Faculdade de Engenharia da Universidade do Porto

Ampulheta Eletrónica

Francisco Cardoso Lima – up201905781

Miguel Augusto Marombal Araújo – up201905077

Relatório de Trabalho Prático realizado no âmbito da Unidade Curricular
Arquiteturas de Computação Embarcada

Outubro 2022

Índice

Lista de figuras

Abreviaturas

TIS	*Time in State (texto não português em itálico)*
TES	*Time enter State*
LED	*Light-emitting diode*
FSM	*Finite State Machine*
GND	*Ground*
PWM	*Pulse Width Modulation*

1. Descrição do Problema

Para o presente trabalho foi proposto modelar e implementar uma ampulheta eletrónica utilizando um microcontrolador Raspberry Pi Pico.
Pretendia-se que, perante a pressão de um botão de START (S2) 6 LEDs se acendessem e, em intervalos de tempo definidos se fossem apagando consecutivamente funcionando assim como um temporizador (do estilo ampulheta) eletrónico. Após se apagarem os 6 LEDs pretendia-se que um sétimo LED se ligasse dando um feedback visual que a ampulheta havia acabado.
Sobre o funcionamento previamente descrito foi proposto implementar uma série de funcionalidades extra.

Nomeadamente, que fosse possível pausar e retomar o funcionamento da ampulheta com recurso a um botão (S2) e, através de um clique duplo num botão (também S2) a ampulheta pudesse ganhar um intervalo extra, ou seja, reacender o último LED anteriormente apagado.

Por fim, foi sugerida a implementação de um modo configuração ao qual o utilizador podia aceder pressionando continuamente o botão S1. No modo de configuração o utilizador teria três distintas possibilidade de configuração do sistema. Na primeira, seria possível controlar qual seria o intervalo de tempo entre o desligar dos LEDs (tal valor poderia ser alternado entre 1, 2, 4 e 8 segundos premindo um botão, neste caso S2). Na segunda configuração seria possível alterar a maneira como os LEDs se desligavam existindo três possibilidades. Na primeira, o LED desligar-se-ia no final do tempo definido. Na segunda, o LED manter-se-ia acesso durante metade do intervalo de tempo definido, na segunda metade do intervalo piscaria se desligar. Na última alternativa, o LED perderia intensidade consoante o tempo for passando, até que no final estaria completamente desligado (dando um efeito de Fade out). Na terceira e última configuração podia ser alterada a maneira como se indicava o término da ampulheta, havendo para tal duas maneiras. A primeira, já descrita, pelo acender de um sétimo LED. Em alternativa, colocando todos os 6 LEDs a piscarem. A navegação entre as três possibilidades de configuração pode ser feita através do botão S1. Para cada respetivo modo o utilizador pressiona o botão S2 para alterar a respetiva funcionalidade. O utilizador terá um dos LEDs 1, 2 ou 3 a piscar indicando-lhe em que modo de configuração se encontra e o LED7 a exemplificar o modo de atuação do sistema na configuração selecionada.

2. Esquema Elétrico

Para a realização do projeto foi proposto o seguinte esquema elétrico. Foram ligados 6 LEDs a pinos e configurados como OUTPUTs do Raspberry Pi Pico em série com resistências de 1K Ω de modo a controlar a corrente que passaria nos LEDs. Posteriormente, foram ligados dois botões de modo que quando não premidos fosse aplicada uma tensão de VCC, correspondente a um valor lógico de High (+3.3V), à entrada do respetivo pino do Raspberry Pi Pico configurado como INPUT, já quando premido é o GND (0V), correspondente ao valor lógico de Low, que é aplicado no pino. Em série com VCC foram colocadas duas resistências de 100k Ω.

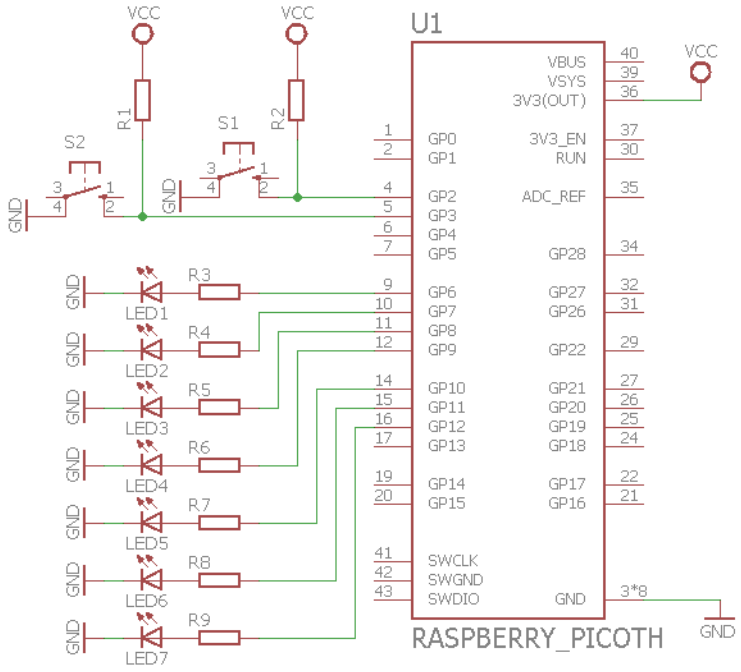

Figura 1. Esquema Elétrico

3. Máquinas de Estado

A fsm[0] é responsável por implementar o funcionamento principal da ampulheta. Devemos realçar a importancia dos varios estados desta maquina fulcral ao funcionamento do programa.

Resumidamente, o estado 0 é o estado inicial e também o estado que realiza o 'reset' do programa.

O estado 1 é responsavél pela temporização do intrevalo de tempo que cada LED demora a apagar, os estados 2 e 3 são os estados de pausa e o estado 4 serve para marcar o final do tempo.

Por fim, os estados 6 e 7 são simples estados de manutenção, que servem para sincronização com outras maquinas de estado, respetivamente fsm[1] e fsm[5].
Já o estado 5 serve para não deixar a maquina de estados evoluir aquando da ativação do modo de configuração, funcionando como um *'freeze'* que guarda o estado onde a maquina parou, retornando a esse estado quando se sai do modo de configuração.

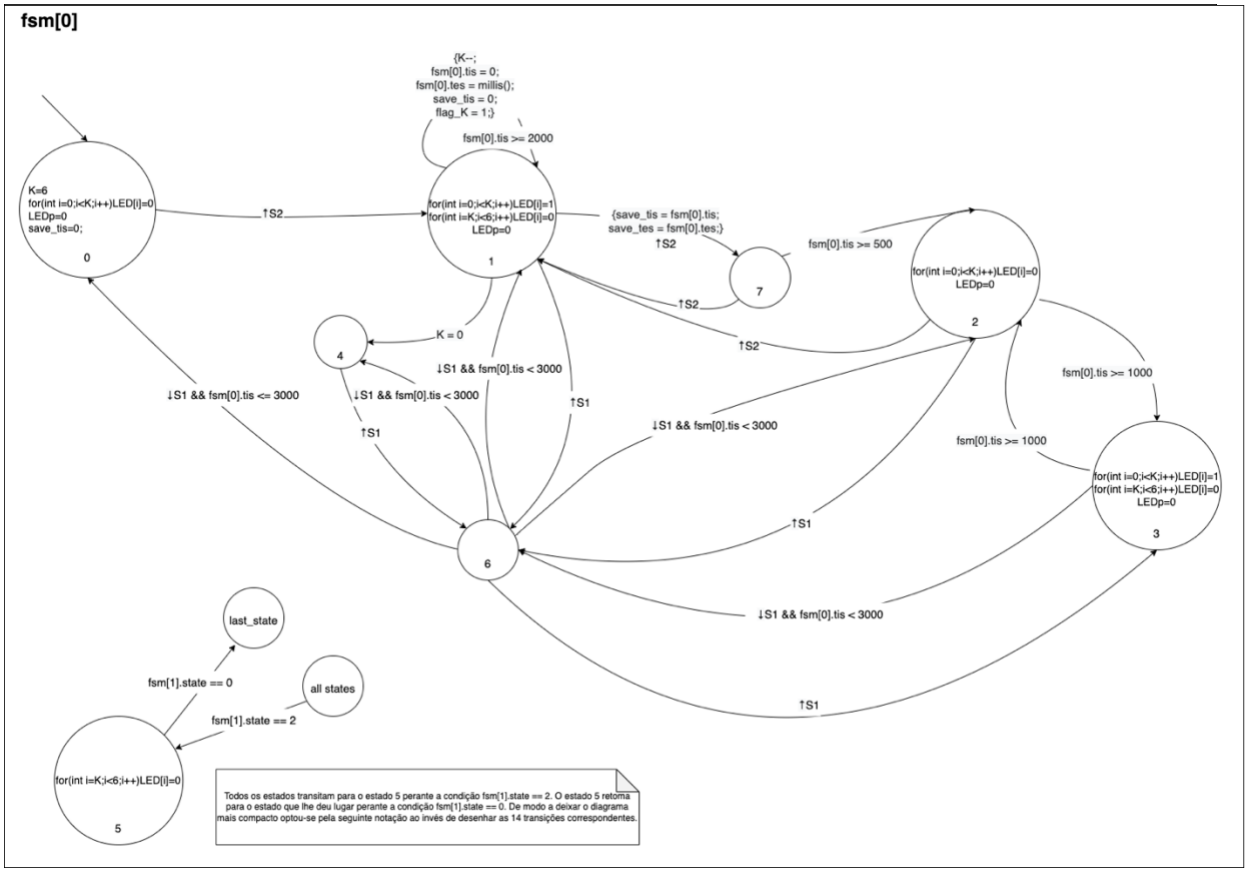

Figura 2. Máquina de Estados fsm[0]

A fsm[5] é responsável por implementar a possibilidade de incrementar um intervalo extra à ampulheta aquando de um duplo clique no botão S2. De notar que foi implementada de modo que tal operação só pudesse ser executada fora do modo de configuração, para evitar que a navegação entre modos de configuração quando rápida seja interpretada como um comando ao incremento de um intervalo extra.

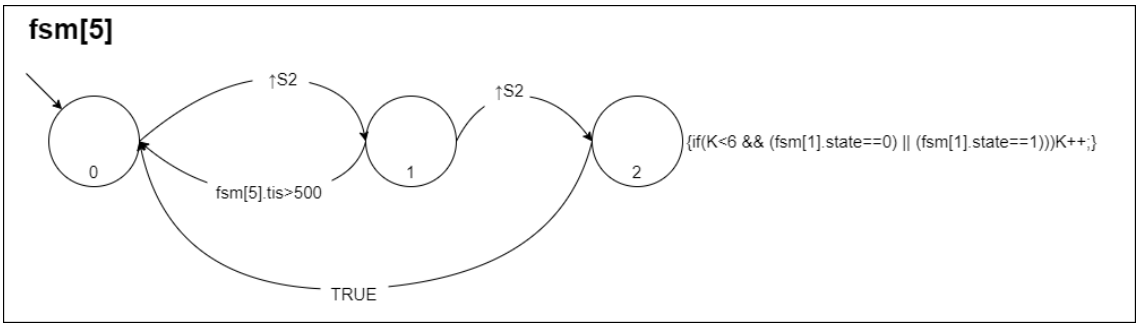

Figura 3. Máquina de Estados fsm[5]

A fsm[1] é responsável por distinguir entre um longo e um curto clique no botão S1. É ela que indica então se se deve entrar em modo de configuração ou não.

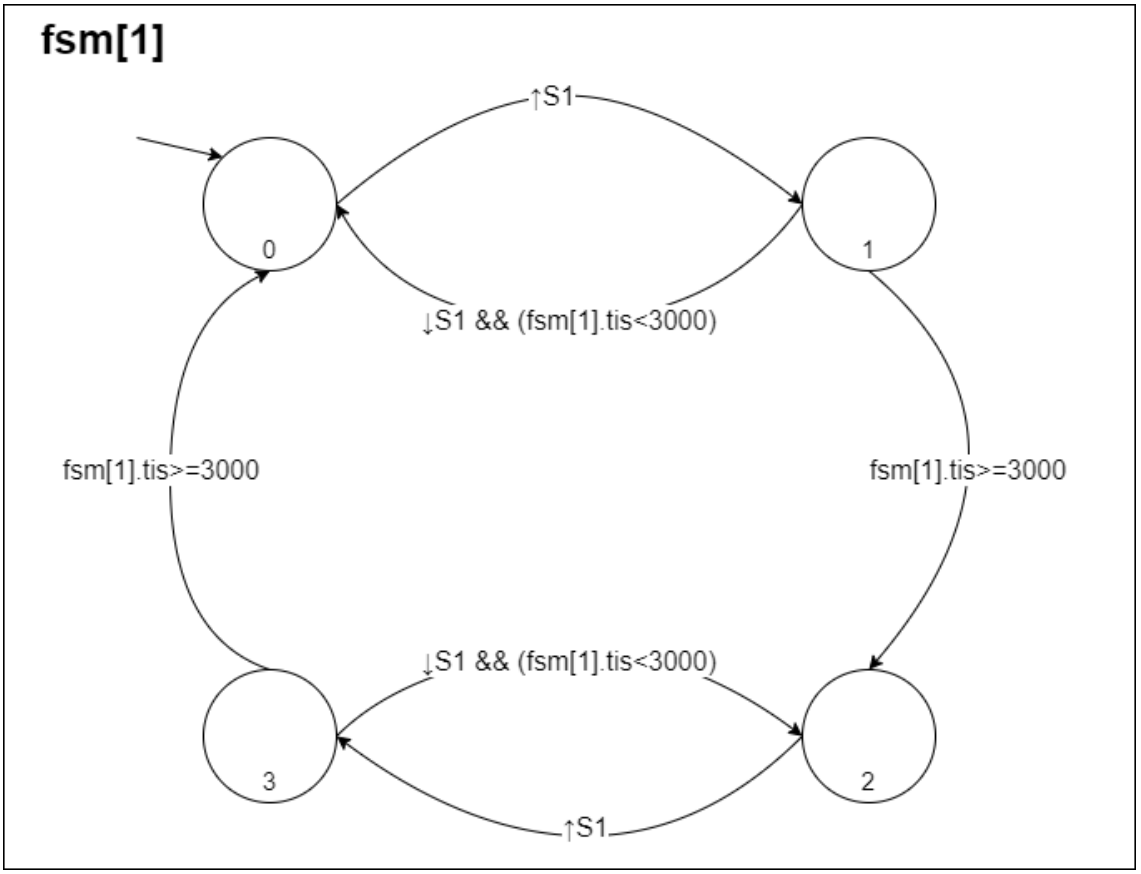

Figura 4. Máquina de Estados fsm[1]

A fsm[2] é responsável por indicar em qual dos três modos de configurações o utilizador se encontra (caso esteja em modo de configuração).

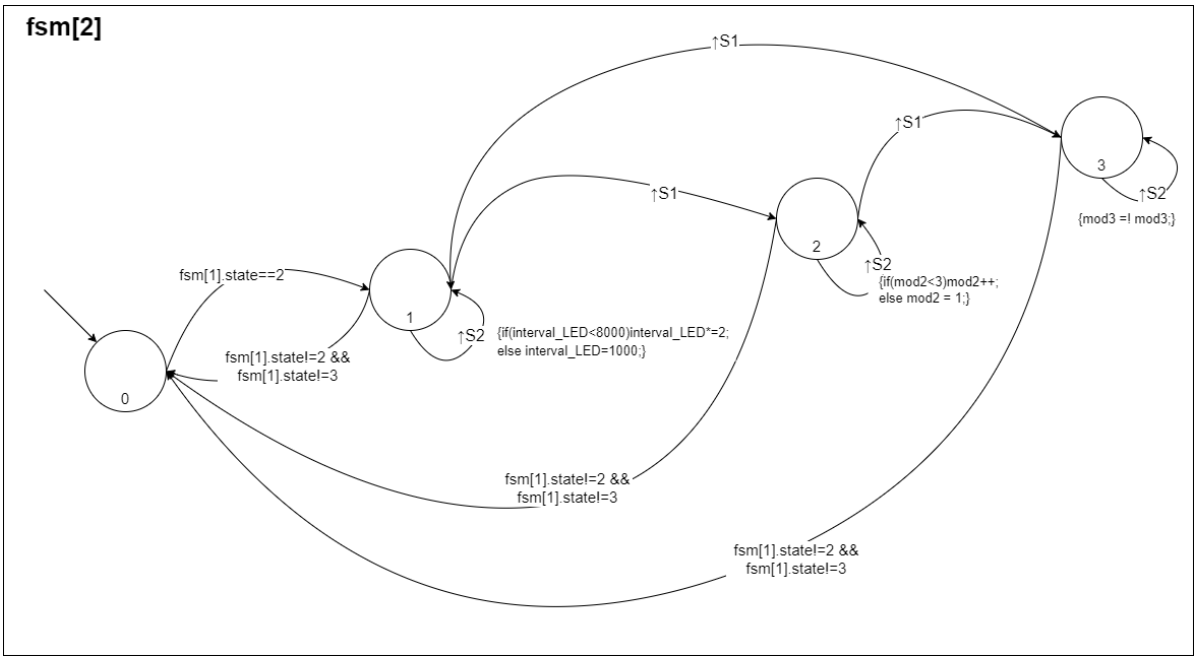

Figura 5. Máquina de Estados fsm[2]

As máquinas de estado fsm[3], fsm[4] e fsm[6] servem para controlar o piscar do LED que fornece ao utilizador a informação de qual dos modos de configuração se encontra.

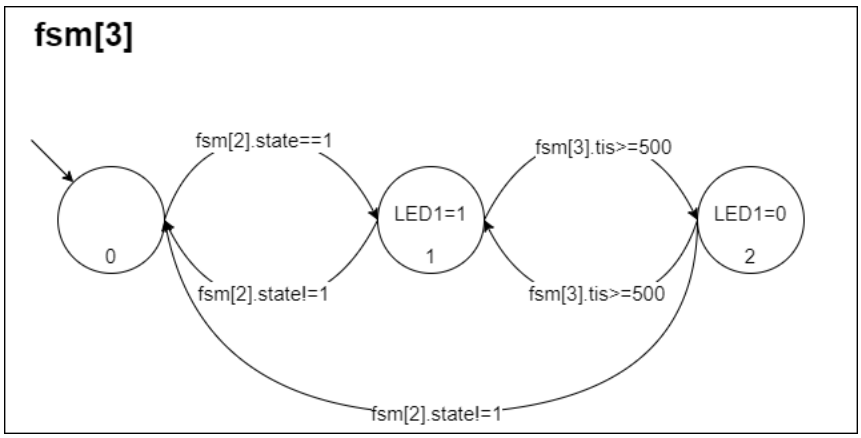

Figura 6. Máquina de Estados fsm[3]

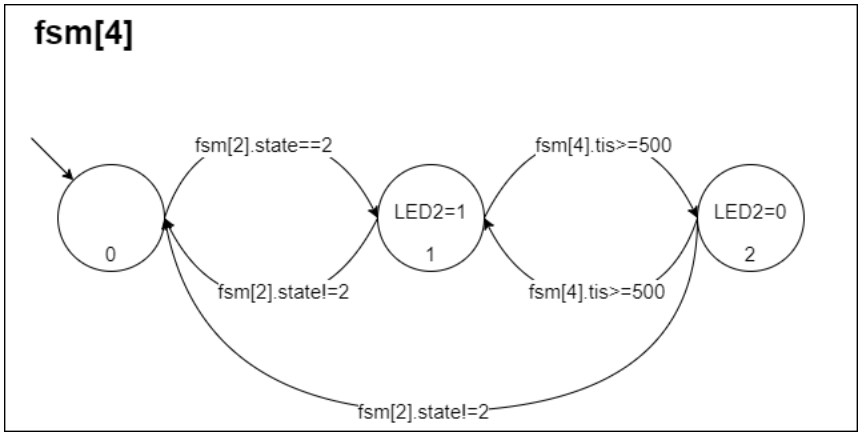

Figura 7. Máquina de Estados fsm[4]

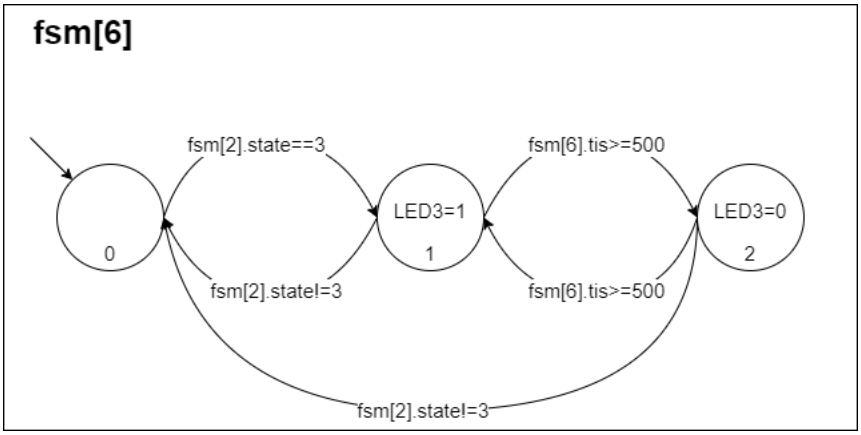

Figura 8. Máquina de Estados fsm[6]

As máquinas de estado fsm[7], fsm[10] e fsm[11] permitem controlar o LED7 de forma ao demonstrar ao utilizador como atuará o sistema na configuração selecionada para cada um dos modos de configuração, respetivamente.

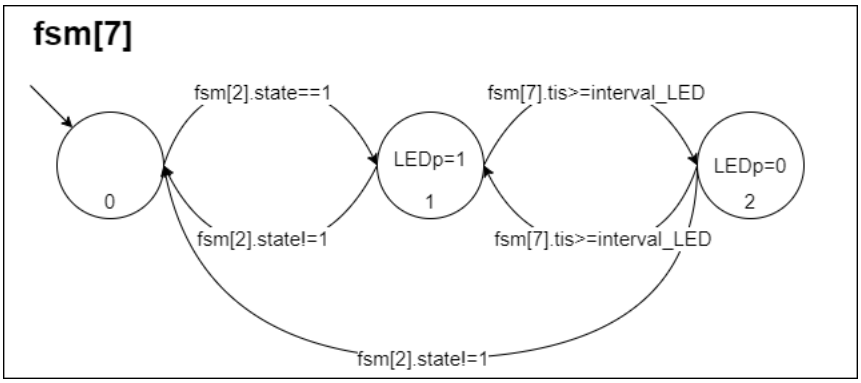

Figura 9. Máquina de Estados fsm[7]

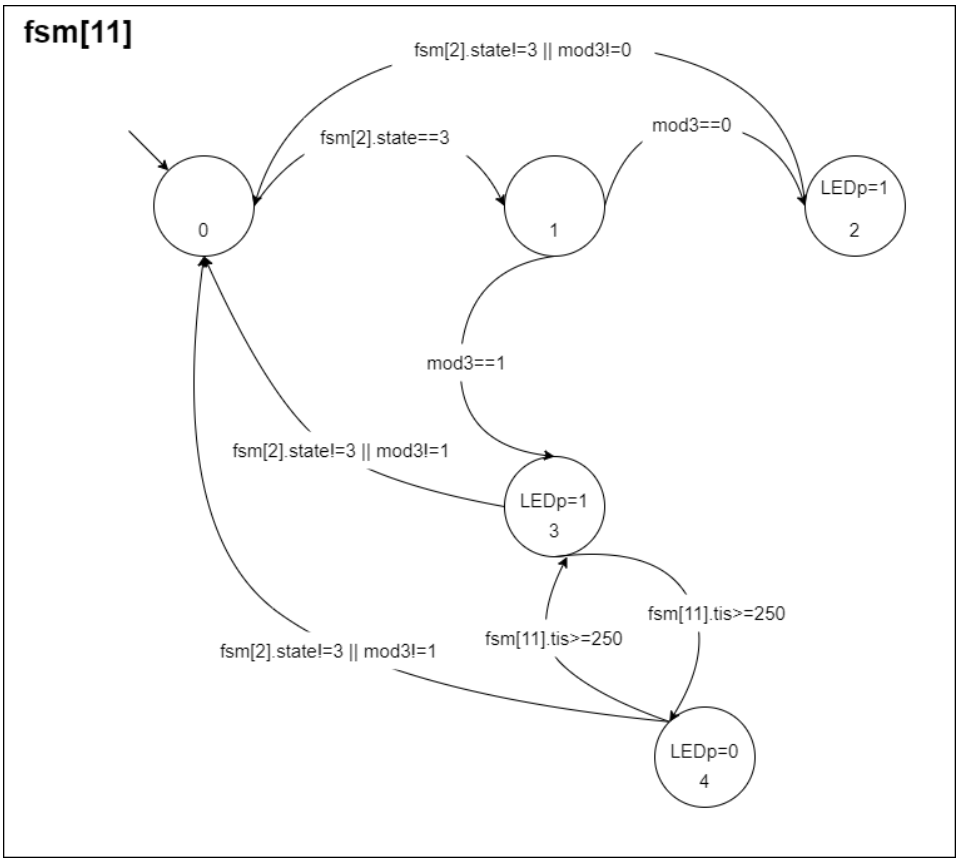

Figura 10. Máquina de Estados fsm[11]

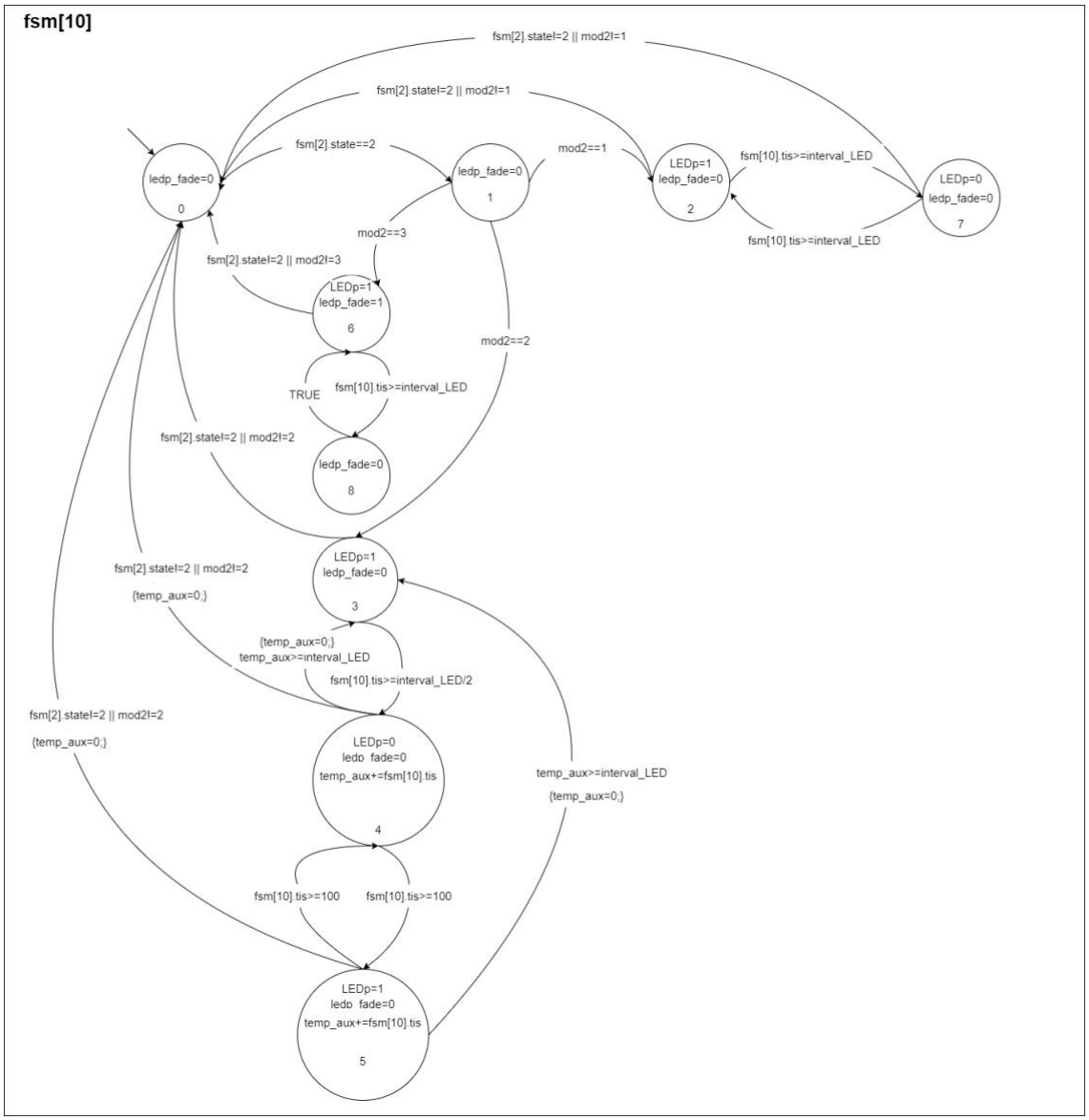

Figura 11. Máquina de Estados fsm[10]

A fsm[8] controla de que modo é apresentado ao utilizador o final do tempo da ampulheta em conformidade com o modo de configuração selecionado.

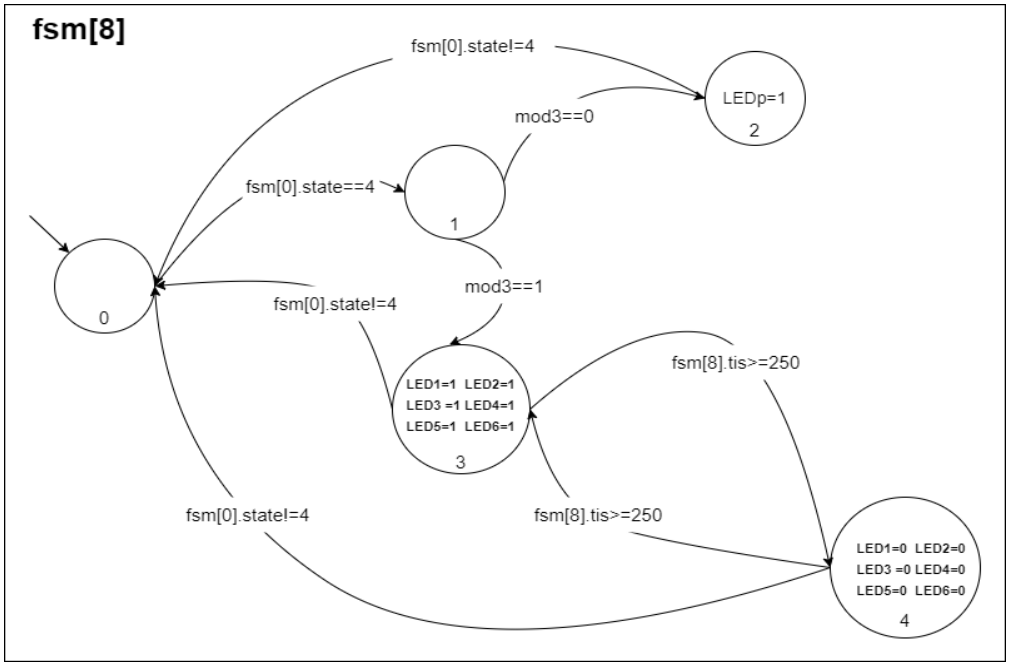

Figura 12. Máquina de Estados fsm[8]

A fsm[9] é responsável por implementar o segundo modo de configuração da maneira como o LED da ampulheta se desliga. Este é o modo em que o LED pisca na segunda metade do intervalo de tempo definido como descrito na “Descrição do Problema” [1].

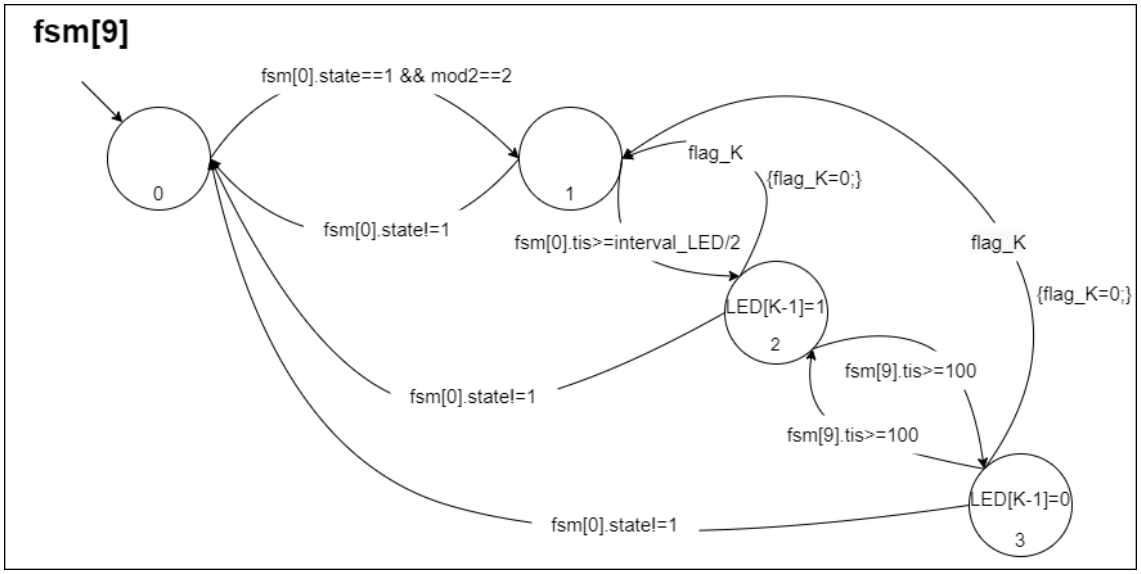

Figura 13. Máquina de Estados fsm[9]

4. Implementação

De modo a implementar a lógica modelada pelas máquinas de estado previamente descritas foi projetado um código cuja sequência de etapas pode ser vista no seguinte diagrama.

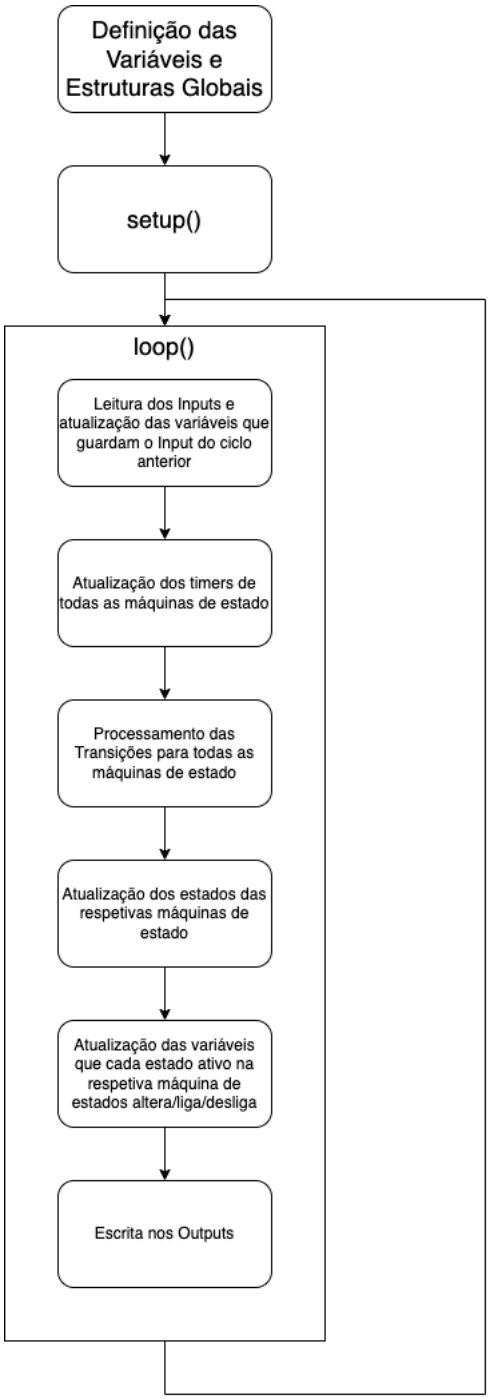

Figura 14. Diagrama dos passos executados pelo código

Inicialmente, são declaradas as variáveis a usar no código bem como a estrutura usada para implementar a máquina de estados. Deste bloco é esta estrutura que merece especial atenção. Foi implementada da seguinte forma:

```
typedef struct{
  int state, state_new;
  unsigned long tes, tis;
} fsm_t;
```

Figura 15. Estrutura implementada para guardar os dados relativos às máquinas de estado

Como pode ser visto, através da estrutura, para cada máquina de estados será guardado o estado atual, estado seguinte, instante em que a máquina entrou no estado atual (*tes*) e tempo há quanto está no estado atual (*tis*).

O Setup() é a função que é executada apenas uma vez e que serve para definir o modo dos Pinos que servem de Inputs e Outputs do sistema e os valores com que devem ser inicializadas as variáveis e máquinas de estado.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(S1_pin, INPUT);
  pinMode(S2_pin, INPUT);
  pinMode(LED1, OUTPUT);
  pinMode(LED2, OUTPUT);
  pinMode(LED3, OUTPUT);
  pinMode(LED4, OUTPUT);
  pinMode(LED5, OUTPUT);
  pinMode(LED6, OUTPUT);
  pinMode(LED7, OUTPUT);

  for(int i = 0; i < 15; i++) set_state(fsm[i], 0);

  Serial.begin(115200);
  interval = 10;
  K = 6;

  interval_LED = 2000;
  temp_aux = 0;
  ledp_fade = 0;

  mod2 = 1;
  mod3 = 0;
  last_state = 0;

  for(int i = 0; i < K; i++) LED[i] = 0;
}
```

Figura 16. Código da função Setup()

Após a execução do Setup() o microcontrolador ficará a executar repetidamente a função loop(), organizada em sub blocos como demonstrado na figura XXX. Como a mesma mostra, são, primeiramente, lidos os Inputs e atualizadas as variáveis que guardam o Input que tinha ocorrido no ciclo imediatamente anterior (tais serão úteis para saber se houve um flanco, isto é, se um botão foi apertado ou largado).

```
// Input Reading and update of "prev" variables
prevS1 = S1;
prevS2 = S2;
S1 = !digitalRead(S1_pin);
S2 = !digitalRead(S2_pin);
```

Figura 17. Estato de código relativo à leitura dos Inputs e atualização das variáveis que retêm os Inputs anteriores

Seguidamente, são atualizados os tempos em que cada máquina de estados está no respetivo estado atual. Este cálculo é feito subtraindo ao instante atual o instante em que se entrou no respetivo estado. A fsm[0] tem uma ligeira nuance face às restantes devido à existência de uma transição do estado 1 para ele próprio que obriga a recolocar o *tis* a 0 quando tal transição acontece. Esse cálculo foi conseguido utilizando uma variável *save_tis* que como foi visto na modelação é manipulado de maneira a cumprir os requisitos propostos.

```
//time management
unsigned long cur_time = millis(); // Only call millis once
fsm[0].tis = cur_time - fsm[0].tes;
if(fsm[0].state == 1){
  fsm[0].tis = cur_time - fsm[0].tes + save_tis;
}
fsm[1].tis = cur_time - fsm[1].tes;
fsm[5].tis = cur_time - fsm[5].tes;
fsm[2].tis = cur_time - fsm[2].tes;
fsm[3].tis = cur_time - fsm[3].tes;
fsm[4].tis = cur_time - fsm[4].tes;
fsm[6].tis = cur_time - fsm[6].tes;
fsm[7].tis = cur_time - fsm[7].tes;
fsm[8].tis = cur_time - fsm[8].tes;
fsm[9].tis = cur_time - fsm[9].tes;
fsm[10].tis = cur_time - fsm[10].tes;
fsm[11].tis = cur_time - fsm[11].tes;
```

Figura 18. Estrato de código relativo à atualização dos timers

Posteriormente são calculadas as transições em cada uma das máquinas de estado. Como exemplo, a figura seguinte mostra o cálculo das transições na fsm[5]:

```
// State Machine 5 Process (DoubleClick)
if((fsm[5].state == 0) && S2 && !prevS2){
  fsm[5].state_new = 1;
}
else if((fsm[5].state == 1) && fsm[5].tis > 500){
  fsm[5].state_new = 0;
}
else if((fsm[5].state == 1) && S2 && !prevS2){
  fsm[5].state_new = 2;
}
else if(fsm[5].state == 2){
  fsm[5].state_new = 0;
}
```

Figura 19. Estrato de código do processamento das transições de uma das máquinas de estado implementadas

Após o processamento das transições, é necessário atualizar os estados em todas as máquinas. Para tal foi usada uma função *set_state()* que verifica se houve uma alteração de estado e quando tal acontece, atualiza o estado, coloca o *tis* a 0 e o *tes* com o valor do instante em que é feita a transição. Nas figuras seguintes é possível ver a função *set_state()* a ser chamada com os devidos parâmetros dentro da função *loop()* e o código da função *set_state()*, respetivamente.

```
// Update the States
set_state(fsm[0], fsm[0].state_new);
set_state(fsm[5], fsm[5].state_new);
set_state(fsm[1], fsm[1].state_new);
set_state(fsm[2], fsm[2].state_new);
set_state(fsm[3], fsm[3].state_new);
set_state(fsm[4], fsm[4].state_new);
set_state(fsm[6], fsm[6].state_new);
set_state(fsm[7], fsm[7].state_new);
set_state(fsm[8], fsm[8].state_new);
set_state(fsm[9], fsm[9].state_new);
set_state(fsm[10], fsm[10].state_new);
set_state(fsm[11], fsm[11].state_new);
```

Figura 21. Estrato de código onde são executados os set_state()

```
void set_state(fsm_t &fsm, int state_new){
  if(fsm.state != state_new){
    fsm.state = state_new;
    fsm.tis = 0;                  /* tis = time in state    */
    fsm.tes = millis();           /* tes = time enter state */
  }
}
```

Figura 20. Código da função set_state

De seguida, são atualizadas as variáveis que cada máquina de estados manipula. A título de exemplo, a seguinte figura mostra como foi implementado para o caso da fsm[11].

```
// Actions that FSM10 states trigger
if(fsm[11].state == 0){}
else if(fsm[11].state == 1){}
else if(fsm[11].state == 2){
  LEDp = 1;
}
else if(fsm[11].state == 3){
  LEDp = 1;
}
else if(fsm[11].state == 4){
  LEDp = 0;
}
```

Figura 22. Estrato de código das ações ativadas por uma das máquinas de estado

Por último, são escritas nas saídas (outputs) os valores respetivos. Como em determinados modos de configuração é necessário criar um efeito de *Fade out*, então, as saídas foram escritas utilizando a função *analogWrite()* [1]. Esta função cria uma onda PWM [2] no pino selecionado, através duma correta configuração do parâmetro do *duty cycle* é possível ser usada para manipular o brilho do LED. Desta forma, primeiro verificamos se estamos num dos modos em que os LEDs se devem desligar com efeito *Fade out*. Depois, verificamos qual o LED que deve realizar o efeito e chamamos a função com o devido *duty cycle*, nas restantes saídas é usada também a mesma função, mas com o duty cycle a 100% (a função aceita valores de *duty cycle* de 0 a 255, no qual 0 corresponde a sempre desligado e 255 sempre ligado). Para calcular o *duty cycle* para o LED que realiza o efeito *Fade out* calculamos a razão entre o tempo que falta para o LED estar completamente desligado e o total de tempo que demora a desligar, isto é [(Intervalo de tempo total – Tempo já passado) / Intervalo de tempo total]. No caso de não se verificar o modo de configuração que exige o efeito de *Fade out*, as saídas são escritas também com recurso a *analogWrite()* com o duty cycle a 100% (255). As duas figuras seguintes ilustram o caso em que não é necessário o efeito de *Fade out* e uma parte do código em que tal efeito é desejado, respetivamente.

```
if((mod2 != 3) || (fsm[0].state != 1)){
  analogWrite(LED1, 255*LED[5]);
  analogWrite(LED2, 255*LED[4]);
  analogWrite(LED3, 255*LED[3]);
  analogWrite(LED4, 255*LED[2]);
  analogWrite(LED5, 255*LED[1]);
  analogWrite(LED6, 255*LED[0]);
}
```

Figura 23. Estrato de código relativo à escrita nos Outputs fora do modo de Fade Out

```
// Set the outputs
if((mod2 == 3) && (fsm[0].state == 1)){
  if(K - 1 == 5){
    analogWrite(LED1, LED[5]*255*(interval_LED-fsm[0].tis)/interval_LED);
    analogWrite(LED2, 255*LED[4]);
    analogWrite(LED3, 255*LED[3]);
    analogWrite(LED4, 255*LED[2]);
    analogWrite(LED5, 255*LED[1]);
    analogWrite(LED6, 255*LED[0]);
  }
  else if(K-1 == 4){
    analogWrite(LED1, 255*LED[5]);
    analogWrite(LED2, LED[4]*255*(interval_LED-fsm[0].tis)/interval_LED);
    analogWrite(LED3, 255*LED[3]);
    analogWrite(LED4, 255*LED[2]);
```

Figura 24. Estrato de código relativo à escrita nos Outputs dentro do modo de Fade Out

5. Montagem (Fotos)

Figura 25. Foto do Grupo com a montagem

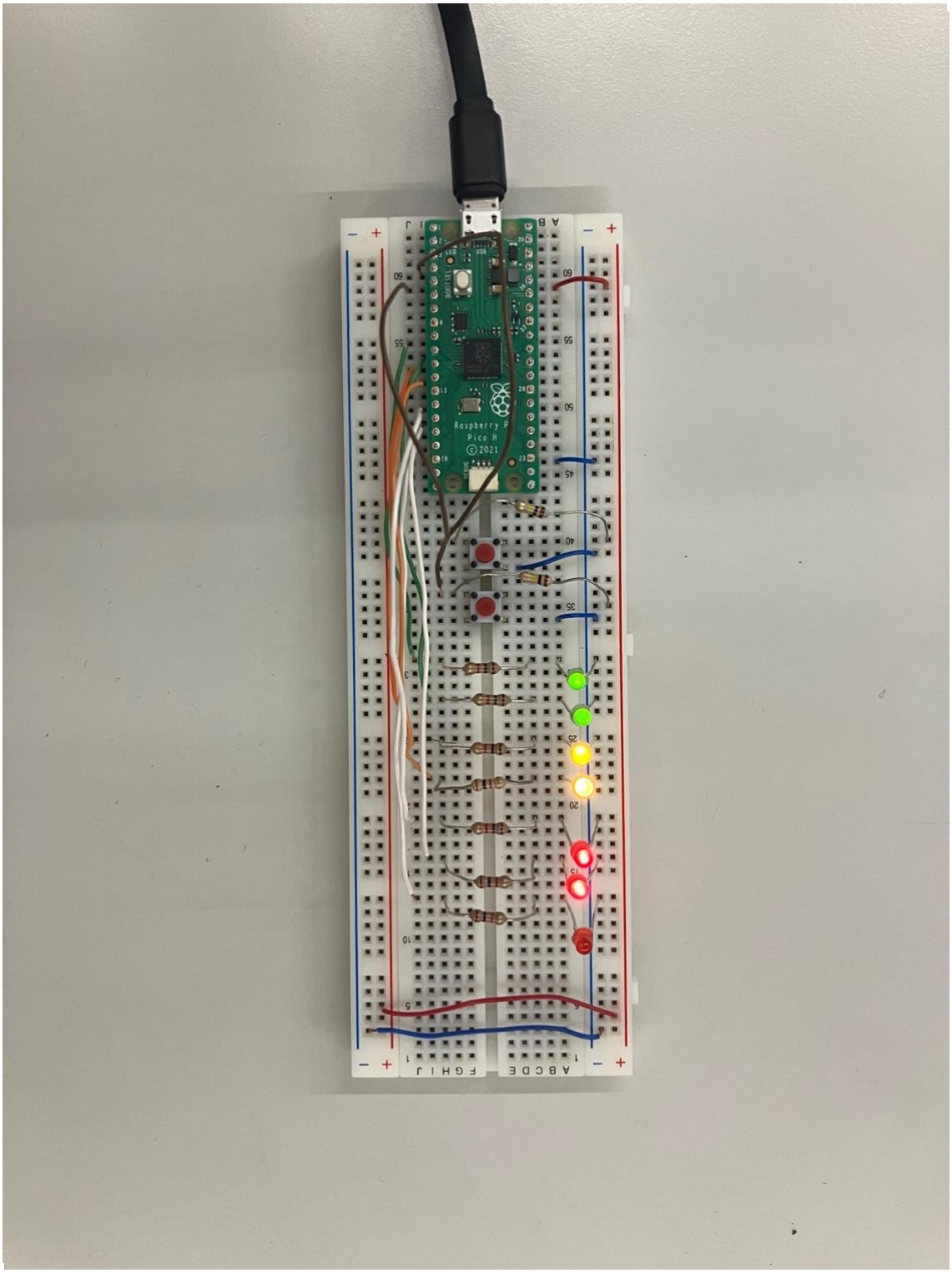

Figura 26. Foto da montagem

Referências

[1] https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/
[2] https://docs.arduino.cc/learn/microcontrollers/analog-output