Universidade Estadual de Campinas - UNICAMP

Projeto 1 - Semáforo com temporização controlada por interrupções

EA076 - Laboratório de Sistemas Embarcados

Alefe Tiago Feliciano RA:230530

Abril, 2022 - Campinas, SP

FEEC - Faculdade de Engenharia Elétrica e Computação

Introdução ao Projeto

Motivação do Projeto

O projeto é uma proposta de familiarização do estudante com conceitos e plataformar de desenvolvimento em Arduino UNO + IDE, utilizando o acionamento de circuitos simples de um sistema embarcado com controles por interupões periódicas para coordenar processos em tempo real. Este trabalho se dispoõe a aplicar os conceitos supracitados para resolver um problema real que será abordado a seguir. Segundo o enunciado, existe uma avenida, chamada Roxo Moreira, que separa a UNICAMP de um bairro nos arredores da cidade universitária. E em determinada altura dessa avenida, mostra-se necessária a implantação de um controle semafórico das vias para diminuir o risco de atropelamento. Assim, foi proposto o desenvolvimento de uma POC (Proof of Concept), ou melhor, uma prova de conceito, para tal controle.

Descrição funcional do projeto

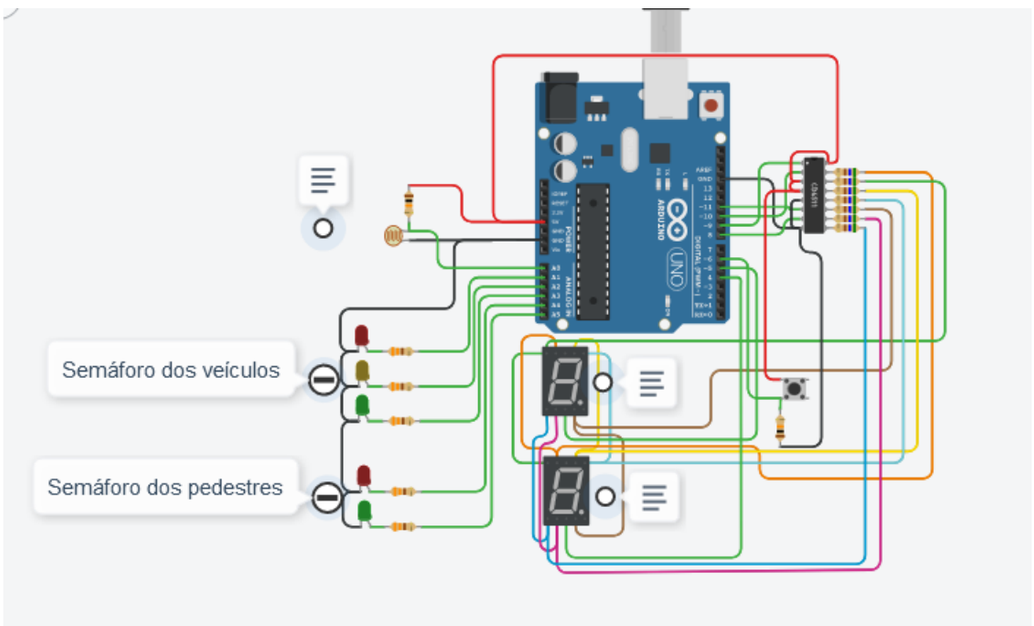
O sistema deve se comportar de determinada maneira a depender das condições externar a ele. Tais condições serão itemizadas a seguir e postas em uma máquina de estados mais a diante.

- Em condições normais, quando de dia, o semáforo está aberto para veículos e fechado para pedestre.
  - Se um pedestre aperta o botão, sendo essa uma forma de alterar o estado do sistema, o sistema:
    - Aguarda 100ms mantendo o estado anterior;
    - Mostra amarelo para os carros;
    - Fecha para os carros e, simultaneamente, abre para os pedestre. Enquanto isso, displays de 7 segmentos exibem o tempo restante para travessia dos pedestres. O display dos veículos inicia a contagem decrescente em 9 e o display dos pedestres a em 5. A cada segundo, o valor mostrado nos displays é decrementado;
    - Ao chegar ao valor zero, o display e a luz vermelha dos pedestres pisca indicando que o semáforo irá fechar em breve. O estado dos itens para veículos se mantem.
    - Quando a contagem associada aos veículos atinge zero, o semáforo fecha para os pedestres e abre para os carros.
  - À noite, o sistema de apertar o botão é desligado e o semáforo somente pisca a luz amarela para os carros e vermelha para os pedestres, retornando ao estado original na manhã seguinte.

O sistema deve ser capaz de ignorar mudanças temporárias na luz. Ou seja, apenas alterações de luminosidade que perdurem por pelo menos alguns segundos podem mudar o modo de operação de diurno para noturno ou vice-versa.

Esquemático Proposto

Nesse primeiro projeto, o aluno não deve realizar modificações no Hardware do sistema. Apenas implementar um algoritmo de controle na plataforma ThinkerCad capaz de atender as especificações descritas no item anterior. Segue o esquemático proposto:



Aqui, temos algumas considerações importantes a se fazer sobre alguns elementos do circuito. Nos próximos parágrafos iremos abordar o sensor de luminosidade, CI decodificador para o controle dos displays e o push Button.

Rotina de Interrupção

As interrupções do temporizador acontecem quando o contador do Timer1 (contador interno do AtMega238p) atinge um valor definido no registrador interno OCR1. Dentro da função que define o comportamento da rotina de interrupção, variáveis são incrementadas para o uso devido em cada parte do programa. Segue a função de controle setup do temporizador bem como a função rotina de interrupção usada.

```
1
2  void configuracao_Timer0(){
3    // Configuracao Temporizador 0 (8 bits) para gerar interrupcoes
       periodicas a cada 8ms no modo Clear Timer on Compare Match (CTC
       )
4    // Relogio = 16e6 Hz
5    // Prescaler = 1024
6    // Faixa = 125 (contagem de 0 a OCR0A = 124)
7    // Intervalo entre interrupcoes: (Prescaler/Relogio)*Faixa =
       (64/16e6)*(124+1) = 0.008s
8
9    // TCCR0A     Timer/Counter Control Register A
10   // COM0A1 COM0A0 COM0B1 COM0B0          WGM01 WGM00
11   // 0      0      0      0          1     0
12   TCCR0A = 0x02;
13
```

```
  // OCR0A     Output Compare Register A
  OCR0A = 124;

  // TIMSK0     Timer/Counter Interrupt Mask Register
  //                      OCIE0B OCIE0A TOIE0
  //                      0      1      0
  TIMSK0 = 0x02;

  // TCCR0B     Timer/Counter Control Register B
  // FOC0A FOC0B          WGM02 CS02 CS01 CS0
  // 0     0          0     1    0    1
  TCCR0B = 0x05;
  ////////////
}
```

```
    // Rotina de servico de interrupcao do temporizador
ISR(TIMER0_COMPA_vect){
  // Aqui, a rotina de interrupcoes s  vai incrementar variaveis
     auxiliares que ser o utilizadas em outras partes do c digo.
  cont++;
  contAux++;
  cont2++;
}
```

Sensor LDR

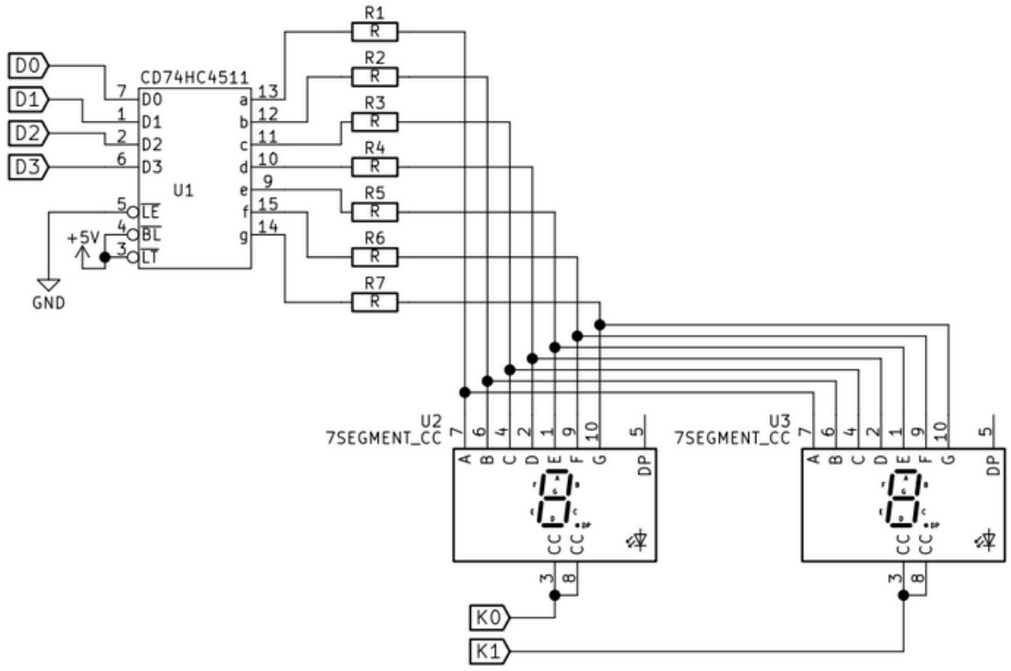
LDR significa Light Dependent Resistor, que é um resistor dependente de luz. O LDR também é conhecido como fotorresistor, e é um tipo de elemento que tem a capacidade de alterar sua resistência dependendo da intensidade da luz que o atinge. O funcionamento do LDR acontece da seguinte forma: Quando partículas de luz (fótons) caem na superfície do sensor, os elétrons presentes no material semicondutor são liberados, então a condutividade do LDR aumenta e, por consequência, sua resistência diminui.

Em condições normais, o material LDR tem uma alta resistência, então quanto maior a porcentagem de luz que atinge o sensor LDR, menor sua resistência! Ou seja, no escuro, a resistência do LDR é máxima e, no ambiente claro, sua resistência será muito menor. Conseguimos ver isso na simulação! Deixando a escala do simulador no modo noturno, temos valores maiores na leitura da tensão que em valores menores! Como a tensão é diretamente proporcional a resistência, temos valores maiores de resistência a noite que de dia.

A leitura do sensor LDR acontece na entrada 14 (GPIO) do Arduíno. Essa entrada é uma entrada do tipo ADC, ou conversor analógico digital, que por um processo de amostragem de sinais gera um valor digital para diferentes níveis de tensão em sua entrada. No caso do sensor LDR, como supracitado, quanto maior a tensão sobre ele, maior é o valor que será colocado na GPIO, por consequência, saberemos se é noite ou dia.

Decodificador 74HCT4511

Nas GPIOS de saída do conjunto B, temos sinais que vão controlar o conjunto composto pelos dois displays de 7 segmentos e o decodificador 74HCT4511. Para que isso aconteça, partimos do seguinte esquema digital para o entendimento do sistema:



Nota-se portanto, que o controle de qual valor irá aparecer em cada display é feito pela combinação dos bits D0:D3 formando em binário o número almeijado. Para facilitar futuramente a impressão de valores nos displays, criei uma função que seta os bits citados acima de acordo com um número alvo de entrada. Segue a lógica proposta:

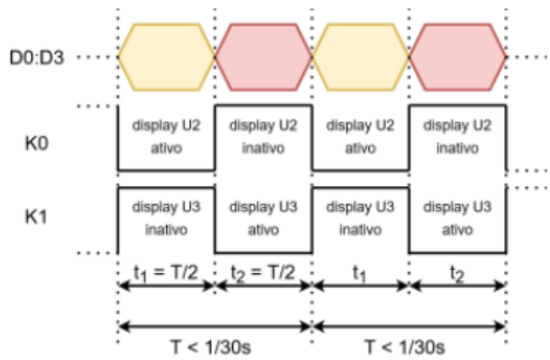
```
1
2  /*Aqui, define-se uma fun   o que para dado valor de entrada,
       mostra-se o valor correspondente nos display's 7 segmentos*/
3
4  void display7Seg(int value){
5    if (value == 9){
6      digitalWrite(11, HIGH);
7          digitalWrite(10, LOW);
8          digitalWrite(9, LOW);
9          digitalWrite(8, HIGH);
10
11   }
12     if (value == 8){
13     digitalWrite(11, HIGH);
14         digitalWrite(10, LOW);
15         digitalWrite(9, LOW);
16         digitalWrite(8, LOW);
17
18   }
19     if (value == 7){
20     digitalWrite(11, LOW);
21         digitalWrite(10, HIGH);
22         digitalWrite(9, HIGH);
```

```
23          digitalWrite(8, HIGH);
24
25      ...
26
27       if (value == 16){
28      digitalWrite(11, HIGH);
29          digitalWrite(10, HIGH);
30          digitalWrite(9, HIGH);
31          digitalWrite(8, HIGH);
32    }
33  }
```

obs: os três pontos no meio do código significa que existem outros números intermediários.

O valor 16 será usado futuramente para apagar o display, visto que por padrão o display não mostra valores maiores que o máximo valor possível para um 7 segmentos, apagando todos os leds internos.

Ainda sobre o controle dos displays de 7 segmentos, precisa-se usar uma lógica para avaliar qual display estará ligado, visto que o decodificador manda a mesma informação para os dois displays. Precisamos usar o conceito de permanencia da visão mostrado na imagem a seguir para implementar uma função de switch entre os displays.



Para isso, utiliza-se a sequência de comandos:

```
1       digitalWrite(changeLedPin0, LOW);
2           digitalWrite(changeLedPin1, HIGH);
3           display7Seg(numLed1);
4           countMiliSecs();
5       digitalWrite(changeLedPin0, HIGH);
6           digitalWrite(changeLedPin1, LOW);
7           display7Seg(numLed0);
8           countMiliSecs();
```

Onde a função countMiliSecs() é a contagem de duas entradas na rotina de interrupção do temporizador, definida por:

```
1
2   void countMiliSecs(){
3     cont=0;
4     while (cont < 2){}
5   }
```

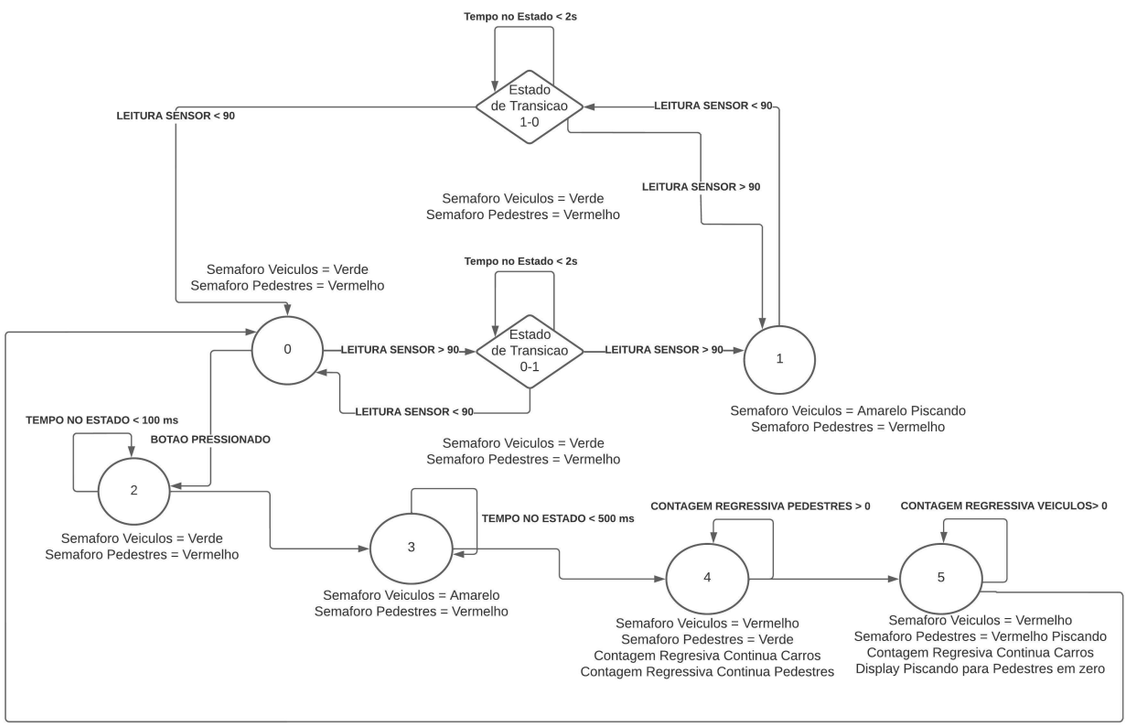
Máquina de Estados Finita

Descrição Textual

O sistema inicializa-se com o semáforo dos veículos aberto e o dos pedestres fechados (estado 0). Nesse momento, o estado pode se alterar de duas formas: se alguma mudança no sensor acontecer ou se o botão for pressionado. Se alguma mudança no sensor acontecer, o sistema vai para um estado intermediário onde vai esperar por 2 s mantendo o estado anterior para avaliar se houve de fato o dia virou noite ou se foi só um evento natural. Se não houve uma mudança, o sistema volta para o estado zero, se houve, o sistema vai para o estado 1 que é o sistema operando em modo noturno com o semáforo dos veículos piscando o amarelo e o semáforo dos pedestres no vermelho. A mesma lógica de transição acontece na volta do estado 1 para o 0 no caso de uma mudança no sensor de luminosidade.

Voltando agora para a segunda possibilidade do estado 0, ou seja, se o botão for pressionado. Se o botão for pressionado, então o o estado vai para o estado 2 onde espera com o estado anterior por 100ms, em seguida vai para o estado 3 onde mantem as proporiedades do estado 2 mas muda o sinal dos veículos para amarelo por 500 ms. Em seguida, vai para o estado 4 onde o sinal fecha para os veículos e abre para os pedestres com uma contagem regressiva até nos dois displays até que o contador regressivo dos pedestres chegue em zero. Quando isso acontece, o sistema entra no estado 5 onde a contagem regressiva dos carros continua mas o semáforo e o display dos pedestres começa a piscar com as propriedades do ultimo estado. Quando a contagem regressiva dos veículos termina, o sistema volta para o estado 0.

Descrição Gráfica



Código e Simulação

O código escrito para o controle do sistema é composto por uma função principal que varre os estados e chama uma segunda função para atribuir os valores nas GPIOS para cada um desses estados conectados, uma função de setup que define algumas condições iniciais para o sistema, funções auxiliares e definições de variáveis. Iniciando pela função principal, temos:

```
1
2  void loop(){
3    //Aqui, insere-se um delay de 1 ms para evitar o problema de
       lentidao no simulador.
4    _delay_ms(1);
5    // Le-se o valor da entrada anal gica a fim de verificar se
       dia ou noite! Se    noite, o valor da leitura ser
6    //maior que 90, se n o ,    menor.
7    readAnalogOut = analogRead(readAnalogPin);
8    if (readAnalogOut < 90){
9            actualState = 1;
10         }
11         else{
12           actualState = 0;}
13   // verifica-se se houve uma mudan a no estado do sistema!
14   verify=verifyChanges(actualState, lastState);
15   if (verify == 0){
16     //Se n o houve mudan a, utilizamos a funcao putState para
         emular o estado atual.
17     putState(actualState);
18     cont2=0;}
19
20   else {
21     // Se houve mudan a, esperamos 2 segundos para garantir que o
         estado n o mudou por um pass ro ou algo que possa
22     //alterar a leitura do sensor momentaneamente. E enquanto
         esperamos, emulamos o valor do estado anterior com o
23     //uso da funcao putState
24     while (cont2 < 250){
25       //para acelerar a simulacao, adicionei mais um delay de 1 ms
26       _delay_ms(1);
27       putState(lastState);
28     }
29     // Por fim, atribuimos a vari vel lastState o valor do
         acutalState para a transferencia instantanea dos estados
30     //passado e futuro.
31   }
32   lastState = actualState;
33 }
```

Percebe-se inicialmente que a função começa com um loop infinito para garantir que a todo momendo que o sistema não esteja dentro de uma outra função, esteja varrendo os estados do mesmo. Em seguida, temos a declaração de um delay de um milisegundo para melhorar um bug no atraso da simulação.

Usamos na main a função de leitura analogica a cada varedura para verificar o estado do sensor de luminosidade e atribuir em seguida o estado, se é dia ou se é noite. Para verificar se houve uma mudança no estado, utilizamos a função a seguir:

```
    int verifyChanges(int actualState, int lastState){
  if (actualState == lastState){
        return 0;
  }
  else{
        return 1;
        }
}
```

Em seguida, a depender do retorno da função, avaliamos se vamos manter o estado anterior ou se vamos entrar em um estado intermediário de alerta, imprimindo nas GPIOS o valor anterior mas avaliando por 3s se aquele estado irá se manter. Se o estado se mantiver, mudamos o estado para o novo estado detectado, se não, voltamos ao estado anterior. Este controle serve para garantir que variações instantaneas de luminosidade não interfiram no estado do sistema.

Para colocar os estados nas GPIOS, utilizamos a função putState(), que é definida por:

```
    void putState(int state){

//Se o estado    1, quer dizer que    dia! Logo o semafaro dos
   carros deve estar aberto e dos pedestres fechados!
//Deve existir tamb m a leitura do bot o por varredura! Assim,
   quando o bot o for pressionado devemos entrar na Rotina
//pressButtonFunction que descreve o funcionamento do sistema quando
    o bot o    pressionado.
        if (state == 1){
    digitalWrite(A3, HIGH);
    digitalWrite(A1, LOW);
    digitalWrite(A1, LOW);
    digitalWrite(A2, LOW);
    digitalWrite(A5, LOW);
    digitalWrite(A4, HIGH);
    //Aqui, os displays s o desligados! Para garantir que nenhum
       n mero esteja sendo transmitido em um momento desnecess rio
       .
    digitalWrite(changeLedPin0, HIGH);
    digitalWrite(changeLedPin1, HIGH);
    if (digitalRead(pushButton) == HIGH){
        pressButtonFunction();
    }
  }
  //Quando o estado    0, ent o    noite! Quando    noite o bot o
     de passagem    desabilitado, logo n o h   uma varredura.

  else if (state==0) {
    digitalWrite(A4, HIGH);
        digitalWrite(A3, LOW);
```

```
25      digitalWrite(A1, LOW);
26          digitalWrite(A5, LOW);
27
28      YellowLedVeic=digitalRead(A2);
29    // Durante a noite o sem foro dos ve culos deve piscar o amarelo
          por 0.5s, ou seja, uma contagem at  63 de um dos
30    //contadore    necess rio.
31    // Se o LED esta apagado a 63*8=500ms ou mais, acende o LED
32    if ((YellowLedVeic==0)&&(cont>=65)) {
33      cont = 0;
34      digitalWrite(A2, HIGH);
35    }
36    // Se o LED esta aceso a 63*8=500ms ou mais, apaga o LED
37    if ((YellowLedVeic==1)&&(cont>=65)) {
38      cont = 0;
39      digitalWrite(A2, LOW);
40    }
41      digitalWrite(changeLedPin0, HIGH);
42      digitalWrite(changeLedPin1, HIGH);
43    }}
```

A função verifica através da entrada se é noite ou dia. Se é noite, pisca o semáforo dos veículos no amarelo e deixa o semáforo dos pedestres aceso no vermelho. Se é dia, deixa o semáforo dos veículos verde, deixa o semáforo dos pedestres no vermelho e varre o estado do botão mecânico. Se o botão for pressionado, entramos na seguinte rotina:

```
1
2   /*A fun   o a seguir implementa o que deve acontecer quando o
       bot o dos pedestres    pressionado*/
3
4   void pressButtonFunction(){
5     // Inicialmente o sistema espera um pouco. No caso, 0.5s
6     countHalfSec();
7
8     //Logo em seguida mostro amarelo para os carros e mantenho o valor
          anterior no semafaro dos pedestres.
9
10    digitalWrite(A3, LOW);
11    digitalWrite(A2, HIGH);
12
13    // Mantenho o amarelo para os carros por 1.5 s
14    countHalfSec();
15
16    // Abro o sem foro para os pedestres e fecho para os carros
17    digitalWrite(A2, LOW);
18    digitalWrite(A1, HIGH);
19    digitalWrite(A4, LOW);
20    digitalWrite(A5, HIGH);
21
22    //Em seguida, come o uma contagem regressiva do 9 ao 5 e do 5 ao
         1 nos displays dos carros e dos pedestres respectivamente.
```

```
23
24    controlDisplays(9,5,1);
25    controlDisplays(8,4,1);
26    controlDisplays(7,3,1);
27    controlDisplays(6,2,1);
28    controlDisplays(5,1,1);
29
30    // Quando o valor do display dos pedestres chega em zero, uso a
         fun   o de controle dos displays com o parametro Type =2
31    // para piscar o display utilzando o valor 16 (dado que valores
         maiores que 9 fazem o display apagar.)
32    digitalWrite(A5, LOW);
33    controlDisplays(4,0,2);
34    controlDisplays(4,16,2);
35    controlDisplays(3,0,2);
36    controlDisplays(3,16,2);
37    controlDisplays(2,0,2);
38    controlDisplays(2,16,2);
39    controlDisplays(1,0,2);
40    controlDisplays(1,16,2);
41
42    // Desligo os displays no final da rotina, j   que os displays
         n  o precisam ficar acesos.
43    digitalWrite(changeLedPin0, HIGH);
44    digitalWrite(changeLedPin1, HIGH);
45  }
```

Onde esperamos um tempo, mostramos o amarelo para os veículos, esperamos novamente, abrimos o semáforo para os pedestres e fechamos para os veículos e iniciamos duas contagens regressivas: de 9 e 5 segundos. Quando a contagem dos pedestres chega em 0, o display pisca com intervalos de 0.5 segundos e termina com a chegada na contagem dos carros em 0. Para esse controle, passamos o parâmetro type da função controlDisplays como 2. Assim, a função sabe que deve piscar. Segue a implementação da lógica:

```
1     void controlDisplays(int numLed0, int numLed1, int type){
2    // Zeramos o contador auxiliar que avalia um segundo de espera.
3    contAux = 0;
4    if (type == 1){
5      while (contAux < 125){
6      digitalWrite(changeLedPin0, LOW);
7          digitalWrite(changeLedPin1, HIGH);
8          display7Seg(numLed1);
9      // Logo depois de exibir o valor esperado em um display,
          alterenamos os displays mudando os valores de changeLedPin0 e
          changeLedPin1
10         countMiliSecs();
11     digitalWrite(changeLedPin0, HIGH);
12         digitalWrite(changeLedPin1, LOW);
13         display7Seg(numLed0);
14         countMiliSecs();
15   }}
```

```
16    else{
17      // Por saber que al entrar nesse trecho de c digo o sem foro
           dos pedestres precisa piscar, faz-se necess rio um conjunto
           de if's para
18      //avaliar se o led estava aceso ou apagado e mudar seu estado.
19      if (digitalRead(A4) == HIGH){
20          digitalWrite(A4, LOW);
21        }
22        else {
23          digitalWrite(A4, HIGH);
24        }
25      while (contAux < 63){
26        // Aqui, entramos na condi  o de quando o tempo de contagem
             entre os estados dos displays    menor que 1s.
27      digitalWrite(changeLedPin0, LOW);
28          digitalWrite(changeLedPin1, HIGH);
29          display7Seg(numLed1);
30          countMiliSecs();
31      // Aqui tamb m, logo depois de exibir o valor esperado em um
           display, alterenamos os displays mudando os valores de
           changeLedPin0 e changeLedPin1
32
33      digitalWrite(changeLedPin0, HIGH);
34          digitalWrite(changeLedPin1, LOW);
35          display7Seg(numLed0);
36          countMiliSecs();
37      }}
38  }
```

A função controla os displays a partir de dois valores de entrada, para cada um dos displays e o modo de operação. Ou seja, se deve ser uma contagem regressiva normal ou se deve piscar os leds internos do display. A lógica de alternar os displays para garantir a permanencia da visão está implementada dentro desssa função, como explicado no tópico de displays acima.

Conclusão

Durante o desenvolvimento dos códigos, a simulação mostra-se grandemente importante para garantir a eficácia e eficiência da solução proposta. Dessa forma, esse primeiro projeto mostrou com uma abordagem real e prática a importância da propototipagem dos sistemas eletrônicos, bem como os desafios da implementação de softwares embarcados para driblar problemas naturais (controle dos estados apesar de variações repentinas e momentaneas de luminosidade) e resolver necessidades humanas de mobilidade e segurança.