Universidade Federal de Sergipe
DComp - Departamento de Computação

Francisco, Renato, Arthur, Carlos Eduardo, Matheus

# Simulador da placa de aprendizagem Apple Juice

São Cristóvão - SE
2026

# Sumário

# 1 Introdução

O presente trabalho descreve o desenvolvimento de um simulador com interface gráfica da placa de aprendizagem Apple Juice, concebido para utilização no laboratório da FnEsc, vinculado ao Departamento de Física da Universidade Federal de Sergipe (UFS).

A placa Apple Juice consiste em um recurso didático destinado ao estudo de circuitos digitais implementados a partir de circuitos integrados discretos, tais como o CD4017, o NE555 e o CD4026. No contexto do simulador, buscou-se reproduzir o comportamento funcional desses componentes, incluindo a possibilidade de utilização de clock externo, o acionamento de sinais de reset e a comutação entre diferentes fontes de clock, com destaque para o uso do temporizador 555 operando em modo astável.

O desenvolvimento do sistema foi realizado no âmbito da disciplina de Programação Orientada a Objetos (POO), sob orientação do professor Carlos Estombelo. Nesse contexto, o projeto teve como objetivo a aplicação prática de conceitos fundamentais da programação orientada a objetos, tais como encapsulamento, herança e polimorfismo, além da utilização de mecanismos de tratamento de exceções por meio de estruturas `try-catch`.

A implementação do simulador foi baseada predominantemente no paradigma orientado a objetos, incorporando, quando necessário, abordagens de natureza procedural para lidar com aspectos específicos da simulação. Dessa forma, o sistema busca não apenas reproduzir o funcionamento da placa física, mas também consolidar, em nível prático, os conceitos teóricos abordados ao longo da disciplina.

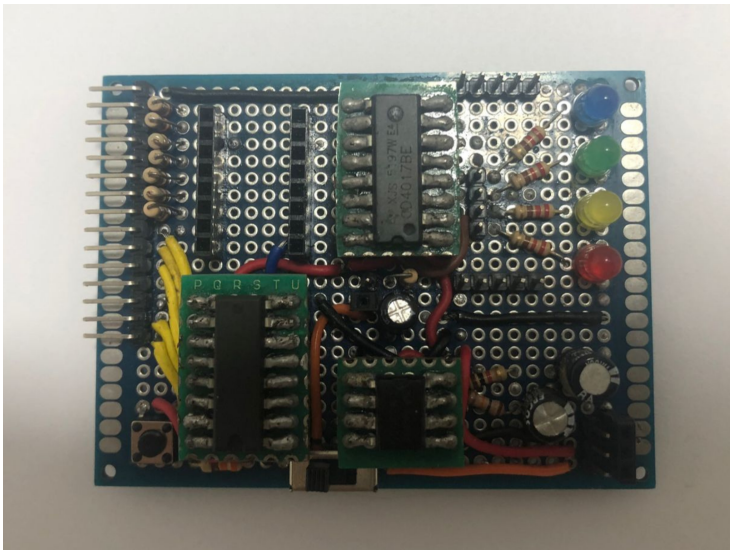

Figura 1: Placa Apple Juice, utilizada para estudo de circuitos digitais – autoria própria.

# 2 Bibliotecas Utilizadas

O projeto utiliza bibliotecas padrão da linguagem C++ para manipulação de entrada e saída, controle de threads, sincronização e operações matemáticas. Além disso, utiliza a biblioteca gráfica Raylib, encapsulada dentro de um namespace próprio.

Essa decisão evita conflitos de nomes e mantém a organização do código, uma vez que a Raylib é escrita em C e não possui namespace nativo.

```
// Bibliotecas padrao do C++
#include <iostream>   // Entrada e saida padrao (cout, cerr, etc.)
#include <string>     // Manipulacao de strings (std::string, std::to_string, etc.)
#include <cmath>      // Funcoes matematicas (pow, sin, etc.)
#include <cstdint>    // Tipos inteiros com tamanho fixo (uint32_t, int64_t, etc.)
#include <mutex>      // Controle de exclusao mutua para threads (std::mutex, std::lock_guard)
#include <atomic>     // Variaveis atomicas para comunicacao segura entre threads (std::atomic)
#include <stdexcept>  // Excecoes padrao (std::invalid_argument)
#include <thread>     // Threads do C++ (std::thread)
#include <chrono>     // Controle de tempo e delays (std::chrono::duration, sleep_for)
#include <cstdlib>    // Funcoes utilitarias gerais da biblioteca C (std::exit, std::rand, std::abs, etc.)

/*
    Biblioteca responsavel pela interface grafica:

    Coloquei esta biblioteca dentro do namespace 'ray' para
       diferencia-la de outras bibliotecas.
    Como ela foi escrita em C, nao possui namespace nativamente,
       ao contrario de bibliotecas C++ como 'iostream',
    que ficam dentro do 'std'.
*/
namespace ray{
    #include <raylib.h> // Biblioteca grafica Raylib
}
```

# 3 Classe Chip555

A classe `Chip555` simula o funcionamento do temporizador 555 operando no modo astável, no qual o circuito gera continuamente um sinal periódico sem necessidade de disparo externo.

## 3.1 Funcionamento no modo astável

No modo astável, o temporizador alterna automaticamente entre dois estados:

- Nível alto (HIGH)
- Nível baixo (LOW)

Essa oscilação ocorre devido ao processo cíclico de carga e descarga de um capacitor através de resistores.

## 3.2 Tempos de carga e descarga

Os tempos em que o sinal permanece em nível alto e baixo são determinados pelos componentes do circuito (resistores e capacitor).

### 3.2.1 Tempo em nível alto

$$t_{high} = 0.693 \cdot (R_1 + R_2) \cdot C \tag{1}$$

### 3.2.2 Tempo em nível baixo

$$t_{low} = 0.693 \cdot R_2 \cdot C \tag{2}$$

## 3.3 Período do sinal

O período total do sinal corresponde à soma dos tempos de nível alto e baixo:

$$T = t_{high} + t_{low} \tag{3}$$

## 3.4 Frequência do sinal

A frequência do sinal gerado é o inverso do período:

$$f = \frac{1}{T} \tag{4}$$

Substituindo os valores de $t_{high}$ e $t_{low}$:

$$f = \frac{1}{0.693 \cdot (R_1 + 2R_2) \cdot C} \tag{5}$$

## 3.5 Aplicação na implementação

As equações apresentadas foram utilizadas diretamente na implementação da classe `Chip555`, permitindo calcular dinamicamente os tempos de comutação do sinal e, consequentemente, o comportamento do clock interno do sistema.

## 3.6 Relação com o Circuito Físico

O funcionamento do temporizador 555 baseia-se na carga e descarga de um capacitor através de resistores.

Durante o tempo $t_{high}$, o capacitor carrega-se através de $R_1$ e $R_2$. Já durante $t_{low}$, ele descarrega-se apenas através de $R_2$.

Esse comportamento é controlado internamente por comparadores e um flip-flop, responsáveis por alternar o estado da saída.

## 3.7 código

Listing 1: Implementação da classe Chip555

```
class Chip555 {
private:
    double R1, R2, C;
    double tHigh = 0.0;
    double tLow  = 0.0;
    double period = 0.0;
    double freq   = 0.0;

    // logaritmo natural de 2
    const double Ln2    = 0.693;
    std::atomic<bool> stateHigh{false};

    void calcTimings() {
        // Modo astavel (aprox): tH = 0.693*(R1+R2)*C; tL =
           0.693*R2*C
        tHigh = Ln2 * (R1 + R2) * C;
        tLow  = Ln2 * (R2) * C;
        period = tHigh + tLow;
        freq = (period > 0) ? (1.0 / period) : 0.0;
```

```
    }

public:
    /*
        Esse construtor cria um objeto Chip555, inicializa seus
           parametros R1, R2 e C, verifica
        se eles sao validos, e calcula os tempos de pulso e
           frequencia para o sinal astavel.
    */
    Chip555(double r1Ohms, double r2Ohms, double cFarads)
        : R1(r1Ohms), R2(r2Ohms), C(cFarads) {
        if (R1 <= 0 || R2 <= 0 || C <= 0) {
            throw std::invalid_argument("R1, R2 e C precisam ser
               > 0");
        }
        calcTimings();
    }

    // Simula um ciclo de clock (HIGH e LOW) com delays
    void pulse() {
        stateHigh = true;
        std::this_thread::sleep_for(std::chrono::duration<double
           >(tHigh));

        stateHigh = false;
        std::this_thread::sleep_for(std::chrono::duration<double
           >(tLow));
    }

    // estes metodos apenas acessam os valores sem altera-los (
       const foi usado aqui como uma aplicacao de seguranca)
    bool isHigh() const {
        return stateHigh;
    }

    double getFrequency() const {
        return freq;
    }

    double getPeriod() const {
        return period;
```

```
    }
};
```

# 4 Classe Chip4017

O `Chip4017` é um contador Johnson de década responsável por ativar sequencialmente suas saídas digitais (Q0 a Q9) a cada pulso de clock recebido.

Seu funcionamento baseia-se no deslocamento de um único nível lógico alto (bit ativo) ao longo das saídas. A cada borda de subida do sinal de clock, o estado ativo avança para a próxima saída, mantendo apenas uma saída em nível alto por vez, enquanto as demais permanecem em nível baixo.

Após a ativação da última saída (Q9), o contador retorna automaticamente ao estado inicial (Q0), caracterizando um ciclo contínuo de contagem de 10 estados.

Além disso, o `Chip4017` possui um sistema de reset assíncrono, que permite forçar o contador imediatamente ao estado inicial, independentemente do clock. Quando o pino de reset é ativado (nível lógico alto), todas as saídas são zeradas, exceto Q0, que é definida como nível alto. Esse mecanismo é fundamental para garantir a inicialização correta do circuito e para possibilitar o controle do ciclo de contagem em aplicações específicas.

Listing 2: Implementação da classe Chip4017

```
class Chip4017 {
private:
    unsigned LimitReset;
    uint32_t Out{0};

public:
    explicit Chip4017(unsigned limitReset)
        : LimitReset(limitReset) {
        if (LimitReset < 1 || LimitReset > 10) {
            throw std::invalid_argument("LimitReset␣precisa␣estar
               ␣entre␣1␣e␣10.");
        }
        reset();
    }

    // metodo que reage ao pulso do clock deslocando o bit mais
       significativo para a direita
    void shift() {
        Out >>= 1;
        if (Out == 0) {
            Out = 1u << (LimitReset - 1);
        }
    }

    // metodo para aplicar o reset no chips
    void reset() {
        Out = 1u << (LimitReset - 1);
    }

    // apenas retornam - nao podem alterar o valor
    uint32_t getOut() const {
        return Out;
    }

    unsigned getLimitReset() const {
        return LimitReset;
    }
};
```

# 5 Classe Chip4026

A classe `Chip4026` representa um contador decimal com saída para display de 7 segmentos, responsável por exibir valores de 0 a 9 de forma sequencial.

## 5.1 Funcionamento geral

O `Chip4026` incrementa seu valor interno a cada pulso de clock recebido, atualizando simultaneamente as saídas que controlam o display.

O comportamento do contador é cíclico, ou seja, após atingir o valor máximo, ele retorna ao estado inicial.

## 5.2 Lógica de contagem

A lógica principal da classe consiste em:

- Incrementar o valor armazenado a cada pulso de clock;
- Limitar a contagem ao intervalo decimal (0 a 9);
- Reiniciar automaticamente o valor para 0 após atingir 9;

Esse comportamento garante a representação correta de um dígito decimal em aplicações digitais.

## 5.3 Sinal de carry

Ao ocorrer a transição de 9 para 0, a classe gera um sinal de *carry*. Esse sinal é utilizado para indicar o término de um ciclo completo de contagem, permitindo a sincronização com outros contadores em sistemas encadeados (como em contadores de múltiplos dígitos).

## 5.4 Extensibilidade e polimorfismo

A utilização de métodos virtuais na implementação permite que a classe `Chip4026` funcione como uma base para especializações. Dessa forma, é possível sobrescrever comportamentos específicos em classes derivadas, mantendo uma interface comum e possibilitando o uso de polimorfismo no sistema.

## 5.5 Código C++

Listing 3: Implementação da classe Chip4026

```
class Chip4026 {
protected:
    bool carryOut = false;      // Indica se houve estouro da
       contagem (Out voltou a 0)
    unsigned int Out = 0;       // Valor atual do display (0 a 9)

public:
    virtual ~Chip4026() = default;

    // Incrementa a contagem. Se atingir 9, reinicia e ativa
       carryOut
    virtual void add() {
        if (Out == 9) {
            Out = 0;
            carryOut = true;
        } else {
            Out++;
            carryOut = false;
        }
    }

    // Reseta o display e desativa o carry
    virtual void reset() {
        Out = 0;
        carryOut = false;
    }

    // Retorna o valor atual do display
    unsigned int getOut() const {
        return Out;
    }

    // Retorna se houve carry na ultima contagem
    bool getCarryOut() const {
        return carryOut;
    }
};
```

# 6 Classes Unidade e Dezena

A classe `Unidade` herda diretamente o comportamento padrão do `Chip4026`, representando o dígito menos significativo.

Listing 4: Implementação da classe que herda de Chip4026

```
class Unidade : public Chip4026 {
public:
    void add() override {
        Chip4026::add();
    }
};
```

Já a classe `Dezena` introduz uma lógica adicional: sua contagem só é incrementada quando ocorre um carry proveniente da unidade. Isso simula corretamente o comportamento de displays em cascata.

Listing 5: Implementação da classe que herda de Chip4026

```
class Dezena : public Chip4026 {
public:
    void addOnCarry(bool carryIn) {
        if (carryIn) {
            add();
        }
    }
};
```

# 7 Renderização do Display de 7 Segmentos

A renderização dos dígitos é feita por meio de funções que desenham segmentos individuais na tela.

Um `enum class` é utilizado para representar os segmentos do display, garantindo maior legibilidade e segurança no acesso aos índices.

A função principal dessa parte do sistema define quais segmentos devem ser ativados para cada número de 0 a 9 e os desenha na tela com base em suas posições e dimensões.

Listing 6: Código do display de 7 segmentos

```
enum class segments{
    a, b, c, d, e, f, g
};

// Desenha um display de 7 segmentos com base no valor (0-9)
static void DrawSevenSegment(ray::Vector2 pos, float size,
   unsigned int value, ray::Color color) {
    float w = size * 0.2f;
    float h = size * 0.05f;
    float gap = size * 0.02f;

    bool seg[7] = {false}; // definindo o valor de todos os
       elementos para 'false'
    switch(value) {
        case 0:
            seg[static_cast<size_t>(segments::a)] = true;
            seg[static_cast<size_t>(segments::b)] = true;
            seg[static_cast<size_t>(segments::c)] = true;
            seg[static_cast<size_t>(segments::d)] = true;
            seg[static_cast<size_t>(segments::e)] = true;
            seg[static_cast<size_t>(segments::f)] = true;
            break;

        ...

        case 9:
            seg[static_cast<size_t>(segments::a)] = true;
            seg[static_cast<size_t>(segments::b)] = true;
            seg[static_cast<size_t>(segments::c)] = true;
            seg[static_cast<size_t>(segments::d)] = true;
            seg[static_cast<size_t>(segments::f)] = true;
            seg[static_cast<size_t>(segments::g)] = true;
            break;
    }

    // vetor que armazena as posicoes dos segmentos
    ray::Vector2 positions[7] = {
        {pos.x + w + gap, pos.y},                       // A
        {pos.x + size - h, pos.y + w + gap},            // B
        {pos.x + size - h, pos.y + size - w - gap},     // C
```

```
        {pos.x + w + gap, pos.y + size - h + 35},        // D
        {pos.x, pos.y + size - w - gap},                 // E
        {pos.x, pos.y + w + gap},                        // F
        {pos.x + w + gap, pos.y + size/2 - h/2 + 20}     // G
    };

    // vetor que armazena as dimensoes dos segmentos
    ray::Vector2 dims[7] = {
        {size - 2*w - 2*gap, h},                         // A
        {h, size/2 - w - gap},                           // B
        {h, size/2 - w - gap},                           // C
        {size - 2*w - 2*gap, h},                         // D
        {h, size/2 - w - gap},                           // E
        {h, size/2 - w - gap},                           // F
        {size - 2*w - 2*gap, h}                          // G
    };

    for(int i=0; i<7; i++) {
        DrawSegment(positions[i], dims[i].x, dims[i].y, seg[i],
           color);
    }
}
```

# 8 Interface Gráfica

A interface gráfica é construída utilizando a biblioteca Raylib.
Os principais elementos visuais incluem:

- LEDs com efeito de brilho (glow);
- Displays de 7 segmentos;
- Painel de fundo estilizado;
- Indicadores de estado do sistema.

O efeito de brilho dos LEDs é simulado através da renderização de múltiplos círculos com transparência variável, criando um efeito visual mais realista.

Listing 7: simulação da formação do efeito de luminosidade dos leds

```
static void DrawLedGlow(ray::Vector2 center, float radius, ray::
   Color core, ray::Color glow) {
    // Ajustes finos do glow
    const int rings = 18;
    const float step = 1.5f;
    const float gamma = 1.5f;

    for (int i = rings; i >= 1; --i) {
        float t = (float)i / (float)rings;
        float r = radius + i * step;

        // Curva de queda do brilho: mais suave perto do LED e
           decai no fim
        float falloff = powf(t, gamma);
        unsigned char a = (unsigned char)(glow.a * falloff * 0.2f
           );

        ray::Color c = glow;
        c.a = a;
        DrawCircleV(center, r, c);
    }

    // Nucleo do LED
    ray::DrawCircleV(center, radius, core);

    // 'brilho interno' para parecer LED real
    ray::Color hi = core;
    hi.a = 90;
    ray::DrawCircleV((ray::Vector2){ center.x - radius * 0.25f,
       center.y - radius * 0.25f }, radius * 0.45f, hi);

    // Borda
    ray::DrawCircleLines((int)center.x, (int)center.y, radius,
       ray::Fade(ray::BLACK, 0.30f));
}
```

# 9 Classe BoardAppleJuice

Essa é a classe principal do sistema, responsável por integrar todos os componentes simulados.

Suas responsabilidades incluem:

- Inicializar a janela gráfica;
- Instanciar os chips simulados;
- Gerenciar a execução do sistema;
- Controlar a interação com o usuário.

## 9.1 Uso de Threads

Uma thread separada é utilizada para simular o funcionamento contínuo do clock gerado pelo 555.

Essa thread:

- Gera pulsos periódicos;
- Atualiza o estado dos chips;
- Sincroniza o acesso aos dados utilizando mutex.

O uso de variáveis atômicas garante comunicação segura entre a thread principal e a thread de simulação.

Listing 8: Thread responsável pelo pulso do 555 e atualização do CD4017

```
std::atomic<bool> running{true};
std::atomic<bool> ligado{false};

std::mutex mtx;

std::thread motor([&]{
    while (running.load()) {
        if (!ligado.load()) {
            std::this_thread::sleep_for(std::chrono::
               milliseconds(30));
            continue;
        }
        // Clock interno do 555 (modo astavel)
        chip555.pulse(); // alterna entre HIGH/LOW
           internamente
```

```
            // Atualiza os chips e displays a cada pulso
            std::lock_guard<std::mutex> lock(mtx);
            chip4017.shift();
            unidade.add();
            ezena.addOnCarry(unidade.getCarryOut());
        }
    });
```

## 9.2 Loop Principal

O loop principal é responsável por:

- Capturar eventos do teclado e mouse;
- Atualizar o estado do sistema;
- Renderizar os elementos gráficos.

Ele também garante o encerramento adequado do programa, evitando loops infinitos.

Listing 9: Loop principal

```
// colocando a condicao '&&' junto ao running.load(), foi
   possivel resolver o problema do loop infinito do programa que
   impedia o mesmo de ser fechado adequadamente
while (running.load() && !ray::WindowShouldClose())
    // condicionais responsaveis pelo controle do simulador
    if (ray::IsKeyPressed(ray::KEY_ENTER)) {
        ligado.store(!ligado.load());

    if (ray::IsKeyPressed(ray::KEY_R)) {
        std::lock_guard<std::mutex> lock(mtx);
        chip4017.reset();
        unidade.reset();
        dezena.reset();

    if (ray::IsKeyPressed(ray::KEY_ZERO)) {
        running.store(false);
        break;

    uint32_t bits = 0;
    {
        std::lock_guard<std::mutex> lock(mtx);
```

```
        bits = chip4017.getOut();

    // Cor padrao para os botoes
    ray::Color btnColor = ray::LIGHTGRAY

    // Botao de reset dos displays
    ray::Rectangle btnReset = { 400, 450, 140, 40 }

    // Responsavel por identificar se o botao esquerdo do mouse
       foi pressionado
    if (ray::IsMouseButtonPressed(ray::MOUSE_LEFT_BUTTON)) {
        ray::Vector2 mouse = ray::GetMousePosition()
        if (mouse.x >= btnReset.x && mouse.x <= btnReset.x +
           btnReset.width &&
            mouse.y >= btnReset.y && mouse.y <= btnReset.y +
               btnReset.height) {
            std::lock_guard<std::mutex> lock(mtx);
            unidade.reset();
            dezena.reset();
        }

    // renderizando as imagens na tela:
    ray::BeginDrawing();
        ray::Rectangle panel = { 60, 80, 1080, 320 };
        DrawPanel(panel)

        // Status para feedback do usuario
        const char* status = ligado.load() ? "LIGADO" : "
           DESLIGADO";
        ray::DrawText(
            ray::TextFormat("Status: %s  |  ENTER liga/desliga  |
               R reset all", status),
            40, 20, 18, ray::Fade(ray::RAYWHITE, 0.85f)
        )
        ray::Color clk = chip555.isHigh() ? (ray::Color){ 50,
           220, 130, 255 } : (ray::Color){ 200, 60, 60, 255 };
        ray::DrawCircle(830, 30, 7, clk);
        ray::DrawText("CLK", 845, 24, 16, ray::Fade(ray::RAYWHITE
           , 0.70f))
        ray::DrawText(
            ray::TextFormat("555: f=%.2f Hz | T=%.3f s", chip555.
```

```
            getFrequency(), chip555.getPeriod()),
        60, 80, 18, ray::Fade(ray::RAYWHITE, 0.55f)
    )

    // LEDs existentes
    float baseY = 280.0f;
    float radius = 32.0f;
    float margem = 120.0f;
    float areaUtil = 1200.0f - 2 * margem;
    float gap = areaUtil / (qtLeds - 1);
    float startX = margem
    float t = (float)ray::GetTime();
    float breathe = 0.5f + 0.5f * sinf(t * 3.2f)
    /*
        Loop que percorre todos os LEDs, desenhando cada um
            com brilho se estiver aceso e exibindo
        seu numero correspondente abaixo.
    */
    for (int i = (int)qtLeds - 1; i >= 0; --i) {
        bool on = ((bits >> i) & 1u) != 0;
        int idx = (int)qtLeds - 1 - i;
        ray::Vector2 c = { startX + idx * gap, baseY }
        ray::Color offCore = (ray::Color){ 120, 125, 135, 255
            };
        ray::Color offGlow = (ray::Color){ 120, 125, 135, 0 }
        unsigned char aGlow = (unsigned char)(70 + 90 *
            breathe);
        ray::Color onCore = (ray::Color){ 70, 255, 130, 220
            };
        ray::Color onGlow = (ray::Color){ 70, 255, 130, aGlow
            }
        if(on) {
            DrawLedGlow(c, radius, onCore, onGlow);
        } else {
            DrawLedGlow(c, radius, offCore, offGlow);
        }
        ray::DrawText(
            ray::TextFormat("L%d", idx + 1),
            (int)(c.x - 14),
            (int)(c.y + 52),
            18,
```

```
                ray::Fade(ray::RAYWHITE, 0.60f)
            );

        // Displays de 7 segmentos
        ray::Vector2 posUnidade = { 950-740, 450 };
        ray::Vector2 posDezena  = { 800-740, 450 };
        float displaySize = 120.0f
        DrawSevenSegment(posDezena, displaySize, dezena.getOut(),
            (ray::Color){70, 255, 130, 255});
        DrawSevenSegment(posUnidade, displaySize, unidade.getOut
            (), (ray::Color){70, 255, 130, 255})
        // Botao de reset
        ray::DrawRectangleRec(btnReset, btnColor);
        ray::DrawText("Reset Display", (int)(btnReset.x + 5), (
            int)(btnReset.y + 5), 18, ray::BLACK)
        // Mensagem de apoio
        ray::DrawText("Dica: aumente C (ex.: 47uF) para ficar
            mais lento; diminua C (ex.: 10uF) para acelerar.", 60,
            360, 16, ray::Fade(ray::RAYWHITE, 0.45f));
    ray::EndDrawing();
});
```

# 10 Segurança e Robustez do Sistema

A implementação do sistema foi desenvolvida com foco em segurança lógica e rigor no controle do fluxo de execução, aspectos fundamentais para garantir confiabilidade em aplicações computacionais.

A utilização do paradigma de Programação Orientada a Objetos (POO) contribui diretamente para esse objetivo, uma vez que promove encapsulamento, controle de acesso aos dados e organização estrutural do código. Esses fatores reduzem a possibilidade de estados inconsistentes e facilitam a manutenção e evolução do sistema.

Como artifício complementar, o programa emprega blocos `try-catch` para capturar exceções relacionadas a parâmetros inválidos e falhas inesperadas durante a execução. Esse mecanismo permite tratar erros de forma controlada, evitando interrupções abruptas e fornecendo maior previsibilidade ao comportamento do sistema.

Dessa forma, a combinação entre boas práticas de projeto (como o uso de POO) e o tratamento explícito de exceções contribui para um sistema mais robusto, seguro e estável.

Listing 10: Função main usando try e catch para tratar erros

```
int main() {
    try {
        // Adicione os valores para simulacao aqui:

        unsigned leds = 4;       // Numero total de LEDs para o
           4017
        double R1 = 1000.0;      // Resistor R1 do 555
        double R2 = 10000.0;     // Resistor R2 do 555
        double C  = 7.37e-6;     // Capacitor do 555

        // Cria o simulador e o executa
        BoardAppleJuice appleJuice(leds, R1, R2, C);
        appleJuice.run();
    }
    catch (const std::invalid_argument& e) {
        std::cerr << "Erro nos parametros do simulador: " << e.
           what() << std::endl;
        return EXIT_FAILURE;
    }
    catch (const std::exception& e) {
        std::cerr << "Erro inesperado: " << e.what() << std::endl
           ;
        return EXIT_FAILURE;
    }
    catch (...) {
        std::cerr << "Erro desconhecido ocorreu!" << std::endl;
        return EXIT_FAILURE;
    }
    // Programa finalizado com sucesso
    return EXIT_SUCCESS;
}
```

Após fazer os processos descritos no README.md sobre 'Como compilar e rodar', é possível o próprio usuário realizar testes sem a interface gráfica usando o comando:

```
make test
```

# 11 Conclusão

O desenvolvimento do simulador da placa de aprendizagem Apple Juice permitiu consolidar conceitos fundamentais de Programação Orientada a Objetos (POO) e aplicar conhecimentos de circuitos digitais de forma prática. A implementação dos chips simulados – `Chip4026`, `Chip555` e `Chip4017` – demonstrou como os componentes digitais podem ser modelados em software, reproduzindo fielmente seu comportamento físico.

O uso de POO garantiu maior organização, modularidade e segurança no sistema, possibilitando a extensão e manutenção do código com facilidade. Os mecanismos de encapsulamento, herança e polimorfismo foram essenciais para criar uma arquitetura flexível, permitindo a especialização de classes e o uso de polimorfismo no controle de displays e contadores.

A implementação de tratamento de exceções por meio de blocos `try-catch` contribuiu para a robustez do simulador, garantindo que erros de entrada ou situações inesperadas fossem tratados de forma segura, sem comprometer a execução contínua do sistema.

A interface gráfica, construída com a biblioteca Raylib, proporcionou uma visualização intuitiva e realista do funcionamento da placa, incluindo a simulação de LEDs com efeito de brilho e displays de 7 segmentos, reforçando a compreensão dos conceitos teóricos.

Por fim, este projeto não apenas replica o funcionamento da placa física, mas também serve como ferramenta didática para estudo de circuitos digitais, integração de componentes e programação orientada a objetos, consolidando a relação entre teoria e prática no contexto do laboratório de física aplicada.

# 12 Repositório original do projeto

O repositório original do projeto pode ser acessado em: Apple Juice Learning Board Simulator no GitHub

# 13 Referências


- STMicroelectronics. **NE555 Timer Datasheet**. Disponível em: `https://www.st.com`. Acesso em: 22 fev. 2026.
- Texas Instruments. **CD4017B Decade Counter/Divider Datasheet**. Disponível em: `https://www.ti.com`. Acesso em: 22 fev. 2026.
- Texas Instruments. **CD4026B Decade Counter/7-Segment Display Driver Datasheet**. Disponível em: `https://www.ti.com`. Acesso em: 22 fev. 2026.
- :contentReference[oaicite:0]index=0. **Raylib Documentation**. Disponível em: `https://www.raylib.com`. Acesso em: 22 fev. 2026.
- :contentReference[oaicite:1]index=1. **The C++ Programming Language**. 4. ed. Boston: Addison-Wesley, 2013.
- :contentReference[oaicite:2]index=2. **Programming: Principles and Practice Using C++**. 2. ed. Boston: Addison-Wesley, 2014.