DANIEL MADEIRA BUENO

Manual de implementação e simulação de projetos utilizando *Arduino UNO* e *Proteus/ISIS*

Manipulação de *LEDS*

Vitória – ES
NOV/2015

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

O Proteus é um software criado pela empresa Labcenter Eletronics, este proporciona ao usuário a possibilidade de montar circuitos eletrônicos, realizar simulações e ainda criar placas de circuito impresso. Um dos componentes do Proteus é o ambiente de desenvolvimento ISIS que tem como função principal auxiliar o desenvolvedor de sistemas embarcados na verificação da parte eletrônica referente aos circuitos dos mesmos, além de possibilitar a realização de simulações (como dito anteriormente) antes de elaborar o hardware de um determinado projeto, podendo proporcionar portanto uma redução no tempo de desenvolvimento e até mesmo do custo envolvido.

POR QUE UTILIZAR O *PROTEUS/ISIS*?

Além dos motivos citados anteriormente, para muitas pessoas não é possível desenvolver todas as ideias que surgem na cabeça por vários motivos, um deles pode ser a falta de acesso aos materiais e equipamentos para o criação de protótipos e realização de testes. Por outro lado, existem ferramentas criadas para justamente suprir esta carência, entre elas está o *Proteus/ISIS*, este conta com diversas ferramentas que estão presentes em laboratórios de eletrônica como osciloscópio, gerador de sinais, voltímetro, amperímetro, além de uma infinidade de componentes como resistores, capacitores, transistores, motores, microcontroladores e outros (muitos outros).

QUAL O OBJETIVO DESTE DOCUMENTO?

Este documento visa ao proporcionar o aprendizado básico para a manipulação deste software bem como uma maneira bastante didática e simples de como realizar pequenos projetos envolvendo a utilização do *Arduino UNO* para aqueles que desejam aprender um pouco sobre o assunto e no momento não possuem conhecimento algum sobre o mesmo. Obviamente existem limitações no que diz respeito à implementação de projetos utilizando o *Proteus/ISIS* (no decorrer dos projetos apresentados serão discutidas algumas limitações caso estas sejam importantes no momento), no entanto, ainda é possível realizar bastante coisa.

O QUE ESTE DOCUMENTO ABRANGE?

Num primeiro momento apresenta-se um pouco da interface do *Proteus/ISIS* e como fazer para selecionar componentes, instalar bibliotecas de componentes, definir propriedades de componentes entre outros.

Posteriormente será mostrado como fazer para importar o código desenvolvido para dentro do *Proteus/ISIS* e simula-lo em um *Arduino UNO* virtual.

Serão desenvolvidos também vários projetos básicos organizados para que exista uma certa progressão no conhecimento adquirido para criação dos mesmos, facilitando portanto, o aprendizado de maneira bastante simples. Neste documento encontram-se projetos envolvendo os *leds* como elementos principais. Esta escolha é justificada pelo fato de que estes são os componentes mais básicos para iniciar os estudos à cerca do desenvolvimento de códigos para microcontroladores em geral. Posteriormente serão adicionados novos projetos envolvendo outros elementos.

SOBRE O AUTOR

Meu nome é Daniel Madeira Bueno, sou engenheiro eletricista graduado pela Universidade Federal do Espírito Santo. Gosto bastante de questões que envolvem sistemas embarcados porém não sou profissional no assunto, portanto organizei este conteúdo da forma que geralmente eu consigo aprender melhor e resolvi compartilhar. Se possível peço que caso existam sugestões, pontos para corrigir ou qualquer outra coisa, entre em contato comigo, ficarei extremamente agradecido em poder aprimorar meu conhecimento e ajudar os outros da mesma forma.

Meu e-mail é daniel_m_bueno@hotmail.com

CONHECIMENTOS BÁSICOS

PROTEUS/ISIS

Primeiramente para entrar no ambiente ISIS deve-se clicar no respectivo ícone localizado na barra de atalhos localizada no canto superior esquerdo. Após realizar este procedimento será aberto o ambiente utilizado para a montagem dos circuitos bem como para realizar as simulações.

Figura 1 – Ícone do ISIS na barra de ferramentas do Proteus

Deve-se ter em mente que existem algumas formas diferentes de realizar a simulação do *Arduino UNO* no *Proteus/ISIS*. Uma delas é procurar o nome do microcontrolador do *Arduino UNO* dentro da biblioteca de componentes do próprio software, no entanto para que este funcione corretamente é necessário adicionar componentes como por exemplo um oscilador de cristal e capacitores, bem como definir várias configurações pertinentes para a devida utilização deste.

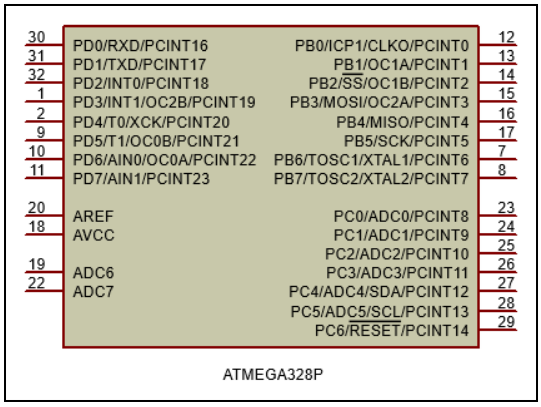

Figura 2 – Microcontrolador ATMEGA328P

Uma outra forma consiste em utilizar a placa do Arduino UNO já pronta dispensando a necessidade de elementos extras, vale lembrar que existem várias bibliotecas contendo placas deste tipo, neste documento será utilizada uma biblioteca que pode ser adquirida em http://blogembarcado.blogspot.com.br/. Para instalar a biblioteca basta fazer o *download* da mesma e copiar os arquivos dentro da pasta *LIBRARY* contida no mesmo local onde foi instalado o *Proteus*.

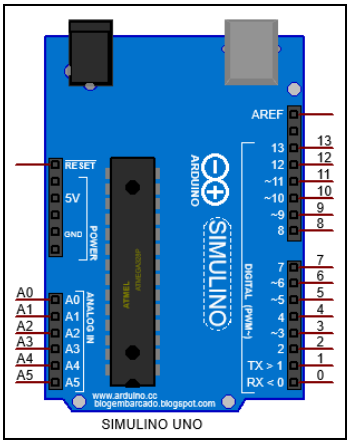

Figura 3 – Placa virtual do *Arduino UNO – SIMULINO UNO*

Para adicionar a placa do *Arduino UNO* que será utilizada deve-se clicar no ícone relativo aos componentes contido na barra de ferramentas localizada originalmente no canto esquerdo da tela. Após este procedimento, clica-se no botão "*P*" para que seja aberta a lista de busca de componentes, estes podem ser localizados através do nome ou por categorias onde são agrupados. Utilizando a biblioteca citada, basta procurar por "Arduino (Blog Embarcado)" e escolher o "*SIMULINO UNO*".

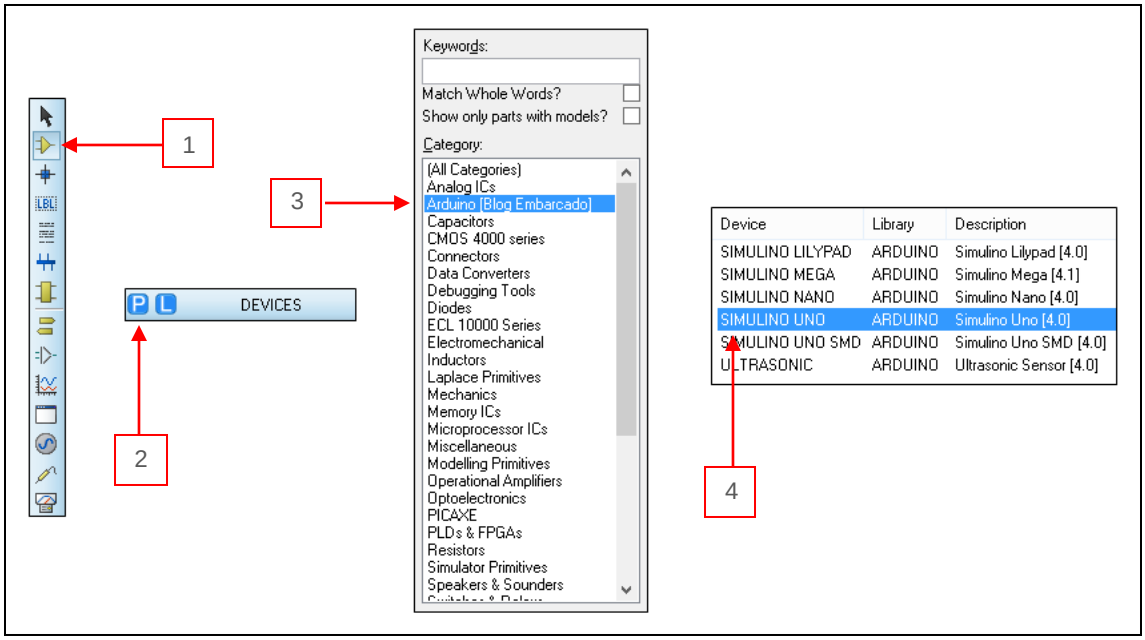

Figura 4 – Adicionando componentes no Proteus

ARDUINO IDE

A IDE do Arduino será utilizada para criar os programas e consequentemente gerar um arquivo com a extensão HEX necessário para a simulação de circuitos com o microcontrolador presente na placa do *Arduino UNO*. Assim que a IDE for iniciada deve-se acessar o item "Preferências/*Preferences*" através do menu "Arquivo/*File*" localizado na barra de ferramentas e marcar a caixa onde está escrito "compilação/*compilation*".

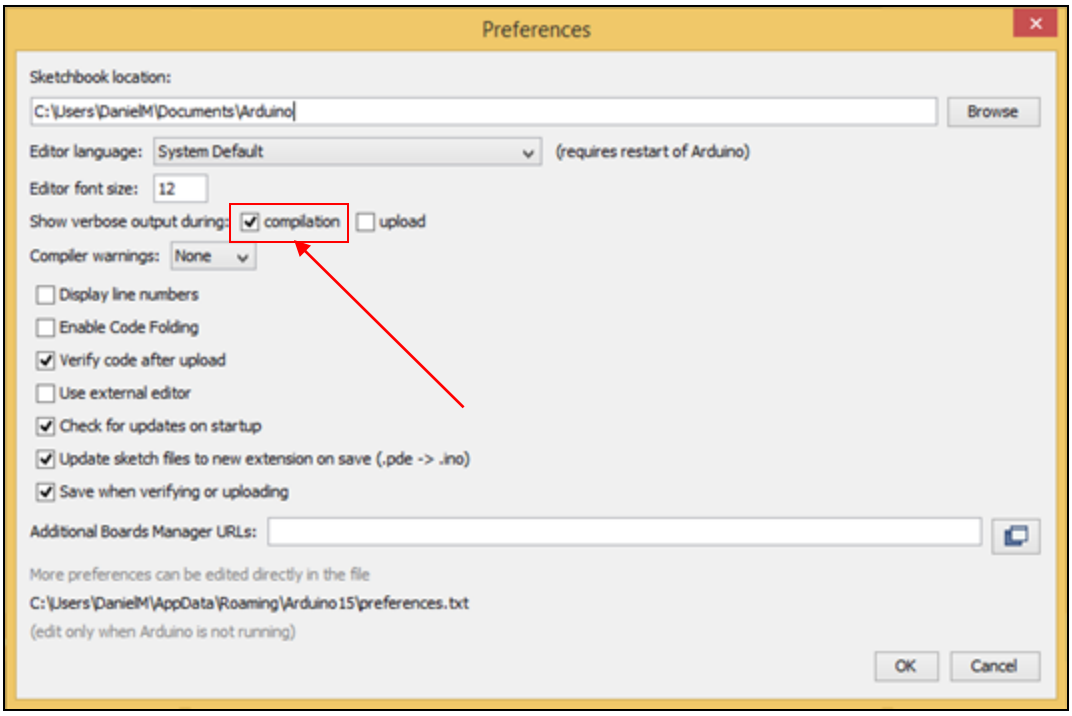

Figura 5 – Alteração na IDE do Arduino

Além desta alteração deve-se também acessar o item "Placa/*Board*" através do menu "Ferramentas/*Tools*" e selecionar a opção referente ao *Arduino UNO*. Neste ponto a IDE está configurada para que seja possível realizar as simulações, bastando simplesmente importar o arquivo com extensão *hex* gerado (este passo será explicado após o primeiro código ser desenvolvido).

MANIPULAÇÃO DE *LEDS*

Neste parte serão apresentados alguns projetos básicos utilizando *leds* de várias maneiras, proporcionando a possibilidade de aprendizado sobre como controlar saídas do Arduino e entradas simples, como por exemplo, botões pressionados. Na questão do hardware, o leitor poderá aprender sobre *leds*, botões e resistências, incluindo resistores de pull-up e pull-down resistores, importantes para garantir que a entrada dispositivos são lidos corretamente. De acordo com o criação dos projetos serão apresentados vários conceitos à respeito de como realizar a programação para o *Arduino UNO*.

1 ACIONAMENTOS BÁSICOS DE LEDS

1.1 ACIONAMENTO SIMPLES

1.1.1 Objetivo

Assim como em outras linguagens de programação utilizadas para diversas finalidades, temos igualmente que propor um primeiro projeto básico análogo ao conhecido *Hello World*. Este consiste em realizar o acionamento de um *led* externo ligado à um dos pinos de *I/O* digitais existentes na placa do *Arduino UNO* ao invés de utilizar o *led* já embutido no pino 13 da mesma. Além de desenvolver de forma clara este simples projeto, espera-se proporcionar o aprendizado também das funções ***setup()***, ***pinMode()***, ***digitalWrite()*** bem como das diretivas *#define* e algumas instruções de como manipular os elementos no Proteus/ISIS e adicionar o arquivo *.hex* ao microcontrolador para realizar a simulação.

1.1.2 Hardware utilizado na simulação

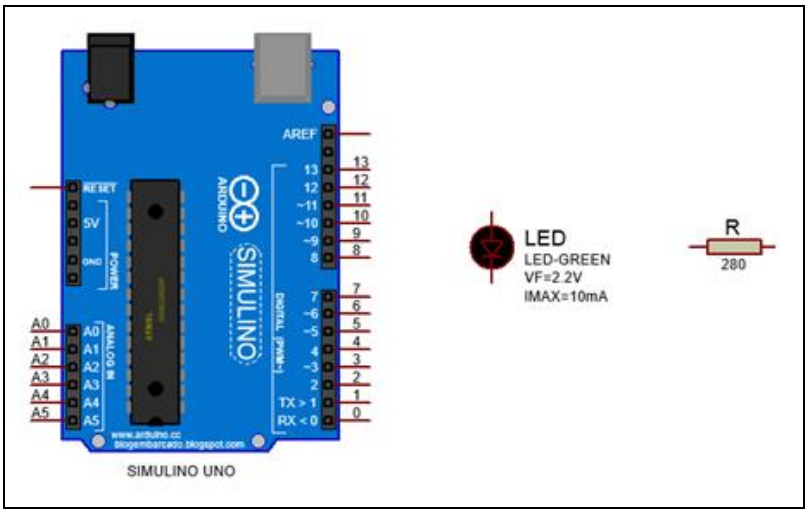

Figura 6 – Hardware utilizado no item 1.1

Como ressaltado anteriormente, na realização deste projeto será utilizado o *SIMULINO UNO*, um ambiente de simulação do *Arduino UNO* no *Proteus/ISIS*. Este permite implementações de diversos tipos de projetos devido à simplicidade que traz através da sua fácil manipulação.

Além do elemento citado anteriormente, será utilizado também um *led*, que pode ser adicionado ao circuito através do mesmo procedimento descrito para o *Arduino UNO* na parte de conhecimentos básicos. Neste projeto optou-se por um *led* verde (o qual pode ser selecionado através do nome *LED-GREEN* no campo

de pesquisa), cuja tensão de funcionamento é 2,2 volts e a corrente máxima que pode ser percorrida através dele são 10mA. Observe que estes e outros parâmetros podem ser alterados por meio da configuração das propriedades do elemento (estas podem ser acessadas com um duplo clique sobre o elemento), conforme a figura a seguir

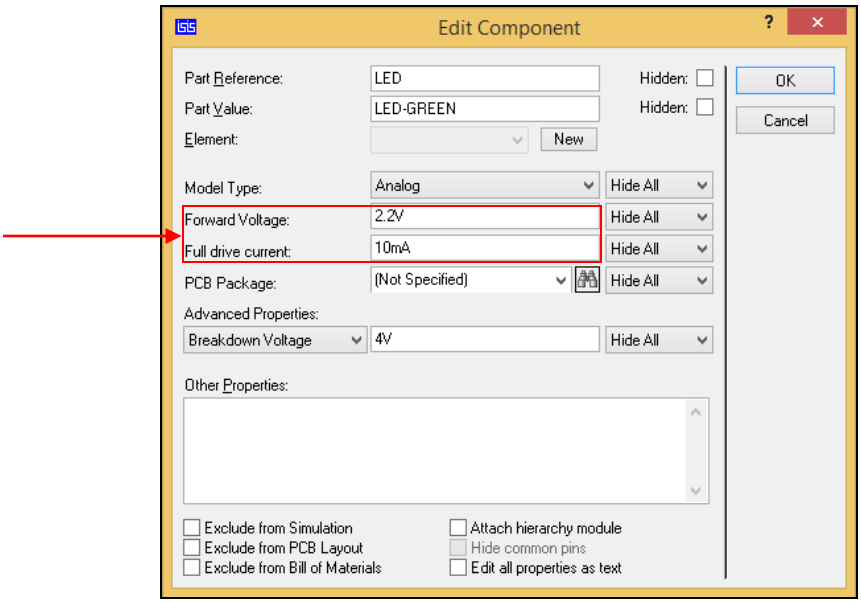

Figura 7 - janela de propriedades do led

Por último, será utilizado um resistor (que pode ser localizado através do nome *RES* no campo de pesquisa) responsável por limitar a corrente que irá passar pelo *led*. De acordo com o *datasheet*, sabe-se que as saídas digitais do *Arduino UNO* disponibilizam em suas saídas, uma tensão de 5 volts e um valor de até 40mA de corrente. Sendo assim para assegurar as condições de funcionamento do *led* utilizado (de acordo com os quesitos definidos na sua janela de propriedades), deve-se escolher um determinado valor de resistência para o resistor em questão e registrá-lo nas propriedades do componente, assim como feito para o *led*. O cálculo do valor da resistência é dado da seguinte maneira:

$$R = \frac{Vard - Vled}{Imáx} = \frac{5 - 2{,}2}{10\ x\ 10^{-3}} = 280\Omega$$

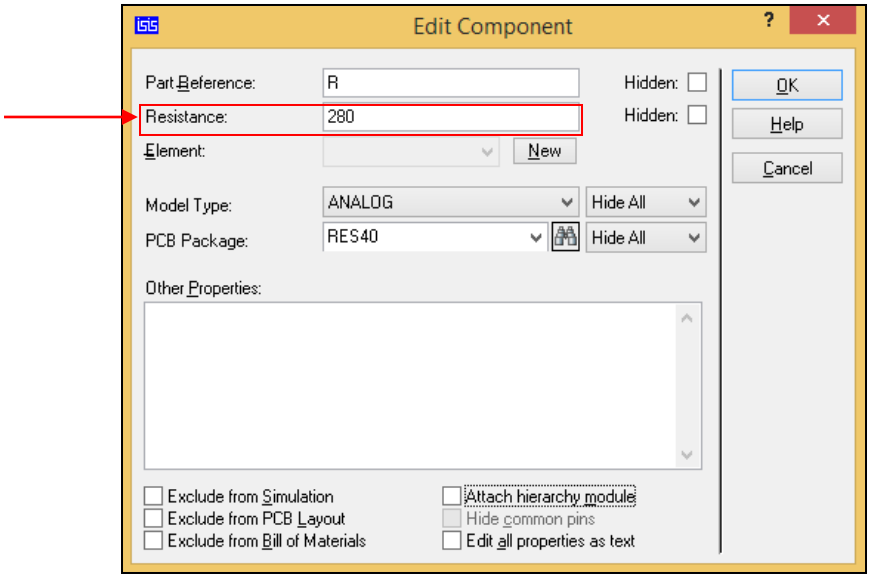

Figura 8 – Janela de propriedades do resistor

1.1.3 Diagrama esquemático das ligações

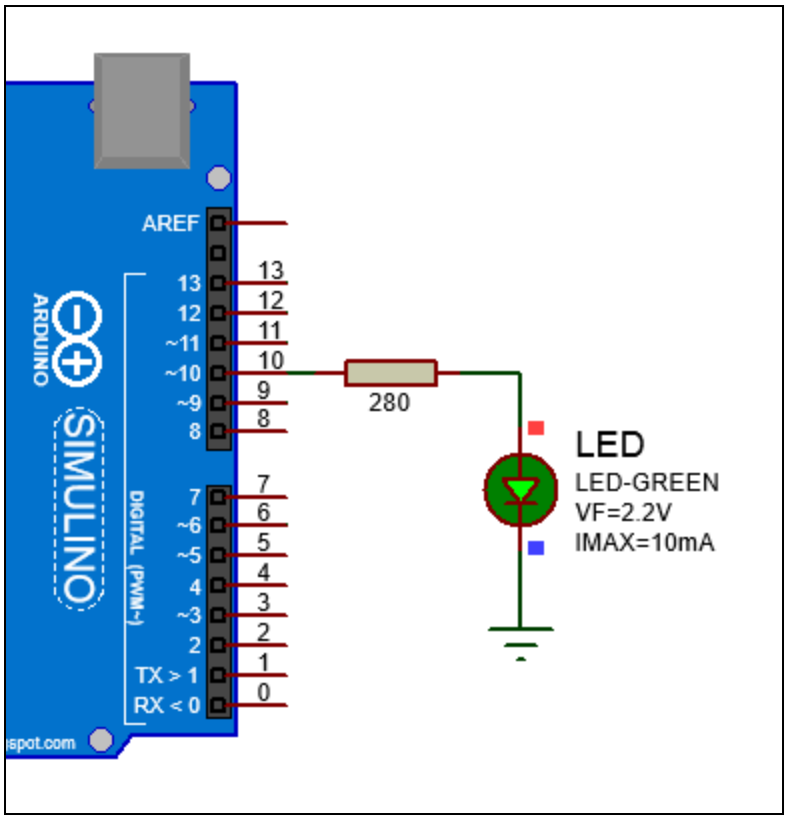

Figura 9 – Diagrama esquemático utilizado no item 1.1

1.1.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente começamos o código utilizando a diretiva *#define* que por sua vez é um componente muito utilizado em C, este permite ao programador dar um nome à uma constante antes do programa ser compilado. Neste programa atribuímos o valor 10 à palavra LED pois conforme o esquemático mostrado, o *led* em questão foi ligado ao pino 10 (uma das funções deste é atuar como pino de *I/O* digital) do *Arduino UNO*, desta forma podemos realizar todo o desenvolvimento da programação nos referindo ao pino 10 através do nome LED. Uma vantagem do uso deste recurso é que caso queiramos trocar o *led* de pino, basta alterar o valor dado na diretiva *#define* sem necessidade de alterar o código.

```
#define LED 10
```

Um código para o *Arduino UNO* deve ter duas funções principais, são elas: a função ***setup()*** e a função ***loop()***, caso contrário o código não conseguirá ser compilado. Nesta primeira etapa usaremos somente a função ***setup()***, pois ela é utilizada somente uma vez e apenas no início do programa (para inicializar variáveis, modos de operação dos pinos, funções específicas de determinadas bibliotecas, etc...). A função ***loop()*** será apresentada no item 1.2.

```
void setup() {
    pinMode (LED, OUTPUT);
    digitalWrite(LED, HIGH);
}
```

A função ***setup()*** neste momento possuirá duas sentenças necessárias para realização do objetivo deste primeiro item. A primeira delas é dada através da função ***pinMode()***, utilizada para definir o modo de operação de um determinado pino, como entrada ou saída. O primeiro argumento da função corresponde ao pino que se quer determinar o modo de operação, neste caso será o pino 10 (cujo nome atribuído anteriormente é LED), enquanto o seguindo argumento diz respeito ao modo de operação do pino citado, para esta etapa, saída (*OUTPUT*).

A segunda sentença presente na função ***setup()*** é dada pela função ***digitalWrite()***, utilizada para determinar se uma tal saída deve permanecer em nível alto ou nível baixo até que outro elemento do programa altere o mesmo. O primeiro argumento desta função corresponde ao pino de saída digital que se deseja manipular (novamente o pino que queremos é o 10), ao passo que o segundo argumento está relacionado de fato ao nível desejado na saída. Como queremos acender um *led* devemos programar para que a saída disponibilize nível alto, proporcionando cerca 5 volts em seu terminal, fazendo com que o *led* seja acionado.

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

A função ***loop()*** será discutida no item 1.2 como citado anteriormente.

1.1.5 Importando o arquivo .hex

Após realizar o desenvolvimento do código utilizando a IDE disponibilizada para o Arduino, deve-se clicar para realizar o upload.

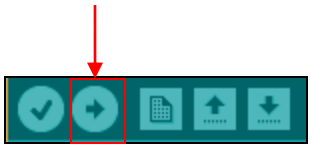

Figura 10 – Barra de ferramentas da IDE do Arduino

Obviamente ocorrerá um erro, em virtude de a placa do *Arduino UNO* não poder ser localizada pela IDE, porém mesmo assim o código será compilado e o arquivo. hex será gerado e estará armazenado no computador como arquivo temporário cujo endereço encontra-se no relatório gerado após a compilação.

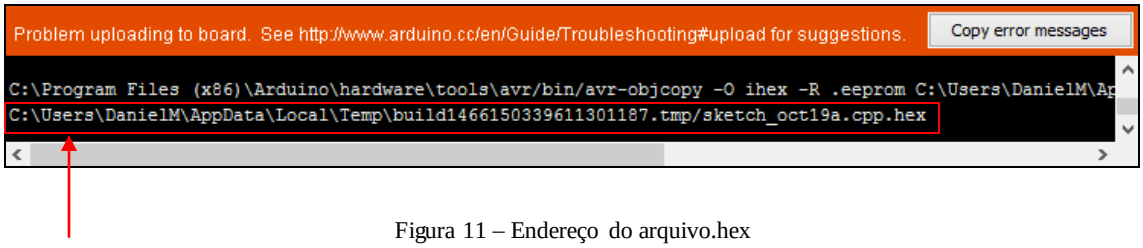

Figura 11 – Endereço do arquivo.hex

Por último basta acessar as propriedades da placa virtual do *Arduino UNO* que foi adicionada no *Proteus/ISIS* e no campo colar o endereço citado (ou o endereço no qual foi colocado o arquivo. hex, caso este tenha sido retirado do seu local de origem) no campo *Program File*.

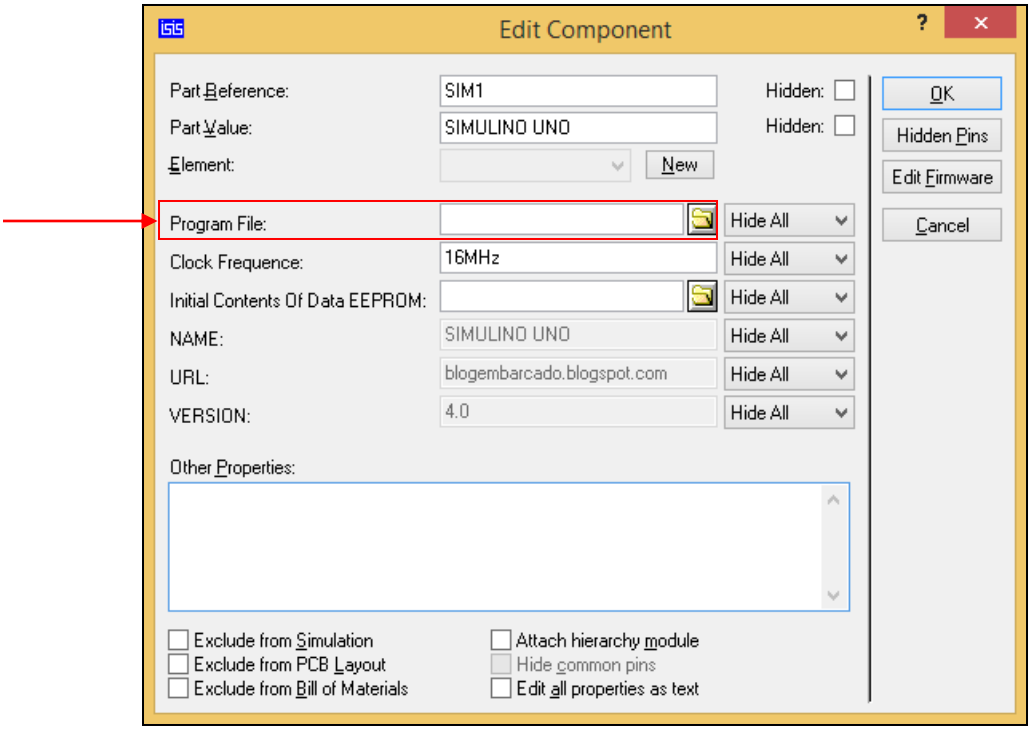

Figura 12 – Adicionando o arquivo.hex no Proteus/ISIS

1.2 ACIONAMENTO TEMPORIZADO

1.2.1 Objetivo

Dando prosseguimento ao projeto realizado no item 1.1, esta aplicação consiste em realizar o acionamento do mesmo *led* externo ligado à um dos pinos digitais existentes na placa do *Arduino UNO*, porém o mesmo deve ser desligado depois de um tempo pré-estabelecido. Nesta etapa, espera-se encadear o aprendizado adquirido no item anterior com o conhecimento que será demonstrado nesta seção, o qual consiste no entendimento das funções ***loop()*** e ***delay()***.

1.2.2 Hardware utilizado na simulação

O hardware utilizado na simulação é o mesmo do item 1.1.

1.2.3 Diagrama esquemático das ligações

O diagrama esquemático das ligações é o mesmo do item 1.1.

1.2.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente começamos o código da mesma maneira do item anterior utilizando a diretiva *#define* para podermos nos referir ao pino 10 utilizando a palavra LED.

```
#define LED 10
```

Em seguida determina-se o modo de operação do pino 10 através da função ***pinMode()*** existente dentro da função ***setup()***.

```
void setup(){
    pinMode(LED, OUTPUT);
}
```

A próxima etapa deste item é fazer com que o *led* funcione como uma espécie de pisca-pisca, sendo ligado e desligado automaticamente e permanecendo em ambos os estados com tempos pré-estabelecidos.

Conforme dito anteriormente, neste momento utilizaremos a função ***loop()***. Tudo que está localizado dentro desta função é executado continuamente, linha por linha desde o início até o final, onde então o código volta a ser executado a partir do seu começo. Este ciclo se repete até o momento em que o *Arduino UNO* é desligado ou resetado.

Dentro da função ***loop()*** existem as funções ***digitalWrite()*** (já demonstrada no item anterior) e a função ***delay()***, utilizada para que o *Arduino UNO* espere um determinado tempo para poder continuar a execução do código. O tempo em questão é passado como parâmetro da função e expresso em milissegundos (neste caso temos 1000 milissegundos ou 1 segundo). Sendo assim, com o auxílio desta função inicia-se o programa acionando o *led*, espera-se um segundo e em seguida desliga-se o *led* e novamente aguarda-se um segundo para então repetir todo o processo.

```
void loop() {
        digitalWrite(LED, HIGH);
        delay(1000);
        digitalWrite(LED, LOW);
        delay(1000);
}
```

1.3 ACIONAMENTO TEMPORIZADO POR MEIO DE BOTÃO

1.3.1 Objetivo

Assim como nos itens 1.1 e 1.2 onde foram realizados acionamentos de um *led* externo ligado à um dos pinos de *I/O* digitais existentes na placa do *Arduino UNO*, este item visa proporcionar novamente o acionamento do *led* citado, porém desta vez por meio de um botão, de modo que ao ser pressionado, o *led* deve permanecer aceso por um tempo pré-estabelecido. Este item visa introduzir a função ***digitalRead()***, declaração de variáveis e o uso da estrutura condicional ***if()***.

1.3.2 Hardware utilizado na simulação

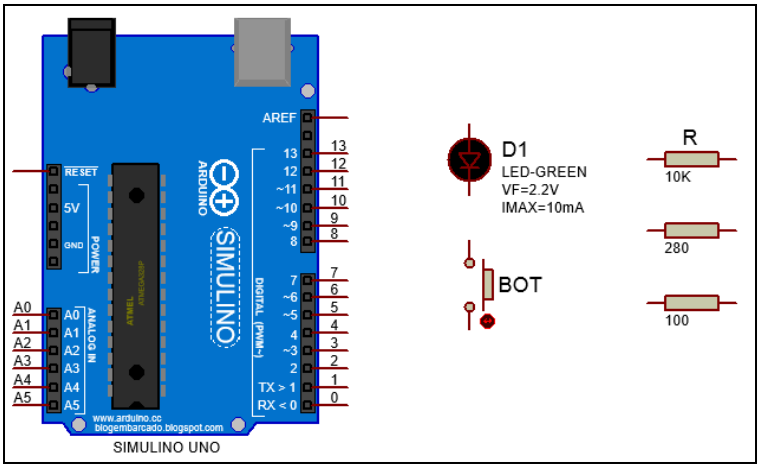

Figura 13 – Hardware utilizado no item 1.3

1.3.3 Diagrama esquemático das ligações

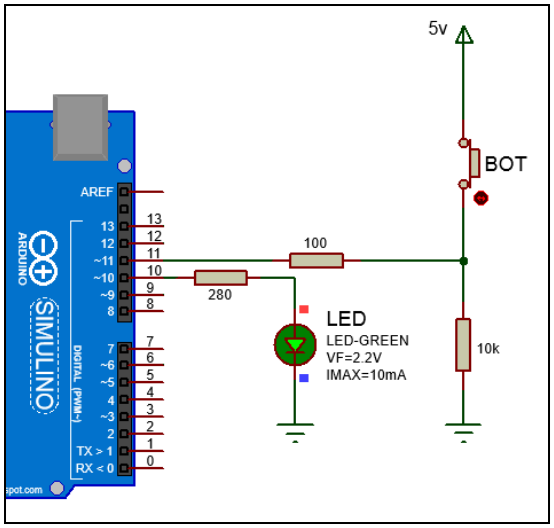

Figura 14 - Diagrama esquemático utilizado no item 1.3

1.3.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para associarmos os elementos aos seus respectivos pinos, ou seja, relaciona-se o *led* (LED) ao pino 10 o botão (BOT) ao pino 11.

```
#define LED 10
#define BOT 11
```

Em seguida é apresentada a primeira declaração de variável dentre os projetos já desenvolvidos, neste caso, foi declarada uma variável do tipo booleana, ou seja, que pode assumir os estados "0" e "1". Como o próprio nome já diz, esta variável será utilizada para armazenar uma informação referente ao estado do botão. Conforme o diagrama esquemático, percebe-se que ao pressionar o botão, o nível lógico no pino ao qual está conectado o mesmo, estará em nível alto, enquanto o resto do tempo este permanecerá em nível baixo.

```
bool estadoBOT;
```

Da mesma forma que declaramos anteriormente o pino 10 como saída digital, desta vez programa-se o pino 11 como entrada digital através da função ***pinMode()*** já utilizada, porém desta vez o segundo argumento da função (*INPUT*) determina o modo de operação do terminal responsável por interagir com o botão.

```
void setup() {

    pinMode(LED, OUTPUT);
    pinMode(BOT, INPUT);

}
```

Num primeiro momento é realizada a leitura do pino onde se encontra o botão através da função ***digitalRead()*** e esta informação é armazenada na variável **estadoBOT**. O ponto principal deste programa está na verificação do estado do botão, isto é, se houve aperto ou não. Esta parte é feita através do uso da

função condicional ***if()*** cujo argumento é a sentença que se deseja testar, caso este seja verdadeiro, serão cumpridas as sentenças internas à função (neste caso, acender o *led* e aguardar dois segundos).No entanto caso a sentença seja falsa o *led* deve ser apagado.

```
void loop() {
    estadoBOT = digitalRead(BOT);

    if (estadoBOT == 1) {
        digitalWrite (LED, HIGH);
        delay (2000);
        digitalWrite (LED,LOW);
    }
}
```

1.4. SEMÁFORO

1.4.1 Objetivo

O projeto proposto neste item consiste em produzir uma aplicação prática a partir dos conhecimentos adquiridos nos itens anteriores. Este é um semáforo que sempre está verde para os veículos, no entanto, é possível através de um botão mudar o regime de operação do mesmo, proporcionando um caminho seguro para pedestres atravessarem a rua. Além dos conceitos já aprendidos, também será demonstrado como se faz o uso da estrutura de repetição ***for()***.

1.4.2 Hardware utilizado na simulação

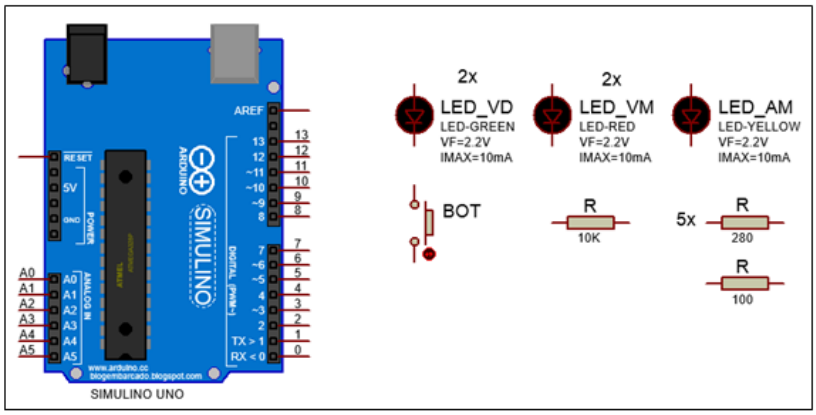

Figura 15 – Hardware utilizado no item 1.4

1.4.3 Diagrama esquemático

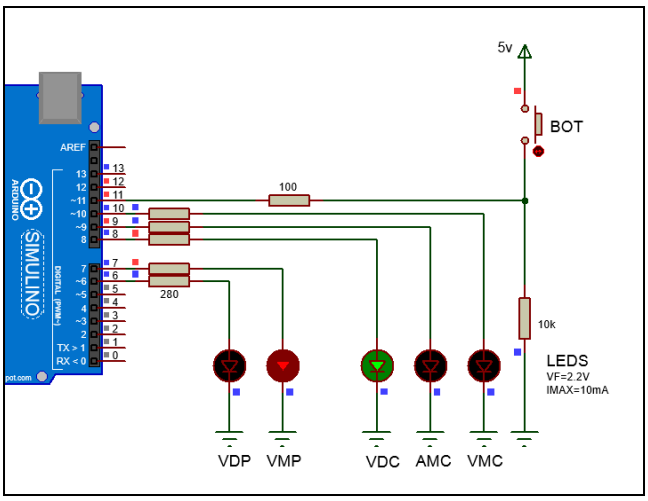

Figura 16 – Diagrama esquemático do item 1.4

1.4.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para associarmos os elementos aos seus respectivos pinos. Neste item os nomes VMC, AMC e VDC representam as luzes vermelha, amarela e verde no semáforo utilizado para o controle do tráfego dos carros, da mesma forma VMP e VDP satisfazem as mesmas funções, no entanto, são para pedestres, e BOT diz respeito ao botão de acionamento utilizado.

```
#define BOT 11
#define VMC 10
#define AMC 9
#define VDC 8
#define VMP 7
#define VDP 6
```

Em seguida é declarada a variável ***estadoBOT***, responsável por armazenar o valor referente ao estado do botão (da mesma maneira realizada no item anterior).

```
bool estadoBOT;
```

Dentro da função ***setup()*** serão definidos os modos de operação dos pinos utilizados, onde os em que estão conectados os 5 *leds* são definidos como saídas e o pino em que se encontra o botão é definido como entrada. Neste momento também são acionados dois *leds*: o verde para os carros e o vermelho para os pedestres.

```
void setup() {

    pinMode(VMC, OUTPUT);
    pinMode(AMC, OUTPUT);
    pinMode(VDC, OUTPUT);
    pinMode(VMP, OUTPUT);
    pinMode(VDP, OUTPUT);
    pinMode(BOT, INPUT);
```

```
    digitalWrite(VMP, HIGH);
    digitalWrite(VDC, HIGH);

}
```

O Primeiro passo para o desenvolvimento do programa propriamente dito é observar o estado do botão e caso este seja acionado deve-se realizar as mudanças necessárias no funcionamento do sistema para atingir o objetivo proposto. Uma vez que o botão é apertado, o *led* verde para veículos é apagado e o amarelo é acionado. Depois de 2 segundos, o amarelo para veículos por sua vez é apagado e o vermelho é acionado. Novamente após 1 segundo (temporização de segurança), o *led* vermelho para pedestres é apagado e o verde é acionado, este estado é mantido por 5 segundos.

Após os acontecimentos descritos anteriormente pode-se ver a presença da estrutura de repetição ***for()***, que por sua vez é executada repetitivamente até que uma determinada condição final seja atendida. O funcionamento da estrutura de repetição ***for()*** é dado da seguinte maneira: primeiramente inicia-se uma variável inteira ***x*** com o valor zero, posteriormente este valor é incrementado de uma unidade sempre que o código interno ao ***for()*** chegar ao final, sendo assim enquanto este valor for menor do que 10, o *led* verde para pedestres ficará alternando entre ligado e desligado como pode-se ver no código abaixo. Após essa sequência de acionamentos e desligamentos, o *led* verde para pedestres (VDP) permanece desligado enquanto *led* vermelho para pedestres (VMP) é acionado e um segundo depois (questão de segurança) o *led* verde para os veículos (VDC) é acionado. Observe que existe um *delay* de cinco segundos após esta última operação, utilizado para garantir um tempo mínimo de tráfego.

```
void loop() {

    estadoBOT = digitalRead(BOT);

    if (estadoBOT == 1) {

        digitalWrite(VDC, LOW);
        digitalWrite(AMC, HIGH);
        delay(2000);

        digitalWrite(AMC, LOW);
        digitalWrite(VMC, HIGH);
        delay(1000);
```

```
        digitalWrite(VMP, LOW);
        digitalWrite(VDP, HIGH);
        delay(5000);

        for (int x=0; x<10; x++) {
            digitalWrite(VDP, HIGH);
            delay(250);
            digitalWrite(VDP, LOW);
            delay(250);
       }

        digitalWrite(VMP, HIGH);
        delay(1000);

        digitalWrite(VMC,LOW);
        digitalWrite(VDC, HIGH);
        delay(5000);
        }
}
```

1.5 DADO DE LEDS

1.5.1 Objetivo

Este projeto visa o desenvolvimento de um dado de 6 faces, onde os números contidos nestas são constituídos por *leds*. O dado é lançado assim que o botão for pressionado e posteriormente seu resultado é apresentado através do acionamento dos *leds* citados. Além dos conceitos aprendidos nos itens anteriores, serão apresentadas as funções ***randomSeed()*** e ***Random()***.

1.5.2 Hardware utilizado na simulação

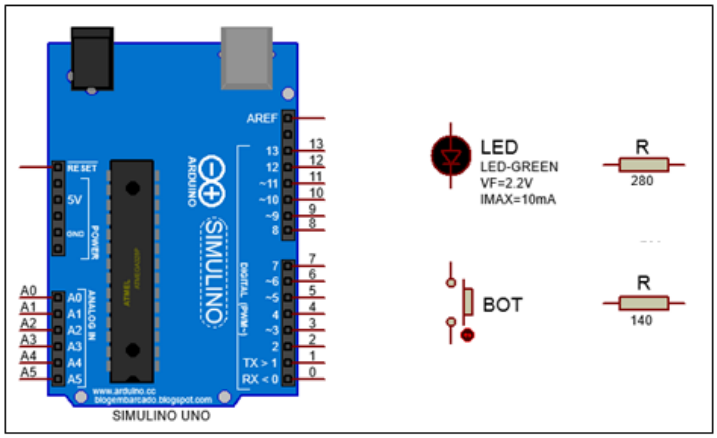

Figura 17 – Hardware utilizado no item 1.5

1.5.3 Diagrama esquemático

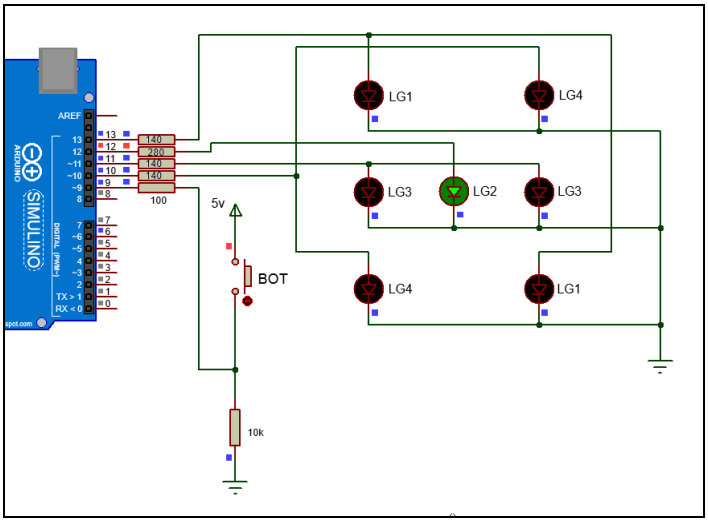

Figura 18 – Diagrama esquemático do item 1.5

1.5.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

O código deste projeto começa a partir da declaração das diretivas *#define*, neste momento são associados os grupos de *leds* e o botão aos respectivos pinos existentes na placa do *Arduino UNO*.

```
#define LG1 13
#define LG2 12
#define LG3 11
#define LG4 10
#define BOT
```

Em seguida são declaradas duas variáveis: uma já conhecida, chamada ***estadoBOT***, responsável por armazenar a informação correspondente ao estado do botão utilizado, e outra chamada ***aleatorio*** será utilizada para conter um valor pseudoaleatório gerado no programa, cuja função é "sortear" o número a ser exibido, esta é do tipo long pois a função ***random()*** utilizada posteriormente retorna um resultado do tipo long.

```
bool estadoBOT;
long aleatorio;
```

Dentro da função ***setup()*** são determinados todos os modos de operação dos pinos utilizados para garantir o funcionamento adequado do projeto. Além das declarações referentes ao comportamento dos pinos, existe também uma função chamada ***randomSeed()*** sendo utilizada responsável por proporcionar uma sequência de números pseudoaleatórios. Observe que o parâmetro utilizado por esta função é um valor lido de uma porta de entrada analógica, logo como ela não está flutuando, ou seja, não está ligada em nada, esta leitura será um valor arbitrário entre 0 e 1023, uma espécie de ruído, utilizado para a geração da sequência. Desta maneira pode-se ter sequências diferentes todas as vezes em que o programa for iniciado (o que não aconteceria caso isso não fosse feito, ou seja, seriam sequências "aleatórias", porém sempre que o programa fosse iniciado, as sequências seriam as mesmas).

```
void setup () {

    pinMode (LG1, OUTPUT);
    pinMode (LG2, OUTPUT);
    pinMode (LG3, OUTPUT);
    pinMode (LG4, OUTPUT);
    pinMode (BOT, INPUT);
    randomSeed(analogRead(0));

}
```

Primeiramente dentro da função ***loop()*** utiliza-se a função ***digitalRead()*** para armazenar a informação relativa ao estado do botão (pressionado ou não) para que através da estrutura condicional ***if()*** este possa ser verificado. Caso o botão tenha sido apertado, a variável ***aleatorio*** recebe um valor pseudoaleatório podendo variar de 1 a 6 proveniente da função ***random()***. (Esta função pode ser utilizada com dois argumentos, sendo que o primeiro é referente ao valor mínimo que pode ser retornado, e o segundo está relacionado ao valor máximo que pode ser retornado pela função, no entanto, este valor deve ser escrito incrementado de uma unidade.

Após a definição do valor contido em ***aleatorio***, são realizadas uma série de análises para saber qual é este valor e assim acender os grupos de *leds* correspondentes ao número sorteado e aguardar 5 segundos até o dado possa ser lançado novamente.

```
void loop() {

    estadoBOT = digitalRead(BOT);

    if (estadoBOT == 1) {

        aleatorio = random(1,7);

        if (aleatorio == 1){
            digitalWrite (LG2, HIGH);
            delay (5000);
         }

         if (aleatorio == 2){
             digitalWrite (LG1, HIGH);
             delay (5000);
```

```
        }

        if (aleatorio == 3){
            digitalWrite (LG4, HIGH);
            digitalWrite (LG2, HIGH);
            delay (5000);
         }

         if (aleatorio == 4){
              digitalWrite (LG1, HIGH);
              digitalWrite (LG4, HIGH);
              delay (5000);
         }

         if (aleatorio == 5){
              digitalWrite (LG1, HIGH);
              digitalWrite (LG4, HIGH);
              digitalWrite (LG2, HIGH);
              delay (5000);
          }

          if (aleatorio == 6){
               digitalWrite (LG1, HIGH);
               digitalWrite (LG3, HIGH);
               digitalWrite (LG4 HIGH);
               delay (5000);
         }

    }

  digitalWrite (LG1, LOW);
  digitalWrite (LG2, LOW);
  digitalWrite (LG3, LOW);
  digitalWrite (LG4, LOW);

}
```

2 ACIONAMENTOS NÃO TEMPORIZADOS POR MEIO DE BOTÃO

2.1 ACIONAMENTO ON/OFF

2.1.1 Objetivo

Assim como o item 1.3 objetivo proposto neste consiste em realizar o acionamento de um *led* externo utilizando um botão, porém com uma diferença: ao pressionar o mesmo, o *led* deve acender e caso este seja pressionado novamente, o *led* deve ser apagado.

2.1.2 Hardware utilizado na simulação

O hardware utilizado na simulação é o mesmo do item 1.3.

2.1.3 Diagrama esquemático das ligações

O diagrama esquemático das ligações é o mesmo do item 1.3

2.1.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para realizar a associação dos elementos utilizados com as respectivas identificações. O botão (BOT) esta relacionado com o pino 11, enquanto o *led* está ligado ao pino 10.

```
#define LED 10
#define BOT 11
```

Em seguida, são criadas variáveis responsáveis pelo funcionamento do programa. A variável ***estadoLED*** é utilizada para conter os valores referentes ao estado (ligado ou desligado) do *led*. As variáveis ***estadoatualBOT*** e ***estadoantBOT*** registram os valores atual e anterior do estado do botão. A variável ***leituraBOT*** é utilizada para armazenar a informação lida através da função ***digitalRead()***, enquanto a variável ***ultDebounceBOT*** fazem parte do cálculo realizado para definir se o tempo que se passou atingiu o valor mínimo estipulado para assegurar que não haja existência de ruído relativo ao acionamento do botão (tempo mínimo em milissegundos está contido na variável ***tempoDeb***).

```
bool estadoLED  = 0;

bool estadoatualBOT = 0;
bool estadoantBOT = 0;

bool leituraBOT = 0;

long ultDebounceBOT = 0;
long tempoDeb = 50;
```

Na função ***setup()*** estão presentes as definições dos modos de operação dos pinos utilizados. O pino 11 é declarado como entrada (botão) e o pino 10 é declarado como saída (*led*).

```
void setup() {

  pinMode(BOT, INPUT);
  pinMode(LED, OUTPUT);

}
```

Como primeiro passo dentro da função ***loop()*** armazena-se o valor relativo ao estado do botão na variável ***leituraBOT*** através da função ***digitalRead()***. Este é comparado com o estado anterior do botão, de modo que caso haja alguma alteração, a variável ***ultDebounceBOT*** recebe um valor em milissegundos proveniente da função ***millis()*** (atuando como uma espécie de *reset* na variável em questão).

Posteriormente compara-se o valor que a função ***millis()*** retorna em um dado momento com o valor armazenado na variável ***ultDebounceBOT*** e caso a diferença entre estes dois valores seja maior do que um tempo pré-estabelecido pela variável ***tempoDeb***, uma nova verificação é realizada entre o valor do estado do botão contido em ***leituraBOT*** e o estado atual do botão, armazenado em ***estadoatualBOT***. Se estes dois

estados forem distintos, a variável ***estadoatualBOT*** é atualizada com o valor referente ao estado do botão. Caso esta variável assuma nível alto, o *led* tem seu estado invertido em relação ao anterior através da função ***digitalWrite()*** e por último a variável ***estadoantBOT*** é atualizada com o valor de ***leituraBOT*****.**

```
void loop() {

    leituraBOT = digitalRead(BOT);

    if (leituraBOT !=  estadoantBOT) {

        ultDebounceBOT = millis();

    }

    if ((millis() – ultDebounceBOT) > tempoDeb) {

       if (leituraBOT != estadoatualBOT) {

           estadoatualBOT = leituraBOT;
         if (estadoatualBOT == 1) {

              digitalWrite(LED, !estadoLED);

           }
       }
    }

    estadoantBOT = leituraBOT;

}
```

3 ACIONAMENTOS UTILIZANDO POTENCIÔMETRO

3.1 ACIONAMENTO COM TEMPORIZAÇÃO VARIÁVEL

3.1.1 Objetivo

A proposta deste item é realizar o acionamento alternado de dois *leds* ligados ao *Arduino UNO*, de maneira que o tempo em que estes se alternem possa ser alterado manualmente (em uma determinada faixa de valores) através de um potenciômetro, além de demonstrar a função ***map()***.

3.1.2 Hardware utilizado na simulação

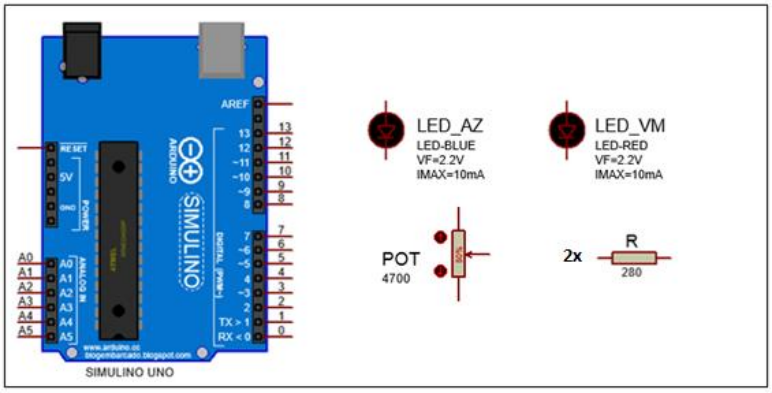

Figura 19 – Hardware utilizado no item 3.1

3.1.3 Diagrama esquemático das ligações

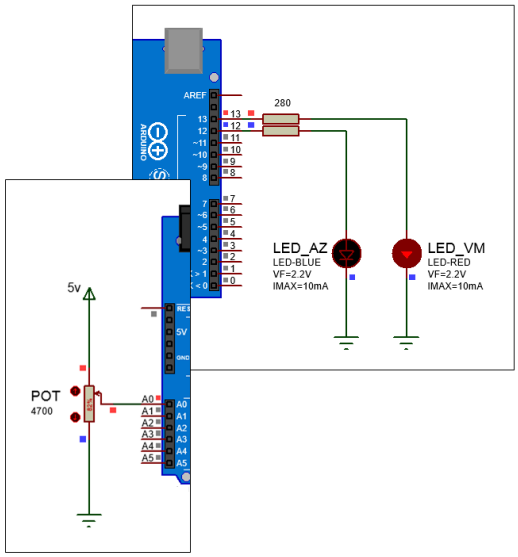

Figura 20 – Diagrama esquemático do item 3.1

3.1.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para realizar a associação dos elementos utilizados com as respectivas identificações. Os *leds* azul (LED_AZ) e vermelho (LED_VM) estão relacionados com os pinos 12 e 13, enquanto o potenciômetro (POT) está ligado ao pino de entrada analógica A0.

```
#define LED_AZ 12
#define LED_VM 13
#define POT 0
```

A variável ***medidaPOT*** é utilizada para conter os valores atuais lidos através da porta de entrada analógica A0, onde está ligado o potenciômetro.

```
int medidaPOT = 0;
```

Na função ***setup()*** estão presentes as definições dos modos de operação dos pinos utilizados para os *leds* como saídas.

```
void setup() {

    pinMode(LED_AZ,OUTPUT);
    pinMode(LED_VM,OUTPUT);

}
```

O primeiro passo na função ***loop()*** é realizar a leitura da porta de entrada analógica A0 e armazenar o conteúdo na variável ***medidaPOT***. Posteriormente utiliza-se a função ***map()*** para mapear os valor armazenado na variável ***medidaPOT*** de um intervalo de 0 a 1023 para um intervalo relativo ao valor em milissegundos que será utilizado como parâmetro para a função ***delay()*****,** neste item, o intervalo escolhido foi de 0 a 5000 (0 a 5 segundos). Em seguida é realizado o mesmo procedimento descrito no item 1.2,

porém o tempo de acionamento e desligamento dos *leds* é controlado através do potenciômetro e ao invés de um *led*, são dois.

```
void loop() {

    medidaPOT = analogRead(POT);
    medidaPOT = map(medidaPOT,0,1023,0,5000);

    digitalWrite(LED_AZ,HIGH);
    delay(medidaPOT);

    digitalWrite(LED_AZ,LOW);
    digitalWrite(LED_VM,HIGH);
    delay(medidaPOT);

    digitalWrite(LED_VM,LOW);

}
```

3.2 BARRA DE LEDS

3.2.1 Objetivo

O objetivo deste item consiste em acionar progressivamente seis *leds* externos ligados ao *Arduino UNO* de acordo com a posição de um potenciômetro.

3.2.2 Hardware utilizado na simulação

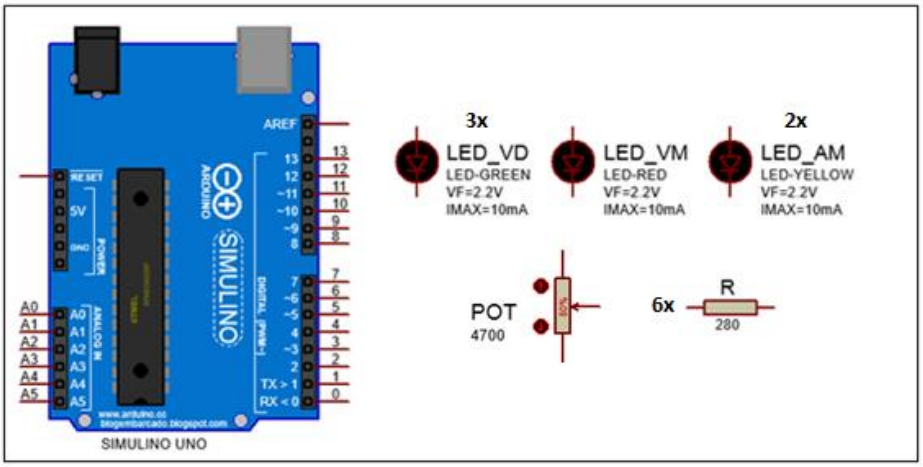

Figura 21 – Hardware utilizado no item 3.2

3.2.3 Diagrama esquemático das ligações

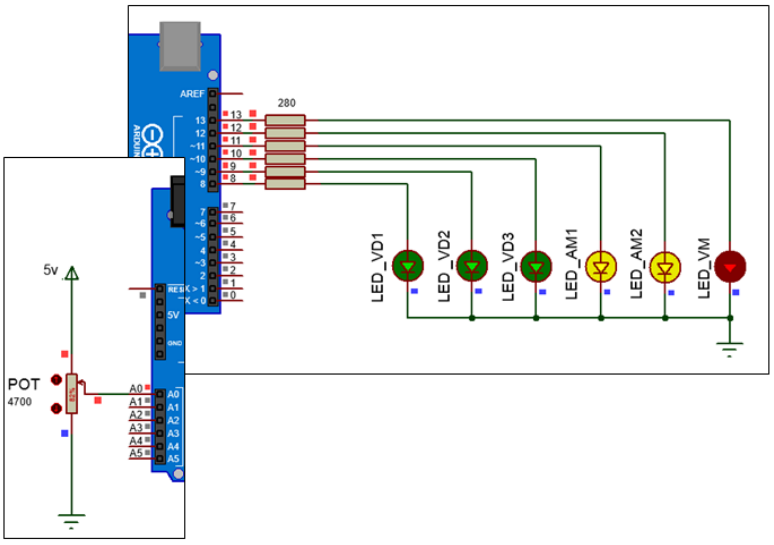

Figura 22 – Diagrama esquemático utilizado no item 3.2

3.2.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para realizar a associação dos elementos utilizados com as respectivas identificações. Os *leds* verdes (VD1,VD2,VD3) estão identificados em ordem crescente de baixo para cima e estão relacionados com os pinos 8, 9 ,10. Os *leds* amarelos (AM1,AM2) também estão identificados em ordem crescente de baixo para cima e estão relacionados com os pinos 11 e 12 enquanto o *led* vermelho (VM) está ligado ao pino 13. O potenciômetro (POT) utilizado está associado ao pino de entrada analógica A0.

```
#define LED_VD1 8
#define LED_VD2 9
#define LED_VD3 10
#define LED_AM1 11
#define LED_AM2 12
#define LED_VM 13
#define POT 0
```

Em seguida, são criadas variáveis responsáveis pelo armazenamento das informações provenientes da leitura da porta de entrada analógica A0 através do potenciômetro. A variável ***medidaPOT*** é utilizada para conter os valores atuais lidos através da porta de entrada analógica citada e a variável ***medidaPOTant*** serve para conter o valor que estava anteriormente contido em ***medidaPOT***.

```
int medidaPOT = 0;
int medidaPOTant = 0;
```

Na função ***setup()*** estão presentes as definições dos modos de operação dos pinos utilizados para os *leds* como saídas.

```
void setup() {

    pinMode(LED_VD1,OUTPUT);
```

```
    pinMode(LED_VD2,OUTPUT);
    pinMode(LED_VD3,OUTPUT);
    pinMode(LED_AM1,OUTPUT);
    pinMode(LED_AM2,OUTPUT);
    pinMode(LED_VM,OUTPUT);

}
```

A primeira linha de código na função ***loop()*** é responsável por realizar a leitura da porta analógica A0 através da função ***analogRead()***, retornar um número que pode variar de 0 a 1023 conforme o nível de tensão (0 a 5 volts) que é estabelecido no pino em questão e registrar este valor na variável ***medidaPOT***. Posteriormente o valor armazenado em ***medidaPOT*** é comparado com o valor contido em ***medidaPOTant*** (lembre-se que esta variável foi iniciada contendo o valor 0) e caso estes valores sejam diferentes torna-se necessário que código inteiro seja analisado (estruturas ***if()*** subsequentes), intervalo por intervalo (definido em porções de mesmo tamanho, iguais a aproximadamente 1023/7) para que seja determinada qual deve ser a sequência a ser utilizada.

As sequências são:

- ***medidaPOT*** ≤ 146 → Todos os leds apagados
- $146 <$ ***medidaPOT*** ≤ 292 → VD1
- $292 <$ ***medidaPOT*** ≤ 438 → VD1, VD2
- $438 <$ ***medidaPOT*** ≤ 584 → VD1, VD2, VD3
- $584 <$ ***medidaPOT*** ≤ 730 → VD1, VD2, VD3, AM1
- $730 <$ ***medidaPOT*** ≤ 876 → VD1, VD2, VD3, AM1, AM2
- ***medidaPOT*** > 876 → VD1, VD2, VD3, AM1, AM2, VM

Após ocorrer alguma mudança no regime de funcionamento da barra de *leds*, a variável ***medidaPOTant*** é atualizada com o valor contido em ***medidaPOT***.

```
void loop() {

    medidaPOT = analogRead(POT);

    if(medidaPOTant != medidaPOT)


        if(medidaPOT <= 146){
```

```
        digitalWrite(LED_VD1,LOW);
        digitalWrite(LED_VD2,LOW);
        digitalWrite(LED_VD3,LOW);
        digitalWrite(LED_AM1,LOW);
        digitalWrite(LED_AM2,LOW);
        digitalWrite(LED_VM,LOW);

    }

    if((medidaPOT > 146) && (medidaPOT <= 292)){

       digitalWrite(LED_VD1,HIGH);
       digitalWrite(LED_VD2,LOW);
       digitalWrite(LED_VD3,LOW);
       digitalWrite(LED_AM1,LOW);
       digitalWrite(LED_AM2,LOW);
       digitalWrite(LED_VM,LOW);

    }

    if((medidaPOT > 292) && (medidaPOT <= 438)){

        digitalWrite(LED_VD1,HIGH);
        digitalWrite(LED_VD2,HIGH);
        digitalWrite(LED_VD3,LOW);
        digitalWrite(LED_AM1,LOW);
        digitalWrite(LED_AM2,LOW);
        digitalWrite(LED_VM,LOW);

    }

    if((medidaPOT > 438) && (medidaPOT <= 584)){

        digitalWrite(LED_VD1,HIGH);
        digitalWrite(LED_VD2,HIGH);
        digitalWrite(LED_VD3,HIGH);
        digitalWrite(LED_AM1,LOW);
        digitalWrite(LED_AM2,LOW);
        digitalWrite(LED_VM,LOW);

     }
```

```
        if((medidaPOT > 584) && (medidaPOT <= 730)){

            digitalWrite(LED_VD1,HIGH);
            digitalWrite(LED_VD2,HIGH);
            digitalWrite(LED_VD3,HIGH);
            digitalWrite(LED_AM1,HIGH);
            digitalWrite(LED_AM2,LOW);
            digitalWrite(LED_VM,LOW);

        }

        if((medidaPOT > 730) && (medidaPOT <= 876)){

            digitalWrite(LED_VD1,HIGH);
            digitalWrite(LED_VD2,HIGH);
            digitalWrite(LED_VD3,HIGH);
            digitalWrite(LED_AM1,HIGH);
            digitalWrite(LED_AM2,HIGH);
            digitalWrite(LED_VM,LOW);

        }

        if(medidaPOT > 876){

            digitalWrite(LED_VD1,HIGH);
            digitalWrite(LED_VD2,HIGH);
            digitalWrite(LED_VD3,HIGH);
            digitalWrite(LED_AM1,HIGH);
            digitalWrite(LED_AM2,HIGH);
            digitalWrite(LED_VM,HIGH);

        }

        medidaPOTant = medidaPOT;
    }
}
```

4 ACIONAMENTOS UTILIZANDO PWM

4.1 ACIONAMENTO SIMPLES

4.1.1 Objetivo

Neste quinto capítulo propõe-se o aprendizado do uso do *PWM*, sendo utilizado neste caso na variação da luminosidade do *led* externo ligado ao pino 10 do *Arduino UNO*. No item 4.1, será demonstrado o uso da função ***analogWrite()***.

4.1.2 Hardware utilizado na simulação

O hardware utilizado é o mesmo do item 1.1.

4.1.3 Diagrama esquemático das ligações

O diagrama esquemático das ligações é o mesmo do item 1.1.

4.1.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para associar o led (representado pela palavra LED) utilizado ao pino 10.

```
#define LED 10
```

Na função ***setup()*** é definido o modo de operação do pino 10. Este pino está configurado para ser uma saída, responsável portanto pelo acionamento do *led* utilizando *PWM*. Além da configuração do pino em questão, foi utilizada a função ***analogWrite()*** responsável por determinar a largura do pulso que irá acionar o *led*, nesta função o primeiro argumento diz respeito ao pino em que se deseja utilizar o *PWM* e o segundo está relacionado ao valor que alterará a largura do pulso. Note que este deve pode variar de 0 a 255

```
void setup() {

    pinMode (LED, OUTPUT);
    analogWrite(LED, 200);

}
```

4.2 VARIAÇÃO DE LUMINOSIDADE UTILIZANDO POTENCIÔMETRO

4.2.1 Objetivo

Este item é uma complementação do projeto desenvolvido no item 4.1, no entanto deseja-se variar a luminosidade do *led* diretamente por meio do uso de um potenciômetro.

4.2.2 Hardware utilizado na simulação

O hardware utilizado na simulação é o mesmo do item 3.1

4.2.3 Diagrama esquemático das ligações

O diagrama esquemático das ligações é o mesmo do item 3.1.

4.2.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para realizar a associação dos elementos utilizados com as respectivas identificações. O potenciômetro está relacionado com pino de entrada analógica A0 (portanto utilizamos o valor "0" na diretiva), enquanto o *led* está ligado ao pino 10.

```
#define POT 0
#define LED 10
```

Em seguida é iniciada a variável ***medidaPOT*** responsável por conter o valor lido a partir da porta de entrada analógica A0.

```
int medidaPOT = 0;
```

Na função ***setup()*** é definido o modo de operação do pino 10. Este pino está configurado para ser uma saída, responsável portanto pelo acionamento do *led* utilizando *PWM*.

```
void setup() {

    pinMode(LED , OUTPUT);

}
```

O valor de tensão é lido na entrada A0 e convertido (em um intervalo de 0 a 1023) através da função ***analogRead()*** e posteriormente armazenado na variável ***medidaPOT***. Para relacionar o valor contido em **medidaPOT** e a saída que deve ser disponibilizada na saída *PWM*, foi utilizada a função ***map()***, esta mapeia o intervalo de 0 a 1023 em um intervalo de 0 a 255, para então ser utilizado para acionar o *led* através da função ***analogWrite()***.

```
void loop() {

medidaPOT =  analogRead(POT);
medidaPOT =  map(medidaPOT,0,1023,0,255);
analogWrite(LED,medidaPOT );

}
```

4.3 VARIAÇÃO DE LUMINOSIDADE UTILIZANDO BOTÕES

4.3.1 Objetivo

Neste item será desenvolvido um projeto semelhante ao proposto no item 4.2, porém serão utilizados dois botões para aumentar ou diminuir a luminosidade do *led*.

4.3.2 Hardware utilizado na simulação

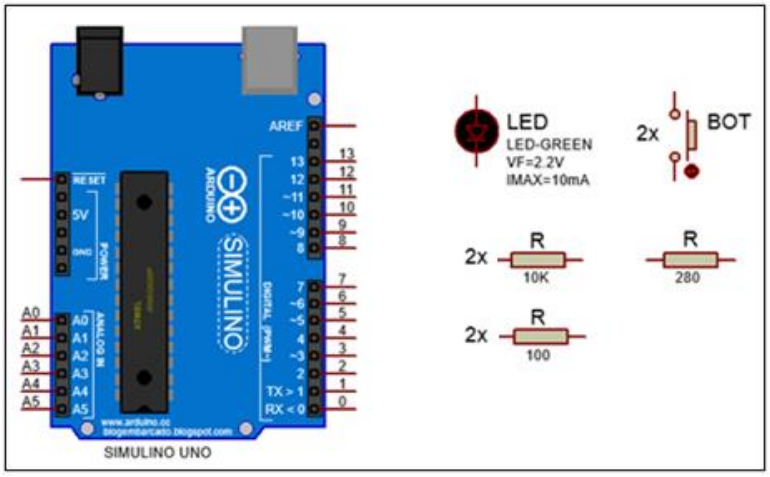

Figura 23 - Hardware utilizado no item 4.3

4.3.3 Diagrama esquemático das ligações

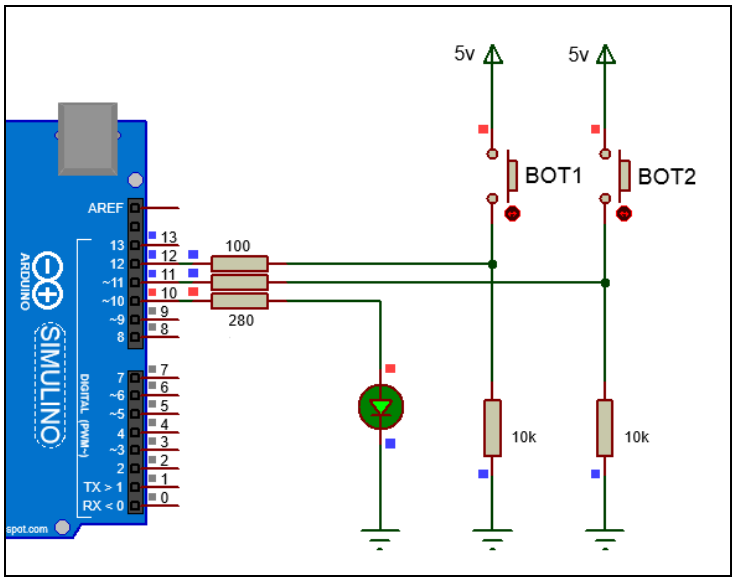

Figura 24 – Diagrama esquemático utilizado no item 4.3

4.3.4 Desenvolvimento do código

No decorrer deste tópico serão explicados todos os passos utilizados para o desenvolvimento do projeto, isto é, o código será comentado de modo que todo conhecimento aplicado seja facilmente compreendido. As trechos demarcados em vermelho contém elementos que não foram citados anteriormente ou são partes de bastante relevância para o funcionamento do programa. O código inteiro está disponibilizado no apêndice A, localizado no final deste documento.

Primeiramente utiliza-se a diretiva *#define* para realizar a associação dos elementos utilizados com as respectivas identificações. Os botões 1 e 2 estão relacionados com os pinos 12 e 11 respectivamente, enquanto o *led* está ligado ao pino 10.

```
#define BOT1 12
#define BOT2 11
#define LED 10
```

Em seguida, são criadas variáveis responsáveis pela manipulação das informações existentes no programa para que os elementos sejam lidos e acionados corretamente (botões e *led*). A variável ***estadoLED*** é utilizada para conter os valores do *PWM* (inteiros de 0 a 255) responsáveis pelo acionamento do *led* com diferentes níveis de luminosidade.

Utilizando a mesma estratégia do item 3.1, são criadas as variáveis ***estadoatual (BOT1/BOT2)*** e ***estadoant(BOT1/BOT2)*** registram os valores atuais e anteriores dos estados de cada botão. As variáveis ***leitura(BOT1/BOT2)*** são utilizadas para armazenar a informação lida através da função ***digitalRead()***. Por último, as variáveis ***ultDebounce(BOT1/BOT2)*** fazem parte do cálculo realizado para definir se o tempo que se passou atingiu o valor mínimo estipulado para assegurar que não haja existência de ruído relativo ao acionamento dos botões (tempo mínimo em milissegundos está contido na variável ***tempoDeb***).

```
int estadoLED  = 0;

bool estadoatualBOT1 = 0;
bool estadoatualBOT2 = 0;

bool estadoantBOT1 = 0;
bool estadoantBOT2 = 0;

bool leituraBOT1 = 0;
```

```
bool leituraBOT2 = 0;

long ultDebounceBOT1 = 0;
long ultDebounceBOT2 = 0;
long tempoDeb = 50;
```

Na função ***setup()*** estão presentes as definições dos modos de operação dos pinos utilizados (botões 1 e 2 como entradas, e o *led* como saída), além de garantir que ao início do programa o *led* esteja apagado.

```
void setup() {

    pinMode(BOT1, INPUT);
    pinMode(BOT2, INPUT);
    pinMode(LED, OUTPUT);

    analogWrite(LED, estadoLED);

}
```

A função ***loop()*** contém a parte principal do código, ou seja, esta é responsável pelo funcionamento propriamente dito do programa. Dentro da função ***loop()*** existem outras duas funções (***verificar(BOT1/BOT2)***) que realizam a verificação dos botões e a partir de então tomam as ações necessárias, aumentando ou diminuindo a luminosidade do *led*. Deve-se ressaltar que estas são praticamente iguais com exceção de algumas linhas de código no final das mesmas.

```
void loop() {

    verificarBOT1();
    verificarBOT2();

}
```

Para criar funções basta escreve-las após a função ***loop()***. Note que os nomes devem ser exatamente os mesmos com o qual estas foram chamadas no programa. As funções ***verificar(BOT1/BOT2)*** contém o mesmo código descrito e explicado no item 2.1 utilizado para tratar os acionamentos via botão, no entanto, ao verificar que o botão de fato permanece estável, a luminosidade do *led* pode ser aumentada caso o botão 1 seja acionado ou diminuída caso o botão 2 seja acionado.

Como dito anteriormente pode-se utilizar valores de 0 a 255 para determinar a largura do pulso, logo, neste projeto a cada vez que um dos botões é utilizado, incrementa-se ou decrementa-se 25 unidades da variável do tipo inteira ***estadoLED***. Posteriormente esta variável é utilizada como parâmetro da função ***analogWrite()***.

```
void verificarBOT1(){

    leituraBOT1 = digitalRead(BOT1);

    if (leituraBOT1 !=  estadoantBOT1) {

        ultDebounceBOT1 = millis();

    }

    if ((millis() - ultDebounceBOT1) > tempoDeb) {

        if (leituraBOT1 != estadoatualBOT1) {

            estadoatualBOT1 = leituraBOT1;
            if (estadoatualBOT1 == 1) {

                estadoLED = estadoLED + 25;
                if (estadoLED >= 255){

                    estadoLED = 255;

                }

                analogWrite(LED, estadoLED);

        }
    }

    estadoantBOT1 = leituraBOT1;

}
```

A seguir está representada apenas a parte da função ***verificarBOT2()*** responsável por decrementar a variável ***estadoLED***.

```
if (estadoatualBOT2 == 1) {
    estadoLED = estadoLED - 25;

     if(estadoLED <= 0){
         estadoLED = 0;
      }
     analogWrite(LED, estadoLED);
}
```

Na figura a seguir estão dispostos 3 gráficos mostrando o efeito da variação do valor de saída do *PWM* para valores distintos do segundo argumento da função ***analogWrite()***, que pode variar de 0 a 255. No primeiro gráfico este é 75, no segundo 150 e no terceiro 225.

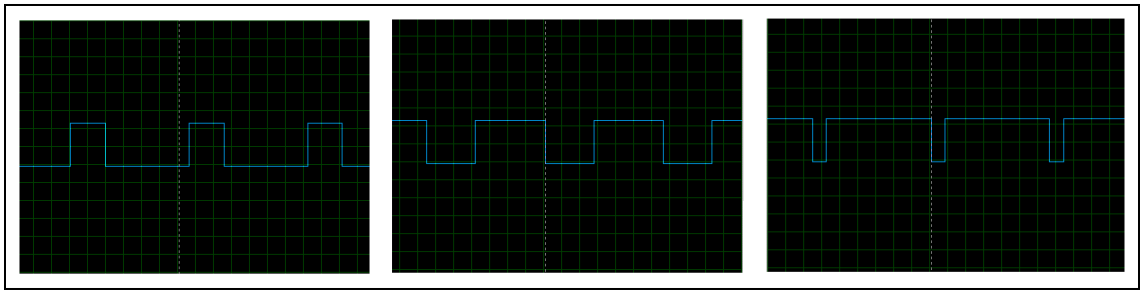

Figura 25 - Largura de pulso para 3 valores distintos da variável estadoLED

APÊNDICE A – Códigos

Este apêndice contém todos os códigos de desenvolvidos nos itens referentes à este documento, no entanto aqui estes não se encontram de forma segmentada (maneira que foram utilizados nos itens já citados).

OBS: Para conferir se os códigos estavam corretos resolvi por várias vezes copia-los diretamente deste documento e colar os mesmos na IDE do Arduino para compilar e assim testar, no entanto, existe a possibilidade de esta acusar alguns erros na compilação devido ao fato de alguns caracteres serem passados de maneira errada, como é o caso do "-" (sinal de menos), portanto caso isso aconteça, basta ir ao local onde está escrito este caractere e troca-lo pelo mesmo porém escrito diretamente na IDE.

CÓDIGO DO ITEM 1.1

```
#define LED 10

void setup() {

pinMode (LED, OUTPUT);
digitalWrite(LED, HIGH);

}

void loop() {
}
```

CÓDIGO DO ITEM 1.2

```
#define LED 10

void setup() {

pinMode(LED, OUTPUT);

}

void loop() {

digitalWrite(LED, HIGH);
```

```
delay(1000);
digitalWrite(LED, LOW);
delay(1000);

}
```

CÓDIGO DO ITEM 1.3

```
#define BOT 11

bool estadoBOT;

void setup() {

pinMode(LED, OUTPUT);
pinMode(BOT, INPUT);

}

void loop() {

estadoBOT = digitalRead(BOT);

if (estadoBOT == 1) {

digitalWrite (LED, HIGH);
delay (2000);

}

else digitalWrite (LED,LOW);

}
```

CÓDIGO DO ITEM 1.4

```
#define BOT 11
#define VMC 10
#define AMC 9
#define VDC 8
#define VMP 7
#define VDP 6
```

```
bool estadoBOT;

void setup() {

pinMode(VMC, OUTPUT);
pinMode(AMC, OUTPUT);
pinMode(VDC, OUTPUT);
pinMode(VMP, OUTPUT);
pinMode(VDP, OUTPUT);
pinMode(BOT, INPUT);

digitalWrite(VMP, HIGH);
digitalWrite(VDC, HIGH);

}

void loop() {

estadoBOT = digitalRead(BOT);

if (estadoBOT == 1) {

digitalWrite(VDC, LOW);
digitalWrite(AMC, HIGH);
delay(2000);

digitalWrite(AMC, LOW);
digitalWrite(VMC, HIGH);
delay(1000);

digitalWrite(VMP, LOW);
digitalWrite(VDP, HIGH);
delay(5000);

for (int x=0; x<10; x++) {
digitalWrite(VDP, HIGH);
delay(250);
digitalWrite(VDP, LOW);
delay(250);
}

digitalWrite(VMP, HIGH);
delay(1000);
digitalWrite(VDC, HIGH);
```

```
delay(5000);

}
}
```

CÓDIGO DO ITEM 1.5

```
#define LG1 13
#define LG2 12
#define LG3 11
#define LG4 10
#define BOT

bool estadoBOT;
long aleatorio;

void setup () {

pinMode (LG1, OUTPUT);
pinMode (LG2, OUTPUT);
pinMode (LG3, OUTPUT);
pinMode (LG4, OUTPUT);
pinMode (BOT, INPUT);
randomSeed(analogRead(0));

}

void loop() {

estadoBOT = digitalRead(BOT);

if (estadoBOT == 1) {

aleatorio = random(1,7);

if (aleatorio == 1){
digitalWrite (LG2, HIGH);
delay (5000);
}

if (aleatorio == 2){
digitalWrite (LG1, HIGH);
delay (5000);
```

```
}

if (aleatorio == 3){
digitalWrite (LG4, HIGH);
digitalWrite (LG2, HIGH);
delay (5000);
}

if (aleatorio == 4){
digitalWrite (LG1, HIGH);
digitalWrite (LG4, HIGH);
delay (5000);
}

if (aleatorio == 5){
digitalWrite (LG1, HIGH);
digitalWrite (LG4, HIGH);
digitalWrite (LG2, HIGH);
delay (5000);
}

if (aleatorio == 6){
digitalWrite (LG1, HIGH);
digitalWrite (LG3, HIGH);
digitalWrite (LG4 HIGH);
delay (5000);
}

}

digitalWrite (LG1, LOW);
digitalWrite (LG2, LOW);
digitalWrite (LG3, LOW);
digitalWrite (LG4, LOW);

}
```

CÓDIGO DO ITEM 2.1

```
#define LED 10
#define BOT 11

bool estadoLED  = 0;
```

```
bool estadoatualBOT = 0;
bool estadoantBOT = 0;

bool leituraBOT = 0;

long ultDebounceBOT = 0;
long tempoDeb = 50;

void setup() {

pinMode(BOT, INPUT);
pinMode(LED, OUTPUT);

}

void loop() {

leituraBOT = digitalRead(BOT);

if (leituraBOT !=  estadoantBOT) {

ultDebounceBOT = millis();

}

if ((millis() – ultDebounceBOT) > tempoDeb) {

if (leituraBOT != estadoatualBOT) {

estadoatualBOT = leituraBOT;

if (estadoatualBOT == 1) {

estadoLED = !estadoLED;
digitalWrite(LED, !estadoLED);

}
}
}

estadoantBOT = leituraBOT;
}
```

CÓDIGO DO ITEM 3.1

```
#define LED_AZ 12
#define LED_VM 13
#define POT 0

int medidaPOT = 0;

void setup() {

pinMode(LED_AZ,OUTPUT);
pinMode(LED_VM,OUTPUT);

}

void loop() {

medidaPOT = analogRead(POT);
medidaPOT = map(medidaPOT,0,1023,0,5000);

digitalWrite(LED_AZ,HIGH);
delay(medidaPOT);

digitalWrite(LED_AZ,LOW);
digitalWrite(LED_VM,HIGH);
delay(medidaPOT);

digitalWrite(LED_VM,LOW);

}
```

CÓDIGO DO ITEM 3.2

```
#define LED_VD1 8
#define LED_VD2 9
#define LED_VD3 10
#define LED_AM1 11
#define LED_AM2 12
#define LED_VM 13
#define POT 0

int medidaPOT = 0;
int medidaPOTant = 0;
```

```
void setup() {

pinMode(LED_VD1,OUTPUT);
pinMode(LED_VD2,OUTPUT);
pinMode(LED_VD3,OUTPUT);
pinMode(LED_AM1,OUTPUT);
pinMode(LED_AM2,OUTPUT);
pinMode(LED_VM,OUTPUT);

}

void loop() {

medidaPOT = analogRead(POT);

if(medidaPOTant != medidaPOT){

if(medidaPOT <= 146){

digitalWrite(LED_VD1,LOW);
digitalWrite(LED_VD2,LOW);
digitalWrite(LED_VD3,LOW);
digitalWrite(LED_AM1,LOW);
digitalWrite(LED_AM2,LOW);
digitalWrite(LED_VM,LOW);

}

if((medidaPOT > 146) && (medidaPOT <= 292)){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,LOW);
digitalWrite(LED_VD3,LOW);
digitalWrite(LED_AM1,LOW);
digitalWrite(LED_AM2,LOW);
digitalWrite(LED_VM,LOW);

}

if((medidaPOT > 292) && (medidaPOT <= 438)){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,HIGH);
```

```
digitalWrite(LED_VD3,LOW);
digitalWrite(LED_AM1,LOW);
digitalWrite(LED_AM2,LOW);
digitalWrite(LED_VM,LOW);

}

if((medidaPOT > 438) && (medidaPOT <= 584)){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,HIGH);
digitalWrite(LED_VD3,HIGH);
digitalWrite(LED_AM1,LOW);
digitalWrite(LED_AM2,LOW);
digitalWrite(LED_VM,LOW);

}

if((medidaPOT > 584) && (medidaPOT <= 730)){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,HIGH);
digitalWrite(LED_VD3,HIGH);
digitalWrite(LED_AM1,HIGH);
digitalWrite(LED_AM2,LOW);
digitalWrite(LED_VM,LOW);

}

if((medidaPOT > 730) && (medidaPOT <= 876)){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,HIGH);
digitalWrite(LED_VD3,HIGH);
digitalWrite(LED_AM1,HIGH);
digitalWrite(LED_AM2,HIGH);
digitalWrite(LED_VM,LOW);

}

if(medidaPOT > 876){

digitalWrite(LED_VD1,HIGH);
digitalWrite(LED_VD2,HIGH);
```

```
digitalWrite(LED_VD3,HIGH);
digitalWrite(LED_AM1,HIGH);
digitalWrite(LED_AM2,HIGH);
digitalWrite(LED_VM,HIGH);

}

medidaPOTant = medidaPOT;
}
}
```

CÓDIGO DO ITEM 4.1

```
#define LED 10

void setup() {

pinMode (LED, OUTPUT);
analogWrite(LED, 200);

}

void loop() {
}
```

CODIGO DO ITEM 4.2

```
#define POT 0
#define LED 10

int medidaPOT = 0;

void setup() {

pinMode(LED,OUTPUT)

}

void loop() {

medidaPOT =  analogRead(POT);
```

```
medidaPOT = map(medidaPOT,0,1023,0,255);
analogWrite(LED,medidaPOT);

}
```

CÓDIGO DO ITEM 4.3

```
int estadoLED  = 0;

bool estadoatualBOT1 = 0;
bool estadoatualBOT2 = 0;

bool estadoantBOT1 = 0;
bool estadoantBOT2 = 0;

bool leituraBOT1 = 0;
bool leituraBOT2 = 0;

long ultDebounceBOT1 = 0;
long ultDebounceBOT2 = 0;

long tempoDeb = 50;

void setup() {

pinMode(BOT1, INPUT);
pinMode(BOT2, INPUT);
pinMode(LED, OUTPUT);
analogWrite(LED, estadoLED);

}

void loop() {

verificarBOT1();
verificarBOT2();

}

void verificarBOT1(){

    leituraBOT1 = digitalRead(BOT1);
```

```
    if (leituraBOT1 !=  estadoantBOT1) {
    ultDebounceBOT1 = millis();
    }

    if ((millis() - ultDebounceBOT1) > tempoDeb) {

    if (leituraBOT1 != estadoatualBOT1) {

    estadoatualBOT1 = leituraBOT1;
    if (estadoatualBOT1 == 1) {

    estadoLED = estadoLED + 25;
    if (estadoLED >= 255){

    estadoLED = 255;

    }

    analogWrite(LED, estadoLED);

    }
    }
    }

    estadoantBOT1 = leituraBOT1;

}
void verificarBOT2(){

    leituraBOT2 = digitalRead(BOT2);

    if (leituraBOT2 !=  estadoantBOT2) {
    ultDebounceBOT2 = millis();
    }

    if ((millis() - ultDebounceBOT1) > tempoDeb) {

    if (leituraBOT2 != estadoatualBOT2) {

    estadoatualBOT2 = leituraBOT1;
    if (estadoatualBOT2 == 1) {

    estadoLED = estadoLED + 25;
    if (estadoLED >= 255){
```

```
    estadoLED = 255;

    }

    analogWrite(LED, estadoLED);

    }
    }
    }

    estadoantBOT2 = leituraBOT2;

}
```