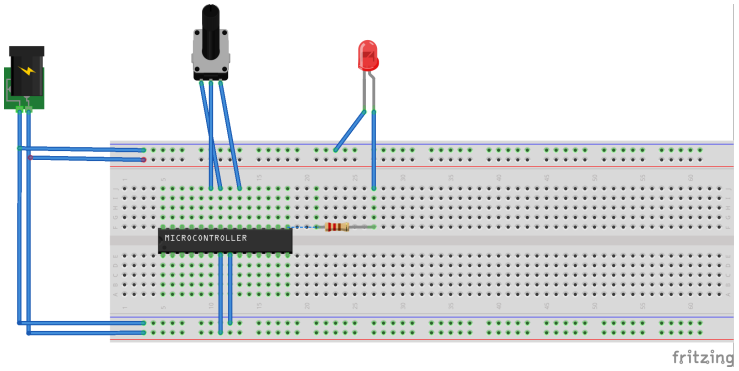
Figura 2 – Esquemático: Fading LED




Fonte: Elaborada pelo autor.

Veja também o circuito montado na protoboard na figura 3:

Figura 3 – Montagem: Fading LED




Fonte: Elaborada pelo autor.

## 2.4 Código em C

**Código-fonte 1** – Fading LED

```
1:
2:   /*
       ***************************************************************

3:   *       Fading Led
4:   *
5:   * Universidade Federal de Goiás
6:   * Microcontrolador Utilizado: AVR ATMega8
7:   * Grupo: Marco Túlio / Vitor do Vale Bernardo / Pablo Silva
8:   * Descrição: O codigo recebe os valores lido na entrada PWM
       atráves de um/
9:   *           Potênciometro e irá ligar o LED proporcionalmente.
10:  *
11:
       ***************************************************************
       */
12:
13: #define FOSC 1000000UL// Clock Speed
14: #define BAUD 1200
15: #define MYUBRR ((FOSC/ (BAUD * (long)16)))
16:
17: #include <avr/io.h>
18: #include <util/delay.h>
19: #include <util/setbaud.h>
20: #include <avr/eeprom.h>
21: #include <avr/interrupt.h>
```

```
22:
23: void USART_Init( unsigned intubrr);
24: void USART_Transmit(unsigned char data);
25: unsigned char USART_Receive(void);
26: uint16_t ReadADC(uint8_t ch);
27: void InitADC();
28:
29:
30: int main(void)
31: {
32:
33:   //PWM Initialisation
34:   TCCR1A = 0b10000001; // fast PWM mode 8-bit on OC1A
35:   TCCR1B = 0b00001010; // prescaling by 8
36:
37:   //Initial value;
38:   OCR1A = 0x00;
39:   DDRB = 0xFF; // set port B for output
40:
41:
42:   unsigned char lei;
43:   InitADC();
44:   USART_Init(MYUBRR);
45:   while(1)
46:   {
47:     OCR1A =ReadADC(0);
48:     _delay_ms(10);
49:   }
50: }
51:
52: void USART_Transmit(unsigned char data)
53: {
54:   while( !( UCSRA & (1<<UDRE)) );
55:   UDR = data;
56: }
57:
58:
59: void USART_Init(unsigned int ubrr)
60: {
61:   UBRRH=(unsigned char)(ubrr>>8);
62:   UBRRL=(unsigned char)ubrr;
63:   UCSRB=(1<<RXEN)|(1<<TXEN);
```

```
64:   UCSRC=(1<<URSEL)|(1<<USBS)|(3<<UCSZ0);
65: }
66:
67: unsigned char USART_Receive(void)
68: {
69:   while(!(UCSRA & (1<<RXC)));
70:   return UDR;
71: }
72:
73:
74: void InitADC()
75: {
76:   ADMUX=(1<<REFS0);              // For Aref=AVcc;
77:   ADCSRA=(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0); //
       Rrescalar div factor =128
78: }
79:
80:
81: uint16_t ReadADC(uint8_t ch)
82: {
83:
84:   //Seleciona o canal de leitura do microcontrolador
85:   ch=ch&0x07;
86:   ADMUX|=ch;
87:   //inicia a conversão
88:   ADCSRA|=(1<<ADSC);
89:   //Aguarda a conversão
90:   while(!(ADCSRA & (1<<ADIF)));
91:   //Limpa a flag
92:   ADCSRA|=(1<<ADIF);
93:   //retorna o dado
94:   return(ADC);
95: }
```

CAPÍTULO

3

# PROEJETO 2: SLEEP MODE

Desenvolva um projeto que desligue um LED e mantenha desligado em um determinado período de tempo. No período em que o LED estiver desligado, todo o sistema deve entrar em sleep mode. Depois de um determinado tempo, o sistema deve ser ativado e ligar o LED novamente.

## 3.1 O que é Sleep Mode?

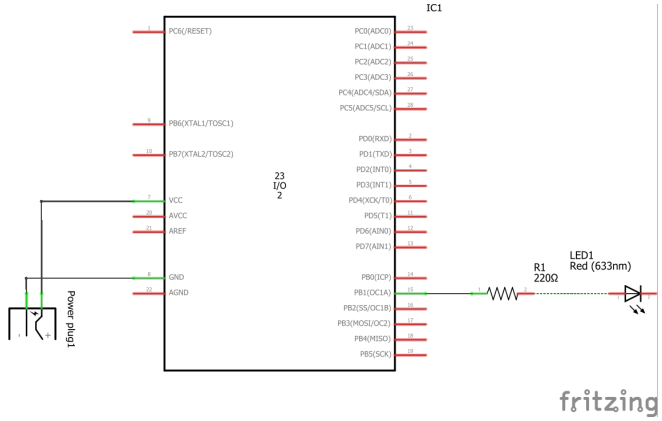
Podemos utilizar a instrução SLEEP para reduzir fortemente o consumo de energia por uma determinada aplicação. Os dispositivos AVR podem ser colocados em diferentes modos de sleep, neste caso o dispositivo AVR que estamos trabalhando é ATMega8.

A biblioteca sleep.h possui diferentes macros para colocar o dispositivo em modo de suspensão. A forma mais simples é opcionalmente configurando o modo de suspensão desejado utilizando a função: Set Sleep Mode, (O padrão é o modo ocioso onde que CPU é coloca no modo de suspensão, mas todos os relógios e periféricos ainda estão em execução) e em seguida devemos ativar o modo de sleep usando a função: sleep$_mode$.

## 3.2 Esquemático e Montagem

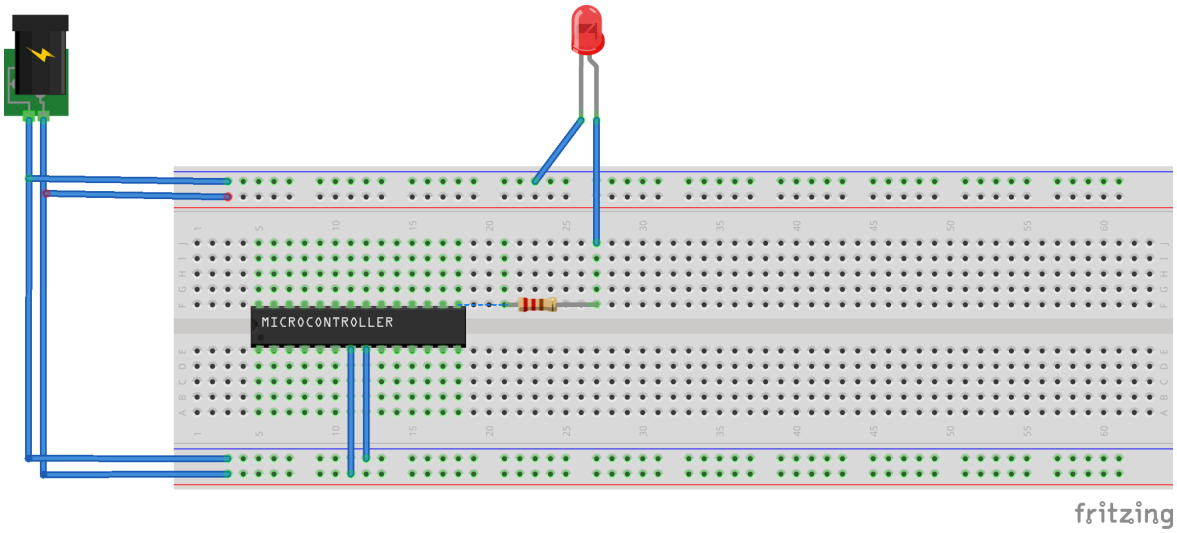
Este projeto tem como objetivo simular o uso do “sleep mode” em um circuito. Quando este modo está ativado, todo o sistema fica em espera, com um baixo uso de recursos, aguardando para voltar ao estado normal quando requisitado. Para o auxílio do projeto, será utilizado um LED para sinalizar quando o sistema entrou e saiu de fato do “sleep mode”. O LED ligado indica que o sistema está operando normalmente. Ao desligar o LED, o sistema deve então simultaneamente entrar em “sleep mode” durante um tempo determinado previamente. Após o fim desse tempo, o sistema então retornará ao seu estado normal e o LED deve ser religado.

Figura 4 – Esquemático: Sleep Mode




Fonte: Elaborada pelo autor.

Figura 5 – Montagem: Sleep Mode




Fonte: Elaborada pelo autor.

**Código-fonte 2** – Led On/Off in Sleep Mode

```
1:
2:
3:  /*
      ***************************************************************************
4:  *       Led On/Off - Utlizando Sleep Mode
5:  *
6:  * Universidade Federal de Goiás
7:  * Microcontrolador Utilizado: AVR ATMega8
8:  * Grupo: Marco Túlio / Vitor do Vale Bernardo / Pablo Silva
9:  * Descrição: Ligar e desligar um LED utilizando o modo sleep.
```

```
10:  *
11:
       **********************************************************************
       */
12:
13: #include <avr/io.h>
14: #include <avr/sleep.h>
15: #include <avr/interrupt.h>
16: #include "util/delay.h"
17:
18: int main(void)
19: {
20:   DDRB = 0xFF;
21:
22:    // infinite main loop
23:    while (1)
24:    {
25:
26:   PORTB = 0x01; //  Liga Led na porta 01
27:   _delay_ms(2000); // Espera 2 segundo
28:   set_sleep_mode(SLEEP_MODE_PWR_SAVE);
29:   cli();  // desligar interrupções
30:   sleep_enable();  // Ativa o Sleep Mode
31:   sei(); //Liga as interrupções
32:   sleep_cpu();
33:   cli();
34:   _delay_ms(5000);
35:   sleep_disable();
36:   sei();
37:
38:    }
39:
40: }
```

CAPÍTULO

4

# PROJETO 3: COMUNICAÇÃO SPI

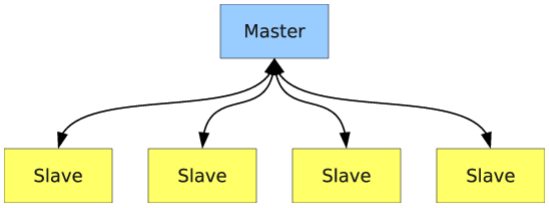
Desenvolva um projeto que realize a comunicação entre dois microcontroladores utilizando o padrão SPI.

## 4.1 O que é o Padrão SPI?

Neste trabalho iremos mostrar uma das diversas tecnologias diferentes existentes para fazer a comunicação serial entre dois dispositivos. Durante o discorrer deste estaremos trabalhando com o padrão Serial Peripheral Interface (SPI).

Na comunicação serial síncrona definimos também o conceito de Mestre-Escravo. Normalmente o gerador do sinal de sincronismo é definido como o Mestre (Master) da comunicação. Para os dispositivos que utilizam do sinal de sincronismo gerado damos a definição de Escravo (Slave). A ligação mais comum desse tipo de comunicação é um Master e vários Slaves.

Figura 6 – Master and Slaves




Fonte: embarcados (2017).

Os pinos básicos de comunicação entre dispositivos utilizando o protocolo SPI e o esquema padrão de ligação são dados conforme abaixo:

O sinal de SS funciona como Seleção de Escravo (Slave Select). É um sinal ativo em nível baixo, o que significa que o dispositivo é selecionado quando este pino se encontra em nível

Figura 7 – Pinos do SPI

| Pino | Nome Padrão | Significado | Nomes Alternativos |
|---|---|---|---|
| Do Master para o Slave | MOSI | Master Output Slave Input | SDO, DO, SO |
| Do Slave para o Master | MISO | Master Input Slave Output | SDI, DI, SI |
| Clock | SCLK | Serial Clock | SCK, CLK |
| Seleção de Slave | SS | Slave Select | CS, nSS, nCS |

Fonte: embarcados (2017).

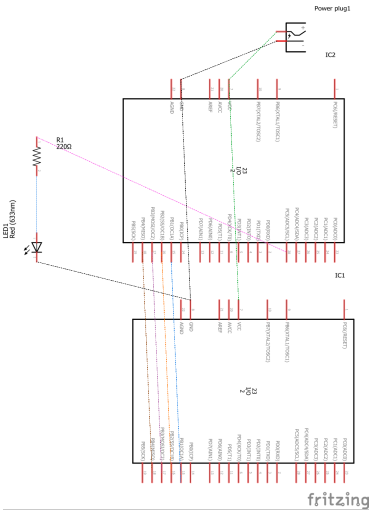
baixo. No entanto, muitos dispositivos utilizam este sinal como sincronismo de frame. Dessa forma, é um sinal importante que deve ser respeitado.

## 4.2 Esquemático e Montagem

Neste projeto realizaremos a comunicação entre dois microcontroladores distintos com o uso do padrão Serial Peripheral Interface (SPI), que trata-se de um protocolo que permite a comunicação de um microcontrolador com diversos outros componentes, formando uma rede. Os dois microcontroladores utilizados serão identificados por mestre e escravo (master / slave). O mestre será encarregado de enviar os comandos através da comunicação serial, enquanto o escravo é encarregado de receber, interpretar e executar uma determinada ação com base nos dados recebidos. Os comandos enviados de um microcontrolador são um indicador de qual será o estado de um LED, ligado ou desligado.

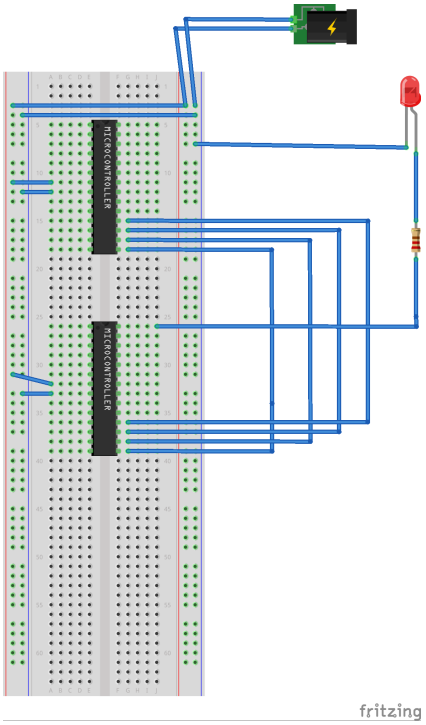
Veja o esquemático na imagem abaixo:

Figura 8 – SPI: Esquemático da montagem




Fonte: Elaborada pelo autor.

Figura 9 – SPI: Montagem na Protoboard




Fonte: Elaborada pelo autor.

## 4.3 Código em C

O código para este projeto é dividio em duas partes: Código do Master e o Código do Slave.

**Código-fonte 3** – MASTER - Comunicação via SPI entre dois dispositivos

```
1:
2:  /*
      ************************************************************************
3:  *       Comununicação entre dois Microcontroladores Utilizando
       SPI (Master)
4:  *
5:  * Universidade Federal de Goiás
6:  * Microcontrolador Utilizado: AVR ATMega8
7:  * Grupo: Marco Túlio / Vitor do Vale Bernardo / Pablo Silva
8:  * Descrição: Codigo Master irá enviar comandos pela comunicaçã
      o SPI para o slave
9:  *              O comando ira dizer o estado do LED: ligado ou
      desligado.
10: *
```

```
11:
      ***********************************************************************
      */
12:
13:
14: #define F_CPU 4000000UL           // clock do microcontrolador
15: #include <avr/io.h>
16: #include <util/delay.h>
17: #include <avr/interrupt.h>
18:
19: ////////////////////////////configuração dos pinos utilizados
      ////////////////////////////////////
20:
21: #define MOSI         PB3
22: #define MISO         PB4
23: #define SCK          PB5
24: #define SS           PB2
25:
26: //
      ////////////////////////////////////////////////////////////////////////////////

27:   void SPI_inicializa(void)
28:   {
29:       DDRD =0xFF;
30:       DDRB |= ((1<<MOSI)|(1<<SCK)|(1<<SS));    //MOSI, SCK and
      SS são saidas(se for usar dois mestres então deve-se o SS
      como entrada)
31:       DDRB &= (~(1<<MISO));                    //MISO é entrada
32:       PORTB |= (1<<SS);                        //inicia com SS
      em nivel alto
33:       //SPE : habilita SPI
34:       //MSTR: modo  Master
35:       //SPIE:  habilita interrupção de SPI
36:       //SPCR = ((1<<SPE)|(1<<MSTR)|(1<SPR1)|(1<SPR0));    //(1<
      SPR1)|(1<SPR0) : FOSC/128
37:       SPCR = ((1<<SPE)|(1<<MSTR)|(1<SPR0));    //(1<SPR0) :
      FOSC/16
38:   }
39:
40:   void SPI_envia_byte(char dados)
41:   {
42:       PORTB &= (~(1<<SS));    //coloca SS em nivel baixo (0),
```

```
      para transferir dados
43:       SPDR = dados;         //inicializa transferencia
44:       while(!(SPSR & (1<<SPIF)));     //espera fim de transmissã
      o
45:       PORTD = SPDR;//escreve no port D o dado recebido
46:       PORTB |= (1<<SS);     //coloca SS em nivel alto (1), pois
      é o fim da transmissão.
47:       _delay_ms(1);//tempo para slave colocar dados no
      registrador
48:   }
49:
50:   int main(void)
51:   {
52:
53:     SPI_inicializa();//inicializa SPI
54:     sei();//habilita interrupções
55:
56:     while(1)
57:   {
58:       SPI_envia_byte(0X01); /*** Envia o primeiro comando:
      LIGAR LED***/
59:       _delay_ms(1000); /*** Delay de 1 segundo ***/
60:      SPI_envia_byte(0x00); /*** Envia o segundo comando:
      DESLIGAR LED ***/
61:       _delay_ms(1000);  /*** Delay de 1 segundo ***/
62:   }
63:
64:     return 0;
65:   }
```

**Código-fonte 4** – SLAVE - Comunicação via SPI entre dois dispositivos

```
1:
2:
3:  /*
      ************************************************************
4:  *       Comununicação entre dois Microcontroladores Utilizando
       SPI (Slave)
5:  *
6:  * Universidade Federal de Goiás
7:  * Microcontrolador Utilizado: AVR ATMega8
```

```
 8: * Grupo: Marco Túlio / Vitor do Vale Bernardo / Pablo Silva
 9: * Descrição: Codigo Slave irá receber os dados enviados pelo
    Master e colocara no PORTD,
10: *            Ligando e desligando o LED.
11: *
12:
    ********************************************************************
    */
13:
14:   #define F_CPU 4000000UL  // Definindo o Clock do
    Microcontrolador
15:   #include <avr/io.h>
16:   #include <avr/interrupt.h>
17:
18:   /*********************************
19:   *  Definindo os Registradores  *
20:   *********************************/
21:
22:   #define MOSI     PINB3 /*******Definindo o PINB3 como
    entrada********/
23:   #define MISO     PINB4 /********Definindo o PINB4 como saída
    ********/
24:   #define SCK      PINB5 /********Definindo o PINB5 como
    entrada********/
25:   #define DDR_SPI  DDRB
26:
27:   char enviar=0;
28:
29:   /*********************************
30:   *  Inicializando          *
31:   *********************************/
32:
33:   void SPI_Slave_inicializa(void)
34:   {
35:       DDR_SPI = (1<<MISO);
36:       SPCR = (1<<SPE)/*|(1<<CPOL)*/|(1<<SPIE); /****Aviva o
    SPI / polaridade do clock / habilita interrupção de SPI ***
    */
37:   }
38:
39:   /**** Vetor de Interrupção *****/
40:   ISR (SPI_STC_vect)
```

```
41:     {
42:       SPDR = enviar; //envia dado anteriormente recebido
43:       PORTD = SPDR;
44:     PORTC = SPDR;
45:       enviar = SPDR;
46:     }
47:
48:     /*********************************
49:     *  Função Principal        *
50:     *********************************/
51:     int main(void)
52:     {
53:         SPI_Slave_inicializa();
54:         DDRD = 0xFF;
55:         DDRC = 0XFF;
56:       sei();
57:
58:       while(1){
59:
60:       }
61:
62:     }
```

# REFERÊNCIAS

AVRPROJECTS. **PROECFading LED PWM**. 2017. Disponível em: <http://www.avrprojects.net/images/fading_led_pwm_pic1.png>. Acesso em: 06 MAR. 2017. Citado na página 23.

EMBARCADOS. **Comunicação SPI**. 2017. Disponível em: <https://www.embarcados.com.br/wp-content/uploads/2014/04/master_slave.png>. Acesso em: 06 MAR. 2017. Citado nas páginas 33 e 34.