UNIVERSIDADE FEDERAL DO PARANÁ
DELT - DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE ENGENHARIA ELÉTRICA

WENDEURICK EMERICK SILVERIO

DESENVOLVIMENTO DE MESA DE LUZ INTERATIVA PARA EXPERIMENTO DE CIÊNCIAS PARA CRIANÇAS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2018

WENDEURICK EMERICK SILVERIO

DESENVOLVIMENTO DE MESA DE LUZ INTERATIVA PARA EXPERIMENTO DE CIÊNCIAS PARA CRIANÇAS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica da Universidade Federal do Paraná, como requisito parcial para a obtenção do título de Engenheiro Eletricista.

Orientador: Dr. James Alexandre Baraniuk

CURITIBA
2018

À memória de Thiago Borges Fonseca.

Gosto que tudo seja real e que tudo esteja certo;
E gosto porque assim seria, mesmo que eu não gostasse.
Por isso, se morrer agora, morro contente,
Porque tudo é real e tudo está certo.
(CAEIRO, Alberto (Fernando Pessoa), 1915).

RESUMO

SILVERIO, Wendeurick. Desenvolvimento de Mesa de Luz Interativa para Experimento de Ciências para Crianças. 2018. 67 f. Trabalho de Conclusão de Curso – Curso de Engenharia Elétrica, Universidade Federal do Paraná. Curitiba, 2018.

O presente trabalho refere-se ao desenvolvimento do protótipo de uma matriz de LED que será parte da exposição interativa "Luz, Ciência e Emoção", idealizada pela arquiteta Dra. Maristela Mitsuko Ono e pelo engenheiro Dr. James Alexandre Baraniuk. A matriz, chamada de "Mesa de Bolinhas", se enquadra no setor artístico da exposição, que traz experimentos envolvendo os conceitos de luz trabalhados nos ensinos pré-escolar e fundamental, e proporcionará uma experiência tangível-visual impactante aos observadores, causando deslumbramento e entusiamos através da interação com arte e tecnologia.
O projeto conta com técnicas de *software* e *hardware* para endereçar 129 LEDs RGBs e controlar seus 387 canais de PWM, mapear 128 sensores reflexivos e submetê-los a rotinas de *debouncing*, implementar máquina de estados finita em *firmware*, desenvolver um teclado analógico com apenas uma GPIO, projetar um conversor de nível lógico não-inversor com um único transistor, entre outras funcionalidades.

Palavras-chave: Matriz de LED. Matriz de sensores. Arte generativa.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	do inglês *Analog to Digital Converter*
API	do inglês *Application Programming Interface*
CPU	do inglês *Central Processing Unit*
FIFO	do inglês *First In, First Out*
FSM	do inglês *Finite-state Machine*
GPIO	do inglês *General Purpose Input/Output*
LDR	do inglês *Light Dependent Resistor*
LED	do inglês *Light Emitting Diode*
MISO	do inglês *Master Input Slave Output, or Master In Slave Out*
MOSI	do inglês *Master Output Slave Input, or Master Out Slave In*
MuMa	Museu de Arte Municipal
NRZ	do inglês *Non-return-to-zero*
OTA	do inglês *Over-the-air*
PCI	Placa de Circuito Impresso
PIBIC	Programa Institucional de Bolsas de Iniciação Científica
PWM	do inglês *Pulse Width Modulation*
RAM	do inglês *Random Access Memory*
RGB	Sistema de cores aditivas, do inglês *Red*, *Green* e *Blue*
SCLK	do inglês *Serial Clock*
SMD	do inglês *Surface Mounting Device*
SPIFFS	do inglês *SPI Flash File System*
SS	do inglês *Slave Select*
TCP	do inglês *Transmission Control Protocol*
TJB	Transistor de Junção Bipolar

UART	do inglês *Universal Asynchronous Receiver-Transmitter*
UDP	do inglês *User Datagram Protocol*
UFPR	Universidade Federal do Paraná
UNESCO	Organização das Nações Unidas para a Educação, a Ciência e a Cultura
USB	do inglês *Universal Serial Bus*

LISTA DE SÍMBOLOS

Ω	Unidade de medida da resistência elétrica
λ	Comprimento de onda

SUMÁRIO

1 INTRODUÇÃO

A luz exerce um papel essencial no nosso cotidiano e está presente das mais diversas formas: iluminação, medicina, pesquisas científicas, geração de energia, telecomunicações, educação, arte, cultura, etc. A Assembleia Geral das Nações Unidas proclamou o ano de 2015 como o Ano Internacional da Luz e das Tecnologias Baseadas na Luz (United Nations, 2014), a fim de reconhecer tal importância para a vida dos cidadãos e para o desenvolvimento futuro da sociedade mundial. No ano da celebração, a UNESCO promoveu uma série de eventos por vários países UNESCO (2015), com o intuito de destacar que o aumento da consciência mundial e o fortalecimento do ensino da ciência e das tecnologias da luz são essenciais para abordar os desafios futuros e atuais, tais como o desenvolvimento sustentável, a energia e as comunicações, assim como para melhorar a qualidade de vida dos países menos desenvolvidos e dos em desenvolvimento.

Baseada em tal iniciativa, a exposição Luz, Ciência e Emoção, idealizada pela arquiteta Dra. Maristela Mitsuko Ono e pelo engenheiro Dr. James Alexandre Baraniuk, traz experimentos envolvendo os conceitos de luz trabalhados nos ensinos pré-escolar e fundamental, cada um com seu grau de impressão aos sentidos. A (Figura 1) apresenta uma parte da mostra que ocorreu entre março e junho de 2017, no Museu de Arte Municipal, em Curitiba.

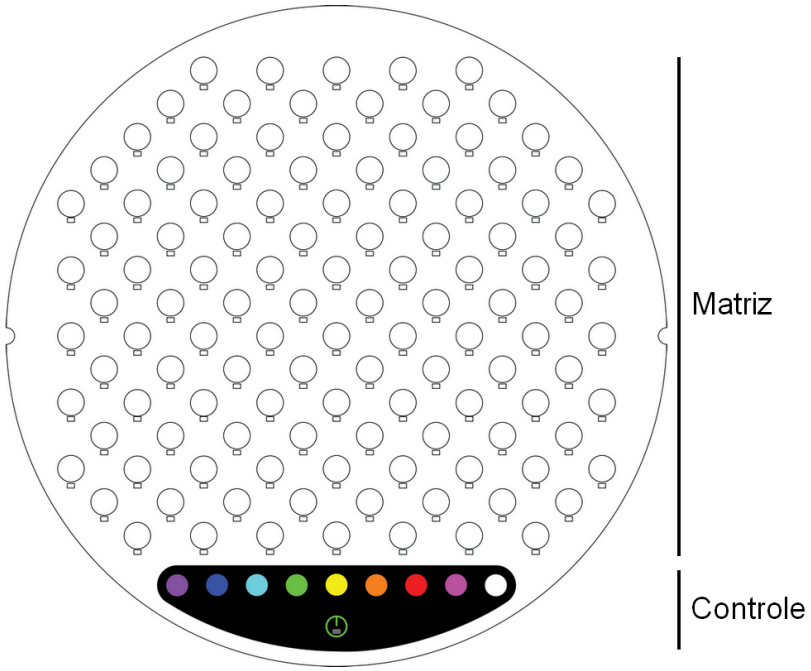
Figura 1 – Parte da exposição "Luz, Ciência e Emoção", no MuMa

Fonte: (UFPR, 2017)

Dentre outros projetos que participou durante o Programa de Iniciação Científica [1], o autor trabalhou na seção Matriz da Mesa de Bolinhas (Figura 2), projeto este que ocupará uma posição de destaque dentro da área artística da mostra. Trata-se de uma matriz de *LEDs* interativa, composta por mais de 100 bolinhas de *ping-pong*, cada uma sobre um correspondente par *LED*-sensor reflexivo. Ela permite que o espectador "pinte com luz" ao passar a mão sobre a mesa, proporcionando uma experiência tangível-visual impactante, causando deslumbramento e entusiasmo através da arte e interação.

O trabalho aqui proposto trata do desenvolvimento do *hardware* e *firmware* embarcado da seção Controle da mesa. Tal setor é responsável pela identificação e tratamento dos sinais de entrada e pelo acionamento da matriz de saída.

Figura 2 – Vista superior da mesa de Bolinhas



Fonte: (MITSUKO, 2016)

1.1 JUSTIFICATIVA

Sendo parte da exposição Luz, Ciência e Emoção, espera-se que o projeto fomente o envolvimento com a arte generativa e com as tecnologias envolvendo luz. Além de poder ser prestigiado durante a mostra, trata-se também de um projeto de código e *hardware* abertos, que serão disponibilizados à comunidade para que possam ser estudados e adaptados às suas necessidades.

Ademais, temas pouco explorados durante a graduação serão implementados, tais como técnica de contorno (*debounce*, identificação de comandos reais sob contextos com

[1] Projeto de Pesquisa: Luz, Ciência e Emoção: Exposição Interativa para Crianças (PIBIC Voluntária: 2016019080)

ruído), padrão de *design* em *software* (FSM, máquina de estados finita) e geração de múltiplos canais de PWM.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Finalizar o *hardware* (seção Controle) e desenvolver o *firmware* embarcado da Mesa de Bolinhas, apresentando um protótipo funcional do projeto.

1.2.1.1 OBJETIVOS ESPECÍFICOS

- Mapeamento da matriz de saída, onde cada *LED* possa ser controlado individualmente;
- Mapeamento da matriz de entrada, onde cada sensor possa ser lido individualmente;
- *Software* microcontrolado que gerencie a interface humano-máquina;
- Leiaute e montagem das Placas de Circuito Impresso;
- Integração da eletrônica com a mecânica do projeto.

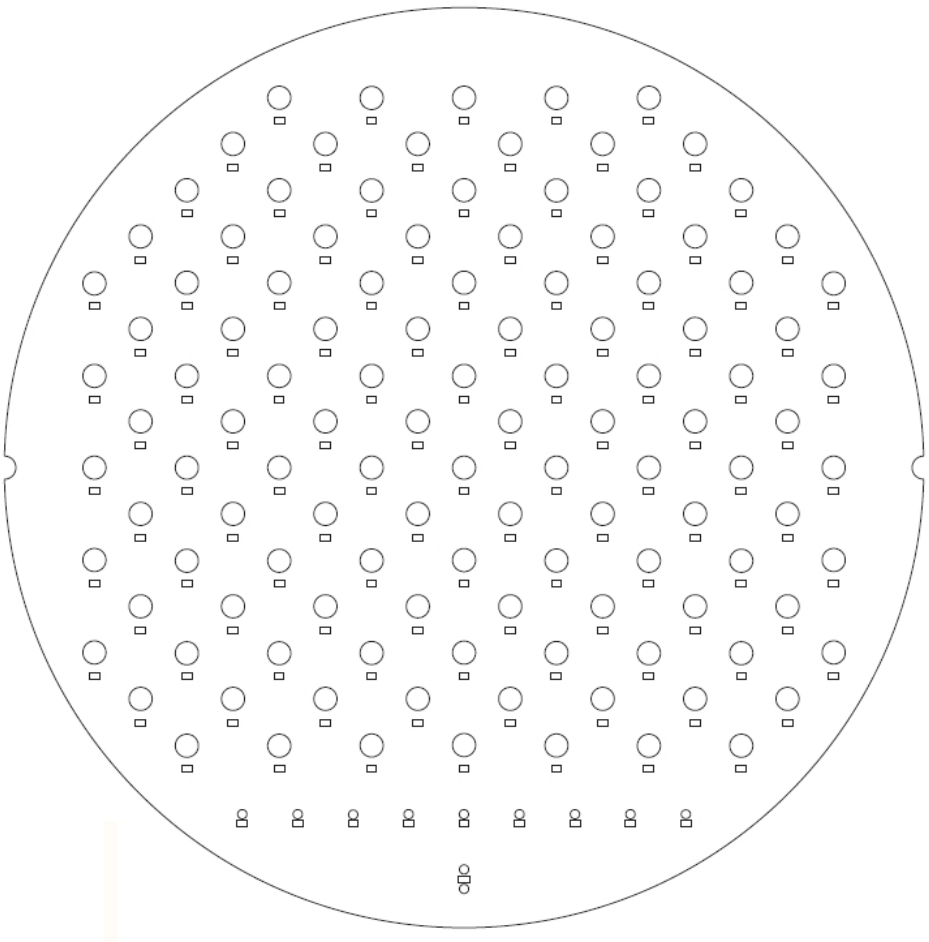
2 REVISÃO TEÓRICA

Este capítulo aborda alguns conceitos dos principais componentes e ferramentas utilizadas, bem como a especificação do projeto.

2.1 PROJETO ARQUITETÔNICO

O projeto arquitetônico definiu a quantidade e a disposição de cada objeto. Ao todo, são 128 sensores e 129 pontos de luz, dispostos sob um tampo acrílico de 1 m de diâmetro, conforme apresentado na Figura 3.

Figura 3 – Disposição dos objetos na Mesa de Bolinhas



Fonte: (MITSUKO, 2016)

A arquiteta também especifica que o funcionamento da mesa deve ser orientado a eventos, provenientes das seções Controle e Matriz (Figura 2):

- A mesa sai do modo ocioso ao manter-se a mão sobre a bolinha *Power* por mais de 1 s: acende-se o seletor de cores e a mesa passa a interpretar os demais comandos;

- Escolhe-se uma cor ao manter-se a mão sobre uma determinada bolinha do seletor de cores por mais de 1 s;
- Ao passar a mão sobre uma determinada bolinha da seção Matriz, a mesma deve responder com a última cor escolhida;
- Uma bolinha da seção Matriz pode ser apagada se a "cor" escolhida tiver sido a da bolinha *Power*;
- Ao manter-se a mão por mais de 3 s sobre a bolinha *Power*, todas as bolinhas da seção Matriz devem ser apagadas; e
- Ao manter-se a mão por mais de 5 s sobre a bolinha *Power*, a mesa entra em modo ocioso: as bolinhas do seletor de cores são apagadas e a mesa passa a ignorar os comandos sobre a Matriz.

A partir da especificação acerca do funcionamento, notam-se alguns desafios do ponto de vista do eletrônico/lógico, tais como:

- A disposição dos objetos e o tamanho da mesa;
- A capacidade de comandar individualmente 129 *LEDs* coloridos (*RGB*), ou seja $129 \times 3 = 387$ canais de cor;
- A capacidade de interpretar os níveis lógicos dos 128 sensores;
- A capacidade de responder, simultaneamente, aos comandos temporais provenientes de cada sensor.

A seleção dos componentes e das ferramentas foi baseada na especificação e nos desafios acima citados e será discutida a seguir nas demais seções deste capítulo.

2.2 SENSOR

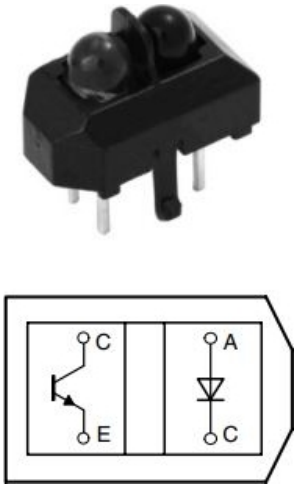
A função do sensor é identificar a posição de um objeto, no caso, a mão do espectador, sobre o tampo acrílico, e para manter-se centrado nos objetivos do projeto arquitetônico, a definição do tipo de sensor foi baseada em dois pontos essenciais:

I. O elemento principal do projeto é a luz que flui da bolinha de *ping-pong*;
II. A exposição pode ser levada a diversos lugares e ser instalada nos mais variados ambientes, como museus, escolas, saguões, etc.

A questão (I) implica que o sensor não deve irradiar ondas eletromagnéticas dentro do espectro visível, que vai de 400 nm a 700 nm (HALLIDAY; WALKER; RESNICK, 2007). Já a questão (II) implica que a instalação será submetida a locais com diferentes graus de iluminação, portanto os sensores devem ser imunes à variação luminosa do ambiente.

Assim sendo, optou-se pelo sensor óptico reflexivo *TCRT5000L*, apresentado na Figura 4, cujas características relevantes para o projeto são apresentadas na Tabela 1.

Figura 4 – Sensor TCRT5000



Fonte: Vishay Semiconductors (2008)

Tabela 1 – Principais características do sensor TCRT5000.

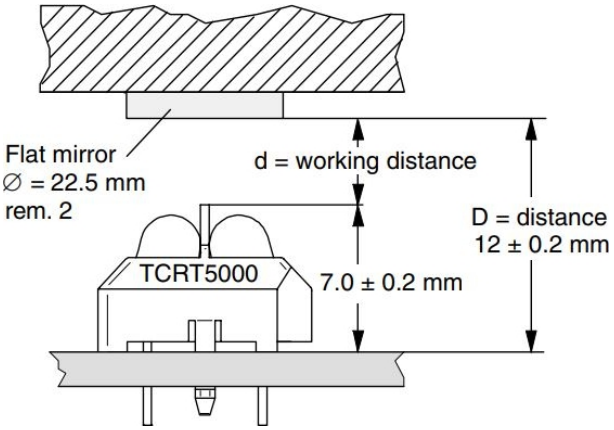
Parâmetro	**Condição**	**Símb.**	**Mín.**	**Típ.**	**Máx.**	**Unidade**
LED - Corrente direta		I_F		60		mA
LED - Queda direta de tensão	$I_F = 60\ mA$	V_F		1,25	1,5	V
LED - Comprimento de onda	$I_F = 100\ mA$	λ_P	940			nm
Sensor - Corrente de coletor	$@5\ V$, $I_F = 10\ mA$, $D = 12\ mm$	I_C	0,5	1	2,1	mA
Sensor - Queda de tensão entre coletor-emissor na saturação	$I_F = 10\ mA$, $I_C = 0{,}1\ mA$, $D = 12\ mm$	$V_{CE_{sat}}$			0,4	V

Fonte: Vishay Semiconductors (2008)

Trata-se de um sensor reflexivo infravermelho (λ_P), ou seja, opera fora do espectro visível, o que contorna a implicação da questão (I) apresentada anteriormente. Além disso, diferentemente de um sensor passivo, por exemplo um LDR, o TCRT5000 opera irradiando luz e, ademais, também contém filtro óptico embutido, absorvendo somente comprimentos de onda próximas a (λ_P). Essa duas características contornam a implicação da questão (II).

A saída transistorizada permite que o sensor seja interpretado de forma lógica. Sua aplicação (Figura 5) vai desde sensoriamento de *encoders* e posições de "fim de curso" até detecção de papéis, cartões, fitas, etc. Foi escolhida a variante "L" (*TCRT5000***L**) por ser a versão com terminais estendidos.

Figura 5 – Aplicação do sensor TCRT5000



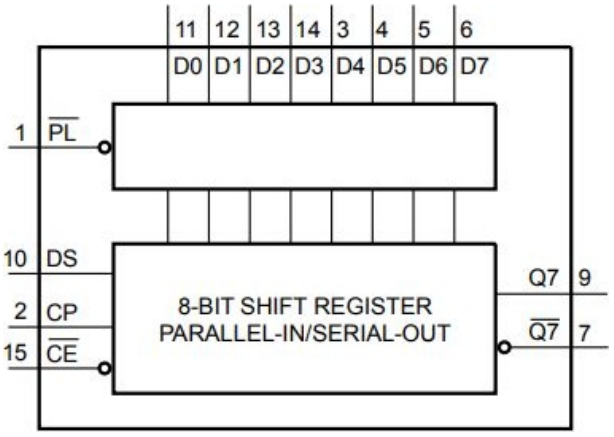
Fonte: Vishay Semiconductors (2008)

2.3 REGISTRADORES DE DESLOCAMENTO

A necessidade de interpretar individualmente os 128 sensores implica na mesma quantidade de entradas do circuito de processamento. Então, optou-se por lidar com entradas virtuais, através de um barramento serial, por meio de registradores de deslocamento. Dessa forma, com poucas GPIOs, o microcontrolador pode interpretar os níveis lógicos de todos os sensores conectados ao barramento.

Devido sua grande difusão no mercado, o registrador de deslocamento escolhido foi o 74HC165. A Figura 6 apresenta o diagrama funcional do circuito integrado.

Figura 6 – Diagrama funcional do 74HC165

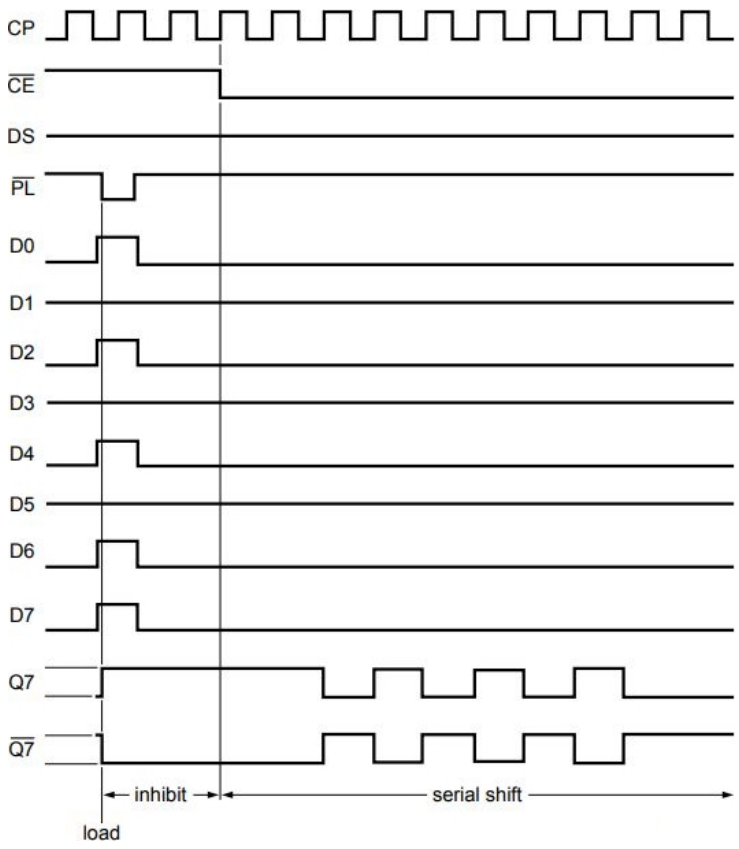


Fonte: NXP Semiconductors (2017)

Trata-se de um registrador de deslocamento com 8 entradas em paralelo (D_N) e saídas complementares (Q_7 e $\overline{Q_7}$) em serial. Basicamente, seu funcionamento (Figura 7) constitui-se na captura instantânea do nível lógico de suas entradas (borda de descida da Entrada Assíncrona

de Carga Paralela, $\overline{P_L}$) e na transmissão desses valores através de deslocamento ordenado ($D_7 \rightarrow D_0$) na saída serial (Q_7), nos eventos de borda de subida na entrada de *clock* (C_P).

Figura 7 – Diagrama temporal do 74HC165



Fonte: NXP Semiconductors (2017)

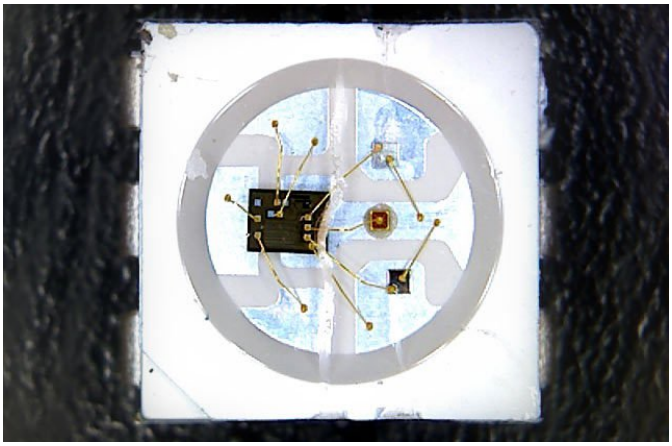
2.4 LED

Para formar a matriz de saída, optou-se pelo LED WS2812S (Figura 8), um dispositivo em encapsulamento SMD5050 com controlador interno, o WS2811: um *driver* especial para LEDs, com 3 saídas de 8 *bits* de resolução, ou seja, pode-se gerar até 16777216 cores do padrão RGB. Esse controlador é muito utilizado em fitas de LEDs endereçáveis (Figura 9), pois permite a conexão de vários controladores em um mesmo barramento serial (protocolo NRZ), podendo-se controlar cada LED individualmente.

O fato de possuir um controlador embutido no próprio encapsulamento implica diretamente sobre o valor do LED, custando um pouco mais que o dobro do valor médio de outros LEDs RGBs mais simples. Apesar disso, o custo acaba sendo compensado pelos benefícios que o *driver* proporciona.

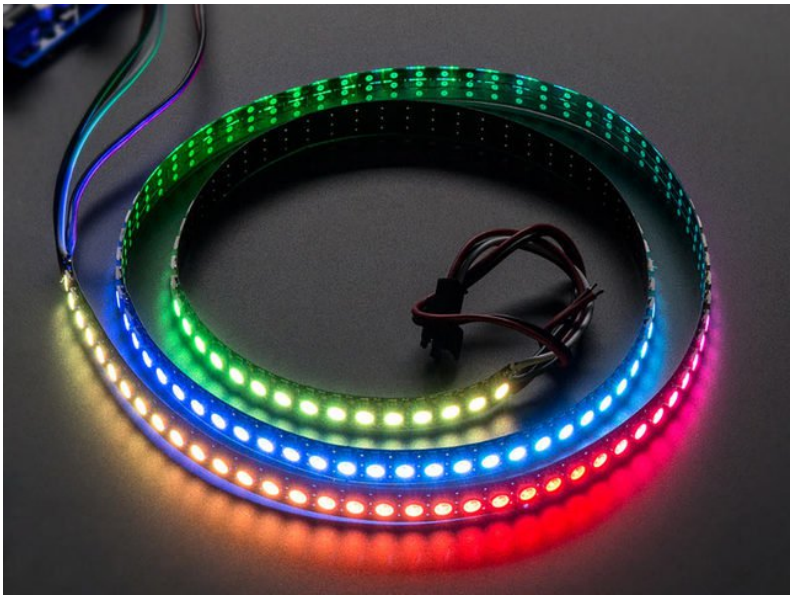
O modelo do LED já havia sido selecionada durante o Programa de Iniciação Científica que o autor participou, já sendo utilizado no circuito da matriz. No projeto aqui proposto, eles serão utilizados no seletor de cores, discutido no Capítulo 3.

Figura 8 – Vista aproximada do LED WS2812 e seu controlador interno



Fonte: Burgess (2017)

Figura 9 – Fita de LED endereçável utilizando o WS2812

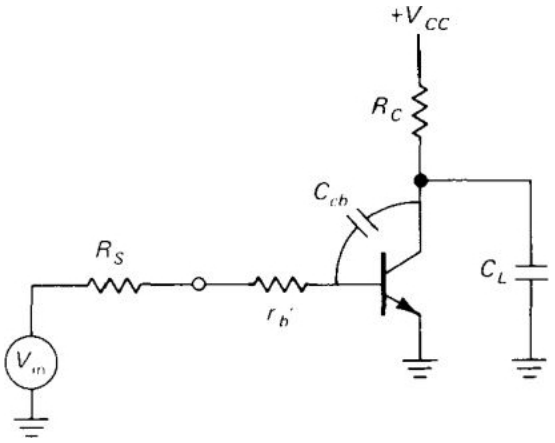


Fonte: Burgess (2017)

2.5 BAKER CLAMP

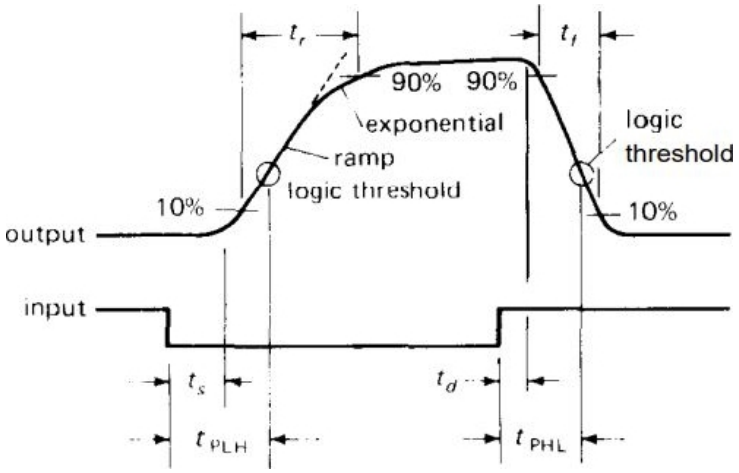
Segundo Horowitz e Hill (1989), os mesmos efeitos que limitam o desempenho de amplificadores lineares em altas frequências (a combinação de capacitância de junção, capacitância de retorno e capacitância parasita) também impõem limitações de velocidade em circuitos digitais de alta frequência. A Figura 10 apresenta um amplificador "emissor comum", que pode atuar como uma chave inversora quando operado em corte e saturação, alimentado por uma fonte de pulsos com tempos de subida e descida extremamente curtos, e a Figura 11 apresenta uma curva típica sobre a carga (R_C) desse tipo de amplificador ao considerar as componentes parasitas.

Figura 10 – Representação de uma chave inversora com TJB



Fonte: (HOROWITZ; HILL, 1989)

Figura 11 – Forma de onda da chave inversora com TJB

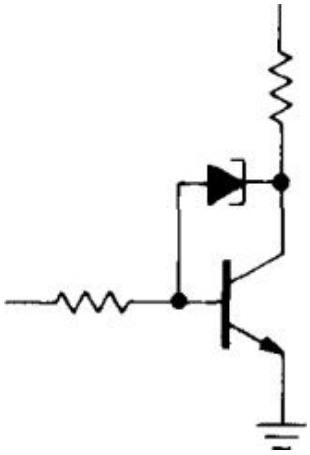


Fonte: (HOROWITZ; HILL, 1989)

Nota-se que, assim que a fonte estabelece o pulso em nível baixo, o transistor leva um certo tempo (t_r) para sair da saturação e alcançar o estado de corte. Esse comportamento deriva-se do acúmulo de carga (C_{cb}) entre o coletor e a base. Dependendo das características físicas do transistor e da aplicação, esse tempo de subida pode vir a prejudicar o desempenho da finalidade do circuito, como é o caso do conversor de nível lógico elaborado para a Mesa de Bolinhas (discutido no Capítulo 3).

Para mitigar tal efeito, Horowitz e Hill (1989) propõem a inclusão de um diodo de *clamping* (também conhecido por "Baker *clamp*"), entre a base e o coletor do TJB (Figura 12). Esse diodo irá desviar a corrente de base quando o transistor estiver se aproximando da saturação e evitará sua saturação, uma vez que a queda direta de tensão do diodo, no caso um diodo Schottky, é menor que a da junção coletor-base. Dessa forma, o TJB pode entrar em corte mais rapidamente, pois o mesmo não chega a entrar no ponto de saturação profunda.

Figura 12 – Chave inversora com o diodo de "Baker *clamping*"



Fonte: (HOROWITZ; HILL, 1989)

2.6 MICROCONTROLADOR

O ESP8266 é o microcontrolador de entrada para a família de 32 bits da fabricante chinesa Espressif Systems. Produzido em escala a partir de 2014, essa família ganhou espaço na área de Internet das Coisas por ser um microcontrolador de baixo custo, de baixo consumo e com suporte à rede 802.11 (*Wi-Fi*). A Tabela 2 apresenta as principais especificações do microcontrolador e a Figura 13 apresenta seu diagrama funcional. Cabe aqui ressaltar que toda a parte de radio-frequência é implementada por *hardware*, simplificando o desenvolvimento de aplicações com comunicação sem fio.

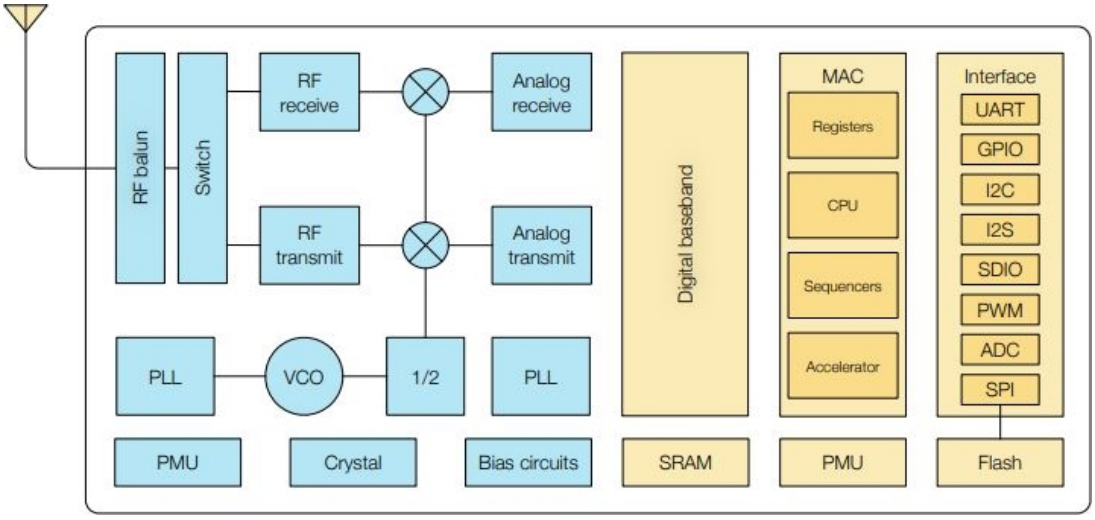
Tabela 2 – Especificação do ESP8266.

Alimentação	3,3 V
Consumo	$10\ \mu A - 170\ mA$
Memória Flash (externa)	16 MB máx. (512 kB normal)
CPU	Tensilica L106 32 bits
Clock	$80 - 160\ MHz$
RAM	$32\ kB - 80\ kB$
GPIOs	17 (compartilhadas com outras funções)
ADC	1 canal (10 *bits* de resolução)
Wireless	Estação, ponto de acesso ou ambos

Fonte: (KOLBAN, 2016)

Embora a comunicação sem fio não fizesse parte das especificações do projeto arquitetônico original, a equipe concordou com a possibilidade de implementá-la em uma aplicação futura. Esse foi o ponto decisivo para a escolha do ESP8266. Assim, a Mesa de Bolinhas poderá ganhar novas funcionalidades sem haver a necessidade de alteração no *hardware*. Isso será discutido na Seção 5.1.

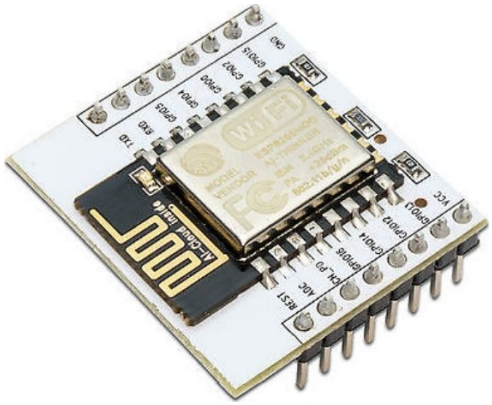
Figura 13 – Diagrama funcional do ESP8266



Fonte: Espressif Systems (2018)

Entre os diversos módulos que incluem o ESP8266, optou-se pelo ESP-12, por ser o que dispõe a maior quantidade de GPIOs (9 ao todo). A Figura 14 apresenta sua placa. Convém salientar que este módulo já conta com memória *Flash* (4 MB) e antena *microstrip*.

Figura 14 – Módulo ESP-12



Fonte: (KOLBAN, 2016)

Por serem compartilhados entre aplicações do usuário e funcionalidades internas, alguns pinos possuem certas particularidades. Os resistores *pull-up/pull-down* (Figura 14) garantem o nível lógico de alguns pinos, porém outros necessitam de uma atenção especial, sobretudo os que definem o modo de funcionamento: operação normal ou modo de gravação, que será discutido no Capítulo 3. A Tabela 3 apresenta a pinagem do módulo ESP-12 e também a função interna cada pino.

Tabela 3 – Pinagem do módulo ESP-12.

Pino	**Descrição**
V_{CC}	Alimentação 3,3 V
GPIO 13	Também usada pela SPI
GPIO 12	Também usada pela SPI
GPIO 14	Também usada pela SPI
GPIO 16	Deve ser conectada ao RESET no modo *Deep Sleep*
CH_{PD}	*Chip enable*: (0) desabilitado; (1) habilitado
ADC	Entrada do ADC
RESET	*Reset* externo: (0) - *reset*; (1) - normal
TXD	Transmissão da UART
RXD	Recepção da UART
GPIO 4	GPIO regular
GPIO 5	GPIO regular
GPIO 0	Modo *flah* se em nível baixo durante a inicialização
GPIO 2	Deve estar em nível alto durante a inicialização
GPIO 15	Deve estar em nível baixo durante a inicialização e gravação
GND	Terra

Fonte: (KOLBAN, 2016)

2.7 *FRAMEWORK*

Sming é um *framework* de código aberto, nativo para a família de microcontroladores ESP8266, desenvolvido na linguagem C++ e com foco em alta eficiência em desempenho e em uso de memória.

Diferente de outros *frameworks* baseados em laço-infinito, a estrutura do Sming é baseada em eventos temporais: as tarefas do usuário são escaladas em uma tabela de tempo. Isso o aproxima de uma das funcionalidades de um Sitema Operacional de Tempo Real, embora não haja, obviamente, um gerenciamento de recursos. Porém, o fato de poder-se escalar as tarefas em uma janela de tempo simplifica a implementação da especificação deste projeto, uma vez que, basicamente, sua funcionalidade também é baseada em eventos temporais, como apresentado no começo do capítulo (Seção 2.1).

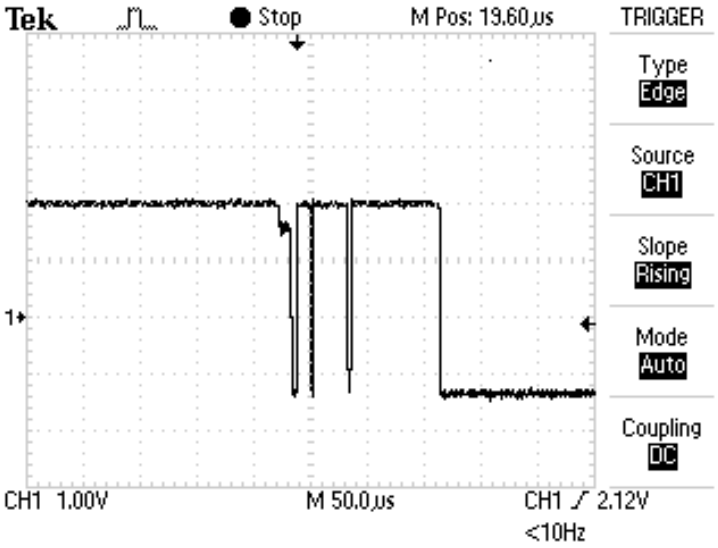
Por tratar-se de uma plataforma de código aberto, este *framework* conta com uma vasta contribuição da comunidade, com uma API de *hardware* robusta e diversas bibliotecas nativas, tais como *bootloader*, sistema de arquivos (SPIFFS), atualização sem fio de *firmware* (OTA) e uma extensa pilha assíncrona de rede (TCP, UDP, *WebSockets*, etc). Esses recursos são úteis para a continuidade do projeto, discutida na Seção 5.1.

2.8 DEBOUNCE

Embora o funcionamento do sensor (Seção 2.2) seja aparentemente simples - isto é, o transistor vai à saturação ao detectar um objeto; ou o transistor permanece em corte se não

houver detecção - podem haver disparos falsos devido à oscilação do sinal durante a interação com a mesa. Essa oscilação é conhecida como *bouncing*: trepidação do nível lógico durante a mudança de estado. A Figura 15 apresenta um exemplo dessa oscilação, quando uma chave física passou do nível lógico alto para o baixo, ao ser pressionada.

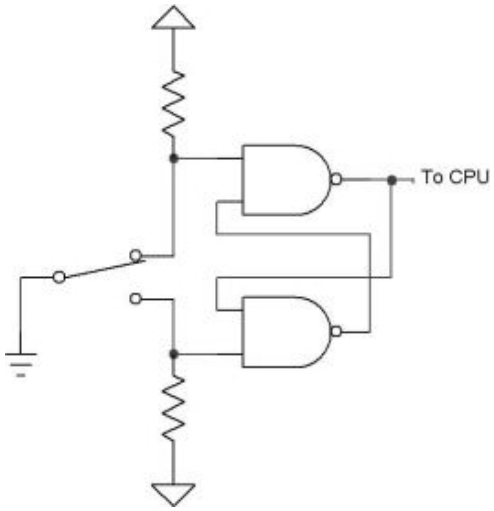
Figura 15 – Oscilação do sinal de uma chave ao ser pressionada



Fonte: Greensted (2010)

Ainda que a Mesa de Bolinhas não conte com botões físicos, o efeito *bouncing* pode vir a ocorrer em razão do limiar de detecção do sensor: o transistor de saída pode ficar operando, ainda que brevemente, no limite da interpretação do seu nível lógico. Para contornar tal situação, assim como nos casos de chaves físicas, será implementado um procedimento de *debouncing*. Ganssle (2007) cita diversas técnicas, desde descarte de tempo, filtros RC (resistor e capacitor), técnicas com *latches* (Figura 16), entre outros, inclusive táticas via *software*.

Figura 16 – Exemplo de circuito de *debounce* com *latches*



Fonte: Ganssle (2007)

No caso da Mesa de Bolinhas, não conviria implementar os modelos de *debouncing* por *hardware*, pois implicaria diretamente no custo do projeto, não só pelo custo dos componentes, mas também pela área de placa ocupada, e também na complexidade do leiaute da PCI. Então, optou-se por implementar uma técnica de contorno via *firmware*. Com isso, não há necessidade de adicionar componentes auxiliares ao circuito dos sensores e registradores de deslocamento. Trata-se de uma técnica baseada em "histerese temporal", onde a referência virtual altera-se de estado somente quando o nível de seu respectivo sensor já estiver estabilizado durante N milissegundos. A implementação dessa técnica será desenvolvida no Capítulo 3.

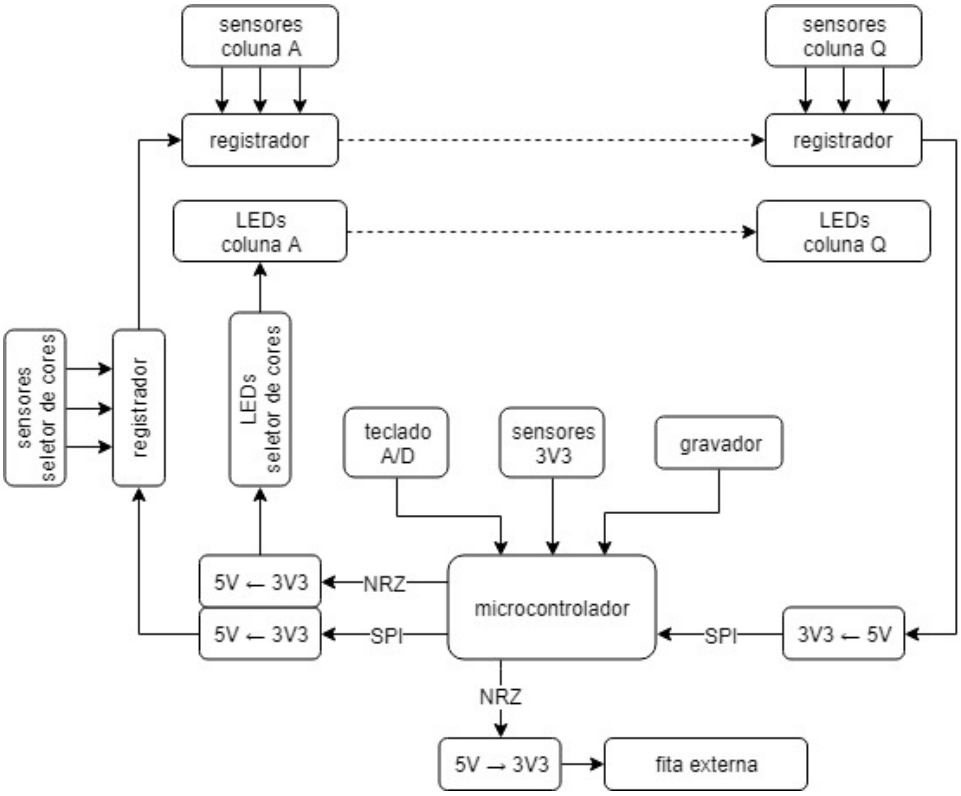
3 DESENVOLVIMENTO

Este capítulo aborda o desenvolvimento do *hardware* do projeto, desde a concepção do projeto eletrônico, levantamento do circuito elétrico, até as técnicas de roteamento das placas de circuito impresso.

3.1 CONCEPÇÃO DO PROJETO

Com base no projeto arquitetônico e a partir do levantamento das tecnologias disponíveis e viáveis (sobretudo pelo custo e pela difusão no mercado), foram definidos os componentes e técnicas a serem utilizadas no projeto eletrônico, cujo diagrama geral é apresentado na Figura 17.

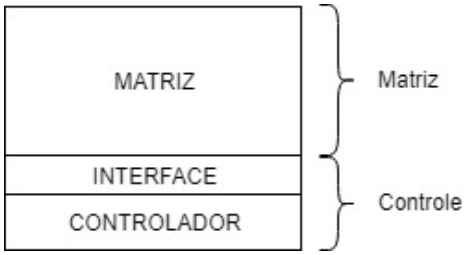
Figura 17 – Diagrama geral do projeto eletrônico



Fonte: (O AUTOR, 2018)

A partir do projeto arquitetônico (Figura 2), que é segmentado em duas partes: Matriz (interação) e Controle (seletor de cores), o projeto eletrônico também foi organizado em 3 subdivisões: Matriz, Interface e Controlador, apresentadas na Figura 18. Dessa forma, pôde-se dar andamento ao projeto por meio de etapas, onde cada fase pode ser considerada um projeto íntegro por si só.

Figura 18 – As 3 subseções principais do projeto eletrônico



Fonte: (O AUTOR, 2018)

As subdivisões têm funções bem definidas, apresentadas logo abaixo, e serão discutidas neste mesmo capítulo nos tópicos a seguir.

- **Matriz:** é o setor que pode ser "pintado com luz", que contém os LEDs e os sensores de interação. É inteiramente comandada pelo Controlador, através da Interface;
- **Interface:** interliga as subseções Matriz e Controlador: captura e propaga o sinal dos sensores, através dos registradores de deslocamento, e forma a ligação sequencial do barramento dos LEDs;
- **Controlador:** processa todos os LEDs e sensores da mesa. Contém também os LEDs e os sensores do seletor de cores, a parte de processamento, os conversores de nível lógico e os periféricos de gravação e depuração.

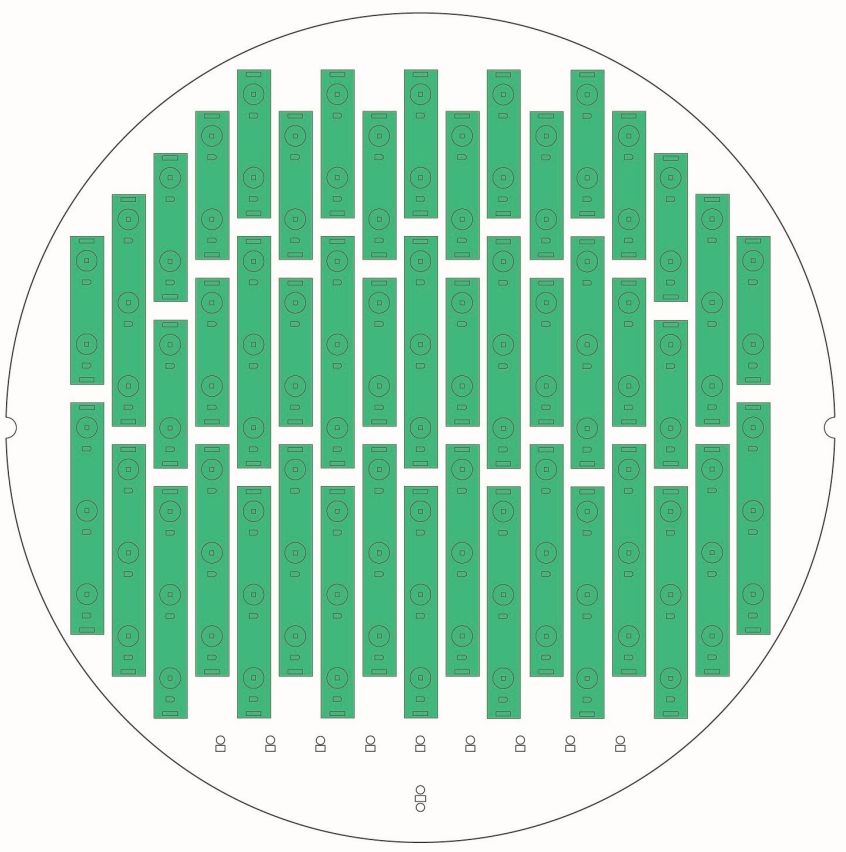
3.2 PROJETO DA MATRIZ

O projeto da Matriz foi concebido durante o Programa de Iniciação Científica do qual o autor participou. Nele, o desafio foi o de contemplar todas as 118 bolinhas de *ping-pong* desta seção (e seus respectivos pares sensor-LED), visando robustez e o menor custo.

Para tal, foi definido que a matriz seria arranjada em 17 colunas verticais, conforme apresentado na Figura 19. Consequentemente, devido à disposição dos objetos (determinada pelo projeto arquitetônico), essas colunas poderiam conter 5, 6, 7 ou 8 bolinhas e, para abrangê-las da maneira mais proveitosa, as colunas foram combinadas por módulos (placas) de 2 ou 3 bolinhas.

Em geral, as empresas fabricantes de placas de circuito impresso consideram cada modelo de placa como um projeto, devido ao corte do painel, procedimento para os testes elétricos, etc. Assim, se utilizando de apenas 2 modelos de placa, foi possível reduzir o custo do projeto sem dispensar a robustez que as PCIs proporcionam.

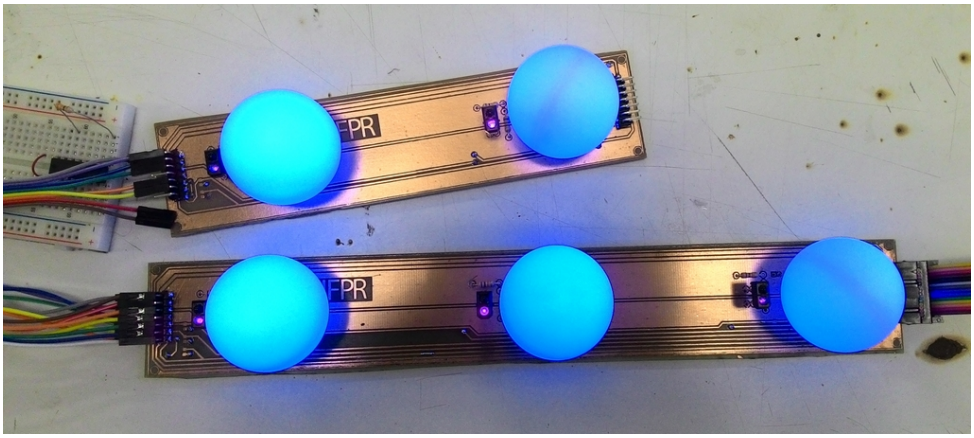
Figura 19 – Disposição das placas da Matriz



Fonte: adaptado de (MITSUKO, 2016)

Uma vez definido que a matriz seria composta por placas com 2 ou 3 bolinhas, iniciaram-se os testes de conceito com o protótipo artesanal, apresentado na Figura 20, produzido no Laboratório de Corrosão de Placas do Departamento de Engenharia Elétrica da UFPR. Nesses ensaios, foram testados os conceitos de endereçamento do LED WS2812, a leitura do nível lógico dos sensores através dos registradores de deslocamento e também o modo de conexão entre as placas.

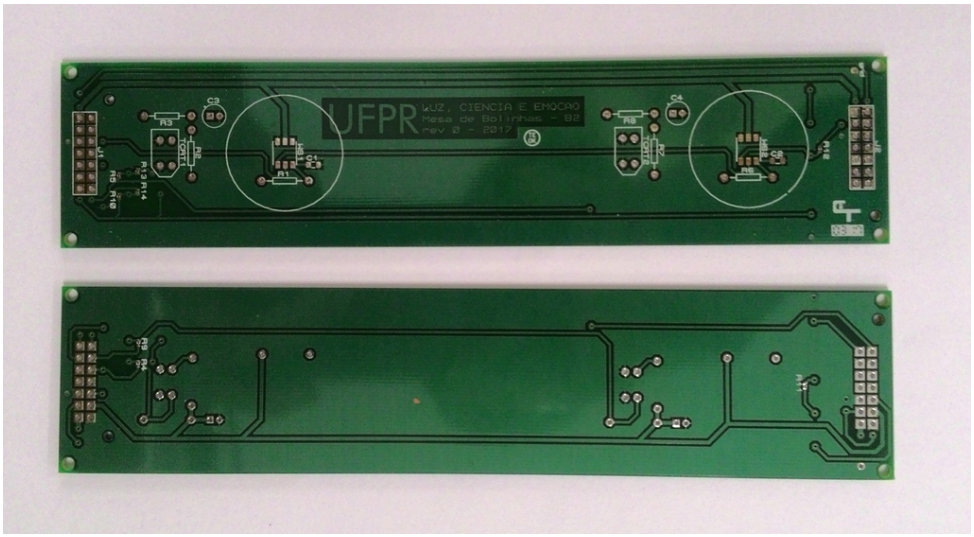
Figura 20 – Protótipo das placas da matriz



Fonte: (O AUTOR, 2017)

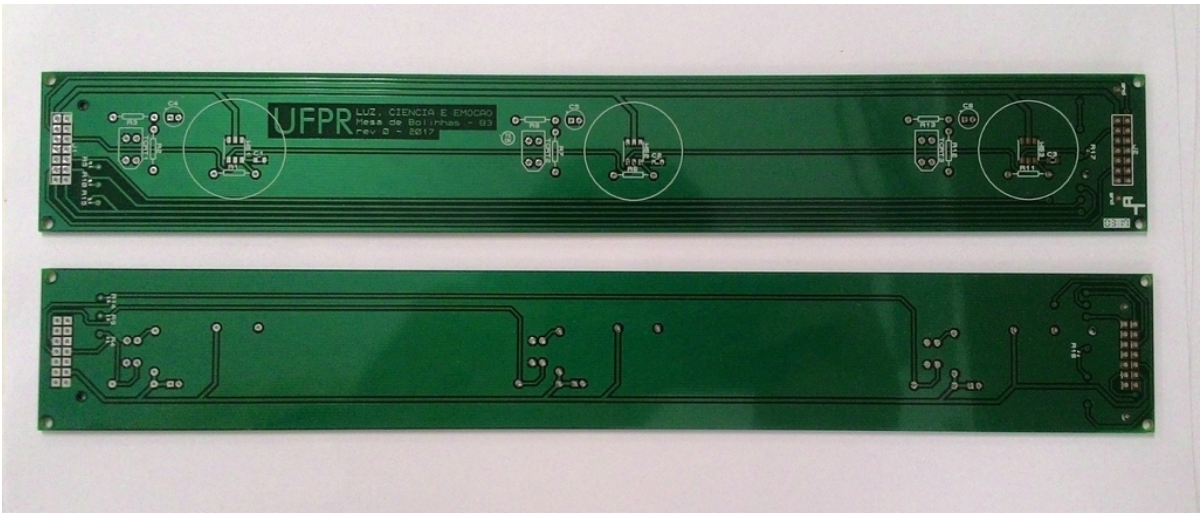
Todos os conceitos propostos foram validados e as 47 placas que compõem a matriz foram confeccionadas em produção industrial. A Figura 21 e a Figura 22 apresentam as faces superior e inferior dos módulos.

Figura 21 – Faces da PCI do módulo de 2 bolinhas



Fonte: (O AUTOR, 2017)

Figura 22 – Faces da PCI do módulo de 3 bolinhas



Fonte: (O AUTOR, 2017)

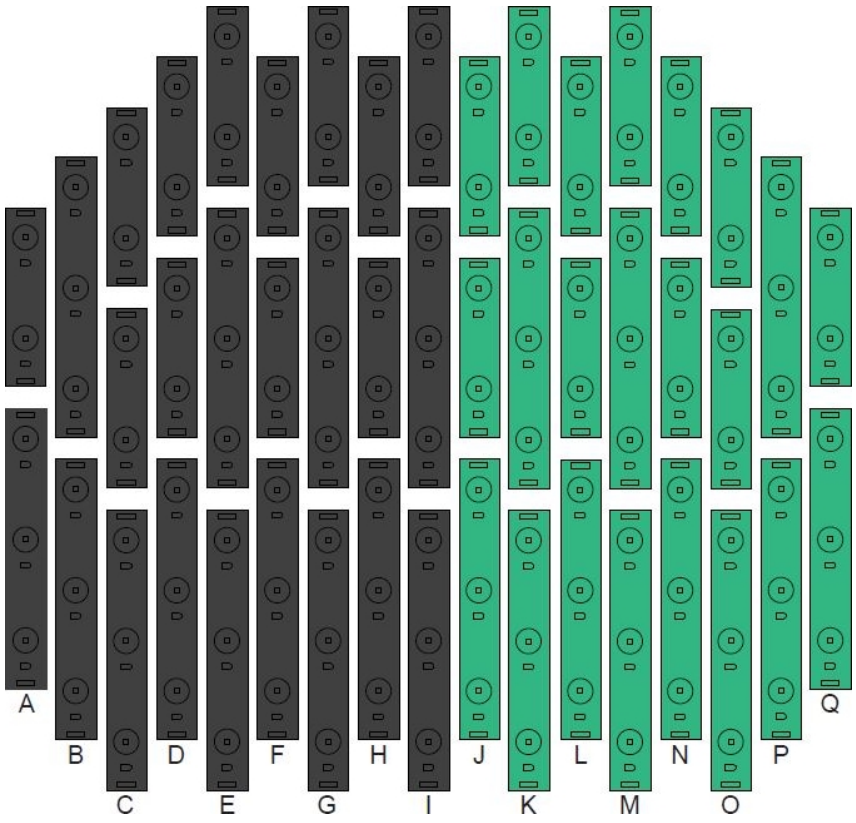
3.3 PROJETO DA INTERFACE

Como já brevemente introduzido pela Seção 3.1 (Concepção do Projeto), a Interface tem a função de interligar as seções Matriz e Controlador. Contudo, deve-se atentar a algumas questões:

I. Tanto o barramento dos sensores, quanto o barramento dos LEDs devem seguir a sequência da esquerda para a direita; de baixo para cima;
II. Por integrar barramentos de frequências razoáveis - NRZ @800 kHz e SPI @4 MHz dos LEDs e sensores, respectivamente - deve-se evitar construir trilhas com quinas abruptas, em função da reflexão do sinal;

A questão (I) foi trabalhada visando a redução de custo, decidiu-se que Interface também seria um projeto modular, isto é, esta seção seria dividida em placas do mesmo modelo. Isso é em razão do valor das placas de circuito impresso ser calculado em função de sua área e que cada projeto possui um pedido mínimo. Em outras palavras, quanto maior a área de uma PCI, maior será o seu custo, e quanto mais modelos de placa, maior será a quantidade a ser adquirida, pois cada modelo possui um pedido mínimo para produção. Sendo assim, a Interface foi projetada para abranger todas as colunas da Matriz em duas "metades", como é apresentado na Figura 23. No caso, uma placa teria a função de operar as primeiras 9 colunas (de A a I) e a outra, as 8 colunas restantes (de J a Q).

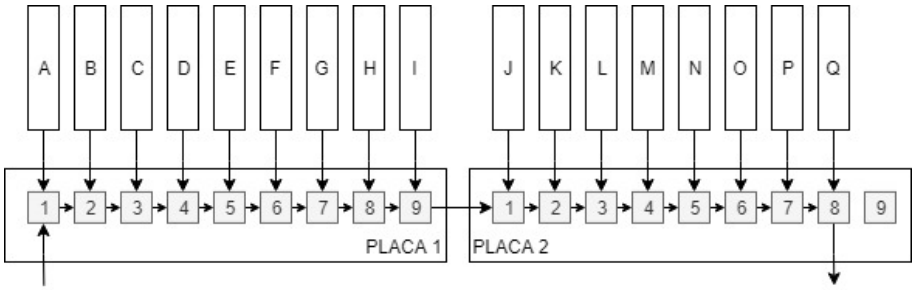
Figura 23 – Divisão das colunas para conexão com a Interface



Fonte: adaptado de (MITSUKO, 2016)

A Figura 24 apresenta o diagrama da formação do barramento dos sensores. Nele, cada coluna da Matriz é conectada a um registrador de deslocamento, em forma sequencial (A:1, ..., I:9; J:1, ..., Q:8). Nota-se também que o último registrador da segunda placa não possui conexão, dado que todas as colunas já foram abrangidas pelos registradores precedentes.

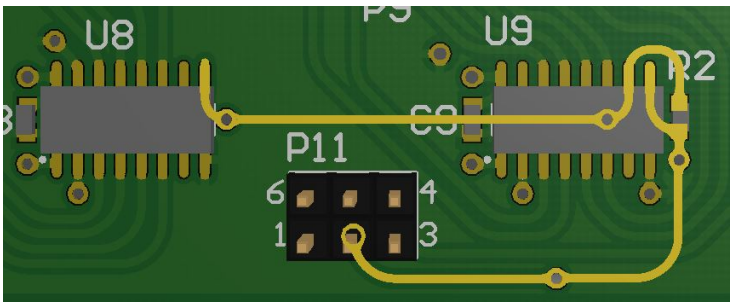
Figura 24 – Diagrama do barramento da Interface



Fonte: (O AUTOR, 2018)

Esse direcionamento do barramento é dado por um resistor de $0\ \Omega$, como apresentado na Figura 25 (componente "R2"). Se montado, o sinal é diretamente conduzido ao conector de saída ("P11"), ignorando o último registrador ("U9") que, neste caso, não seria montado.

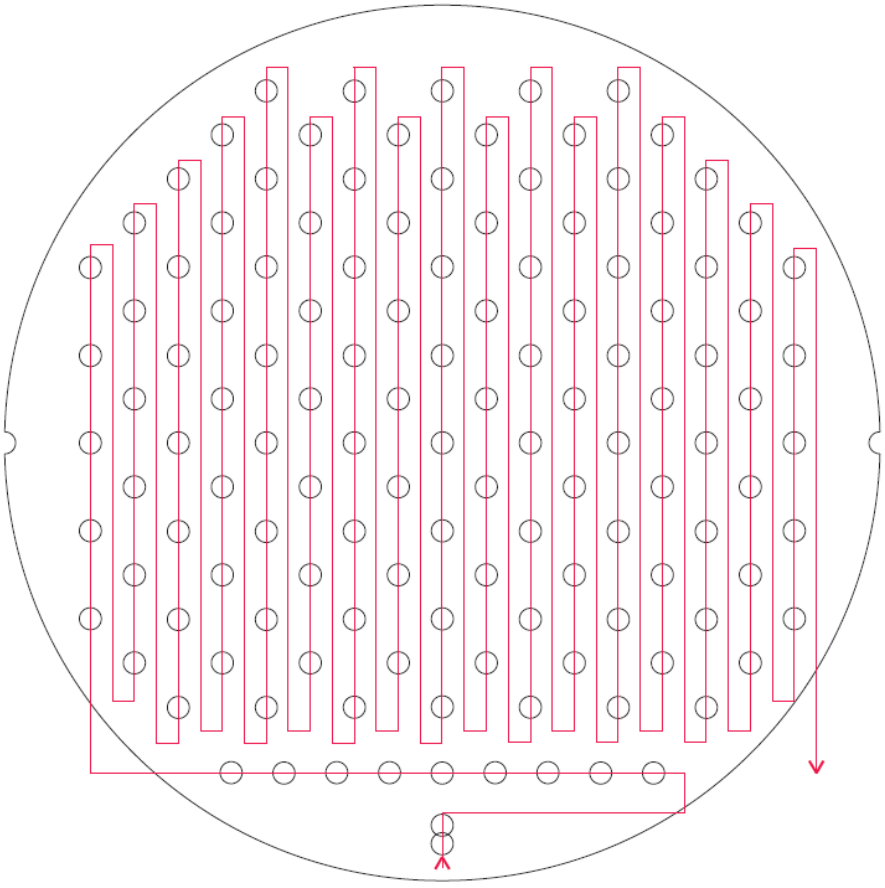
Figura 25 – Direcionamento do barramento dos sensores



Fonte: (O AUTOR, 2018)

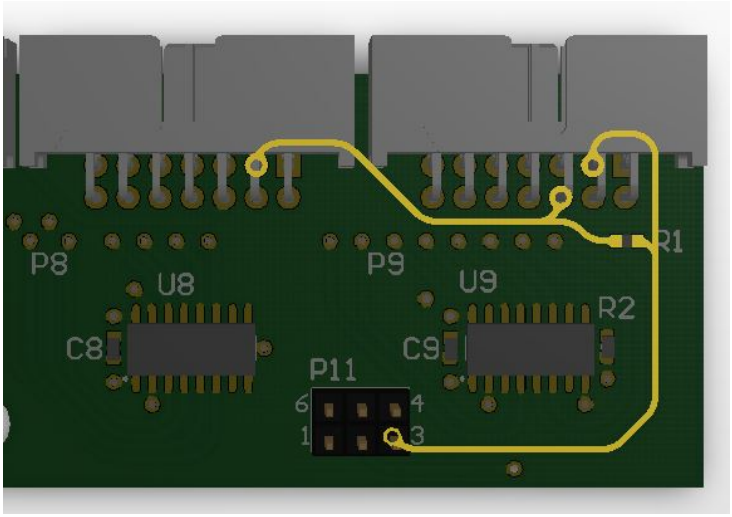
De iguais modos, o barramento dos LEDs foi projetado para seguir a mesma sequência (Figura 26) e o direcionamento entre a última coluna ("P9") ou o conector de saída ("P11") também é dado através de um resistor de $0\ \Omega$ ("R1"), como apresentado na Figura 27.

Figura 26 – Sentido do barramento dos LEDs



Fonte: adaptado de (MITSUKO, 2016)

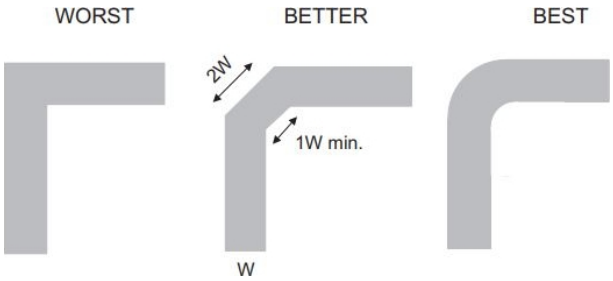
Figura 27 – Direcionamento do barramento dos LEDs



Fonte: (O AUTOR, 2018)

Já em relação à questão (II), sobre a possibilidade de reflexão de sinais devido à perturbação da capacitância e auto-indutância da trilha, seguiu-se a recomendação de leiaute de (Texas Instruments Incorporated, 2015), apresentada na Figura 28.

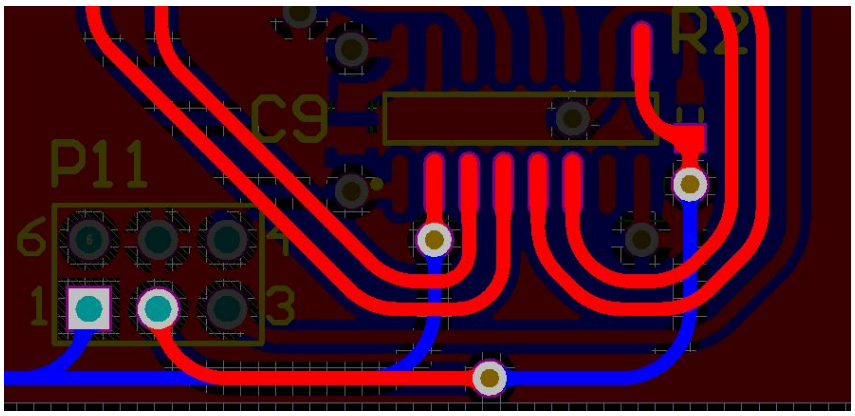
Figura 28 – Recomendação de leiaute para as trilhas de sinal



Fonte: Texas Instruments Incorporated (2015)

Tal técnica foi aplicada não só nas trilhas de sinais, mas também em todas as ocasiões possíveis, como apresentado na Figura 29.

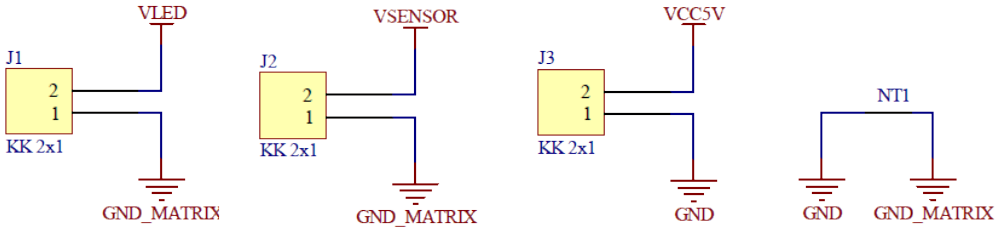
Figura 29 – Trilhas arredondadas



Fonte: (O AUTOR, 2018)

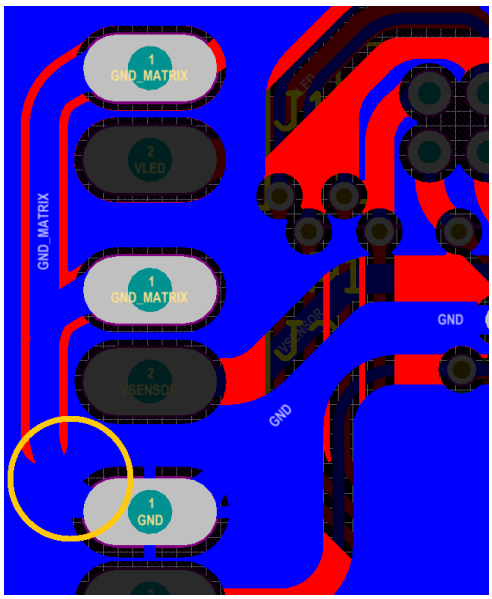
Além disso, para evitar a propagação do ruído da parte de potência, a alimentação do circuito lógico ("VCC5V") foi separada da alimentação dos LEDs ("VLED") e da dos sensores ("VSENSOR"), conforme apresentado na Figura 30. Ademais, a conexão física entre as massas (terra) é feita somente em um único ponto, assim, o retorno da alimentação de potência ("GND_MATRIX") permanece confinado, não circulando sobre o plano de terra ("GND"). Essa técnica é conhecida por *"net-tie"* e está destacada na Figura 31.

Figura 30 – Separação das alimentações da Interface



Fonte: (O AUTOR, 2018)

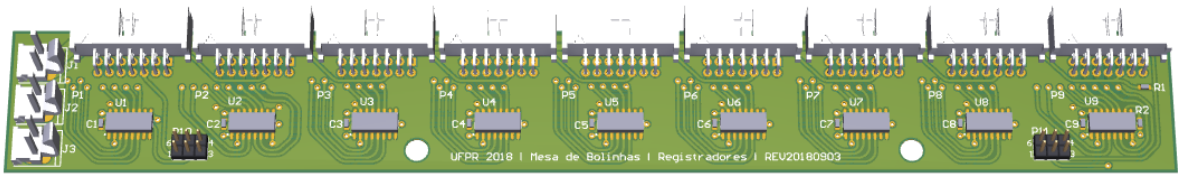
Figura 31 – Separação física das massas



Fonte: (O AUTOR, 2018)

Por fim, a Figura 32 apresenta o modelo 3D da placa de interface. Trata-se de uma PCI face-dupla, 245 mm x 32,76 mm. Os conectores de alimentação (à esquerda da placa) são da série KK Molex®, que impedem a inversão de polaridade na conexão com fonte de alimentação externa, e os conectores das colunas (parte superior da placa) são do padrão *Latch-header*, recomendado para aplicações com limitação de espaço. Cada placa suporta até 9 colunas de até 8 sensores, e as PCIs podem ser ligadas em série para compartilharem o mesmo barramento.

Figura 32 – Modelo 3D da Interface

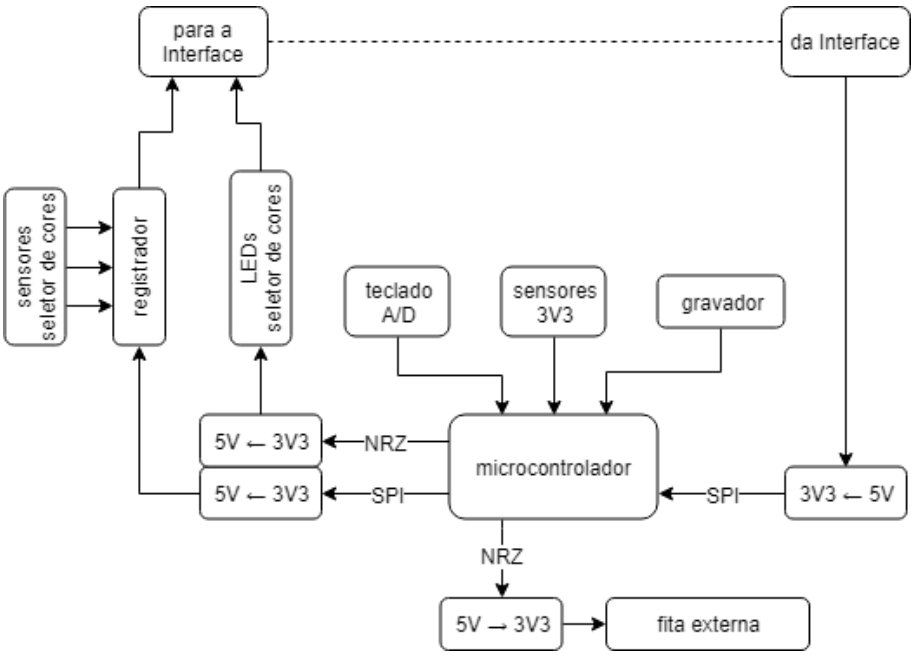


Fonte: (O AUTOR, 2018)

3.4 PROJETO DO CONTROLADOR

O Controlador é responsável por comandar os eventos de interação, através da Interface, da seção Matriz. Ademais, engloba também o seletor de cores, circuito de gravação, teclado para depuração e suporte para um barramento externo. A Figura 33 apresenta seu diagrama simplificado.

Figura 33 – Diagrama da seção Controlador



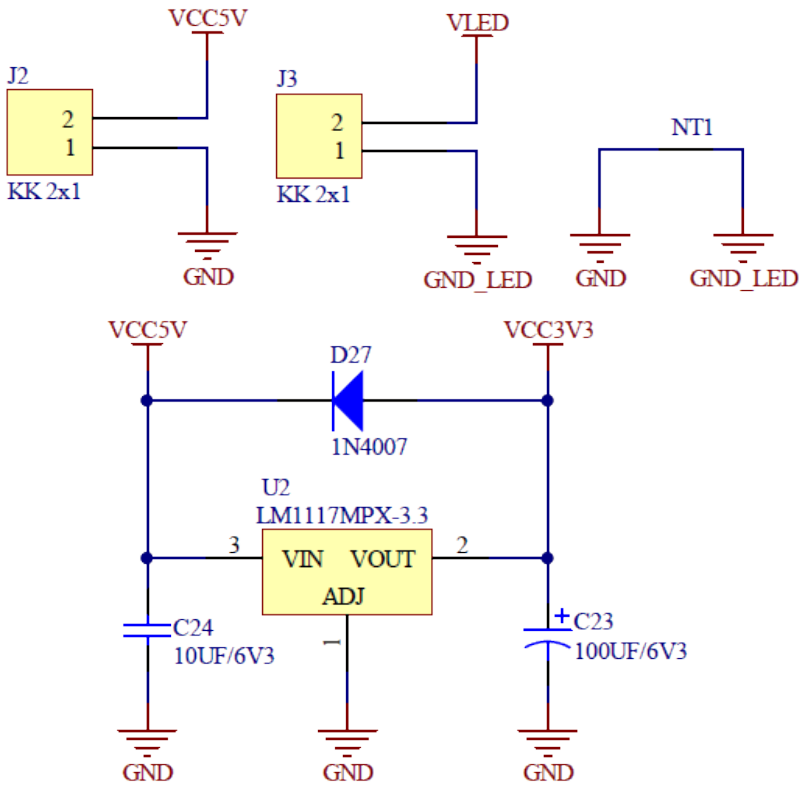
Fonte: (O AUTOR, 2018)

3.4.1 ALIMENTAÇÃO E CONVERSÃO DE NÍVEL LÓGICO

Da mesma forma que na Interface, a alimentação do Controlador também foi dividida, separando os circuitos de potência ("VLED") e lógico ("VCC5V"), inclusive com *net-tie* isolando fisicamente as massas. Porém, diferente da Matriz, onde a quantidade de LEDs e sensores é muito maior que a do seletor de cores do Controlador, esses circuitos compartilham a mesma alimentação de potência neste projeto. A Figura 34 apresenta o circuito de entrada de alimentação.

Como o microcontrolador operar em 3,3 V, houve a necessidade de incluir um regulador para a faixa especificada. Por se tratar de um circuito de carga simples, isto é, de baixo consumo, optou-se por um regulador linear de saída fixa (componente "U2"). Para protegê-lo da descarga das capacitâncias conectadas à sua saída (por exemplo, o capacitor "C23"), foi incluído um diodo de *bypass* ("D27") para desviar a corrente reversa sobre regulador, no caso da fonte externa ser desconectada e a tensão de saída resultar em um potencial mais alto que o de entrada. Esse diodo também possui a função de proteger o regulador caso o desenvolvedor queira alimentar a placa diretamente através do circuito de gravação, a ser discutido mais adiante.

Figura 34 – Entrada de alimentação do Controlador

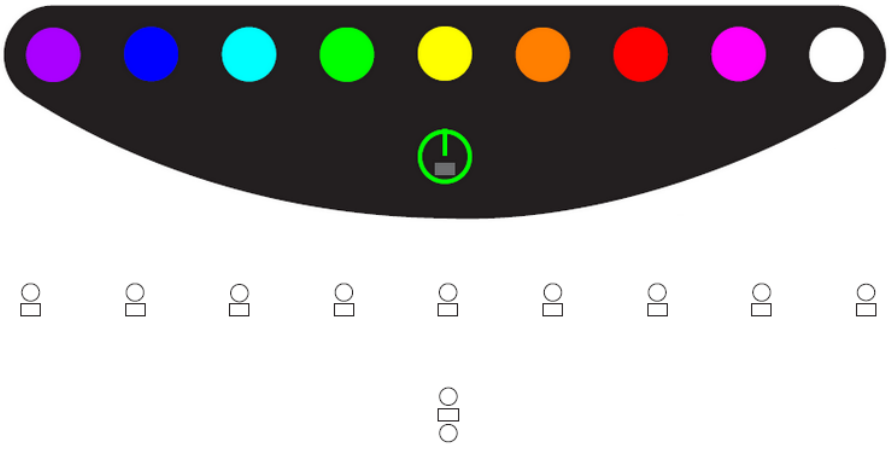


Fonte: (O AUTOR, 2018)

3.4.2 SELETOR DE CORES

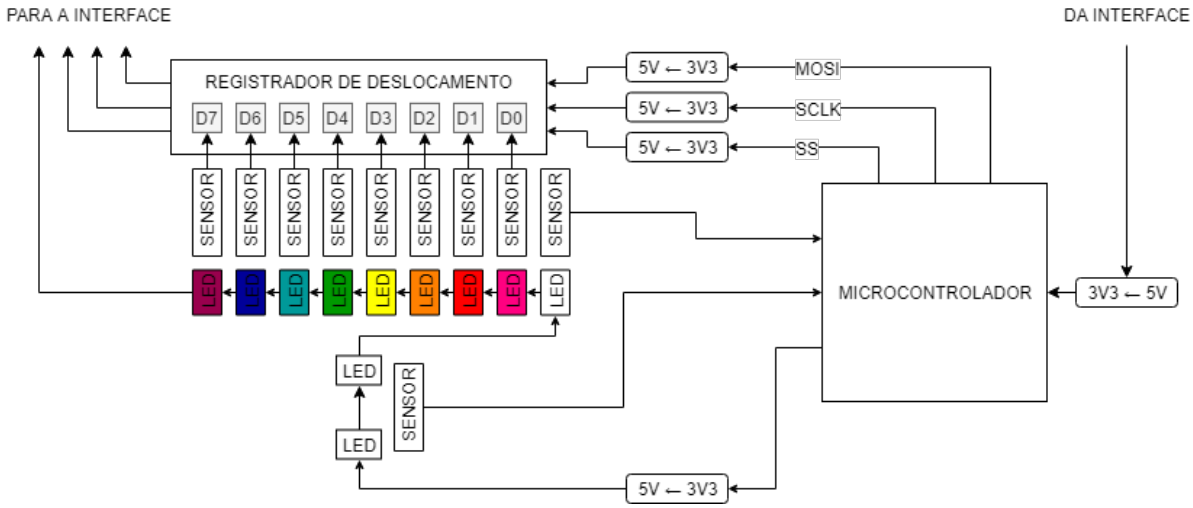
O seletor de cores forma a primeira etapa dos barramentos, seja dos LEDs endereçáveis ou dos registradores de deslocamento. É composto por 10 sensores reflexivos e 11 LEDs endereçáveis, como apresentado na Figura 35, e a Figura 36 apresenta seu diagrama.

Figura 35 – Vista do seletor de cores



Fonte: adaptado de (MITSUKO, 2016)

Figura 36 – Vista do seletor de cores



Fonte: (O AUTOR, 2018)

Assim como feito com as colunas da Matriz, por meio das placas da Interface, o registrador de deslocamento é responsável por coletar os níveis lógicos de 8 dos 10 sensores do seletor de cores, restando 2 sensores fora do barramento. Assim, optou-se por conectá-los diretamente aos pinos do microcontrolador, visto que haviam GPIOs livres.

Devido aos LEDs e registradores de deslocamento serem alimentados em 5 V, enquanto que o microcontrolador é alimentado em 3,3 V, foi necessário adequar os sinais dos barramentos através de conversores de nível lógico. Essa conversão se deu de duas formas, nos seguintes sinais:

- De 3,3 V para 5 V:
  - Saída de dados do barramento NRZ (LEDs WS2812)
  - Saída de dados do barramento SPI (MOSI)
  - Saída de clock do barramento SPI (SCLK)
  - Sinal de seleção de periférico do barramento SPI (SS)
- De 5 V para 3,3 V:
  - Entrada de dados do barramento SPI (MISO)

Os conversores de nível de 3,3 V para 5 V foram projetados na topologia não-inversora, com transistor NPN operando nas regiões de corte e saturação: mantêm-se base e coletor polarizados - base em 3,3 V e coletor em 5 V - enquanto que a polarização da junção base-emissor (V_{BE}) é dada pela aplicação do sinal no emissor.

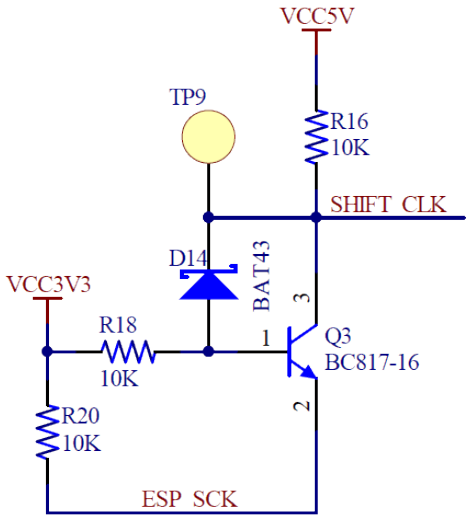
A Figura 37 apresenta o esquemático do conversor para o sinal de *clock* do barramento SPI. Nota-se que, quando o sinal "ESP_CLOCK" (saída SCLK do microcontrolador ESP8266) vai a 1-lógico (3,3 V), não há diferença de potencial entre a base e o emissor ($V_{BE} = 0\ V$), portanto, o transistor está em corte, logo a saída "SHIFT_CLK" (entrada de *clock* do *shift-register*) vai a 5 V (1-lógico).

No caso da aplicação de 0 V (0-lógico) por "ESP_CLOCK", a junção base-emissor

estará diretamente polarizada, o que, neste caso, levará o transistor à saturação. Com isso, a tensão em "SHIFT_CLK" será a diferença de potencial entre coletor e emissor (V_{CE}) somada à queda de tensão sobre o *mosfet* interno da GPIO do microcontrolador, resultando em um potencial relativamente próximo a 0 V (0-lógico).

Cabe aqui ressaltar inclusão do diodo de *Baker clamping* para minimizar os efeitos das capacitâncias parasitas do transistor.

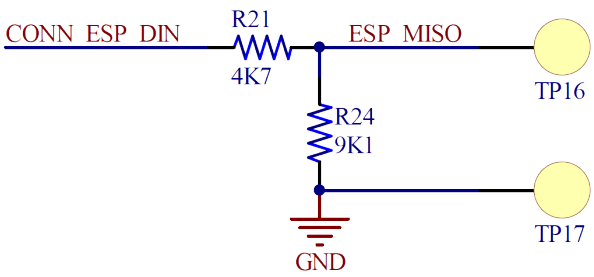
Figura 37 – Conversor de nível lógico da linha de *clock* do barramento SPI



Fonte: (O AUTOR, 2018)

Já para a situação de transformação de 5 V para 3,3 V, como é o caso do retorno do sinal dos registradores de deslocamento das placas de interface (sinal MISO), a implementação do conversor de nível lógico foi dada por um simples divisor resistivo, como apresentado na Figura 38.

Figura 38 – Conversor de nível lógico da linha MISO do barramento SPI



Fonte: (O AUTOR, 2018)

Quando aplicado 5 V (1-lógico) em "CONN_ESP_DIN" (entrada de dados do conector, linha MISO do barramento SPI), a tensão em "ESP_MISO" (linha MISO do microcontrolador) será aproximadamente 3,3 V (1-lógico), dado pela Equação (1). De mesmo modo, quando

aplicado 0 V na entrada do conversor, não haverá queda de tensão sobre o resistor de saída ("R24"), portanto a tensão em "ESP_MISO" também será 0 V (0-lógico).

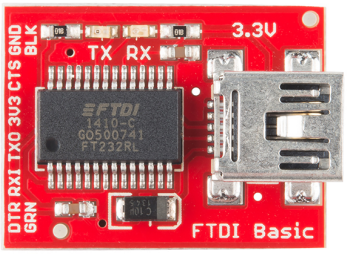
$$V = 5\ V \cdot \frac{9100\ \Omega}{9100\ \Omega + 4700\ \Omega} = 3{,}297\ V \tag{1}$$

3.4.3 GRAVADOR

O microcontrolador ESP8266 possui alguns modos de operação, entre eles, o modo normal e o de gravação via UART, que são determinados pelo estado de alguns pinos durante a inicialização do sistema (*boot*), como brevemente introduzido na Seção 2.6.

Além de fornecer uma via para a comunicação serial com um adaptador USB, o circuito apresentado a seguir (Figura 40) também tem a função de forçar o microcontrolador a entrar em modo de gravação conforme os comandos de *"reset"* e *"data terminal ready"* (DTR) do adaptador. Dessa forma, não há a necessidade de uma operação manual para determinar os estados dos pinos de programação para carregar um novo *firmware* no microcontrolador. Para tal, basta somente que o adaptador disponha de uma linha DTR, como o exemplo apresentado na Figura 39. Cabe aqui ressaltar que, assim como citado na Subseção 3.4.1, todo o circuito de 3,3 V pode ser alimentado diretamente pelo adaptador USB (desde que seja 3,3 V), sem a necessidade de uma fonte externa. Isso simplifica o desenvolvimento e a depuração do projeto, quando não houver a necessidade de testar o restante do circuito.

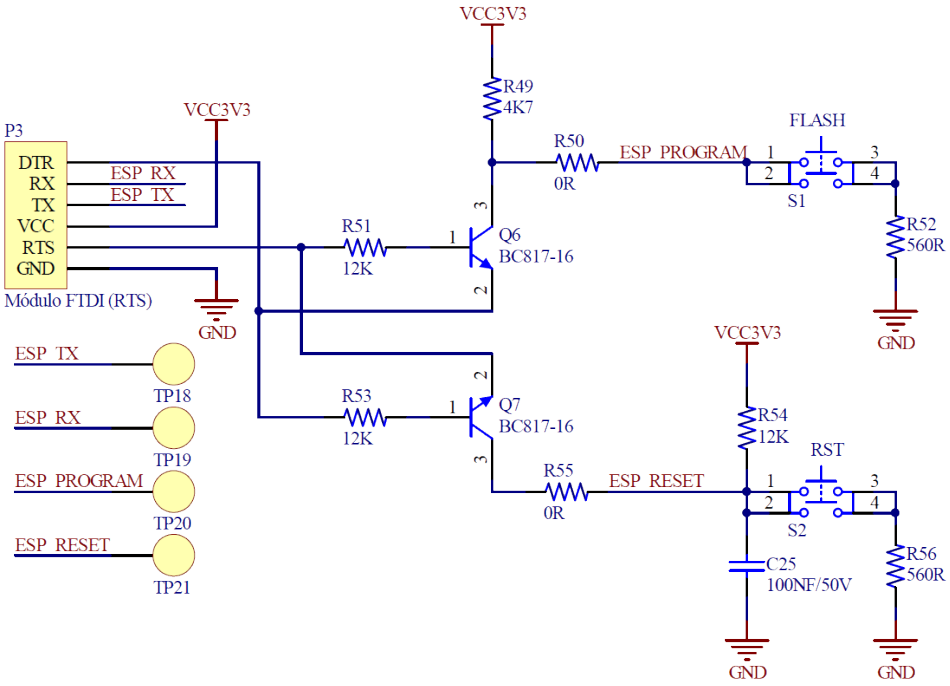
Figura 39 – Exemplo de adaptador serial-USB com linha DTR



Fonte: SparkFun (2015)

Sobretudo, para que o microcontrolador entre em modo de gravação, o pino "GPIO 0" deve estar em 0-lógico durante a inicialização. Se estiver em 1-lógico durante o *reset*, o ESP8266 inicializa em modo normal, executando o programa armazenado na memória interna.

Figura 40 – Esquemático do circuito de gravação



Fonte: (O AUTOR, 2018)

A Tabela 4 apresenta os estados dos pinos do microcontrolador em função dos níveis do adaptador USB, através do circuito da Figura 40. Contudo, se o desenvolvedor não dispor de um adaptador USB com sinal DTR, ainda há a possibilidade de pôr o microcontrolador em modo de gravação de maneira manual, através dos botões "S1" e "S2".

Tabela 4 – Tabela verdade do gravador automático.

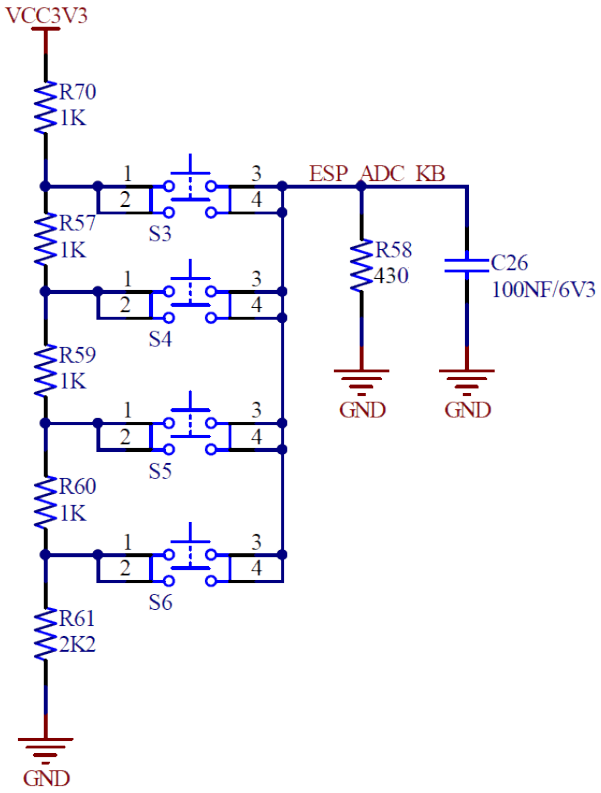
Adaptador USB		**ESP8266**	
DTR	**RST**	**GPIO 0**	**Reset**
0	0	1	1
0	1	0	1
1	0	1	0
1	1	1	1

Fonte: (O AUTOR, 2018)

3.4.4 TECLADO AD

O "Teclado AD" (ou teclado analógico) tem a função de servir como interface durante as etapas de desenvolvimento/depuração. Com apenas um único pino do microcontrolador é possível ler diversas teclas, desde que seja um pino com função Conversor Analógico-Digital. A Figura 41 apresenta o esquemático do circuito.

Figura 41 – Esquemático do circuito do teclado analógico



Fonte: (O AUTOR, 2018)

Seu funcionamento se dá por meio da combinação de uma cadeia de resistores. Através da Lei de Ohm, pelas combinações série-paralelo, é possível estimar qual botão está sendo pressionado, limitando-se a um botão por vez.

A Tabela 5 apresenta os valores de tensão e as unidades estimadas pelo conversor interno do microcontrolador, um ADC de 10 *bits* de resolução, com faixa de operação de 0 V a 1 V.

Tabela 5 – Valores de tensão e unidades do teclado analógico.

Chave	Tensão (V)	Valor ADC
S3	0,94	960
S4	0,54	551
S5	0,37	379
S6	0,27	279

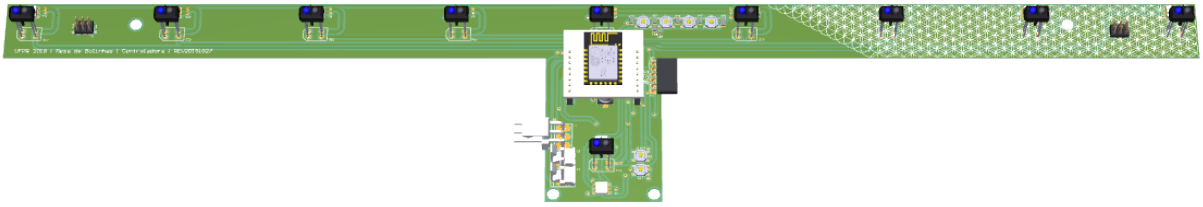
Fonte: (O AUTOR, 2018)

3.4.5 PROJETO DA PCI DO CONTROLADOR

Em sua totalidade, não há um padrão de repetição de funcionalidades como nos outros setores (Interface e Matriz), então a placa do Controlador não foi projetada de maneira modular, senão em placa única que, inclusive, resultou em uma PCI relativamente grande (500 mm x 90

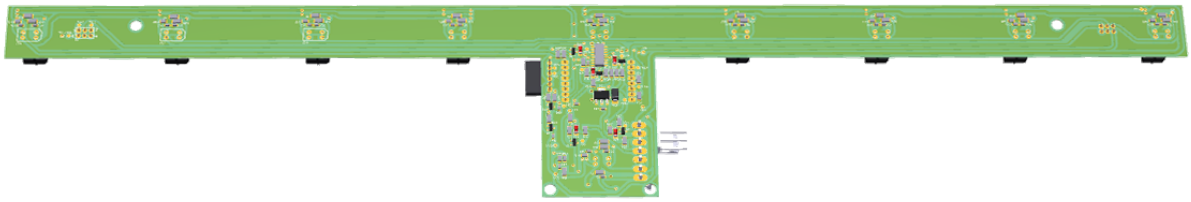
mm), a ser discutida na Capítulo 4.A Figura 42 apresenta a visão da face superior do modelo 3D da placa, e a Figura 43 apresenta a visão da face inferior.

Figura 42 – Visão superior do modelo 3D da placa do Controlador



Fonte: (O AUTOR, 2018)

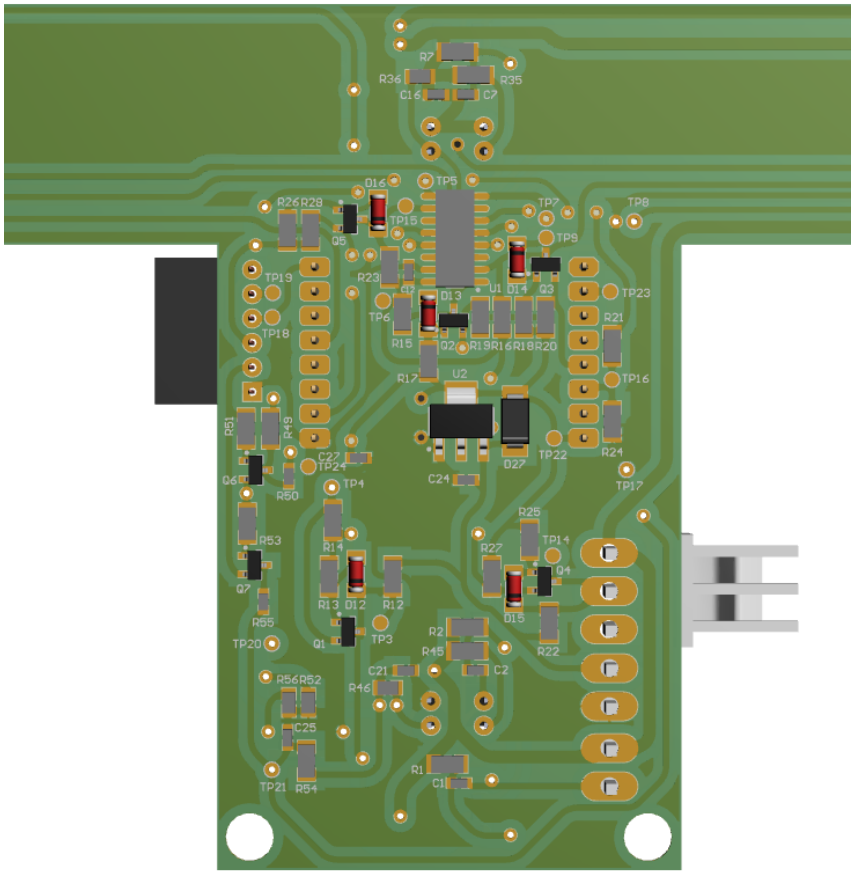
Figura 43 – Visão inferior do modelo 3D da placa do Controlador



Fonte: (O AUTOR, 2018)

Assim como nos projetos anteriores, a placa do Controlador também foi projetada utilizando-se de técnicas para evitar a reflexão dos sinais, separação das alimentações dos circuitos de potência e utilização de conectores que impedem inversão de polaridade. Além disso, com exceção dos sensores e conectores, todos os componentes são de montagem em superfície (SMD), como apresentado na Figura 44.

Figura 44 – Visão inferior do modelo 3D da placa do Controlador - detalhe para os componentes SMDs



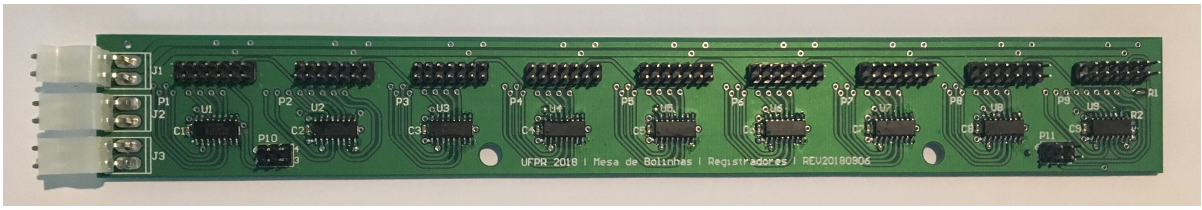
Fonte: (O AUTOR, 2018)

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Devido à complexidade dos circuitos, as placas desta etapa do projeto não foram produzidas como protótipos artesanais, como foi feito com as PCIs da Matriz durante o Programa de Iniciação Científica. As placas aqui projetadas possuem uma grande quantidade de vias (furos metalizados que interligam as faces), trilhas e isolação (separação mínima entre as trilhas) muito pequenas, o que tornaria inviável sua fabricação no Laboratório de Corrosão por falta de material adequado. Portanto, os projetos dessas PCIs foram enviados diretamente para a fabricação industrial.

A Figura 45 apresenta a placa de circuito impresso da Interface, já com seus componentes montados. Foi fabricada por uma empresa da região de Nova Trento, em Santa Catarina, a de orçamento mais acessível entre as quase 20 empresas contatadas. Trata-se de uma produção industrial, inclusive com a garantia de integridade por meio de testes elétricos.

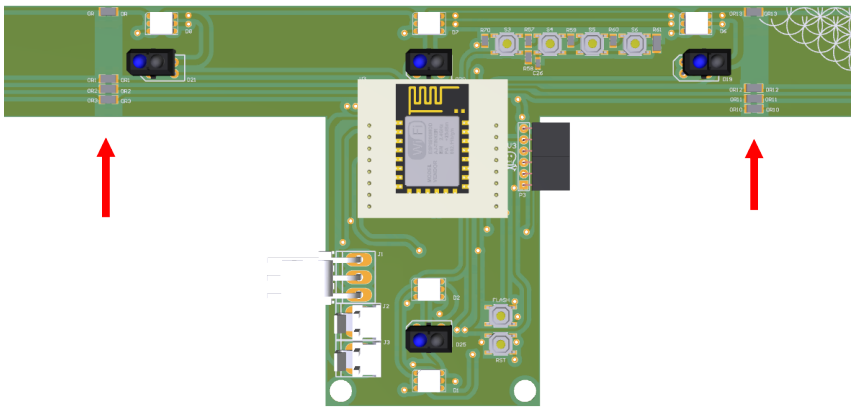
Figura 45 – Placa de circuito impresso da Interface



Fonte: (O AUTOR, 2018)

Com relação à placa de circuito impresso da seção de controle, após contatar mais de 20 empresas, somente cerca de 3 delas teriam capacidade de produzir PCIs com tamanhas dimensões (500 mm x 90 mm), sendo que os valores ultrapassavam os R$1000,00. Então, decidiu-se reprojetar a placa do Controlador, na verdade, separá-la em outras 3 PCIs, como apresentado na Figura 46. Dessa forma, as PCIs teriam as dimensões adequadas para produção na maioria das empresas. Ademais, as placas seriam conectadas por meio de resistores de 0Ω (encapsulamento SMD 1206), dispensando o uso de fios.

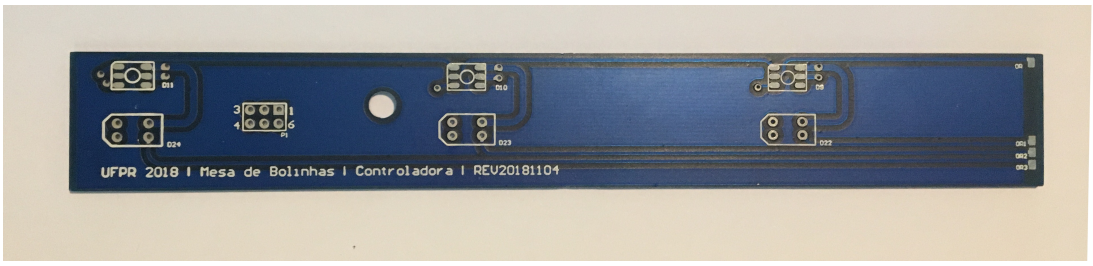
Figura 46 – Reprojeto da placa de controle, detalhe para os locais de corte e resistores de conexão



Fonte: (O AUTOR, 2018)

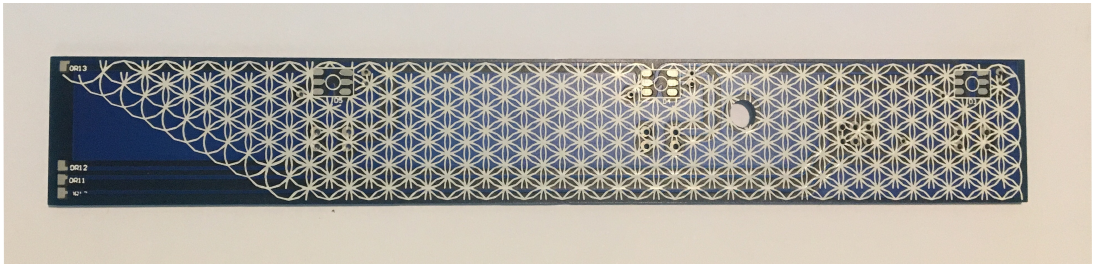
Contudo, como mencionado no Capítulo 3, cada modelo de placa é considerado um projeto íntegro pelas fabricantes de PCIs e, em virtude do pedido mínimo, o valor total ainda seria alto. Então, as placas foram enviadas para a fabricação em uma universidade da região de Curitiba, que conta com um laboratório de produção de protótipos, inclusive com confecção de vias metalizadas. As Figuras 47, 48 e 49 apresentam os protótipos fabricados.

Figura 47 – Placa Esquerda do Controlador



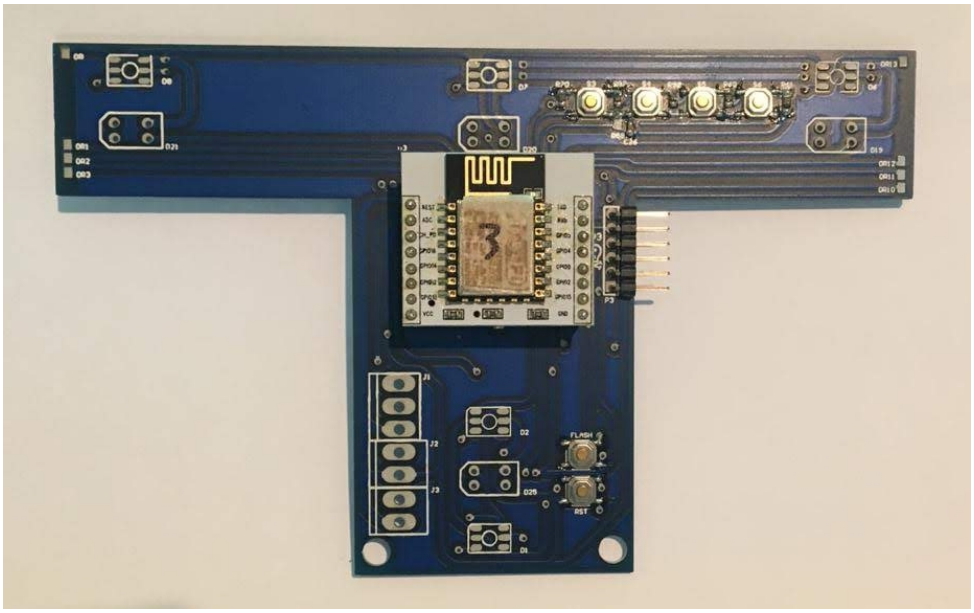
Fonte: (O AUTOR, 2018)

Figura 48 – Placa Direita do Controlador



Fonte: (O AUTOR, 2018)

Figura 49 – Placa Central do Controlador, parcialmente montada

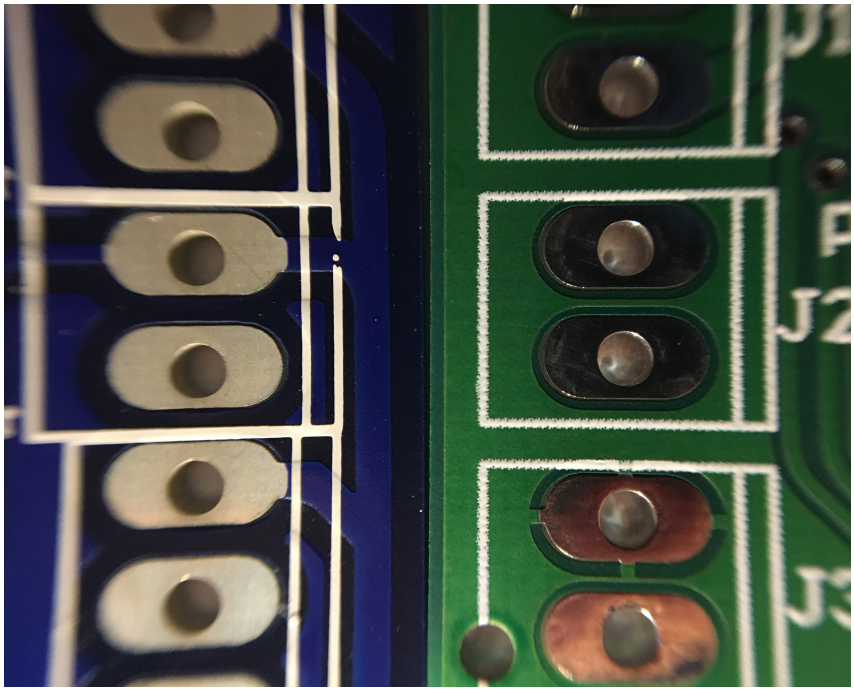


Fonte: (O AUTOR, 2018)

Visto a carência de várias técnicas de roteamento do *software* de leiaute utilizado durante o Programa de Iniciação Científica, o autor optou por usar um *software* mais avançado para o desenvolvimento deste projeto (Interface e Controlador). Entretanto, devido à inexperiência com o novo programa e também à sua complexidade, maior que a do *software* anterior, o andamento do projeto já vinha atrasado em relação ao cronograma. Além disso, houve dificuldade na montagem das placas do Controlador. Por se tratarem de protótipos produzidos de maneira semi-industrial, o acabamento da superfície exposta (terminais de cobre) é de qualidade muito inferior ao das placas fabricadas de forma industrial (PCIs da Interface). A Figura 50 [1] apresenta a comparação dos dois modelos: à esquerda, o protótipo do Controlador, e à direita, a placa da Interface, produzida de forma industrial. Nota-se que o acabamento da placa do Controlador é mais fosco que o da Interface, mais granuloso. Esse material é de difícil aderência ao estanho através do ferro de solda. Talvez, o mais indicado seria a montagem por meio de uma estação de solda por ar quente, que alcança de maneira uniforme toda a área a ser soldada, além do fluxo especial da pasta de solda. Entretanto, nem o autor, nem a universidade dispõem desse equipamento.

[1] A imagem está distorcida devido à lente de aproximação.

Figura 50 – Comparação do acabamento dos terminais de solda



Fonte: (O AUTOR, 2018)

Em função da placa de controle não ter sido finalizada dentro do período do cronograma, não foi possível testar o *firmware* desenvolvido, que já estava em uma etapa avançada, porém ainda não havia sido testado, uma vez que dependia dos periféricos do Controlador. O código das implementações se encontram na seção de Apêndices, sendo: inicialização dos periféricos internos, inicialização dos periféricos externos, técnica de *debounce* e fila de eventos. Faltando apenas a máquina de estados finita dos LEDs.

5 CONCLUSÃO

O principal objetivo deste trabalho era apresentar um protótipo funcional para a Mesa de Bolinhas. Embora não tenha sido atingido em sua totalidade, o projeto se encontra em uma etapa avançada. Entre os objetivos específicos apresentados, foram finalizadas as etapas do mapeamento da matriz de saída (LEDs), do mapeamento da matriz de entrada (sensores da matriz e do seletor de cores), do leiaute da placa de interface, do leiaute da placa de controle, do *firmware* gerenciador da matriz de entrada, da montagem das placas de interface, sendo ainda pendente a montagem da placa de controle, a integração com a mecânica e a finalização das rotinas da máquina de estados finita.

Para o autor, o projeto serviu de grande aprendizado de técnicas e ferramentas não abordadas durante a graduação, tais como: projeto de leiaute de placa de circuito impresso a nível intermediário, programação do microcontrolador ESP8266, programação orientada a eventos temporais, implementação de fila (FIFO), implementação de técnica de *debouncing* para múltiplas entradas, implementação de máquina de estados finita em *software*, entre outras.

5.1 TRABALHOS FUTUROS

Para que o projeto continue avançando, pretende-se, além de finalizar a montagem proposta neste trabalho, implementar novas funcionalidades, funções estas que não necessitam de alteração de *hardware*, visto que já foram consideradas no projeto dos circuitos:

- Gravação OTA (*over-the-air*): com essa funcionalidade, o microcontrolador poderá ser reprogramado sem a necessidade do adaptador USB: a gravação será via Wi-Fi, dispensando a remoção do tampo acrílico todas as vezes que uma atualização for necessária;
- Protocolo ArtNet: permitirá que a mesa seja comandada remotamente através de programas de mapeamento de luz (via Wi-Fi, sobre a camada de UDP). Assim, a matriz poderá servir como tela para animações e exibição de textos e imagens;
- Fita de LED externa: foi implementado em *hardware* uma saída para fitas de LED com suporte ao protocolo NRZ. Dessa forma, a mesa também poderá controlar outras instalações, sem a necessidade de um circuito dedicado

Além disso, o *hardware* e o código serão documentados e disponibilizados à comunidade, para que o projeto possa ser estudado, replicado e modificado conforme suas necessidades.

Referências

BURGESS, P. **Adafruit NeoPixel Überguide**. 2017. Disponível em: <https://learn.adafruit.com/adafruit-neopixel-uberguide>. Acesso em: 22 de novembro de 2018. Citado na página 9.

Espressif Systems. **ESP8266EX Datasheet**. 2018. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 19 de novembro de 2018. Citado na página 12.

GANSSLE, J. **A Guide to Debouncing**. 2007. Disponível em: <https://cseweb.ucsd.edu/classes/sp07/cse140L/debounce.pdf>. Acesso em: 22 de novembro de 2018. Citado 2 vezes nas páginas 14 e 15.

GREENSTED, A. **Switch Debouncing**. 2010. Disponível em: <http://www.labbookpages.co.uk/electronics/debounce.html>. Acesso em: 22 de novembro de 2018. Citado na página 14.

HALLIDAY, D.; WALKER, J.; RESNICK, R. **Fundamentals of Physics**. 8. ed. [S.l.]: Wiley, 2007. Citado na página 5.

HOROWITZ, P.; HILL, W. **The Art Of Electronics**. 2. ed. New York: Cambridge University Press, 1989. Citado 3 vezes nas páginas 9, 10 e 11.

KOLBAN, N. **Kolban's Book on ESP8266**. 1. ed. Texas: [s.n.], 2016. Citado 3 vezes nas páginas 11, 12 e 13.

NXP Semiconductors. **74HC165; 74HCT165 - 8-bit parallel-in/serial out shift register**. 2017. Disponível em: <https://assets.nexperia.com/documents/data-sheet/74HC_HCT165.pdf>. Acesso em: 15 de novembro de 2018. Citado 2 vezes nas páginas 7 e 8.

SPARKFUN. **SparkFun FTDI Basic Breakout - 3.3V**. 2015. Disponível em: <https://www.sparkfun.com/products/9873>. Acesso em: 24 de novembro de 2018. Citado na página 30.

Texas Instruments Incorporated. **SNx4HC165 8-Bit Parallel-Load Shift Registers**. 2015. Disponível em: <http://www.ti.com/lit/ds/symlink/sn74hc165.pdf>. Acesso em: 24 de novembro de 2018. Citado na página 23.

UNESCO. **International Year of Light**. 2015. Disponível em: <http://www.light2015.org/Home/Event-Programme.html>. Acesso em: 15 de novembro de 2018. Citado na página 1.

United Nations. International year of light and light-based technologies 2015: Resolution adopted by the general assembly on 20 december 2013. n. 3, p. 1–3, 2014. Citado na página 1.

Vishay Semiconductors. **Reflective Optical Sensor with Transistor Output**. 2008. Disponível em: <https://www.vishay.com/docs/83760/tcrt5000.pdf>. Acesso em: 15 de novembro de 2018. Citado 2 vezes nas páginas 6 e 7.

Apêndices

APÊNDICE A – plomodefs.h

```
#ifndef PLOMODEFS_H
#define PLOMODEFS_H

////////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////////
// nativas
//#include <stdlib.h>
#include <cstdint>
// usuário
#include <plomotypes.h>
////////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// variáveis públicas
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////////////

/**
*  \def ABS(x)
*   @brief      Calcula o valor absoluto do argumento \a x.
*   @param      x valor de entrada.
*   @return     valor absoluto de \a x.
*/
#define ABS(x) ( ((x)>0) ? (x) : -(x) )

/**
```

```
*  \def MAX(x, y)
*   @brief      Determina o maior valor entre os argumentos \a x e \a y.
*   @param      x primeiro valor.
*   @param      y segundo valor.
*   @return     maior valor entre \a x e \a y.
*/
#define MAX(x, y) ( (x)>(y) ? (x) : (y) )

/**
*  \def MIN(x, y)
*   @brief      Determina o menor valor entre os argumentos \a x e \a y.
*   @param      x primeiro valor.
*   @param      y segundo valor.
*   @return     menor valor entre \a x e \a y.
*/
#define MIN(x, y) ( (x)>(y) ? (y) : (x) )

/**
*  \def GET_ARRAY_LENGTH(arr)
*   @brief      Determina a quantidade de elementos de um array.
*   @param      arr array a ser consultado.
*   @return     quantidade de elementos no array.
*/
#define GET_ARRAY_LENGTH(arr) (sizeof(arr)/sizeof((arr)[0]))

#ifndef NDEBUG
#if defined(USE_FULL_ASSERT)
/**
*   @brief      Callback da macro \a assert_param, em caso de erro.
*   @note       Deve ser implementada pelo usuário para tratamento do erro.
*/
void assert_failed();
/**
*  \def assert_param(expr)
*   @brief      Testa a expressão é verdadeira. Se for falsa, chama a função de c
*   @param      Expressão a ser testada.
*/
#define assert_param(expr) ((expr) ? (void)0 : assert_failed());
#endif // USE_FULL_ASSERT
```

```
#endif // NDEBUG

//////////////////////////////////////////////////////////////////
// protótipo das funções
//////////////////////////////////////////////////////////////////

#endif // PLOMODEFS_H
```

APÊNDICE B – plomotypes.h

```
#ifndef PLOMOTYPES_H
#define PLOMOTYPES_H

//////////////////////////////////////////////////////////////////
// definições
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
//////////////////////////////////////////////////////////////////

typedef enum {ERROR = 0, SUCCESS = !ERROR} ErrorStatus;
typedef enum {RESET = 0, SET = !RESET} FlagStatus;
typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;

#endif // PLOMOTYPES_H
```

APÊNDICE C – tabmask8.h

```
#ifndef MESA_TABMASK_H
#define MESA_TABMASK_H
//////////////////////////////////////////////////////////////////
// bibliotecas
//////////////////////////////////////////////////////////////////
// nativas
#include <cstdint>
// usuário
#include <plomodefs.h>
//////////////////////////////////////////////////////////////////
// definições
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// variáveis globais
//////////////////////////////////////////////////////////////////
const uint8_t TABMASK_8[8] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
//////////////////////////////////////////////////////////////////
// macros
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// protótipo das funções públicas
//////////////////////////////////////////////////////////////////

#endif //MESA_TABMASK_H
```

APÊNDICE D – sensors.h

```
#ifndef MESA_SENSORES_H
#define MESA_SENSORES_H
////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////
// nativas
// usuário
#include <plomodefs.h>
////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////
// Quantidade de sensores.
#define NUM_SENSORS (128)
// Período de chamada da tarefa: 10ms
#define SENSOR_TASK_TIME_IN_MS (10)
////////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
// variáveis globais
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
// protótipo das funções públicas
////////////////////////////////////////////////////////////////////
void init_sensors();
void sensor_task();

#endif //MESA_SENSORES_H
```

APÊNDICE E – hardware.h

```
#ifndef MESA_HARDWARE_H
#define MESA_HARDWARE_H

////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////
// nativas
// usuário
#include <plomodefs.h>
////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////

typedef enum tagGPIO
{
    GPIO_STRIP_DOUT = (uint8_t)16,
    GPIO_SPI_SCK = (uint8_t)14,
    GPIO_SPI_MISO = (uint8_t)12,
    GPIO_SPI_MOSI = (uint8_t)13,
    GPIO_SPI_SS = (uint8_t)15,
    GPIO_LEDS_DOUT = (uint8_t)2,
    GPIO_PROGRAM = (uint8_t)0,
    GPIO_LEFTOVER_SENSOR = (uint8_t)4,
    GPIO_POWER_SENSOR = (uint8_t)5
} GPIO;

////////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
// variáveis globais
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
// macros
```

```
//////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////
// protótipo das funções públicas
//////////////////////////////////////////////////////////////////
void init_gpio();

#endif //MESA_HARDWARE_H
```

APÊNDICE F – application.cpp

```
////////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////////
// nativas
#include <SmingCore/SmingCore.h>
// usuário
#include <user_config.h>
#include <plomodefs.h>
#include <hardware.h>
#include <uart.h>
#include <sensors.h>
#include <matrix.h>
////////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// variáveis privadas do módulo
////////////////////////////////////////////////////////////////////////
Timer debounceTimer;
////////////////////////////////////////////////////////////////////////
// variáveis globais
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// protótipo das funções públicas
////////////////////////////////////////////////////////////////////////
```

```
/**
* @brief Inicialização do sistema.
* @param None.
* @return None.
*/
void init()
{
    // Inicialização das portas.
    init_gpio();

    // Inicializa a UART.
    init_uart();

    // Inicializa o módulo ARTNET.
    //init_artnet();

    // Inicialização dos sensores / SPI.
    init_sensors();

    // Inicializa a máquina dos LEDs.
    init_matrix();

    ////////////////////////////////
    // Tarefas periódicas.

    // Sensores
    debounceTimer.initializeMs(SENSOR_TASK_TIME_IN_MS, sensor_task).start();
}
```

APÊNDICE G – hardware.cpp

```
////////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////////
// nativas
#include <SmingCore/Digital.h>
#include <Wiring/WConstants.h>
// usuário
#include <hardware.h>
////////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// variáveis globais
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// variáveis privadas do módulo
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// protótipo das funções privadas
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// implementação do módulo
////////////////////////////////////////////////////////////////////////
```

```
/**
 * @brief Inicialização das portas.
 * @param None.
 * @return None.
 */
void init_gpio()
{
    // LEDs
    pinMode(GPIO_STRIP_DOUT, OUTPUT);
    pinMode(GPIO_LEDS_DOUT, OUTPUT);

    // Sensores
    pinMode(GPIO_POWER_SENSOR, INPUT);
    pinMode(GPIO_LEFTOVER_SENSOR, INPUT);

    // Demais portas não necessitam de inicialização.
}
```

APÊNDICE H – sensors.cpp

```
////////////////////////////////////////////////////////////////////
// bibliotecas
////////////////////////////////////////////////////////////////////
// nativas
#include <SmingCore/SPI.h>
// usuário
#include <sensors.h>
#include <matrix.h>
#include <tabmask8.h>
#include <hardware.h>
////////////////////////////////////////////////////////////////////
// definições
////////////////////////////////////////////////////////////////////
// Configurações do SPI
#define SPI_FREQ_HZ 4000000
#define SPI_BYTE_ORDER MSBFIRST
#define SPI_MODE SPI_MODE2

// Recarga do timer de debounce: 10 x 10ms = 100ms.
#define DEBOUNCE_TIMER_RELOAD_VALUE ((uint16_t)(10))

// Cada registrador é responsável por 1 coluna. São 17 colunas (matriz) + 1 selet
#define NUM_SHIFT_REGITERS (17+1)

// Quantidade de sensores individuais (power button, leftover).
#define NUM_SINGLE_SENSORS (1+1)

// Os bits de "sensorMask" são o espelho dos sensores.
// 128 sensores / 8 bits = 16 bytes
#define MASK_SIZE (16)

// Valor das entradas inválidas.
#define INVALID_INPUT ((uint8_t)0xFF)
////////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////
// dados personalizados, enumerações
////////////////////////////////////////////////////////////////

typedef struct tagSENSOR
{
    uint16_t timer;
    bool previousLevel;
} SENSOR;

////////////////////////////////////////////////////////////////
// variáveis globais
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
// variáveis privadas do módulo
////////////////////////////////////////////////////////////////

/**
 * Máscara dos sensores. Cada bit corresponde a um sensor da mesa.
 */
uint8_t sensorMask[MASK_SIZE];

/**
 * Estruturas de controle da máquina de estados do debounce.
 */
SENSOR sensors[NUM_SENSORS];

/**
 * Entradas mapeadas na máscara dos sensores.
 * Cada byte corresponde a uma entrada (entradas flutuantes, inclusive).
 */
const uint8_t mapped_inputs_in_sensor_mask[8*NUM_SHIFT_REGITERS+NUM_SINGLE_SENSORS]
{
            // Power button
            0, // input 0

            // Leftover sensor
```

```
1, // input 1

// Seletor de cor
2, // input 2
3, // input 3
4, // input 4
5, // input 5
6, // input 6
7, // input 7
8, // input 8
9, // input 9

// Coluna A
10, // input 10
11, // input 11
12, // input 12
13, // input 13
14, // input 14
INVALID_INPUT, // input 15
INVALID_INPUT, // input 16
INVALID_INPUT, // input 17

// Coluna B
15, // input 18
16, // input 19
17, // input 20
18, // input 21
19, // input 22
20, // input 23
INVALID_INPUT, // input 24
INVALID_INPUT, // input 25

// Coluna C
21, // input 26
22, // input 27
23, // input 28
24, // input 29
25, // input 30
26, // input 31
```

```
27, // input 32
INVALID_INPUT, // input 33

// Coluna D
28, // input 34
29, // input 35
30, // input 36
31, // input 37
32, // input 38
33, // input 39
34, // input 40
INVALID_INPUT, // input 41

// Coluna E
35, // input 42
36, // input 43
37, // input 44
38, // input 45
39, // input 46
40, // input 47
41, // input 48
42, // input 49

// Coluna F
43, // input 50
44, // input 51
45, // input 52
46, // input 53
47, // input 54
48, // input 55
49, // input 56
INVALID_INPUT, // input 57

// Coluna G
50, // input 58
51, // input 59
52, // input 60
53, // input 61
54, // input 62
```

```
55, // input 63
56, // input 64
57, // input 65

// Coluna H
58, // input 66
59, // input 67
60, // input 68
61, // input 69
62, // input 70
63, // input 71
64, // input 72
INVALID_INPUT, // input 73

// Coluna I
65, // input 74
66, // input 75
67, // input 76
68, // input 77
69, // input 78
70, // input 79
71, // input 80
72, // input 81

// Coluna J
73, // input 82
74, // input 83
75, // input 84
76, // input 85
77, // input 86
78, // input 87
79, // input 88
INVALID_INPUT, // input 89

// Coluna K
80, // input 90
81, // input 91
82, // input 92
83, // input 93
```

```
84, // input 94
85, // input 95
86, // input 96
87, // input 97

// Coluna L
88, // input 98
89, // input 99
90, // input 100
91, // input 101
92, // input 102
93, // input 103
94, // input 104
INVALID_INPUT, // input 105

// Coluna M
95, // input 106
96, // input 107
97, // input 108
98, // input 109
99, // input 110
100, // input 111
101, // input 112
102, // input 113

// Coluna N
103, // input 114
104, // input 115
105, // input 116
106, // input 117
107, // input 118
108, // input 119
109, // input 120
INVALID_INPUT, // input 121

// Coluna O
110, // input 122
111, // input 123
112, // input 124
```

```
            113, // input 125
            114, // input 126
            115, // input 127
            116, // input 128
            INVALID_INPUT, // input 129

            // Coluna P
            117, // input 130
            118, // input 131
            119, // input 132
            120, // input 133
            121, // input 134
            122, // input 135
            INVALID_INPUT, // input 136
            INVALID_INPUT, // input 137

            // Coluna Q
            123, // input 138
            124, // input 139
            125, // input 140
            126, // input 141
            127, // input 142
            INVALID_INPUT, // input 143
            INVALID_INPUT, // input 144
            INVALID_INPUT // input 145
};

////////////////////////////////////////////////////////////////
// macros
////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////
// protótipo das funções privadas
////////////////////////////////////////////////////////////////
void update_sensor_mask();
void config_bit_in_sensor_mask(uint8_t inputIndex, uint8_t value);
bool get_level_from_sensor_mask(uint8_t sensorId);
void rising_edge_callback(uint8_t sensorId);
void falling_edge_callback(uint8_t sensorId);
```

```
////////////////////////////////////////////////////////////////
// implementação do módulo
////////////////////////////////////////////////////////////////

/**
 * @brief Inicialização dos sensores reflexivos (barramento SPI e entradas extras)
 * @param None.
 * @return None.
 */
void init_sensors()
{
    // Configuração dos pinos.
    pinMode(GPIO_SPI_MISO, INPUT);
    pinMode(GPIO_SPI_MOSI, OUTPUT);
    pinMode(GPIO_SPI_SCK, OUTPUT);
    pinMode(GPIO_SPI_SS, OUTPUT);
    pinMode(GPIO_POWER_SENSOR, INPUT);
    pinMode(GPIO_LEFTOVER_SENSOR, INPUT);

    // SS em nível alto.
    digitalWrite(GPIO_SPI_SS, HIGH);

    // Inicializa o periférico.
    SPI.begin();

    // Limpa o buffer.
    memset(sensorMask, 0x00, sizeof(sensorMask));

    ////////////////////////////
    // Inicialização da máquina.

    // Atualiza o buffer.
    update_sensor_mask();

    // Configura os parâmetros.
    for (uint8_t id = 0; id < NUM_SENSORS; ++id)
    {
        // Recarrega o timer.
```

```
            sensors[id].timer = DEBOUNCE_TIMER_RELOAD_VALUE;
            // Inicializa o estado anterior.
            sensors[id].previousLevel = !get_level_from_sensor_mask(id);;
        }
    }

    /**
     * @brief Atribui o bit referente ao índice do sensor na mascara de sensores.
     * @param inputIndex índice da entrada.
     * @param value valor de teste.
     */
    void config_bit_in_sensor_mask(uint8_t inputIndex, uint8_t value)
    {
        // Verifica se o índice é válido
        if (inputIndex < sizeof(mapped_inputs_in_sensor_mask))
        {
            // Índice válido.
            ////////////////

            // Remapeia o índice.
            inputIndex = mapped_inputs_in_sensor_mask[inputIndex];

            // Verifica se é uma entrada válida.
            if (inputIndex != INVALID_INPUT)
            {
                // O bit não deve ser ignorado.
                ///////////////////////////////

                // Mapeia o bit no buffer.
                auto maskIndex = (uint8_t)(inputIndex / 8);
                uint8_t bitPos = TABMASK_8[inputIndex % 8];

                assert(maskIndex < sizeof(sensorMask));

                // Testa se o valor é verdadeiro.
                if (value)
                {
                    // Valor verdadeiro. Sinaliza o bit.
                    ////////////////////////////////////
```

```
                sensorMask[maskIndex] |= bitPos;
            }
            else
            {
                // Valor falso. Limpa o bit.
                ////////////////////////////
                sensorMask[maskIndex] &= (uint8_t)(~bitPos);
            }
        }

    }
}

/**
 * @brief Verifica o valor de um determinado sensor na máscara de sensores.
 * @param sensorId índice do sensor.
 * @return valor do sensor (true, se em nível alto; false, se em nível baixo).
 */
bool get_level_from_sensor_mask(uint8_t sensorId)
{
    // Mapeia o índice.
    auto maskIndex = (uint8_t)(sensorId / 8);
    uint8_t bitPos = TABMASK_8[sensorId % 8];

    assert(maskIndex < sizeof(sensorMask));

    return (bool)(sensorMask[maskIndex] & bitPos);
}

/**
 * @brief Atualiza o buffer \a sensorMask.
 * @param None.
 * @return None.
 */
void update_sensor_mask()
{
    // Índice das entradas.
    uint8_t inputIndex = 0;
```

```
    ////////////////////////////////////////////////////////////////////////////
    // Leitura dos sensores conectados diretamente aos pinos do microcontrolador.

    // Atribui o valor do primeiro sensor (power button).
    config_bit_in_sensor_mask(inputIndex++, digitalRead(GPIO_POWER_SENSOR));

    // Atribui o valor do segundo sensor (seletor de cor).
    config_bit_in_sensor_mask(inputIndex++, digitalRead(GPIO_LEFTOVER_SENSOR));

    ////////////////////////////////////////////////////////////////
    // Faz a leitura sequencial dos registradores de deslocamento.

    // Configura o barramento.
    SPI.beginTransaction(SPISettings(SPI_FREQ_HZ, SPI_BYTE_ORDER, SPI_MODE));

    // Faz os registradores capturarem o estado das portas (borda de descida).
    digitalWrite(GPIO_SPI_SS, LOW);

    // Percorre todos os registradores de deslocamento.
    for (uint8_t bufferIndex = 0; bufferIndex < NUM_SHIFT_REGITERS; ++bufferIndex)
    {
        // Lê o registrador atual.
        uint8_t collumn = SPI.read8();

        // Máscara dentro do registrador atual.
        uint8_t fastMask = 0b00000001;

        // Armazena seus bits na máscara dos sensores.
        for (uint8_t i = 0; i < 8; ++i)
        {
            // Atribui o valor do sensor ao buffer.
            config_bit_in_sensor_mask(inputIndex++, collumn & fastMask);

            // Próximo bit.
            fastMask <<= 1;
        }

    }
```

```
    // Libera os registradores.
    digitalWrite(GPIO_SPI_SS, HIGH);

    // Encerra a recepção.
    SPI.endTransaction();
}

/**
 * @brief função de callback para uma borda de subida identificada.
 * @param sensorId índice do sensor que gerou o evento.
 */
void rising_edge_callback(uint8_t sensorId)
{
    if (sensorId < NUM_SENSORS)
    {

    }
}

/**
 * @brief função de callback para uma borda de descida identificada.
 * @param sensorId índice do sensor que gerou o evento.
 */
void falling_edge_callback(uint8_t sensorId)
{
    if (sensorId < NUM_SENSORS)
    {

    }
}

/**
 * @brief rotina da tarefa periódica do módulo dos sensores.
 */
void sensor_task()
{
    // Atualiza o buffer.
    update_sensor_mask();
```

```
    // Percorre o buffer inteiro.
    for (uint8_t id = 0; id < NUM_SENSORS; ++id)
    {
        // Captura o nível atual.
        bool level = get_level_from_sensor_mask(id);

        // Referencia o sensor.
        SENSOR &thisSensor = sensors[id];

        // Verifica se houve mudança do nível desde a última execução da tarefa.
        if (level != thisSensor.previousLevel)
        {
            // Houve mudança do nível do sensor.
            ////////////////////////////////////

            // Recarrega o timer.
            thisSensor.timer = DEBOUNCE_TIMER_RELOAD_VALUE;

            // Atualiza o estado anterior.
            thisSensor.previousLevel = level;
        }
        else
        {
            // Não houve mudança do nível do sensor.
            ////////////////////////////////////////

            // Verifica se o timer está rodando.
            if (thisSensor.timer)
            {
                // O timer está rodando.
                ////////////////////////

                // Decrementa o timer e verifica se houve timeout.
                if (--thisSensor.timer == 0)
                {
                    // Houve timeout.
                    /////////////////

                    // Executa a função de callback conforme o nível.
```

```
                    if (level)
                    {
                        // Borda de subida.
                        ////////////////////

                        rising_edge_callback(id);
                    }
                    else
                    {
                        // Borda de descida.
                        /////////////////////

                        falling_edge_callback(id);
                    }
                }
            }
        }

    }
}
```