CENTRO FEDERAL DE EDUCACAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA
DEPARTAMENTO DE EDUCAÇÃO SUPERIOR - DEPES
ENGENHARIA ELETRÔNICA

LUCAS DE FREITAS FERNANDES

DESENVOLVIMENTO DE UM SOFTWARE DE SIMULAÇÃO DE CIRCUITOS DIGITAIS COM INTEGRAÇÃO PARA DISPOSITIVOS FÍSICOS

TRABALHO DE CONCLUSÃO DE CURSO

RIO DE JANEIRO
2025

LUCAS DE FREITAS FERNANDES

Desenvolvimento de um Software de Simulação de Circuitos Digitais com Integração para Dispositivos Físicos

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Eletrônica, do Departamento de Educação Superior, do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca.

Orientador: Profa. Dra. Aline Gesualdi Manhães

RIO DE JANEIRO
2025

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

F363 Fernandes, Lucas de Freitas
Desenvolvimento de um software de simulação de circuitos digitais com integração para dispositivos físicos / Lucas de Freitas Fernandes – 2025.
xii, 61f.: il. (algumas color.) , enc.

Projeto Final (Graduação). Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2025.
Bibliografia: f. 60-61.
Orientador: Aline Gesualdi Manhães.

1. Engenharia eletrônica. 2. Circuitos digitais. 3. Simulação (computador digital). I. Manhães, Aline Gesualdi (Orient.). II. Título.

CDD 621.381

Elaborada pelo bibliotecário Leandro Mota de Menezes – CRB/7 nº 5281

AGRADECIMENTOS

A Deus, por me sustentar em todas as coisas.
A minha família, pelo suporte em todos os momentos.
A Anna, minha companheira, pela compreensão nos momentos de ausência, e por me emprestar um pouco do seu bom gosto na visão sobre o projeto.
A minha orientadora, Aline, sem a qual esse projeto não teria sido possível, por toda a orientação, pelas trocas, e por todos os momentos dedicados.
A professora Luciana, pela oportunidade de discussão e pelo direcionamento de materiais.
A todos aqueles com quem conversei e que puderam compartilhar ideias e conhecimentos.
A esta instituição, e a todos com quem pude aprender um pouco mais durante essa longa jornada.

RESUMO

O aprendizado de eletrônica digital por parte de estudantes de cursos profissionalizantes, idealmente, deve envolver a compreensão dos circuitos e também a familiarização com a prática de montagem. No entanto, em muitos contextos educacionais pode ocorrer uma limitação quanto ao acesso a laboratórios e materiais para a experimentação. Muitos professores tem utilizado simuladores de circuitos digitais como uma ferramenta de apoio para o ensino, porém, os simuladores tradicionais não são capazes de suprir a necessidade básica de contato com circuitos físicos. Este trabalho propõe o desenvolvimento de um novo simulador, chamado SimClick, que oferece o recurso de integração de circuitos físicos e simulados através da conexão de dispositivos como o Raspberry Pi. O objetivo é permitir que uma nova ferramenta possa auxiliar no ensino da eletrônica digital, de forma a tornar a aprendizagem mais interativa por meio de uma abordagem híbrida.

Palavras-chave: Simulador; Circuitos Digitais; Software; Hardware; Integração.

ABSTRACT

The learning process of digital electronics by students in professional training programs should ideally involve an understanding of circuits and also the practical experience in circuit assembly. However, in many educational contexts, access to laboratories and materials for experimentation is often limited. Many educators have adopted digital circuit simulators as an educational tool, although conventional simulators are not capable of addressing the need for students to interact with physical circuits. This work proposes the development of a new simulator, named SimClick, that offers the possibility of integrating physical and simulated circuits through the connection of devices such as the Raspberry Pi. The goal is to provide a new tool that can aid in the education of digital electronics, making the learning process more interactive through a hybrid approach.

Keywords: Simulator; Digital Cicuits; Software; Hardware; Integration.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

API	Interface De Programação De Aplicações (Inglês: *Application Programming Interface*) 16, 17
CSS	Folhas De Estilo Em Cascata (Inglês: *Cascading Style Sheets*) 14, 15
GPIO	Entrada/Saída De Uso Genérico (Inglês: *General Purpose Input/Output*) 13, 31, 32, 45, 47, 51, 52, 54, 58
GUI	Interface Gráfica Do Usuário (Inglês: *Graphical User Interface*) 11, 13, 16, 17, 18
HTML	Linguagem De Marcação De HiperTexto (Inglês: *HypeText Markup Language*) 14, 15
I2C	Circuito InterIntegrado (Inglês: *Inter-Integrated Circuit*) 33, 58
IOT	Internet Das Coisas (Inglês: *Internet of Things*) 27
JS	JavaScript 15, 16, 25
JSON	Notação De Objeto Do JavaScript (Inglês: *JavaScript Object Notation*) 25
LAN	Rede De Área Local (Inglês: *Local Area Network*) 27, 33
LED	Diodo Emissor De Luz (Inglês: *Light-Emitting Diode*) 46, 47, 50, 51, 52, 54, 55, 56
MDNS	Sistema De Nomes De Domínio Multicast (Inglês: *Multicast Domain Name System*) 59
MVP	Produto Mínimo Viável (Inglês: *Minimum Viable Product*) 34
PWM	Modulação Por Largura De Pulsos (Inglês: *Pulse Width Modulation*) 33, 58
RPC	Chamada De Procedimento Remoto (Inglês: *Remote Procedure Call*) 18
SCLD	Simulador De Circuito Lógico Digital 11
SPI	Interface Serial De Periféricos (Inglês: *Serial Peripheral Interface*) 32
TCP	Protocolo De Controle De Transmissão (Inglês: *Transmission Control Protocol*) 33
TS	TypeScript 16
UART	Receptor / Transmissor Assíncrono Universal (Inglês: *Universal Asynchronous Receiver / Transmitter*) 33

Capítulo 1

Introdução

No ensino da manipulação de circuitos lógicos digitais em cursos profissionalizantes existe o desafio de trabalhar com duas abordagens que são essenciais para a familiarização dos estudantes com a disciplina. Primeiramente, a formalização teórica — que abrange a matemática e as técnicas de cálculo — e, em segundo lugar, o desenvolvimento do conhecimento prático por meio da montagem dos circuitos.

Uma das categorias de ferramentas que vem sendo bastante explorada, e que auxilia na experimentação e na verificação da aplicação das teorias, é a dos simuladores de circuitos. Em especial nos últimos anos, tendo sido impulsionada pelo isolamento imposto durante a pandemia do COVID-19, muitos professores e instituições encontraram nesses simuladores uma forma alternativa de desenvolver experiências, principalmente no contexto da limitação de acesso aos laboratórios.

Pesquisas recentes têm explorado bastante os diversos aspectos da experiência de simulação de circuitos digitais e as formas de interação dos usuários com essas ferramentas. Alguns estudos enfatizam o uso de simuladores que possam ser acessados diretamente pelo navegador de internet, enquanto outros trabalham com a instalação de programas no computador, ou ainda focam em aspectos colaborativos.

Embora sejam inegáveis as vantagens de poder trabalhar com componentes completamente virtuais — seja em termos de custo, facilidade, ou na versatilidade de acesso —, também é importante destacar que a interação com componentes físicos desempenha um papel fundamental no processo de aprendizagem. Isso se deve tanto aos fatores lúdicos, que tornam a visualização e interação com os sistemas mais intuitivas, quanto à necessidade de familiarização com os componentes que os profissionais irão interagir no mercado de trabalho.

Com o intuito de combinar os benefícios providos pelos simuladores digitais com aqueles obtidos ao utilizar componentes físicos, este trabalho propõe o desenvolvimento de um software simulador de circuitos lógicos digitais, chamado SimClick, que se destaca por sua capacidade de integrar elementos físicos na simulação. Esse re-

curso é possibilitado através da conexão com placas de desenvolvimento — como o Raspberry Pi — e da utilização de suas portas de entrada e saída para realizar a interação com os circuitos montados.

1.1 Objetivo

A elaboração do presente trabalho tem como propósito o desenvolvimento de um programa para a simulação de circuitos lógicos digitais que una, de maneira inovadora, a simulação virtual — com experiência semelhante a de simuladores tradicionais — e a interação direta com circuitos físicos, por meio de placas de desenvolvimento, como, por exemplo, o Raspberry Pi. A proposta busca enriquecer o ambiente de aprendizado, de forma que o usuário possa não apenas visualizar e interagir com circuitos em um ambiente digital, mas também experimentar a montagem prática desses circuitos através de interfaces que se conectem com elementos reais.

O simulador a ser desenvolvido visa apoiar estudantes e profissionais ao promover um novo recurso para ser utilizado no processo de compreensão dos conceitos de eletrônica digital, permitindo que o estudo teórico possa ser imediatamente correlacionado com a prática. Para isso, o simulador contará com uma interface intuitiva, inspirada em simuladores de mercado e aplicações web como quadros colaborativos, que promoverá uma experiência de uso simples e dinâmica.

Ao combinar a simulação virtual com a montagem prática de circuitos, o projeto busca suprir uma lacuna existente nos métodos convencionais de ensino de eletrônica digital, proporcionando uma abordagem híbrida que amplie a experiência de estudantes e contribua para a formação de novos profissionais.

1.2 Motivação

A motivação para o desenvolvimento deste projeto surge a partir da análise e da constatação de que os simuladores tradicionais disponíveis para o trabalho com circuitos digitais, embora eficazes em permitir a realização de testes com acesso a uma ampla variedade de componentes, não proporcionam a experiência prática necessária a futuros profissionais que desejam atuar em ramos da eletrônica digital. Também é importante destacar que, em muitos contextos educacionais, o acesso limitado a

laboratórios práticos impede que os alunos possam interagir diretamente com componentes reais, o que compromete sua formação e entendimento no que tangem a esses aspectos.

Além disso, a ampliação do acesso a tecnologias proporcionadas por dispositivos controladores de baixo custo tornou viável a integração entre simulação digital e a interação com hardware físico. Essa convergência oferece uma oportunidade para inovação nos métodos tradicionais de ensino, permitindo que os estudantes possam desenvolver habilidades práticas ao mesmo tempo em que compreendem conceitos teóricos em situações reais.

1.3 Estrutura do Trabalho

Nesta seção, descrevemos a organização geral do trabalho e pontuamos os principais tópicos e a sequência lógica de conteúdos abordados.

Iniciamos na seção **Fundamentação Teórica** (2) com uma revisão dos conceitos fundamentais de eletrônica digital, essenciais para a compreensão dos temas discutidos ao longo do trabalho.

Em seguida, em **Trabalhos Relacionados e Programas de Referência** (3), apresentamos uma síntese de estudos anteriores que abordaram o desenvolvimento de simuladores de circuitos digitais, destacando também outras aplicações que foram analisadas e serviram de inspiração para o desenvolvimento do SimClick.

Na seção **Desenvolvimento do Software** (4) detalhamos aspectos relevantes do desenvolvimento do programa, incluindo as tecnologias empregadas e como foi projetada a arquitetura do sistema. Também discutimos brevemente algumas limitações presentes na implementação atual.

Na sequência, **Apresentação do Programa** (5) explora a interface e as funcionalidades do software do ponto de vista do usuário, utilizando exemplos práticos para ilustrar o comportamento e a utilização de alguns dos recursos oferecidos.

Resultados (6) apresenta uma discussão sobre os resultados gerais obtidos, complementada por dois casos de uso que demonstram circuitos simulados e a integração com componentes externos.

Por fim, encerramos com as conclusões do projeto (7), apontando possíveis direções para desenvolvimentos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Circuitos Lógicos Digitas

O estudo das eletrônicas analógica e digital se diferem principalmente pela natureza de como os sinais são quantificados em suas medidas para os objetivos de cada disciplina. Isso é, de acordo com a forma adotada, teremos dois modos de representação das quantidades desses sinais, sendo correspondentemente chamadas de representação analógica e representação digital.

Na representação analógica, os sinais são continuamente variáveis, podendo assim, a qualquer instante, ter o valor do sinal assumindo qualquer valor contínuo dentro dos limites definidos para o sistema. Dessa forma, não existem níveis predefinidos ou "saltos" entre os valores possíveis.

Em contrapartida, a representação digital de sinais trabalha com valores discretos, baseando-se em níveis predefinidos. Valores que são fisicamente contínuos, são aproximados para valores correspondentes a cada um desses níveis. Conforme afirmado por (TOCCI; WIDMER; MOSS, 2018, p. 16), "a representação digital é o resultado da atribuição de um número de precisão limitada a uma quantidade continuamente variável". Em geral, em sistemas digitais, são empregados circuitos que operam com duas faixas de valores — seus limites contínuos definidos de acordo com o projeto —, que representam os estados lógicos ALTO e BAIXO.

Olhando para o estudo de sistemas a partir da ótica digital, também é importante abordarmos a questão das representações numéricas adotadas. Sabendo que os sistemas operam com dois estados lógicos possíveis, adotamos um sistema numérico conveniente por trabalhar também apenas com dois valores de algarismos, o binário, sendo esses dois valores "0" e "1", respectivamente correspondentes aos níveis BAIXO e ALTO. Cada dígito representado nesse sistema é chamado de *bit*, termo derivado de "dígito binário" (do inglês: *binary digit*).

Sistemas montados para tratar com sinais digitais consideram suas entradas e são

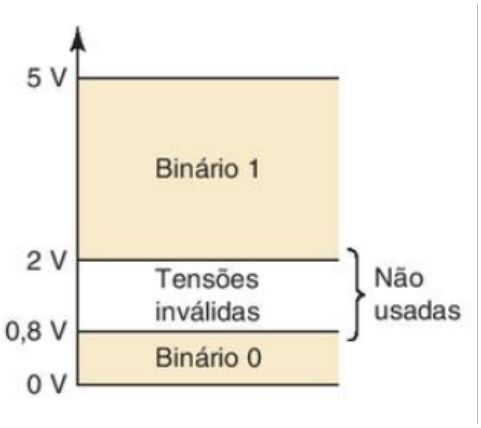


Figura 1: Exemplo de níveis lógicos de tensão para um dispositivo. Extraído de (TOCCI; WIDMER; MOSS, 2018, p. 7).

capazes de efetuar operações lógicas por meio de seus componentes. Essas operações, por sua vez, podem ser descritas por meio de uma formalização matemática que será abordada mais a frente. Devido a essa natureza particular de processamento das entradas, que obedecem a um conjunto de regras lógicas, denominamos, alternativamente, os circuitos digitais como circuitos lógicos.

2.2 Álgebra Booleana

Conforme citado anteriormente, um circuito lógico permite receber entradas de sinais digitais e obter uma saída através de um comportamento interno de processamento que equivale a uma combinação de operações básicas que podemos descrever como operações lógicas.

Um destaque para a importância desses conceitos é que a partir desses comportamentos dos circuitos digitais somos capazes de representar estados de elementos reais que desejamos que sejam manipulados, e fazer com que o circuito execute as ações desejadas em resposta a esses estados por meio da codificação das expressões lógicas através dos elementos do circuito. Dessa forma, associamos os estados ALTO e BAIXO dos circuitos com os estados lógicos “verdadeiro” e “falso”, e podemos construir expressões do tipo: “se está escuro (‘verdadeiro’ ou ‘falso’), a luz deve ser acesa (resultado)”.

Além disso, para que seja facilitado o trabalho de construção e análise dessas expressões lógicas, necessitamos de notações mais práticas para a representação. Nesse contexto, podemos trabalhar com a notação de expressões booleanas, onde

usaremos símbolos específicos para cada operação válida — descritas a seguir —, e representaremos as entradas, isto é, os elementos que queremos medir, como variáveis que poderão assumir os estados “verdadeiro” ou “falso”. A partir dessa notação seremos capazes de melhor analisar a relação entre as entradas e saídas.

A álgebra booleana possui três operações básicas para descrever as relações entre as entradas e saídas de suas expressões. A partir dessas operações básicas, também podemos compor algumas outras operações derivadas que abordaremos em seguida. As operações básicas disponíveis são chamadas de **NOT** (“NÃO” ou inversão), **OR** (“OU”) e **AND** (“E”), sendo cada uma descrita por um símbolo correspondente nas expressões.

A operação NOT é responsável por inverter uma entrada, ou seja, a operação aplicada uma entrada “verdadeira” resulta em uma saída “falsa”, e vice-versa. Essa operação pode ser representada como uma apóstrofe ao final ($'$), ou como uma barra vertical sobre o operando. Um exemplo das duas formas pode ser visto nas equações 2.1 e 2.2, que descrevem o equivalente a “x é igual ao INVERSO de A”.

$$x = A' \tag{2.1}$$

$$x = \overline{A} \tag{2.2}$$

A operação OR deve resultar em “verdadeiro” quando pelo menos um de seus operandos for “verdadeiro”. É representada em expressões por um sinal mais ($+$). Um exemplo na equação 2.3, que descreve “x é igual a A OU B”.

$$x = A + B \tag{2.3}$$

Por fim, a operação AND representa que o resultado deve ser “verdadeiro” apenas quando todos os seus operandos também o forem. A operação AND é expressa pelo sinal de ponto multiplicativo ($\cdot$). O exemplo da equação 2.4 representa a expressão lógica “x é igual a A E B”.

$$x = A \cdot B \tag{2.4}$$

2.3 Portas Lógicas e Tabelas Verdade

Na montagem dos circuitos digitais, as operações lógicas descritas pelas operações básicas que foram apresentadas e outras operações, mais complexas, serão sintetizadas através de componentes lógicos específicos que representam cada uma. A esses componentes chamamos “portas lógicas”, e cada uma é denominada conforme a operação que representa.

Outra ferramenta útil para a análise de circuitos digitais é a “tabela verdade”, que é a representação em formato tabular de todos os valores possíveis para as entradas do circuito e de todos os valores correspondentes assumidos pelas saídas.

A seguir apresentamos as tabelas verdade e os símbolos correspondentes para as portas lógicas de cada uma das operações. Estão listadas as operações básicas e também algumas operações que podem ser compostas a partir dessas três. Para as tabelas utilizamos “0” para nível BAIXO e “1” para ALTO.

Operação NOT

A operação NOT trabalha apenas com um operando, por isso a porta lógica é representada apenas com uma entrada. Terá o sinal na saída invertido em relação à entrada.

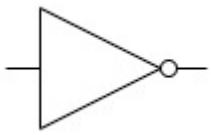


Figura 2: Porta lógica NOT.

A	$\overline{A}$
0	1
1	0

Tabela 1: Tabela verdade para operação NOT.

Operação OR

A porta lógica básica para a representação da operação OR possui duas entradas, embora se possa construir outras abstrações da mesma operação para mais entradas. Nesse caso, a saída será ALTO sempre que qualquer das entradas for ALTO.

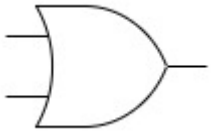


Figura 3: Porta lógica OR.

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

Tabela 2: Tabela verdade para operação OR.

Operação AND

Da mesma forma que para a operação OR, a porta lógica básica correspondente a operação AND terá duas entradas. Na necessidade de abstração para mais entradas, todas precisam ser ALTAS para que a saída seja ALTAS.

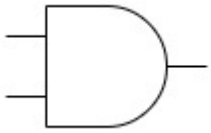


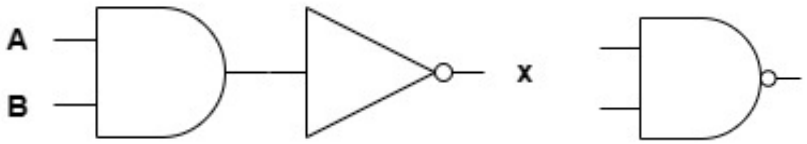
Figura 4: Porta lógica AND.

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 3: Tabela verdade para a operação AND.

Operação NAND

Começando a tratar de operações mais complexas, que representam combinações de operações simples, temos a chamada NAND, que é o resultado da inversão do resultado de uma operação AND.



(a) **(b)**

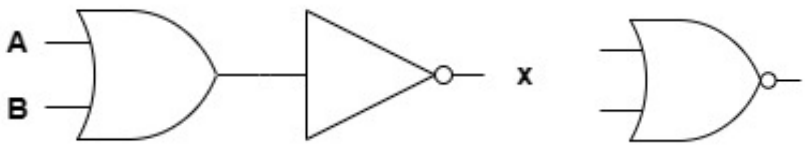
Figura 5: Porta lógica NAND.
(a) Circuito para composição da porta lógica.
(b) Símbolo da porta lógica.

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Tabela 4: Tabela verdade usada para a operação NAND.

Operação NOR

Similar ao que a operação NAND representa para a operação AND, a operação NOR é o resultado da inversão da saída de uma operação OR.



(a) **(b)**

Figura 6: Porta lógica NOR.
(a) Circuito para composição da porta lógica.
(b) Símbolo da porta lógica.

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Tabela 5: Tabela verdade usada para a operação NOR.

Operação XOR

A operação XOR ("*eXclusive OR*", "OU exclusivo" em inglês) é uma operação que representa que apenas um de seus operadores pode ser ALTO por vez — de forma exclusiva — para que o resultado seja ALTO. É formada pela expressão 2.5, mas também possui um símbolo específico que pode ser utilizado, conforme a equação 2.6.

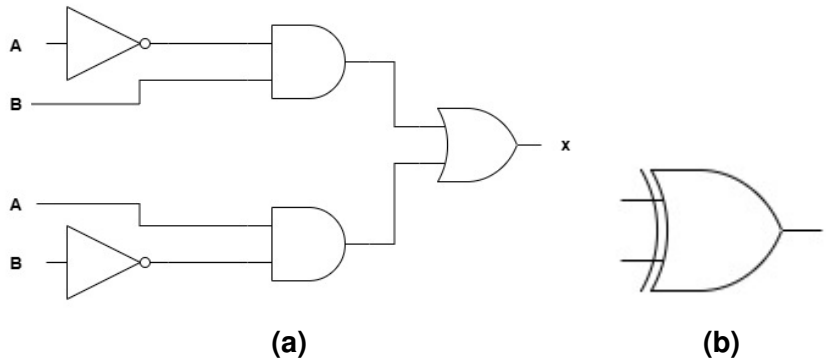


Figura 7: Porta lógica XOR.
(a) Circuito para composição da porta lógica.
(b) Símbolo da porta lógica.

$$x = A \cdot \overline{B} + \overline{A} \cdot B \tag{2.5}$$

$$x = A \oplus B \tag{2.6}$$

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 6: Tabela verdade usada para a operação XOR.

Capítulo 3

Trabalhos Relacionados e Programas de Referência

Nos últimos anos, tem se observado o desenvolvimento de novos simuladores de circuitos digitais, de forma similar ao proposto neste projeto, por meio de diversas abordagens e perspectivas que enfatizam distintos aspectos de aplicação. No caso do SimClick, foram usados como base de inspiração tanto simuladores de circuitos lógicos discutidos em diversos estudos, como também alguns simuladores de mercado, os quais foram analisados para verificar quais características seriam mais apropriadas para a finalidade do projeto.

Recentemente, o programa Simulador de Circuito Lógico Digital (SCLD) foi desenvolvido por (SANTOS, 2023), capaz de trabalhar com componentes como portas lógicas, fontes e *clocks* (fontes de pulsos temporizadas). Esse trabalho buscou tornar fácil o acesso ao simulador e ao compartilhamento de circuitos por estudantes, permitindo que a aplicação possa ser executada totalmente através de um site na web, o que reduz as exigências de hardware do computador do usuário e a necessidade de instalação de programas.

Em sua proposta, (SANTOS, 2023) também faz referência a dois outros projetos de simuladores que serviram como base de inspiração e que cabem ser citados aqui pela diferença em suas abordagens. O simulador descrito por (FRANCO, 2016) também se utiliza de tecnologias web que exploram conceitos que continuaram a ser amplamente aplicados no mercado de desenvolvimento na última década. Esse simulador, por sua vez, teve um foco na criação de funcionalidades diferentes, tendo como uma de suas principais preocupações possibilitar aos usuários que trabalhem de forma colaborativa em tempo real. O outro projeto citado, foi aquele proposto por (SOUZA DE LUCENA, 2013), onde foi adotada uma abordagem mais tradicional de desenvolvimento, sendo essa a da criação de um simulador instalável localmente na máquina do usuário. Um ponto importante sobre esse último é que, ao se basear primariamente em programas simuladores mais tradicionais de mercado, a sua implementação da Interface Gráfica do Usuário (inglês: *Graphical User Interface*) (GUI) incorporou diversas ferramentas

que são mais comuns nesse meio, como o desenho de fios por meio da marcação de quinas, além da implementação de atalhos e macros.

Um simulador comercial, com foco educacional, que oferece uma versão demonstrativa para acesso através da web, e que, na versão paga, pode ser instalado em dispositivos com Windows e Mac, é o Logic.ly. Esse simulador dispõe de uma grande variedade de componentes para a montagem — incluindo flip-flops — e apresenta elementos visuais bastante lúdicos. Ele conta com uma dinâmica de conexão através de nós, similar à adotada por (SANTOS, 2023), e conta também com o recurso de salvar os circuitos montados como arquivos, para carregamento posterior.

Além destes, foi avaliado o simulador PSIM, um simulador comercial que, embora não seja focado exclusivamente no trabalho com circuitos digitais, dispõe de recursos para esse tipo de simulação. A análise concentrou-se na forma como são realizadas as interações da interface, especialmente em como os elementos se conectam e na utilização da área de trabalho, que permite a adição de componentes alinhados a uma grade de referência.

No que tange à interface, não apenas simuladores foram incluídos na análise, mas também duas aplicações web populares na categoria de quadros colaborativos: Miro e FigJam. Essas ferramentas permitem que múltiplos usuários acessem um quadro compartilhado e adicionem conteúdos como post-its, anotações e gráficos, sendo amplamente utilizadas em processos de ideação e organização de projetos. A principal funcionalidade que serviu de inspiração na implementação do simulador proposto, foi a concepção do quadro “infinito”, que pode ser arrastado, ampliado ou reduzido para facilitar a adição e visualização do conteúdo.

Para o desenvolvimento do simulador SimClick, optou-se pelo desenvolvimento de um programa desktop, aproximando-se um pouco mais do fluxo de trabalho típico em muitos simuladores de mercado. Essa abordagem favorece, também, o uso do recurso inovador de conexão com dispositivos físicos, já que os dispositivos podem ser conectados a mesma rede que o computador. Contudo, ainda visando aproveitar as vantagens da versatilidade proporcionada por interfaces desenvolvidas com tecnologias web, adotou-se uma abordagem híbrida, que permite a aplicação dessas tecnologias no contexto de uma aplicação desktop.

Capítulo 4

Desenvolvimento do Software

4.1 Tecnologias Utilizadas

O programa foi pensado como uma aplicação voltada para desktop, sendo essa uma abordagem comum para simuladores. Ainda assim, desejou-se poder utilizar os ricos recursos disponíveis para o desenvolvimento de interfaces para web, como o acesso a frameworks JavaScript para o trabalho com interfaces altamente dinâmicas.

Um fator importante também na escolha de tecnologias foi poder trabalhar com uma separação lógica entre o código responsável por realizar a modelagem e cálculo dos estados do circuito, e entre o código responsável pela representação da GUI.

Tendo em conta esses pontos, optou-se por trabalhar com uma combinação da linguagem Go e de uma camada de interface web baseada no framework Svelte para JavaScript. Essa combinação foi possibilitada através do framework Wails para para desenvolvimento de aplicações desktop multiplataformas (Windows, Mac e Linux).

A fim de manter uma tipagem forte no código responsável pela interação da interface gráfica, assim como foi feito no código que interage com o sistema devido ao uso da linguagem Go, optou-se por utilizar a linguagem TypeScript em conjunto com o framework Svelte.

Para a construção da interface, além do framework Svelte, também foi feito o uso da biblioteca Pixi.js devido a alta quantidade de elementos gráficos que precisam ser exibidos para a representação dos circuitos simulados. A escolha dessa biblioteca se deu por permitir uma utilização facilitada da tecnologia de renderização de gráficos 2D com WebGL.

Também foi desenvolvido, de forma desacoplada, um módulo de comunicação modelo para representar a conexão de placas de desenvolvimento. Foi escolhido, em primeiro momento, fazer-se uma integração com placas Raspberry Pi. Assim, dada a disponibilidade de bibliotecas e facilidade de acesso às portas de Entrada/Saída de Uso Genérico (inglês: *General Purpose Input/Output*) (GPIO) da placa, utilizou-se a

linguagem Python para o desenvolvimento desse módulo.

O software na placa Raspberry Pi é executado sobre a distribuição padrão do sistema operacional Linux oferecida para a placa, chamada de Raspberry Pi OS.

Nas próximas seções apresentamos brevemente cada uma das tecnologias empregadas e discutimos mais detalhadamente sobre a arquitetura do simulador, limitações, e o módulo de comunicação desenvolvido em conjunto com o hardware utilizado.

4.1.1 Tecnologias Web: HTML, CSS e JavaScript

Sites da web, em suas partes que são processadas pelo navegador de internet dos usuários, e que chamamos de front-end, utilizam principalmente três tecnologias para compor suas interfaces, conteúdo, elementos visíveis e invisíveis e adicionar interatividade a essas páginas. Essas três tecnologias são as linguagens HTML, CSS e JavaScript.

A Linguagem de Marcação de HiperTexto (inglês: *HypeText Markup Language*) (HTML) é a responsável pela estruturação do conteúdo de páginas web. Ela é essencial para realizar a marcação de quais elementos estarão presentes em uma página (texto, vídeos, áudios, links, etc) e sua hierarquia dentro da estrutura de conteúdos. A identificação apropriada de cada elemento, conforme sua categoria é importante, não apenas para a exibição desses conteúdos, mas também para que possam ser acessados adequadamente por meio de ferramentas de acessibilidade que são utilizadas por diversos usuários conforme suas necessidades. A referência a “HiperTexto” se deve a natureza da forma que a web é organizada, onde um documento (página) pode referenciar outros através de hiperlinks (chamados comumente de links).

A segunda tecnologia, CSS — Folhas de Estilo em Cascata (inglês: *Cascading Style Sheets*) (CSS) —, é uma linguagem de estilização de conteúdo. Ela é utilizada para dar a forma visual desejada para os conteúdos marcados através do HTML na página, e cumpre esse papel por meio da alteração de propriedades como tamanho, cor e posição dos elementos. O termo “cascata”, utilizado no nome, é devido a forma como a aplicação de estilos através do CSS respeita a hierarquia dos elementos descrita no HTML. Sendo assim, elementos mais internos recebem estilos dados a elementos mais externos, seus ancestrais, isso facilita a definição de blocos e seções na página em que estilos sejam comuns.

Por fim, JavaScript (JS), o terceiro elemento da tríade, é uma linguagem de programação de propósito geral, mas, que no contexto dos navegadores de internet, onde foi concebida inicialmente, assume como papel principal adicionar interatividade aos elementos descritos através do HTML e estilizados via CSS, alterando suas propriedades em resposta a eventos. Com JS, elementos podem ser adicionados ou removidos das páginas, bem como ter suas formas de exibição alteradas, tudo em tempo de execução. Outro ponto, também importante para a interatividade das páginas, é permitir que dados sejam enviados ou retornados de outros sistemas de forma dinâmica.

JavaScript foi introduzido em um navegador pela primeira vez 1995, através do Netscape. Diversos novos recursos foram propostos e implementadas pelos navegadores ao longo dos anos, tendo sido padronizada pela organização Ecma International em 1997, sob o nome de ECMAScript. A partir de 2015, com o lançamento da versão ES2015 do padrão, também chamado de ES6, que trouxe uma quantidade considerável de novos recursos e acréscimos na sintaxe, a linguagem passou a ter novas versões da especificação publicadas anualmente.

4.1.2 Svelte

Svelte é um framework voltado para a construção de interfaces web reativas. É uma tecnologia que permite a construção de páginas a partir da escrita de componentes que combinam HTML, CSS e JavaScript com alguns outros recursos de sintaxe. Algumas das vantagens do desenvolvimento através de componentes é poder reutilizar funcionalidades e partes da interface, e também contar com uma gestão mais declarativa dos dados que são exibidos na aplicação, diferente da abordagem tradicional com JS onde é necessário escrever explicitamente os procedimentos de atualização dos dados em cada local.

Após o desenvolvimento da aplicação, o código é passado por um compilador que transforma os componentes em código JS, que manipulará a página de forma dinâmica durante a execução. Esse é um diferencial do Svelte em relação a outros frameworks e bibliotecas voltados para a construção de interfaces reativas, como Angular ou React. Devido a essa etapa de compilação, não é necessário que haja a existência de um ambiente de execução do framework quando a página é carregada pelo navegador, o que torna o JavaScript gerado mais simples e menos exigente de recursos da

máquina do usuário.

4.1.3 TypeScript

A linguagem TypeScript (TS) é definida como um *superset* da linguagem JavaScript, isto é, uma linguagem compatível com os recursos originais, mas que os amplia. A principal adição, e de onde deriva o “Type” do nome TypeScript, são os recursos para tipagem do código.

Em JavaScript as variáveis são dinâmicas, isso quer dizer que não há imposição um tipo de dados específico que seja atribuído a cada variável, por isso, o tipo do valor associado a uma variável pode ser alterado durante a execução do código. Já no TypeScript, ao criar uma variável, ela será associada a um tipo específico, que valores posteriormente atribuídos deverão respeitar.

Esse controle da tipagem, embora torne a escrita do código mais rígida, permite que sejam realizadas checagens estáticas para verificar erros que poderiam ocorrer devido a utilização de um tipo incorreto em alguma operação.

Após a escrita do código, este deve ser passado por uma etapa de compilação (nesse caso chamada de transpilação) onde o TS será transformado em JS. Isso permitirá que o código seja executado pelos navegadores de internet, que possuem suporte nativo para a execução do JavaScript.

No projeto do SimClick, o TypeScript foi utilizado em conjunto com o Svelte no desenvolvimento da GUI.

4.1.4 Pixi.js

Pixi.js é uma biblioteca para JavaScript que facilita a renderização de elementos gráficos de forma 2D em páginas web com a tecnologia WebGL. O uso do WebGL permite uma alta performance de renderização gráfica no contexto de páginas web, já que são utilizados recursos para aproveitamento do processamento gráfico do dispositivo.

Interagir diretamente com a API — Interface de Programação de Aplicações (inglês: *Application Programming Interface*) (API) — do WebGL é bastante complexo, envolvendo uma abundância de cálculos e etapas para a renderização de elementos

simples. Utilizando a biblioteca Pixi.js, podemos ter acesso a esses recursos, focando em elementos 2D, através de uma API facilitada.

A biblioteca foi utilizada no contexto do simulador desenvolvido para a construção de um *canvas* (quadro de trabalho), que permite ao usuário a montagem e visualização dos circuitos de forma intuitiva.

4.1.5 Linguagem Go

Go, também conhecida popularmente como Golang, é uma linguagem de programação que começou a ser desenvolvida internamente no Google em 2007, e que foi disponibilizada para o público em 2009. É uma linguagem fortemente tipada, com sintaxe inspirada em C, e que apresenta diversos recursos como um *garbage collector* (recurso que limpa a memória de itens que não estão mais sendo utilizados) e suporte nativo para concorrência.

A linguagem tem sido largamente adotada no mercado para diversas áreas como o desenvolvimento de serviços de rede, linhas de comando e desenvolvimento web. A escolha pela linguagem no projeto se deu em parte pelo seu modelo simplificado de representação de dados, pela facilidade de desenvolvimento, o suporte nativo para a execução de testes, e o suporte para integração de tecnologias de interface web com aplicações desktop, por meio do framework Wails.

4.1.6 Wails

Wails é um framework para o desenvolvimento de aplicações multiplataforma que permite a integração de código escrito em Go com interfaces desenvolvidas em tecnologias web, como JavaScript.

O modelo de aplicações no Wails se baseia em ter uma aplicação com código escrito em Go empacotada juntamente com os recursos estáticos de uma aplicação web que possa ser executada por um navegador. A aplicação invoca o renderizador web nativo do sistema — diferente para cada sistema operacional — para o processamento da GUI. Também é disponibilizada uma biblioteca para interação com menus, caixas de diálogo e eventos, tanto através do Go, como do JavaScript.

A aplicação web que compõe a interface é capaz de se comunicar com a aplicação

Go por meio de um modelo de Chamada de Procedimento Remoto (inglês: *Remote Procedure Call*) (RPC), onde as funções expostas no código em Go são disponibilizadas para chamada direta no ambiente de execução do JavaScript.

4.2 Arquitetura

O programa simulador SimClick foi estruturado como uma aplicação para desktop que opera em conjunto com programas instalados nas placas de desenvolvimento que se deseja conectar à simulação. Os programas desenvolvidos para as placas são chamados de “módulos de comunicação”, e são desenvolvidos especificamente para cada modelo ou família de placas de desenvolvimento suportadas pelo simulador.

O módulo de comunicação adequado deve ser carregado e executado nas placas desejadas. Adicionalmente, a placa e o computador que estiver executando o simulador devem ser conectados à mesma rede local, para permitir a comunicação.

O software simulador é dividido em duas partes principais, que interagem para permitir a simulação dos circuitos pelo usuário. A seguir, essas partes serão denominadas como “módulo back-end” e “módulo front-end”. Elas correspondem, respectivamente à aplicação Go e à aplicação baseada em Svelte, que executam de acordo com o padrão sugerido pelo framework Wails. A comunicação entre esses dois módulos é realizada através de chamadas do módulo front-end para funções no modelo RPC, que executam procedimentos correspondentes no módulo back-end.

O módulo back-end realiza dois papéis na execução do programa, sendo cada um desses dirigido por um submódulo. O primeiro é o módulo de simulação de circuitos, responsável por representar internamente a modelagem do circuito que está sendo simulado e calcular os valores dos sinais para todos os pontos requisitados. O segundo submódulo é um servidor responsável por efetuar a comunicação com os dispositivos de hardware.

O módulo back-end também realiza algumas funções básicas de integração com o sistema operacional da máquina hospedeira, como a abertura e gravação de arquivos, diretamente ou através da biblioteca de tempo de execução disponibilizada pelo Wails.

Quanto ao módulo front-end, esse possui o principal papel de renderizar e executar a lógica referente à GUI que é apresentada para interação, e requisitar os recursos necessários do módulo back-end, conforme o usuário realiza ações no programa.

A base da interface, como menus, caixas de diálogo e navegação entre telas diferentes da aplicação, é coordenada através da estrutura de componentes montada no framework Svelte. Já a renderização do *canvas* para montagem do circuito, visualização e manipulação dos componentes, é feita dentro de um contexto WebGL acessado através da biblioteca Pixi.js.

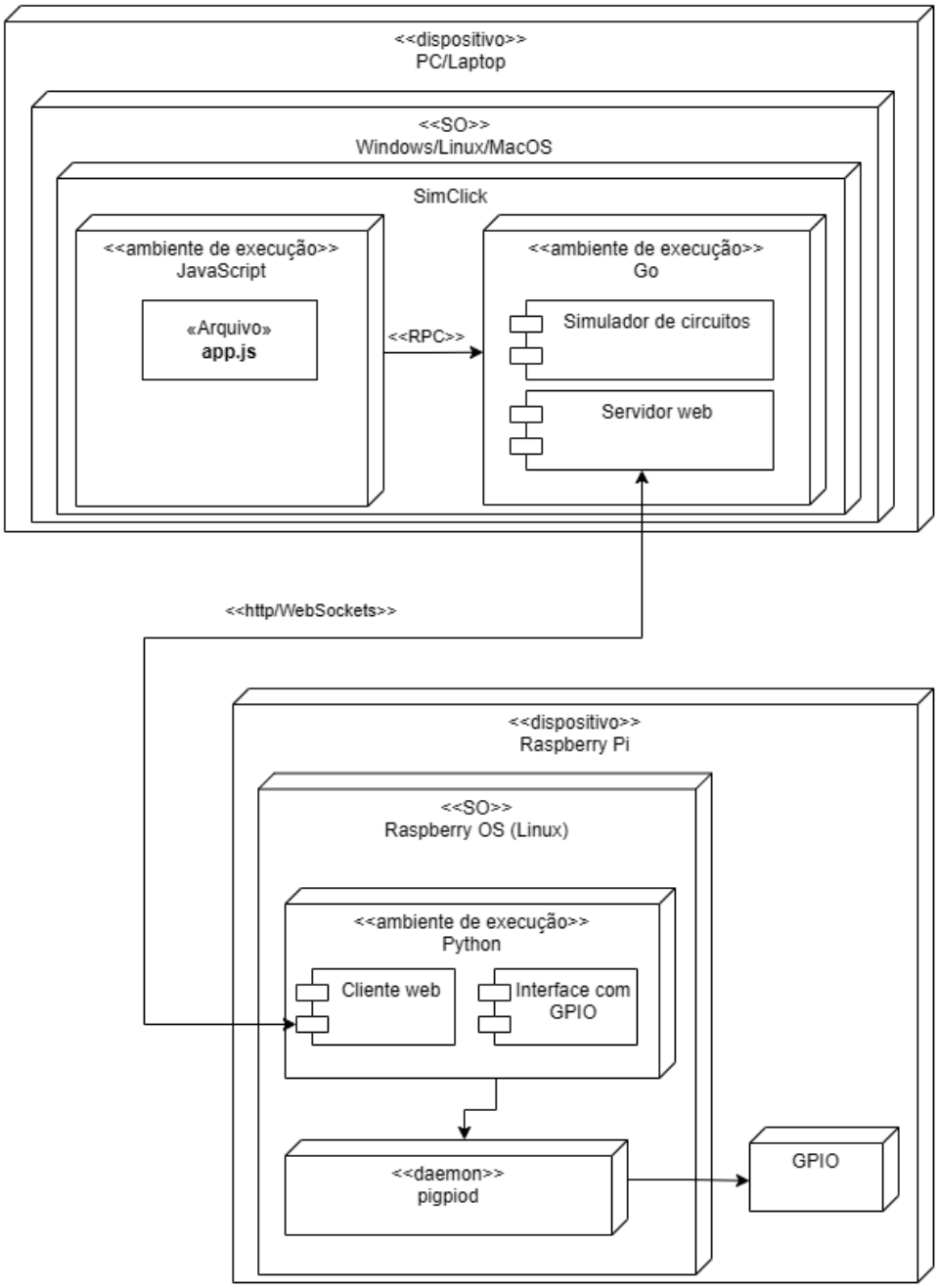


Figura 8: Arquitetura da aplicação SimClick e comunicação com dispositivos.

4.3 Representação de Estados Lógicos

Para representação dos estados lógicos no sistema, além dos estados de nível ALTO e BAIXO, foi definido também um estado que representa uma indeterminação. Esse estado é utilizado sempre que o estado “real” de um nó no circuito não poderia ser definido. Sendo assim, os três estados utilizados no programa foram definidos internamente como *HIGH* (para ALTO), *LOW* (para BAIXO) e *UNKNOWN* (para “desconhecido” ou indeterminado).

Ao serem adicionados ao *canvas*, por não estarem ainda conectados a outros componentes, muitos elementos possuem seus nós definidos como indeterminados. Por isso, estão detalhados a seguir os comportamentos iniciais para os elementos disponíveis no programa.

As *fontes*, ao serem adicionadas ao circuito, tem seu estado inicial e saída definidos como *LOW*. Porém, as *fontes externas*, aquelas que precisam ser associadas a um pino disponível de um dispositivo conectado, ficam em estado *UNKNOWN* enquanto ainda não estiverem associadas. *Fontes de pulsos* (ou *clocks*) também ficam inicialmente em estado *LOW* até que seja definido o período desejado.

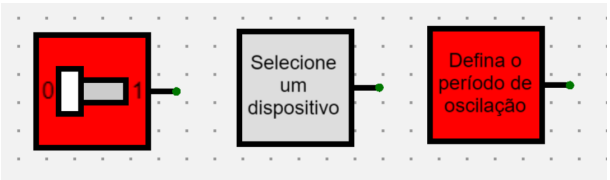


Figura 9: Fontes em seus estados iniciais. Da esquerda para a direita: fonte interna, fonte externa, fonte de pulsos (*clock*).

Todos os *fios*/*nós* possuem sempre o mesmo estado dos terminais de saída aos quais estão conectados, sendo que só podem se conectar a uma saída por vez. Da mesma forma, quando não estão conectados a um terminal de saída de algum componente, seu estado é definido como *UNKNOWN*.

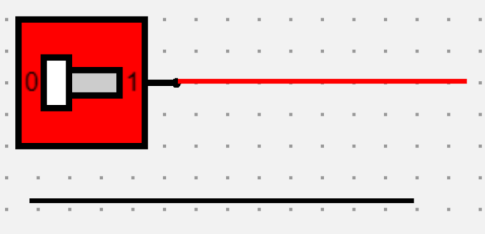


Figura 10: Acima, um nó conectado, assumindo o estado *LOW* da saída da fonte. Abaixo, um nó desconectado, em estado *UNKNOWN*.

Os indicadores de *saídas*, tanto as internas (que existem somente na simulação), quanto as externas (as que podem ser associadas a pinos dos dispositivos conectados), recebem sempre os estados definidos pelos *nós* conectados aos seus terminais de entrada. Caso estejam desconectados, o estado será definido como *UNKNOWN*. No caso das *saídas externas*, enquanto estão no estado *UNKNOWN*, não enviam atualizações de sinal para o dispositivo associado, caso haja.

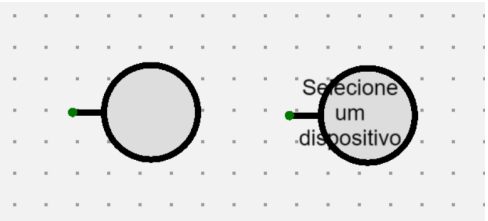


Figura 11: Saídas em estados iniciais. Da esquerda para a direita: saída interna, saída externa.

As portas lógicas tem seus estados no terminal de saída definidos a partir de suas tabelas verdade que são computadas de acordo com a conexão dos terminais de entrada da porta lógica. Devido ao estado *UNKNOWN*, que pode ser fornecido nas conexões de entrada, caso estejam desconectadas ou conectadas a um *nó* que esteja nesse estado, são necessárias algumas adições de estados possíveis às tabelas verdade de cada porta lógica em relação as tabelas que levam em conta apenas sinais ALTO e BAIXO.

Com esses três estados possíveis, as tabelas verdade para as portas lógicas foram definidas como apresentadas a seguir. Nas tabelas aqui listadas, o valor *LOW* está representado por “0”, *HIGH* por “1”, e o estado *UNKNOWN* por “X”.

Porta Lógica NOT

Para a porta lógica NOT (inversora), havendo apenas uma entrada, o sinal de saída só será indefinido quando o sinal de entrada for indefinido.

A	$\overline{A}$
0	1
1	0
X	X

Tabela 7: Tabela verdade usada na implementação da porta lógica NOT.

Porta Lógica OR

Para a porta lógica OR, qualquer entrada que seja definida como *HIGH*, irá fazer com que o sinal de saída seja definido como *HIGH*. Se as duas entradas forem indefinidas, ou, se a entrada conhecida estiver em nível lógico *LOW*, não será possível determinar o nível de saída da porta lógica.

A	B	$A+B$
0	0	0
0	1	1
0	X	X
1	0	1
1	1	1
1	X	1
X	0	X
X	1	1
X	X	X

Tabela 8: Tabela verdade usada na implementação da porta lógica OR.

Porta Lógica AND

Para a porta lógica AND, é necessário que todas as entradas tenham sinal lógico *HIGH* para que a saída seja definida como *HIGH*. Sendo assim, sempre que houver uma entrada definida como *LOW*, necessariamente, a saída deverá ser *LOW*. Por isso, o sinal de saída será indefinido quando todas as entradas forem indefinidas, ou, quando uma entrada for indefinida e a outra entrada estiver em nível lógico *HIGH*.

A	B	$A \cdot B$
0	0	0
0	1	0
0	X	0
1	0	0
1	1	1
1	X	X
X	0	0
X	1	X
X	X	X

Tabela 9: Tabela verdade usada na implementação da porta lógica AND.

Porta Lógica NAND

Sendo ela a inversão da porta AND, para a porta lógica NAND, a única combinação de entradas que levará a uma saída com nível lógico *LOW*, será quando ambas forem *HIGH*. Havendo qualquer entrada definida como *LOW*, ocasionará que a saída seja *LOW*. Os casos adicionais, onde não é possível determinar todas as entradas, e não há a presença definida de um sinal *LOW*, terão a saída desconhecida (*UNKNOWN*).

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
0	X	1
1	0	1
1	1	0
1	X	X
X	0	1
X	1	X
X	X	X

Tabela 10: Tabela verdade usada na implementação da porta lógica NAND.

Porta Lógica NOR

A porta lógica composta NOR funciona como uma associação de uma porta OR seguida de uma NOT, por isso, quando houver qualquer entrada com sinal lógico *HIGH*, a saída correspondente será *LOW*. Nos casos com entradas indefinidas, onde não exista uma entrada *HIGH*, as saídas serão indefinidas. Apenas para ambas as entradas em estado *LOW*, a saída será *HIGH*.

A	B	$\overline{A+B}$
0	0	1
0	1	0
0	X	X
1	0	0
1	1	0
1	X	0
X	0	X
X	1	0
X	X	X

Tabela 11: Tabela verdade usada na implementação da porta lógica NOR.

Porta Lógica XOR

A porta lógica XOR permite apenas sinal lógico *HIGH* na saída quando somente uma das entradas é *HIGH*. Como trabalhamos com portas lógicas de duas entradas, é necessário que para obter estado *HIGH*, as duas entradas tenham estados opostos. Sendo assim, caso qualquer das entradas não esteja definida, o sinal de saída também será indefinido.

A	B	$A \oplus B$
0	0	0
0	1	1
0	X	X
1	0	1
1	1	0
1	X	X
X	0	X
X	1	X
X	X	X

Tabela 12: Tabela verdade usada na implementação da porta lógica XOR.

4.4 Armazenamento de Circuitos

O armazenamento e uso posterior de circuitos criados é possibilitado através do recurso de gravação e carregamento de arquivos, que foi desenvolvido para trabalhar com arquivos customizados utilizando a extensão "*.sck*". Para a representação interna desses arquivos, foi escolhido o formato JSON, com algumas propriedades definidas da forma que será abordada a seguir. O formato JSON — Notação de Objeto do

JavaScript (inglês: *JavaScript Object Notation*) (JSON) — é um formato de texto para a codificação de informações, que é baseado na notação de objetos existente na sintaxe do JS.

A estrutura básica do conteúdo dos arquivos é um objeto contendo as propriedades "*circuit_name*", "*nodes*", "*components*", "*connections*" e "*node_links*". "*circuit_name*" é associado a uma *string* contendo o nome interno do circuito, que é definido pelo usuário. As outras chaves são listas de elementos necessários para o carregamento do circuito.

```
{
   "circuit_name":"Meu circuito",
   "nodes":[],
   "components":[],
   "connections":[],
   "node_links":[]
}
```

A lista armazenada na chave "*nodes*" contém objetos que representam os ramos dos nós do circuito. Cada elemento é descrito por um "*id*", para identificação do ramo, e uma lista de dois pontos associada a chave "*points*". Esses pontos são as marcações de posicionamento do início e fim do ramo no *canvas*.

Cada ponto é descrito por coordenadas x e y, respectivamente referentes ao eixo horizontal e vertical a partir do ponto inicial de referência do quadro. Essas coordenadas aumentam nos sentidos horizontal para a direita e vertical para baixo. A mesma descrição de pontos é utilizada para outros elementos descritos a seguir.

```
{
     "id":"1738031445554--244",
     "points":[
        { "x":532.29, "y":301.59 },
        { "x":666.29, "y":334.39 }
     ]
}
```

A chave “*components*” armazena a listagem de componentes adicionados ao circuito. Cada um é associado a uma identificação única (“*id*”), um tipo (“*type*”), que descreve qual é o componente, e uma posição de referência. Os tipos são mapeados para cada componente que é possível acrescentar ao circuito, conforme a tabela 13.

```
{
    "id":"1738031377656--0",
    "type":"EXTERNAL_SOURCE",
    "position":{ "x":432, "y":262.8 }
}
```

Tipo	Componente
AND	Porta lógica AND
OR	Porta lógica OR
NOT	Porta lógica NOT
SOURCE	Fonte de sinal lógico
OUTPUT	Indicador de saída
EXTERNAL_SOURCE	Fonte de sinal associável a dispositivo externo
EXTERNAL_OUTPUT	Saída associável a dispositivo externo
CLOCK_SOURCE	Fonte de pulsos

Tabela 13: Tipos utilizados para descrição dos componentes.

Além da listagem dos componentes e dos ramos existentes no circuito, também é necessário o armazenamento de quais são as conexões entre os ramos e as entradas e saídas dos componentes. A lista dessas conexões é o que é associada a chave “*connections*”. Cada conexão é representada por um objeto que indica o *id* do ramo (chave “*node*”), o *id* do componente (chave “*component*”), o tipo da conexão (chave “*type*”), isso é, se é para uma entrada (“*INPUT*”) ou saída (“*OUTPUT*”) do componente, e qual o índice numérico (chave “*index*”) referente à entrada ou saída.

```
{
     "node":"1738031445554--244",
     "component":"1738031377656--0",
     "type":"OUTPUT",
     "index":0
}
```

Por fim, a última listagem é a chamada “*node_links*”, em que cada item é uma sublista que representa quais ramos estão conectados entre si no circuito, formando um único nó. Cada nó é listado por seu “*id*”.

```
[ "1738031461500--1050", "1738031445554--244" ]
```

4.5 Comunicação entre Simulador e Hardware

Um ponto muito importante na implementação do programa é a forma como é realizada a conexão e comunicação entre o simulador e as placas de desenvolvimento. Alguns pontos a se considerar são as diferenças entre protocolos e a forma de envio de dados, assim como o impacto em fatores como latência e a facilidade de implementação para novos dispositivos.

A seguir abordaremos mais sobre a escolha do protocolo WebSockets como base para a comunicação entre os dispositivos, e entraremos em detalhes sobre a padronização que foi elaborada para a troca de mensagens.

4.5.1 Avaliação de Protocolos

O primeiro ponto de avaliação para a implementação do recurso de comunicação foi a definição de qual protocolo de comunicação seria utilizado. Partimos do pressuposto de que seria utilizada uma comunicação por Rede de Área Local (inglês: *Local Area Network*) (LAN), devido a uma rede desse tipo ser bastante acessível no contexto de casas e salas de aula, podendo ser providenciada através de redes cabeadas ou Wi-Fi, e, nessa última forma, sendo possível até mesmo configurar uma rede local através do uso de um celular como roteador.

Para a escolha do protocolo, consideramos o desempenho demonstrado em pesquisas de diversos protocolos de camada de aplicação comumente utilizados em aplicações de Internet das Coisas (inglês: *Internet of Things*) (IoT). Focamos no segmento de IoT por haver grande quantidade de material lidando de forma específica com a comunicação entre dispositivos como microcontroladores.

Um dos requisitos considerados como importantes para a escolha foi a possibilidade da realização de comunicação bidirecional, já que isso permitiria que fossem enviadas tanto atualizações de sinais para saídas nas placas quanto o recebimento

de novas leituras, sem a necessidade de recorrer a uma técnica de *polling*, que é quando se realizam requisições de forma periódica para buscar por atualizações.

Outro requisito, foi não necessitar da utilização de um *broker* como intermediário para as mensagens. Devido ao programa ser projetado para instalação local em uma máquina, e para adição simplificada de novas placas para conexão, considerou-se que a adição de um *broker* tornaria a distribuição e configuração mais complexas, dificultando a adoção do programa por usuários mais leigos.

Tomando os pontos descritos como base, e a análise feita por (GUPTA; M, 2021), que compara os protocolos em camada de aplicação CoAP, MQTT, WebSockets, XMPP e AMPQ, foi definido que WebSockets seria o mais apropriado para a aplicação desejada.

O protocolo WebSockets foi pensado inicialmente para aplicação na web em páginas que requerem manter um fluxo constante de atualização de dados. Uma comunicação via WebSockets é iniciada através de uma conexão HTTP tradicional, porém, ao receber a requisição, o servidor solicita que seja feito um *upgrade* na conexão. Dessa forma, a requisição é mantida em aberto e se estabelece um canal bidirecional de comunicação. Esse fluxo é demonstrado na figura 12, e, uma apresentação mais extensiva dos detalhes do protocolo pode ser encontrada no documento de padronização (RICE, 2025).

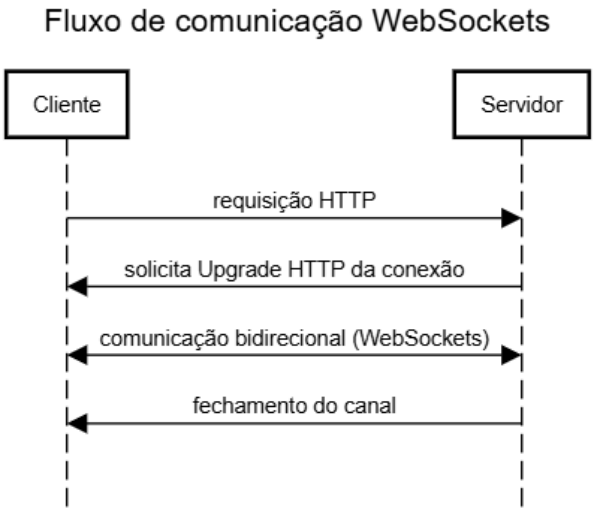


Figura 12: Fluxo de comunicação do protocolo WebSockets.

4.5.2 Padronização de Mensagens

Sendo um dos objetivos do projeto obter facilidade de extensão com relação aos dispositivos integráveis ao simulador, é importante que todos os aspectos da forma de conexão sejam padronizados. Por isso, além do estabelecimento de WebSockets como protocolo comum, também deve ser estabelecido o formato padrão através do qual as mensagens devem trafegar, tanto no sentido do simulador para o dispositivo, quanto no sentido inverso.

Sendo assim, foi estabelecido um conjunto de comandos que o simulador pode enviar para o dispositivo e comandos que o dispositivo pode enviar ao simulador, seja como resposta (assíncrona) ou como informações. Essas interações serão descritas a seguir.

Logo após o estabelecimento da conexão, o dispositivo deve enviar informações sobre si mesmo que serão utilizadas pelo simulador como parâmetros para a presente interação. O envio dessas informações deve ser feita pelo dispositivo através do comando que foi definido como *META*. Esse comando permite associar um valor a uma chave através do seguinte formato.

```
META chave:valor
```

Os dados utilizados, e que o dispositivo deve enviar, são os correspondentes as chaves *Name* e *GPIO*. Para *Name*, o valor deve ser o nome pelo qual o dispositivo deverá ser identificado. *GPIO* deve ser associado a uma lista de números inteiros delimitados por vírgula, que correspondem a quais pinos estão disponíveis na placa para utilização como entrada ou saída pelo simulador.

No exemplo a seguir o dispositivo define sua identificação como *RPi* e informa que os pinos de 1 a 5 estão disponíveis para uso como entradas ou saídas.

```
META Name:RPi
META GPIO:1,2,3,4,5
```

Do lado do simulador, para interagir com um pino específico do dispositivo, conhecendo já os que estão disponíveis, deve-se enviar uma mensagem inicial contendo o comando *MODE* para definir o modo de operação do pino desejado — entrada (*IN*)

ou saída (*OUT*). O exemplo a seguir define o pino 2 como entrada e os pinos 4 e 5 como saídas.

```
MODE 2 IN
MODE 4 OUT
MODE 5 OUT
```

Para um pino que foi definido como entrada, uma leitura pode ser solicitada pelo simulador através do comando *READ*. A resposta deve ocorrer de forma assíncrona, através de uma mensagem enviada pelo dispositivo, que informará o pino que foi lido e o valor correspondente — ALTO (*HIGH*) ou BAIXO (*LOW*). O exemplo a seguir indica uma solicitação e resposta para leitura do pino 2.

Simulador:

```
READ 2
```

Dispositivo:

```
READING 2:HIGH
```

Para um pino que tenha sido definido como saída, um valor de sinal ALTO (*HIGH*) ou BAIXO (*LOW*) pode ser definido através do comando *WRITE*. No exemplo a seguir, o valor ALTO é associado ao pino 4 e o valor BAIXO ao pino 5.

```
WRITE 4 HIGH
WRITE 5 LOW
```

4.6 Módulo de Comunicação para Raspberry Pi

O simulador SimClick foi desenvolvido com a ideia de ser facilmente integrável a placas de desenvolvimento, para permitir a extensão dos circuitos simulados através de circuitos físicos conectados a essas placas. Para a conexão de uma nova placa com o programa, é necessário que o hardware suporte o programa a partir de um módulo de comunicação desenvolvido especificamente para esse dispositivo, a fim de que a comunicação ocorra da forma que foi padronizada.

A título de modelo para futuras implementações, e também como uma forma de avaliar as funcionalidades de integração do programa com um dispositivo real, o primeiro dispositivo escolhido foi o Raspberry Pi.

Raspberry Pi é não apenas um dispositivo, mas uma família de microcomputadores montados em placas de circuitos, geralmente com um tamanho próximo ao de um cartão de crédito (o tamanho varia de acordo com o modelo). São dispositivos que tornam bastante acessível a prototipação de alguns circuitos ou a montagem de projetos educacionais. Se tratando de microcomputadores, a família de dispositivos possui suporte a instalação de sistemas operacionais Linux adaptados para o hardware.

No presente projeto foram utilizados dois modelos do Raspberry Pi para teste do módulo de comunicação desenvolvido.

O primeiro dispositivo foi um Raspberry Pi B+, que é equipado com $512MB$ de memória RAM, um processador BCM2835 com arquitetura ARM de 32bits, e uma GPIO com um conjunto de 40 pinos. Esse modelo possui entrada para conexão de rede via cabo Ethernet apenas, por isso, foi acoplado um *dongle* adaptador a uma de suas portas USB, para permitir que se conecte em rede via Wi-Fi.

O segundo modelo utilizado foi um Raspberry Pi 4 B, com $1GB$ de memória RAM, processador BCM2711 com arquiterura ARM de 64 bits, e uma GPIO também com 40 pinos. Esse modelo, além da entrada para cabo Ethernet, também apresenta um receptor para Wi-Fi interno.

Dessa forma, foi desenvolvido um módulo de comunicação para a utilização dos dispositivos Raspberry Pi. A implementação foi montada em compatibilidade com o modelo de comunicação que foi detalhado anteriormente no documento.

A primeira etapa, anterior ao desenvolvimento e aplicação do software nos dispositivos, foi a configuração da plataforma. Os dispositivos foram carregados, através dos cartões SD, com o sistema operacional oficial mais recente disponibilizado pela fabricante em sua página, chamado Raspberry Pi OS. Esse sistema é uma distribuição Linux customizada e baseada do Debian 12, no momento com kernel 6.6. No modelo 4B, optou-se pela versão de 64 bits do sistema e com “desktop” (essa é a versão com ambiente gráfico de trabalho). Já para o modelo B+, o sistema foi carregado na versão 32 bits “Lite” (sem ambiente gráfico). Optou-se por tal por motivo de otimização, dada a velocidade reduzida do dispositivo. Os dispositivos foram então configurados para acesso automático a uma rede Wi-Fi, que permitiu o acesso facilitado através do computador para carregar o módulo desenvolvido.

Conforme discutido anteriormente, o módulo desenvolvido para um dispositivo tem

os papéis de se comunicar com o simulador via WebSockets e de repassar os comandos para os pinos da GPIO. O módulo foi implementado em Python, utilizando a versão 3.11, sendo essa a já disponibilizada pelo sistema na versão utilizada.

A comunicação com a GPIO foi facilitada através do uso da biblioteca *pigpio*. Para o uso dessa biblioteca, é necessário que um *daemon* (programa com execução de fundo no sistema) chamado *pigpiod* esteja em execução, já que ele funciona como um intermediário para o acesso a GPIO.

Para a criação do cliente de conexão com o servidor WebSockets, foi escolhida a biblioteca *websockets*, homônima ao protocolo, em sua versão 13.1.

Ao realizar a conexão, as mensagens informacionais (*META*) sobre o nome do dispositivo e pinos disponíveis para uso são enviadas ao servidor. O nome que é enviado para o dispositivo é baseado no nome do usuário configurado para o sistema operacional e no modelo do dispositivo.

4.7 Limitações

A implementação atual do simulador apresenta algumas limitações quanto aos recursos e com relação a comunicação que realiza com os dispositivos. Parte das limitações existentes foram consideradas no projeto inicial, considerando-se que, para os objetivos atuais de utilização, não apresentam grande impacto. Outras limitações surgiram como parte da complexidade de desenvolvimento e, nesses casos, optou-se por adotar soluções mais simples para o momento, e planejar como pontos de melhorias futuras. A seguir, apresentamos uma discussão sobre algumas dessas limitações.

4.7.1 Funcionalidades dos Dispositivos

Um dos primeiros pontos a serem considerados ao se propor a comunicação do programa simulador com placas de desenvolvimento são as funcionalidades nativas presentes nas placas. A tomar como exemplo as placas Raspberry Pi, que foram utilizadas, além do uso padrão de cada pino da GPIO como entrada ou saída simples, também são oferecidos recursos de hardware para a utilização de diversos protocolos de comunicação em pinos específicos, como SPI, I2C e UART — Interface Serial de Periféricos (inglês: *Serial Peripheral Interface*) (SPI); Circuito InterIntegrado (in-

glês: *Inter-Integrated Circuit*) (I2C); Receptor / Transmissor Assíncrono Universal (inglês: *Universal Asynchronous Receiver / Transmitter*) (UART). Em placas de hardware como essas, é comum que diversos pinos tenham funções especiais que podem ser ativadas, como por exemplo, o uso de Modulação por Largura de Pulsos (inglês: *Pulse Width Modulation*) (PWM).

Esse tipo de funcionalidade é extremamente variável, com cada modelo de placa sendo diferente no que disponibiliza. Por sua vez, oferecer suporte para cada funcionalidade dessas através da interação com o simulador demandaria uma adaptação tanto das interfaces de interação com usuário, e de comunicação com os dispositivos, como também requereria uma complexidade bem grande na implementação de cada módulo de comunicação. Consideradas essas questões, e avaliando o que seria o essencial para atender aos objetivos propostos para o programa, optamos por trabalhar nesse momento apenas com a funcionalidade básica de utilização dos pinos como entradas e saídas simples, algo que pode mais facilmente ser abstraído para diferentes plataformas de hardware.

4.7.2 Latência de Comunicação

Há uma grande vantagem no uso de uma rede LAN para a comunicação entre os dispositivos com relação ao fator de acesso e praticidade, grandes impulsionadores no contexto educacional. Em contrapartida, a utilização de uma rede desse tipo apresenta também alguns obstáculos, particularmente se considerarmos conexões não cabeadas através de Wi-Fi.

Muitas aplicações eletrônicas atuais, também em ambientes profissionais, fazem uso de dispositivos que trabalham em altas frequências. Nesses cenários, e também em aplicações de equipamentos que trabalham com o conceito de tempo real, é necessário uma alta precisão na contagem dos períodos de transição entre estados nos circuitos.

Quando lidamos com redes LAN, e utilizamos protocolos como o Protocolo de Controle de Transmissão (inglês: *Transmission Control Protocol*) (TCP), base para o WebSockets, que foi escolhido para comunicação, não temos a facilidade de contar com tempos extremamente definidos na transmissão de pacotes. Há sempre uma variação possível nos tempos de transmissão, que podem ser afetados pelo meio de

transmissão (cabos ou ar), distância, interferências que gerem perdas de pacotes, e diversos outros fatores. Assim, um dos elementos a se considerar, é essa dificuldade de trabalhar com tempos precisos em meio a latência de comunicação.

Posto que os propósitos do programa desenvolvido são inicialmente focados na ampliação da interatividade com os circuitos, e facilitar o aprendizado por meio de uma visualização mais acessível, decidiu-se que seria mais viável trabalhar apenas com baixas frequências.

Sendo assim, a taxa de atualização visual dos estados lógicos dos componentes foi limitada a uma velocidade máxima de $10Hz$, variável. As fontes de pulsos (*clocks*) foram limitadas à frequência máxima de $5Hz$. E também, a frequência dos sinais lidos e enviados para os dispositivos foi limitada a uma taxa de $20Hz$. Essa última imposta também para evitar o congestionamento dos canais de comunicação.

4.7.3 Validação de Circuitos Carregados

Sendo o simulador também proposto como um Produto Mínimo Viável (inglês: *Minimum Viable Product*) (MVP), algumas validações referentes a casos específicos de execução não foram desenvolvidas. Um exemplo é com relação aos arquivos carregados de circuitos salvos anteriormente.

É necessário que a estrutura básica dos arquivos, bem como sua formatação conforme apresentada anteriormente, seja cumprida para que a leitura dos mesmos possa ser feita pelo programa simulador. Apesar disso, uma validação de consistência mais interna dos dados dispostos no arquivo não é realizada. Isso não tende a causar problemas com arquivos que tenham sido produzidos diretamente através da função de salvamento do programa, porém, um arquivo que seja alterado por um agente externo é passível de levar ao carregamento de um circuito em um estado inconsistente, como, por exemplo, indicar a conexão entre elementos que não estejam visualmente ligados.

Como a ocorrência de problemas referentes a essa questão é dependente de interferência externa no programa, e que o mesmo é executado totalmente de forma local pelo usuário, não se considerou esse como um problema de alto impacto pelo momento, mas há de se considerar uma validação mais completa em versões futuras.

Capítulo 5

Apresentação do Programa

Nesta seção apresentamos o programa, com sua interface, funcionalidades e principais interações possíveis.

5.1 Tela Inicial

Ao iniciar o programa SimClick, o usuário é apresentado a uma tela com a logo do programa e botões para duas opções: “Novo Circuito”, para iniciar a montagem de um circuito do zero; e “Abrir circuito”, para escolher e carregar um circuito já montado a partir de um arquivo salvo anteriormente.

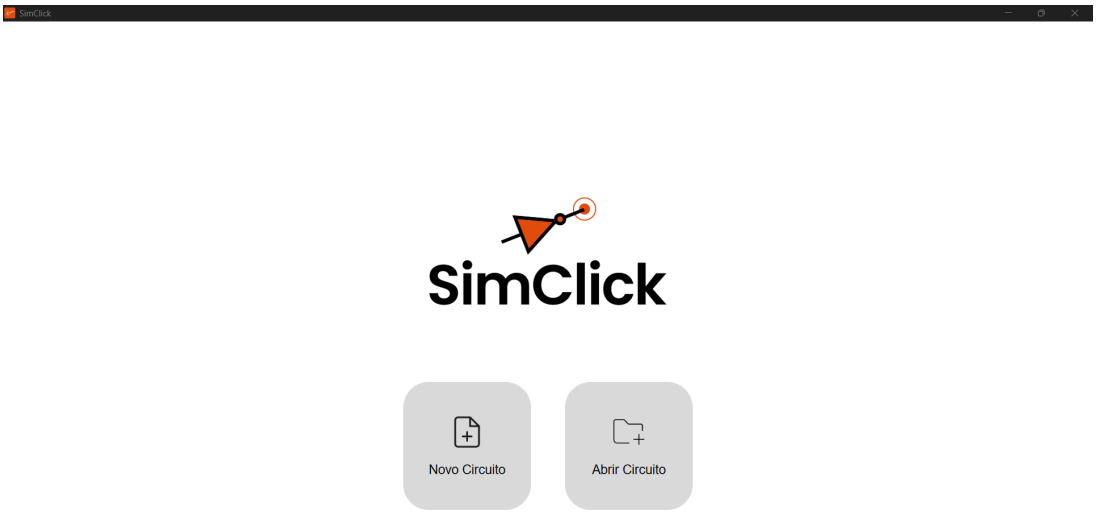


Figura 13: SimClick: Tela inicial do programa.

Escolhendo por iniciar um novo circuito, será apresentada diretamente a tela de montagem de circuitos, onde o quadro estará vazio. Se optar por abrir um circuito já existente, uma janela para seleção do arquivo será apresentada, como na figura 14. Após a seleção e confirmação do arquivo a ser carregado, a tela de montagem de circuitos será exibida com o circuito escolhido já carregado no quadro.

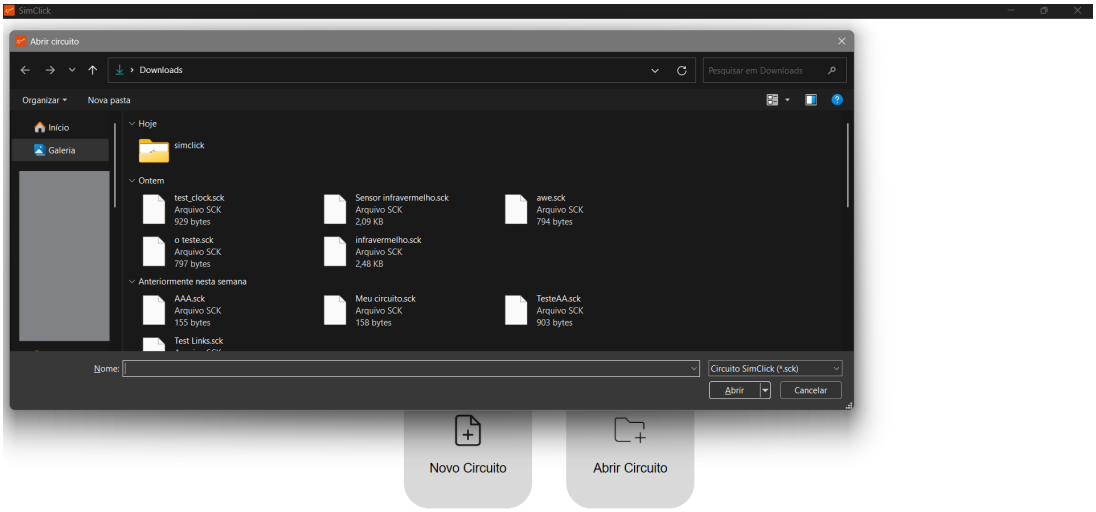


Figura 14: SimClick: Escolha de arquivo de circuito para carregamento.

5.2 Montagem e Edição de Circuitos

A tela de montagem de circuitos é a área onde ocorrerão a maior parte das interações do usuário. Ela é composta principalmente de um quadro movimentável onde o usuário poderá colocar os componentes que deseja adicionar ao circuito. Essa tela também é composta de diversos menus e indicadores que dão acesso a várias funcionalidades.

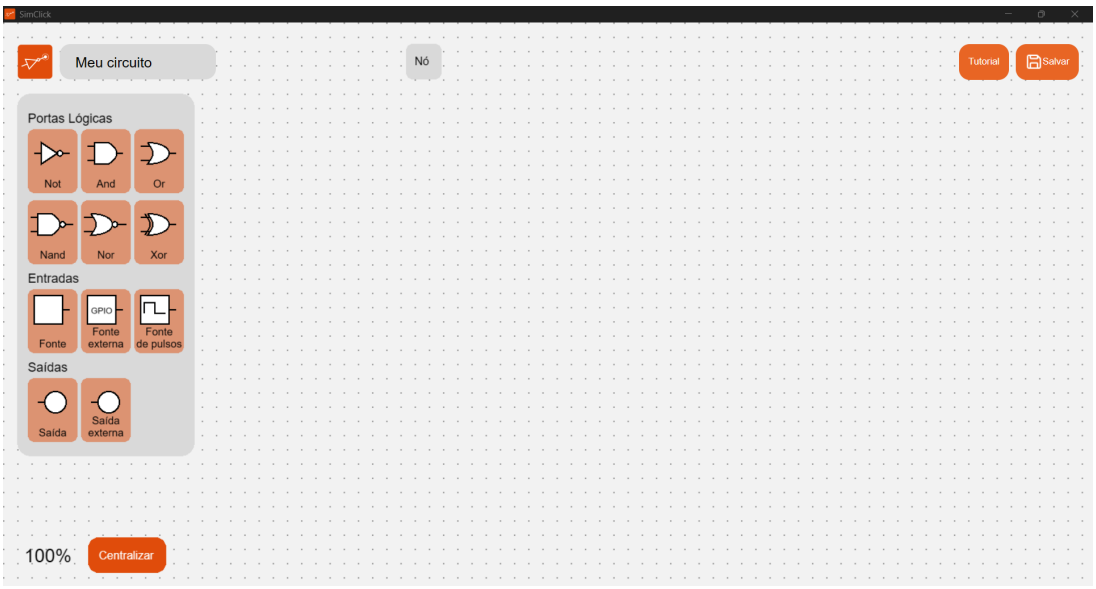


Figura 15: SimClick: Tela de montagem de circuito.

No canto superior esquerdo da tela é exibido um botão com a logo do programa

que, ao ser clicada, oferece uma forma rápida para o usuário retornar para a tela inicial.

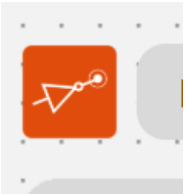


Figura 16: SimClick: Botão para retorno à tela inicial.

Ao lado direito do botão de retorno, há uma caixa de texto que exibe o nome atribuído ao circuito. O usuário é livre para modificar esse nome clicando na caixa de texto e digitando o nome desejado, esse nome não precisa ser o mesmo que o nome do arquivo que será salvo. Essa caixa também possui o papel de indicar visualmente se o circuito foi alterado de alguma forma e essas mudanças ainda não foram salvas em um arquivo. Nesse caso, haverá um asterisco (*) ao final do texto exibido na caixa.

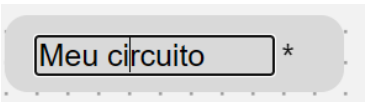


Figura 17: SimClick: Caixa de alteração de nome de circuito.

Ainda do lado esquerdo da interface, abaixo do nome do circuito, há a paleta de componentes. Essa é uma caixa que lista todos os componentes possíveis de serem adicionados ao circuito simulado, separados por categorias: “Portas Lógicas”, “Entradas” e “Saídas”. Para adicionar qualquer desses componentes ao circuito, o usuário deve posicionar o cursor sobre o componente desejado na paleta, clicar com o botão esquerdo do mouse e manter pressionado enquanto arrasta o cursor para o local do quadro em que deseja posicionar o componente. Ao soltar o botão, um novo componente do tipo escolhido será adicionado ao local indicado.

Figura 18: SimClick: Paleta de componentes.

A parte central da interface é o que chamamos de quadro ou *canvas*. Essa área permite o posicionamento dos componentes para a montagem dos circuitos simulados. O quadro oferece bastante flexibilidade para a interação do usuário com o circuito, podendo ser ampliado ou reduzido, e arrastado em qualquer direção. O fundo do quadro é branco, porém ele também é composto de uma grade formada por pontos cinza espaçados igualmente no sentido vertical e horizontal. Essa grade existe para facilitar a visualização de quando o quadro é deslocado.

Para deslocar a visualização do quadro na direção desejada, o usuário deve posicionar o cursor sobre o quadro e segurar o botão direito do mouse enquanto arrasta na direção que deseja. O nível de ampliação da visualização do quadro pode ser alterado segurando a tecla *Ctrl* e girando a roda de *scroll* do mouse, sendo o sentido para cima correspondente à ampliação, e o sentido oposto à redução. O nível de ampliação atual pode ser visualizado na parte inferior esquerda da interface.

Caso o usuário deseje voltar a posição inicial de visualização do quadro, pode pressionar o botão “Centralizar” que fica ao lado do indicador do nível de ampliação. Ao clicar, além do ajuste da posição, o valor de ampliação também retorna para 100%.

Figura 19: SimClick: Indicador de nível de ampliação e botão de centralização.

Na parte superior direita da interface há mais dois botões: “Tutorial” e “Salvar”. O botão “Tutorial” abre uma janela modal que exibe uma descrição dos principais

comandos para o usuário. “Salvar” abre uma janela para escolha de diretório para salvamento do circuito como um arquivo, conforme a figura 20. Caso o circuito já tenha sido salvo, ou um arquivo tenha sido carregado, o último nome do arquivo será sugerido para o salvamento. Em caso contrário, o nome do circuito será sugerido como nome de arquivo.

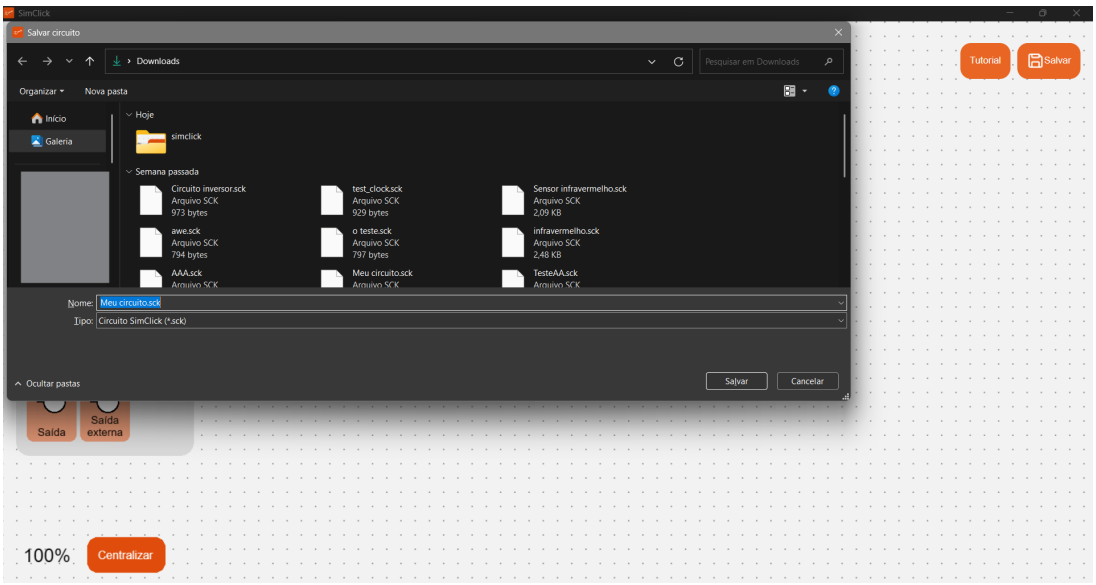
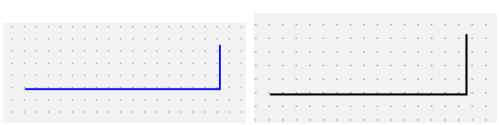


Figura 20: SimClick: Janela de seleção de salvamento de arquivo.

Por fim, o último elemento de interface visível nessa tela é o botão “Nó”, exibido no centro da área superior. Esse botão é utilizado para acionar a ferramenta de desenho de fios.

Para realizar a criação de um ramo, primeiramente clicamos no botão e verificamos que o cursor é alterado para uma cruz. Em seguida, deve-se clicar no local do quadro em que se deseja posicionar o ponto inicial do ramo. Ao deslocar o cursor, uma visualização temporária do ramo que está sendo criado será exibida como na figura 21a. Quando o ramo estiver na posição desejada, clicamos novamente no quadro para que o ponto final seja posicionado. O ramo será criado no quadro a partir dos pontos indicados.



(a) **(b)**

Figura 21: SimClick: Criação de ramo. (a) Visualização temporária. (b) Ramo finalizado.

Para interligar componentes é necessário adicionar fios entre eles. Primeiro, adicionamos os componentes que desejamos ao quadro. Em seguida, ativamos a ferramenta de desenho de ramos e escolhemos como pontos inicial e final os círculos verdes exibidos nos terminais dos componentes. Esses círculos são os pontos de conexão. Após serem conectados a um fio, esses pontos de conexão assumem a cor preta.

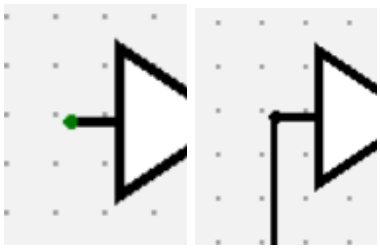


Figura 22: SimClick: Ponto de conexão em componente. (a) Aberto. (b) Conectado.

Também é possível interligar fios para criar nós com mais de um ramo, que conectem vários componentes. Para isso, ao adicionar um novo ramo, seu ponto inicial ou final deve estar sobre o fio já existente ao qual se deseja conectá-lo.

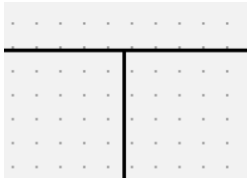


Figura 23: SimClick: Fio com múltiplos ramos interligados.

Além de adicionar elementos ao circuito, também é possível removê-los. Para remover um componente ou nó, primeiro o selecionamos clicando sobre ele com o botão esquerdo do mouse. Um elemento que esteja selecionado irá ficar levemente transparente. Para excluir o elemento, pressionamos a tecla *Del* no teclado.

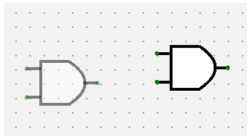


Figura 24: SimClick: Componente selecionado à esquerda e não selecionado à direita.

Outra forma de selecionar elementos é através da criação de uma área de seleção.

Primeiro clicamos com o botão direito em uma área vazia do quadro, em seguida movemos o cursor mantendo o botão pressionado. Irá ser exibida uma caixa azul limitada pelos pontos inicial e atual do cursor. Ao soltar o botão do mouse, todos os elementos que estiverem completamente posicionados dentro da caixa serão selecionados. Clicar sobre uma área vazia do quadro irá cancelar a seleção atual, e pressionar o botão *Del* sempre removerá todos os elementos selecionados.

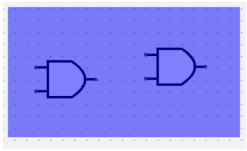


Figura 25: SimClick: Área de seleção.

Um componente pode ser movido, ao clicar sobre ele com o botão esquerdo do mouse e mover o cursor para arrastá-lo para uma nova posição. Após soltar o botão, o componente será reposicionado e, caso esteja conectado a algum fio, essas conexões serão desfeitas.

5.3 Interação com Componentes

A seguir, veremos alguns exemplos de montagem de circuitos passo-a-passo, para ver em mais detalhes as características de componentes específicos que podemos adicionar aos circuitos.

Iniciaremos com a criação de um circuito simples, que recebe um sinal lógico de uma fonte, inverte esse sinal em uma porta NOT, e exibe o sinal alterado em uma saída. Primeiro, começamos com um novo circuito vazio.

Para estruturar o circuito, adicionaremos primeiro ao quadro os componentes que serão utilizados. Arrastamos para o quadro, a partir da paleta de componentes, uma *Fonte*, uma porta *NOT*, e uma *Saída*.

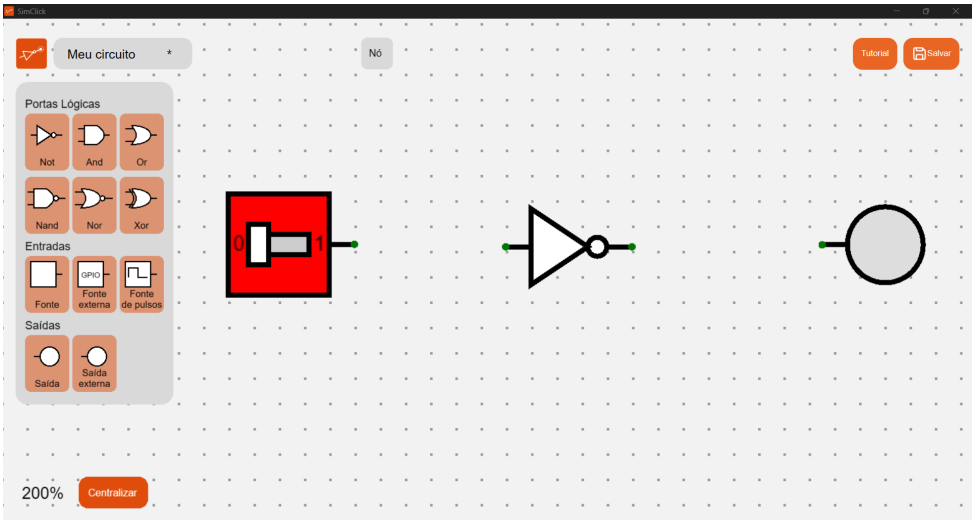
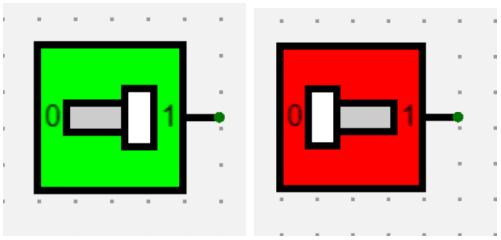


Figura 26: Componentes adicionados ao quadro: Fonte, Porta Lógica NOT, Saída.

Tendo o componentes já posicionados no quadro, podemos observar algumas coisas sobre suas disposições iniciais.

A fonte apresenta um seletor que indica se o sinal fornecido no momento tem o estado lógico alto ou baixo. Como inicialmente tem a marcação do seletor no "0", e sua cor de fundo está em vermelho, isso indica que o sinal inicial é baixo. Podemos alterar o sinal para alto ou vice-versa ao clicar no seletor.



(a) **(b)**

Figura 27: Fonte fornecendo sinal com estados alto (a) e baixo (b).

Também podemos verificar o componente indicador de saída. Enquanto está desconectado, sua cor de fundo é cinza, o que indica que o estado atual é desconhecido.

Para seguir com a montagem, vamos realizar as conexões entre os componentes criando fios. Clicamos no botão "Nó", em seguida, clicamos no ponto de conexão da fonte (em verde) e no ponto de conexão da entrada da porta inversora. Os pontos de conexão irão mudar de cor para preto, o que indica que foram conectados. Também, o fio estará com a mesma cor da fonte, o que indica que o sinal, alto ou baixo está sendo transmitido.

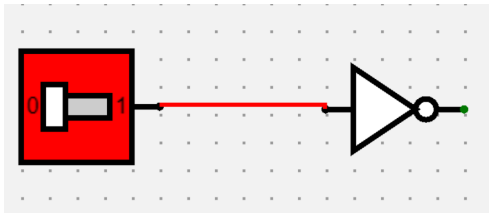
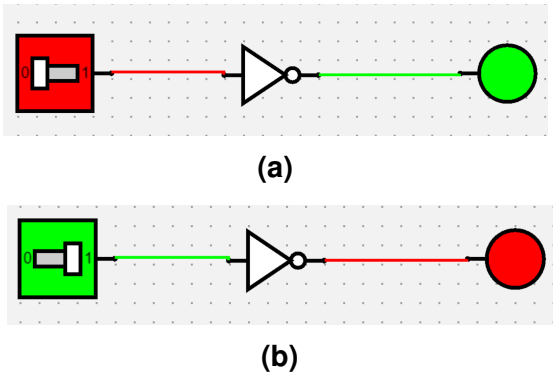


Figura 28: Fio conectando fonte e porta NOT.

Conectamos agora a saída da porta NOT com a entrada do componente indicador de saída através da criação de outro fio, com o mesmo procedimento. Podemos perceber que a cor indicada pelo novo fio e pelo indicador de saída será referente ao sinal lógico oposto àquele que sai da fonte. Isso mostra o funcionamento da lógica da porta inversora. Se alterarmos o sinal fornecido pela fonte através do seletor, iremos verificar que o sinal de saída do circuito é atualizado de acordo com essa entrada.



(a)

(b)

Figura 29: Circuito com componentes conectados. Fonte negativa em (a) e positiva em (b).

Por fim, podemos nomear o circuito, através da caixa de edição de nome. O asterisco ao final da caixa indica que o circuito ainda não foi salvo. Vamos salvar clicando no botão “Salvar” e escolhendo um local e nome de arquivo na janela de seleção.

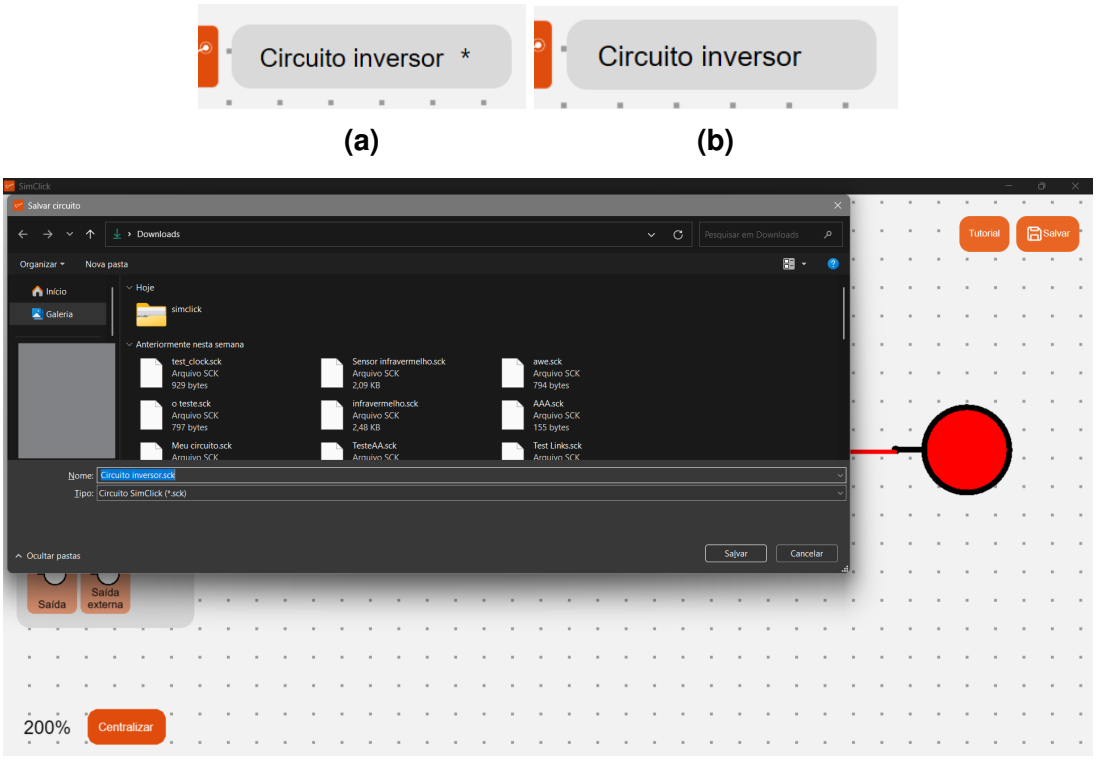


Figura 30: (a) Mostra o indicativo de alterações não salvas. Em (b) não há o indicativo de alterações, é o esperado após realizar salvamento através da janela apresentada em (c).

Para o exemplo seguinte, vamos analisar como adicionar conexão com dispositivos físicos, externos à simulação. Partiremos do circuito montado anteriormente, para isso, vamos utilizar a função de carregamento de circuito a partir da tela inicial do SimClick.

Com o circuito carregado, vamos realizar a troca do componente da Fonte. Vamos clicar para selecionar e em seguida pressionar "*Del*" no teclado para remover o componente. Perceba que ao remover a fonte, os sinais do circuito retornaram ao estado de indefinição.

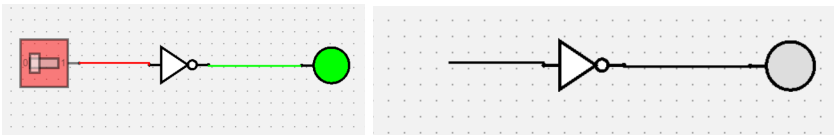


Figura 31: Remoção de fonte do circuito. (a) Fonte selecionada. (b) Fonte removida e estados indeterminados nos componentes.

Em seguida, vamos arrastar, a partir da paleta, o componente Fonte externa. Essa

fonte inicialmente não está associada a nenhum dispositivo, por isso, além do texto “Selecione um dispositivo”, ela terá cor cinza de fundo, indicando que também está indefinida.

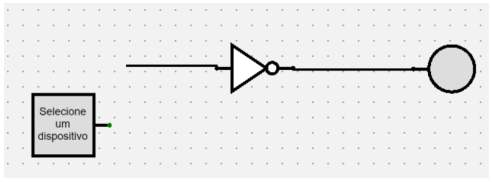


Figura 32: Fonte externa adicionada ao circuito.

Para utilizar essa fonte precisamos realizar a leitura de um sinal a partir de um dispositivo físico. Considerando que temos um dispositivo Raspberry Pi conectado a mesma rede local que nosso computador, e que ele está executando o programa módulo de conexão adequado, o dispositivo se conectará automaticamente ao programa, sendo apenas necessário indicar que queremos utilizar sua GPIO.

É necessário também que haja um circuito montado do qual a placa possa realizar a leitura do sinal. Para isso, considere que temos um circuito com um botão (ou chave) que fecha ou abre a conexão entre um pino com sinal $3.3V$ no Raspberry e o pino 17 da GPIO.

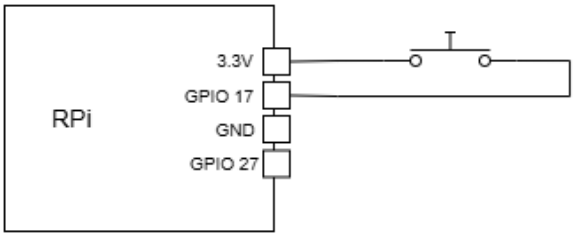


Figura 33: Diagrama de circuito para leitura de botão.

Com o circuito preparado, seguimos para indicar o uso do dispositivo para o simulador. Clicamos duas vezes com o botão esquerdo do mouse no componente Fonte externa. Um modal será aberto com um seletor de dispositivo. Ao escolher o Raspberry na lista, os pinos disponíveis da GPIO são exibidos como botões.

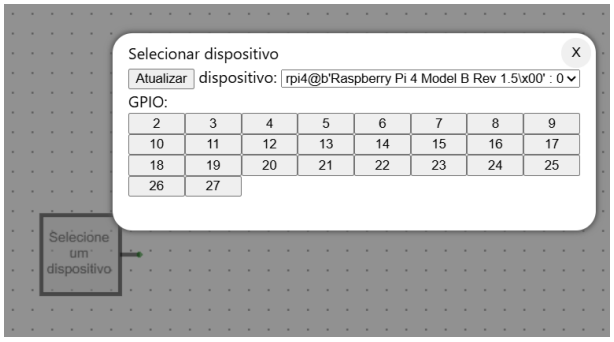
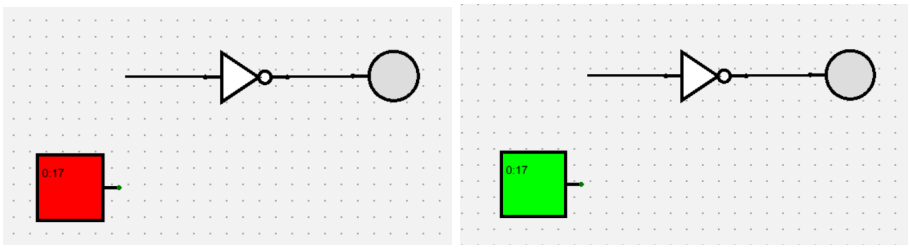


Figura 34: SimClick: Seleção de dispositivo e pino da GPIO.

Escolhendo então o pino 17 verificamos que a fonte começa a fornecer um sinal de acordo com a leitura do pino. Pressionando o botão no circuito montado, verificamos a alteração do sinal da fonte no simulador.



(a) **(b)**

Figura 35: SimClick: Leitura da fonte a partir do dispositivo.
(a) Estado ALTO. (b) Estado BAIXO.

Para completar a conexão com o circuito, apenas precisamos ligar a fonte com o restante do circuito através de um novo fio, interligado com o anterior onde estava a fonte antiga.

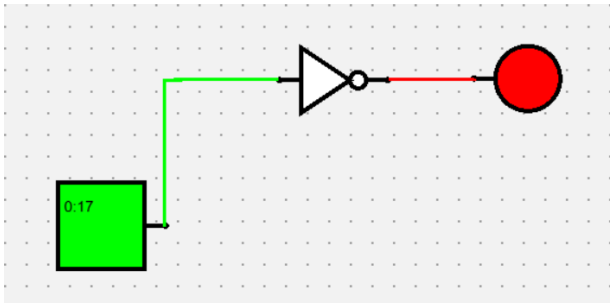


Figura 36: SimClick: Conexão da fonte externa com o circuito.

Agora, vamos adicionar uma saída para o circuito físico. Precisamos ser capazes de visualizar o sinal no circuito montado, por isso vamos atualizá-lo acrescentando um LED — Diodo Emissor de Luz (inglês: *Light-Emitting Diode*) (LED) — em série com

um resistor de proteção ligados ao pino 27 da GPIO, conforme o diagrama da figura 37.

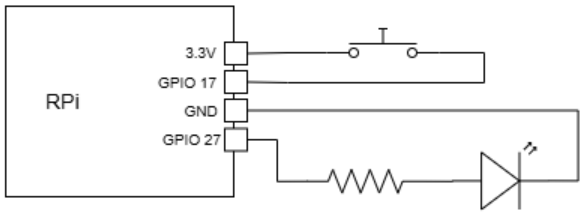


Figura 37: Diagrama de circuito para leitura de botão e saída para LED.

No simulador, arrastamos um componente Saída externa da paleta para o circuito. Não vamos remover a saída que já temos, apenas acrescentar essa nova. Criamos um novo fio conectado ao ramo que vai para a saída anterior.

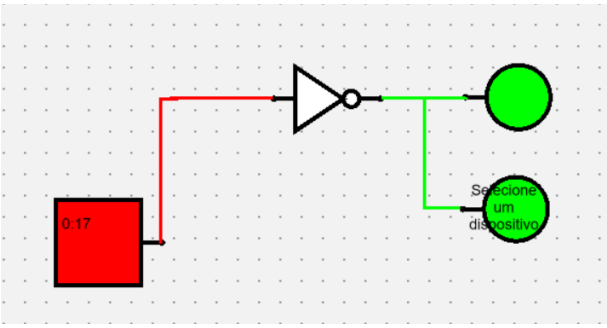


Figura 38: Diagrama de circuito para leitura de botão e saída para LED.

Agora, para associar essa saída ao pino da GPIO, faremos o mesmo processo que foi feito com a fonte. Clicamos duas vezes no componente Saída externa e o modal de seleção é aberto. Basta escolher o pino 27 do Raspberry Pi.

Agora conforme interagimos com o botão, podemos ver a atualização na simulação sendo refletida também no circuito, acendendo e apagando o LED.

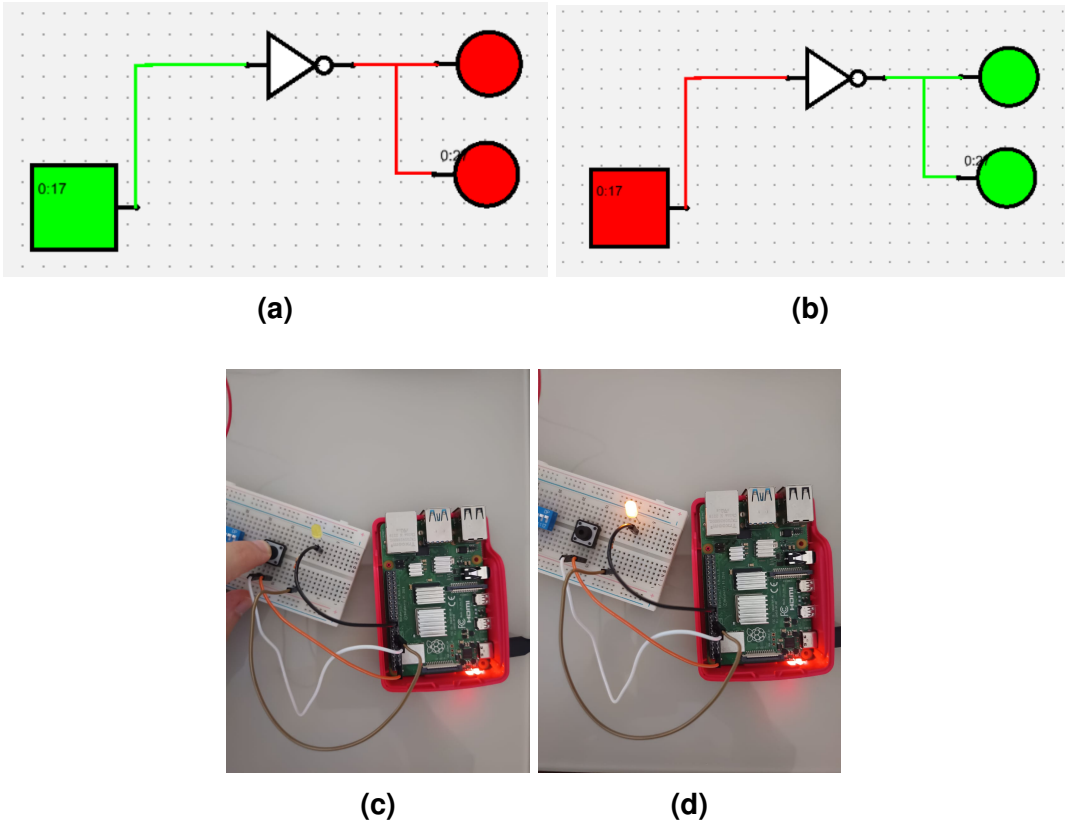


Figura 39: Simulação e circuito montado correspondente. Botão clicado em (a) e (c). Botão não clicado e LED aceso em (b) e (d).

Podemos alterar o nome desse novo circuito simulado e salvar em um novo arquivo, se desejarmos.

Vamos agora partir de um novo circuito para analisar o componente de Fonte de pulsos, também chamado de *clock*. Além dessa fonte, acrescentaremos uma Saída, para visualizar o resultado. Criamos um fio para conectar ambos os componentes.

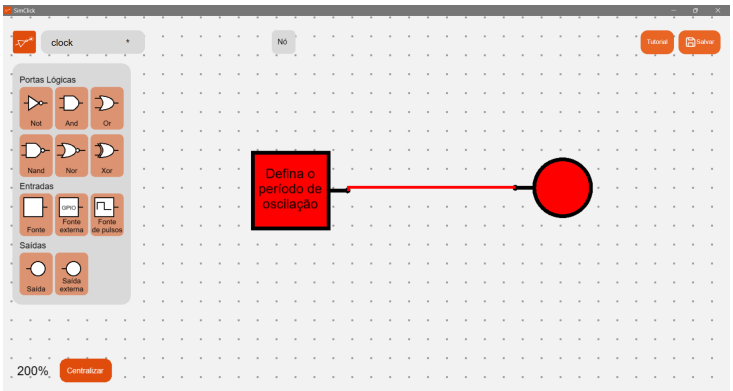


Figura 40: Circuito com fonte de pulsos (*clock*).

Ao ser adicionada ao circuito, a fonte ficará inicialmente no estado lógico baixo.

Será exibido para que seja selecionado o período de oscilação. Clicamos duas vezes na fonte, para configurar o comportamento.

No modal que abre é solicitado o período em milissegundos, vamos colocar 1000, equivalente a 1 segundo. Isso quer dizer que em metade desse tempo (meio segundo), a fonte ficará em nível alto e, na outra metade, em nível baixo.

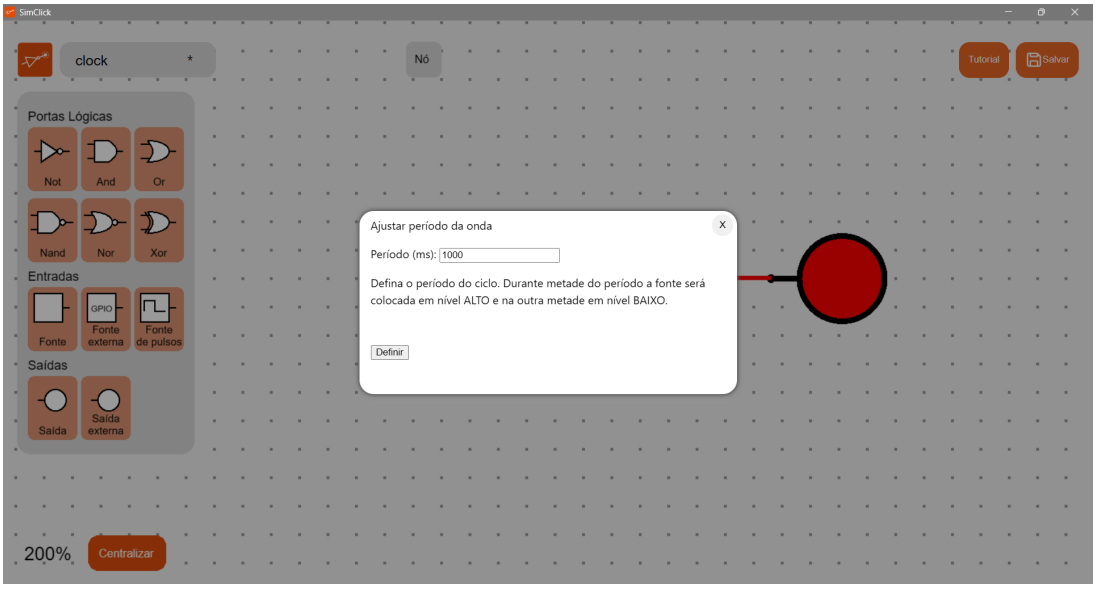


Figura 41: Configuração de tempo para *clock*.

Esse componente permite que executemos ações temporizadas em nossos circuitos.

Capítulo 6

Resultados

Com o programa desenvolvido e através da descrição de suas funcionalidades apresentadas aqui anteriormente, bem como a análise do comportamento dos componentes mostrada nos exemplos de simulação e montagem anteriores, temos disponíveis poderosas ferramentas para aplicação em contextos educacionais e de ideação de projetos de circuitos.

A seguir são propostos dois cenários de montagem, onde é possível visualizar a utilização dos recursos oferecidos pelo SimClick para interagir com componentes eletrônicos e verificar o funcionamento prático de circuitos.

6.1 Casos de Uso

6.1.1 Detecção de Obstáculos

Um dos casos de uso propostos para demonstração da aplicabilidade do programa é a montagem de um circuito para realizar a detecção de obstáculos através da leitura de um sensor infravermelho. O sensor utilizado para esse experimento é o HW-201, ele possui 3 pinos para conexão sendo os dois primeiros para a alimentação (VCC e GND), e o último referente à saída. Esse dispositivo pode trabalhar na faixa de tensão de $3.3V$ a $5V$ e, para facilitar a compatibilidade com o Raspberry Pi no circuito montado, optou-se por trabalhar diretamente com $3.3V$.

O dispositivo funciona a partir do acionamento de um LED infravermelho, que emite um sinal que pode ser detectado por um fotodiodo também presente na placa do sensor, quando o sinal luminoso é refletido por um obstáculo. A distância de detecção pode ser regulada através de um trimpot, sendo o mínimo $2cm$ e o máximo $30cm$. Quando ocorre a detecção de um obstáculo dentro da distância ajustada, a saída é colocada em sinal baixo.

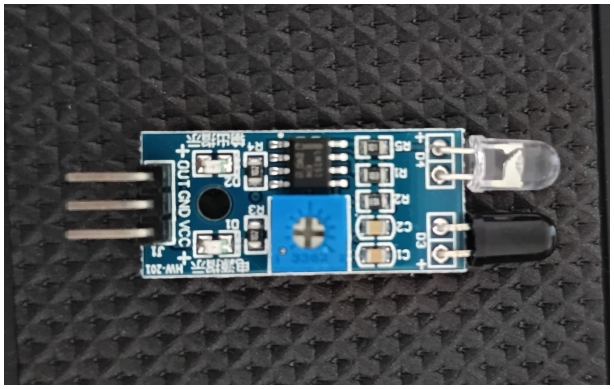


Figura 42: Sensor de obstáculos infravermelho HW-201.

Para esse caso de uso, desejamos realizar a leitura do sinal vindo do sensor, processá-lo em um circuito simulado, e acionar um LED físico. Adicionalmente a detecção, o acionamento deve depender de que esteja habilitado na simulação.

Para a simulação, conforme indicado no diagrama da figura 43, adicionamos uma fonte externa, que foi associada à leitura do sensor através do pino 17 da GPIO. Como a saída do sensor fica em estado baixo quando um obstaculo é detectado, acrescentamos uma porta inversora para tornar o sinal alto. Acrescentamos também uma saída interna para facilitar a visualização do sinal que é recebido do sensor. Foi colocada também uma fonte controlada internamente pela simulação e ambas as fontes — interna e externa — foram conectadas às entradas de uma porta AND. Dessa forma, o acionamento da saída para o LED só será feito se, além de um obstáculo ser detectado, a fonte interna também estiver em estado alto.

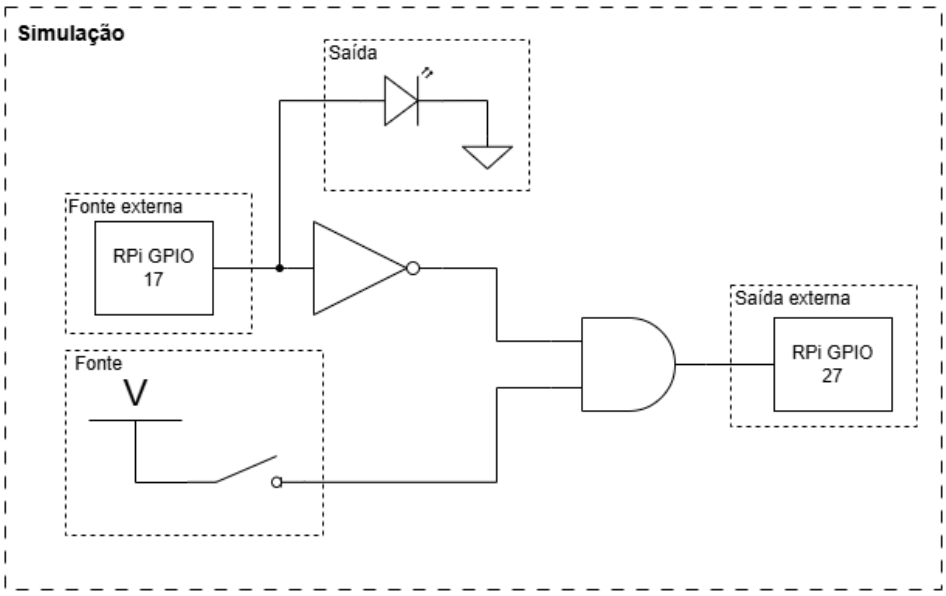


Figura 43: Diagrama de circuito para detecção de obstáculos com sensor IR: parte simulada.

O circuito na protoboard foi montado a partir do diagrama da figura 44. Primeiramente alimentamos o sensor com a alimentação de $3.3V$ do Raspberry Pi e, em seguida, ligamos o pino 17 da GPIO à saída do sensor. Como o Raspberry Pi possui uma conexão de *pull-down* interna que é configurada pelo módulo de comunicação, não é necessário acrescentar esse circuito extra a protoboard para fazer a leitura. Conectamos também um LED, através do pino 27, em série com um resistor de 330Ω para limitação da corrente.

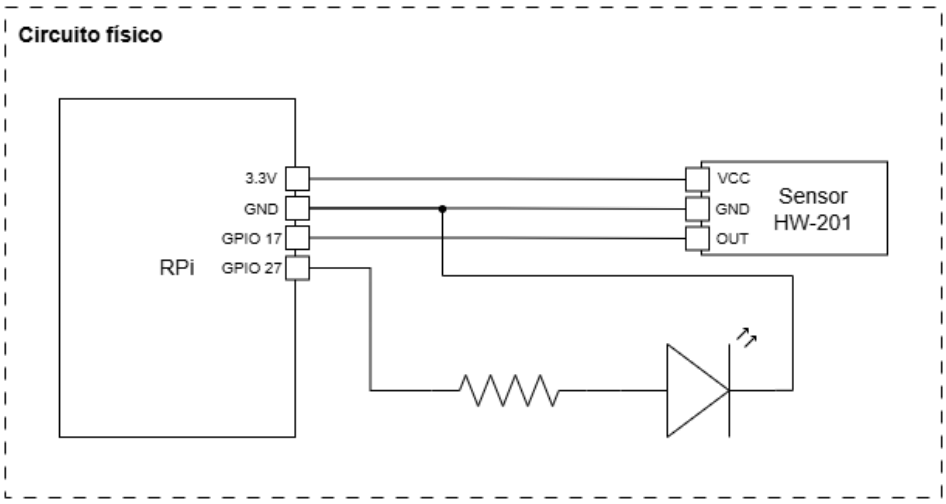


Figura 44: Diagrama de circuito para detecção de obstáculos com sensor IR: parte montada.

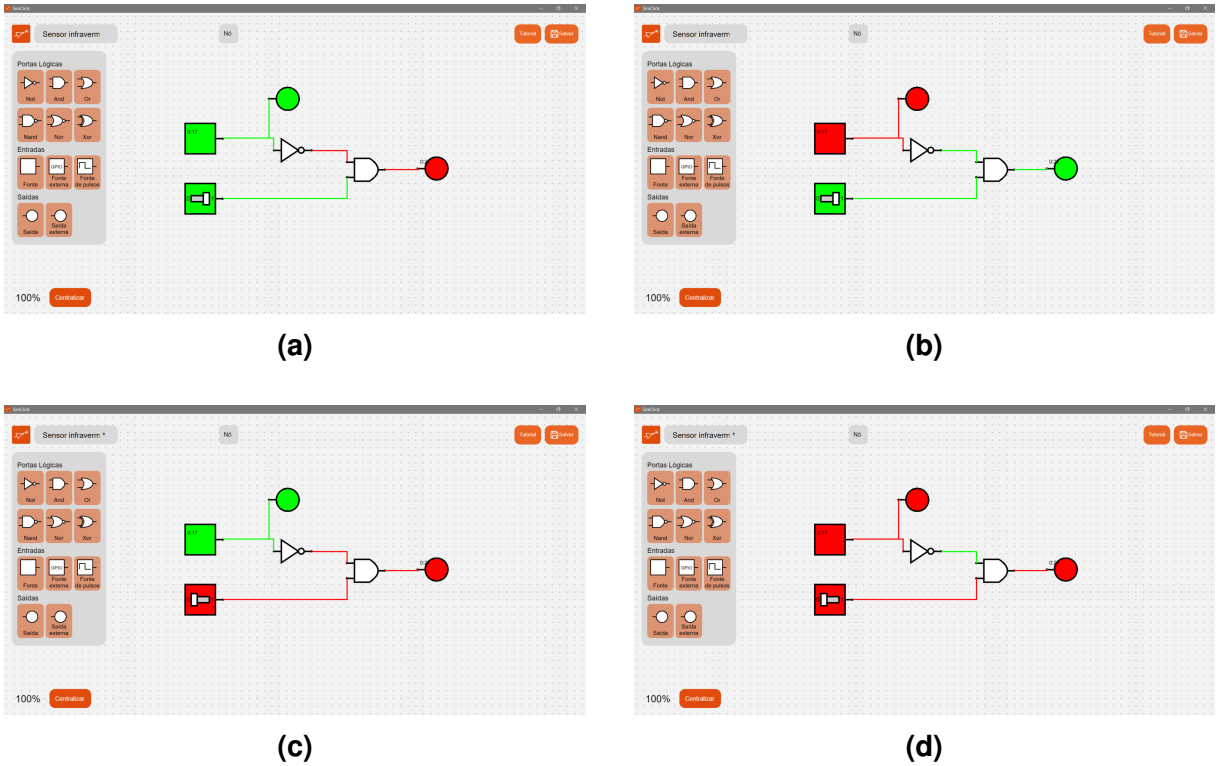


Figura 45: Simulação montada para leitura do sensor de obstáculos infravermelho. (a), (b) Fonte interna ativa. (a) sensor sem detecção de obstáculo, saída final baixa. (b) sensor detectando obstáculo, saída final alta. (c) e (d) fonte interna baixa, em (c) não há leitura de obstáculo, em (b) sim, porém, saída final de ambos baixa.

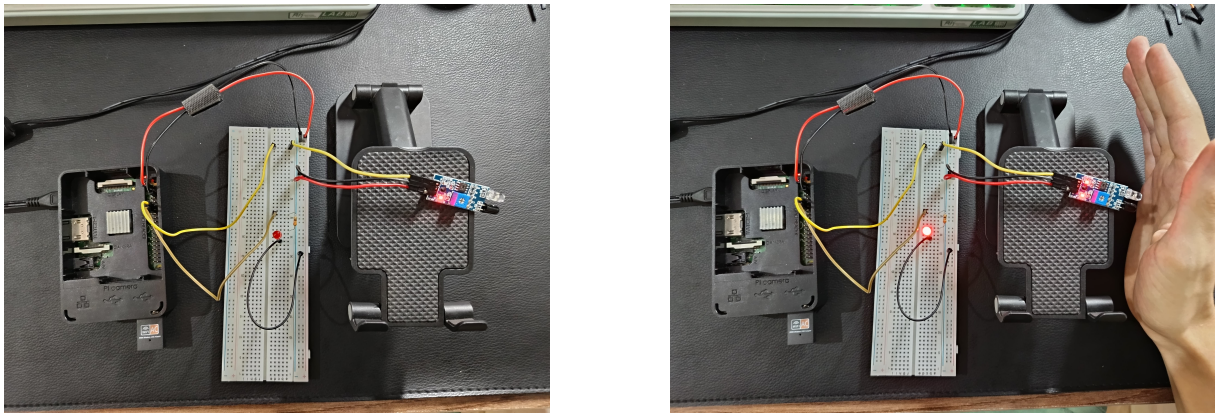


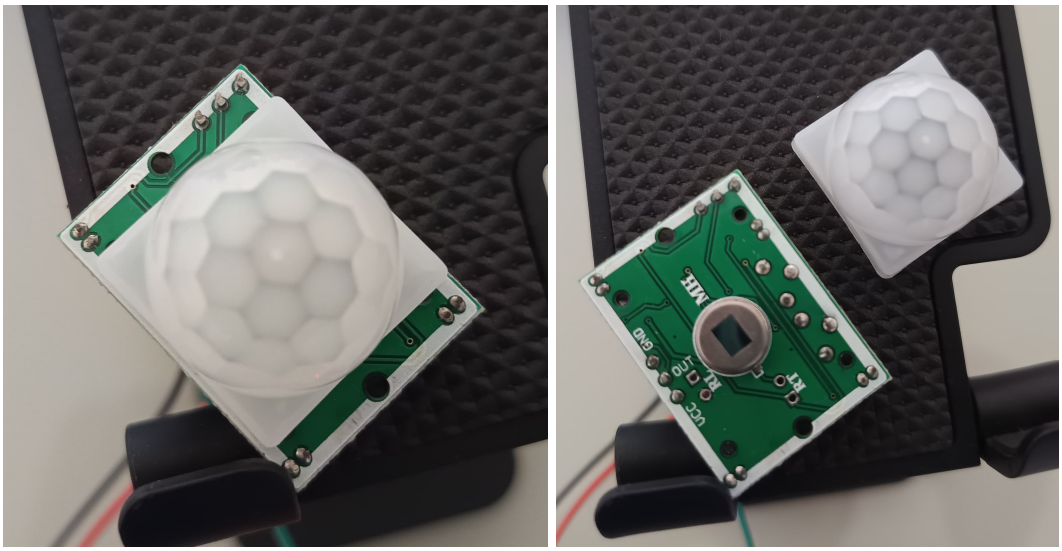
Figura 46: Montagem do circuito. (a) Sem detecção de obstáculo, LED apagado. (b) Obstáculo detectado, LED acesso.

6.1.2 Sistema de Alarme

Um segundo caso de uso que propomos como demonstração é a montagem de um circuito que poderia ser utilizado como base para um sistema de alarme doméstico.

Supondo uma separação e distância entre os circuitos responsáveis pela detecção de presença e acionamento dos dispositivos sonoros e visuais do alarme, assim como sua ativação e desativação, a montagem foi planejada com o uso de dois dispositivos Raspberry Pi, que devem realizar sua comunicação através do circuito simulado. Utilizaremos como componentes um sensor de movimento, uma chave seletora, um *buzzer* ativo e dois LEDs.

A detecção de presença no ambiente será feita através do sensor de movimentação, que será o PIR HC-SR501, capaz de detectar variações no espectro de ondas infravermelhas emitidas pelos elementos a sua volta. Há também uma capa semi translúcida para acoplamento a placa do sensor que permite melhor difusão dos raios infra vermelhos. Esse sensor deve ser alimentado por uma fonte de $5V$, mas seu sinal de saída, que indica a detecção, é nivelado em $3,3V$. Esse será o único componente conectado ao primeiro dispositivo Raspberry.



(a) **(b)**

Figura 47: Sensor de movimentação PIR HC-SR501. (a) Sensor com a capa difusora acoplada e (b) desmontada com as indicações da placa e sensor visível.

No segundo dispositivo serão conectados os LEDs com resistores em série, o *buzzer* ativo, e a chave seletora. A chave será conectada em um pino da GPIO para ser lida e permitir a definição do alarme como ativo ou não. Os outros elementos serão ativados por pinos da GPIO individuais. O esquema completo de montagem, para o circuito de ambos os dispositivos foi disposto no diagrama da figura 49.

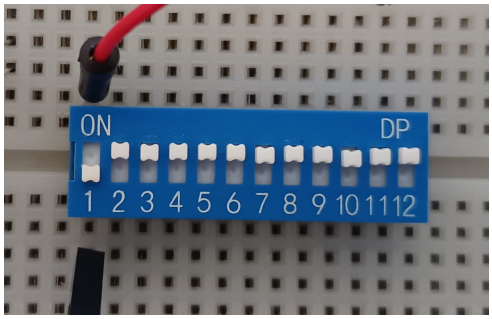


Figura 48: Chave seletora. Foi utilizada apenas a chave numerada como 1.

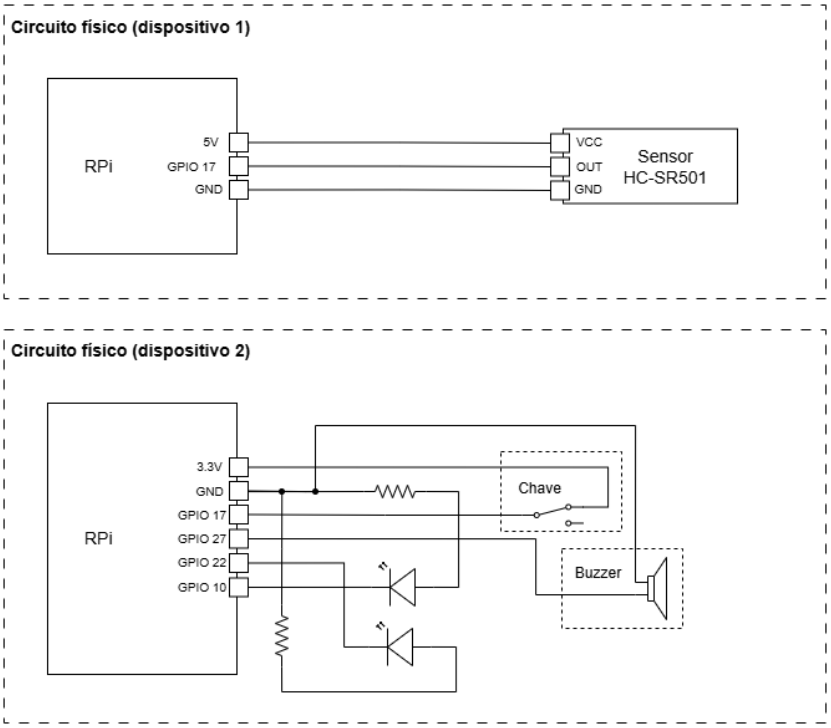


Figura 49: Diagrama de circuito de alarme: trechos montados na protoboard.

A simulação foi planejada com duas entradas de fontes externas, para a leitura do sensor pelo primeiro dispositivo, e para a leitura da chave de ativação do alarme, no segundo. Ambas precisam ser positivas para que o alarme seja acionado, por isso, devem ser associadas a uma porta lógica AND. Também foram criadas duas fontes internas de pulsos, essas serão necessárias para configurar o tempo de piscamento dos LEDs — que serão alternados —, e o tempo de ativação e desativação do alarme sonoro. Dessa forma, o diagrama foi feito conforme mostra a figura 50.

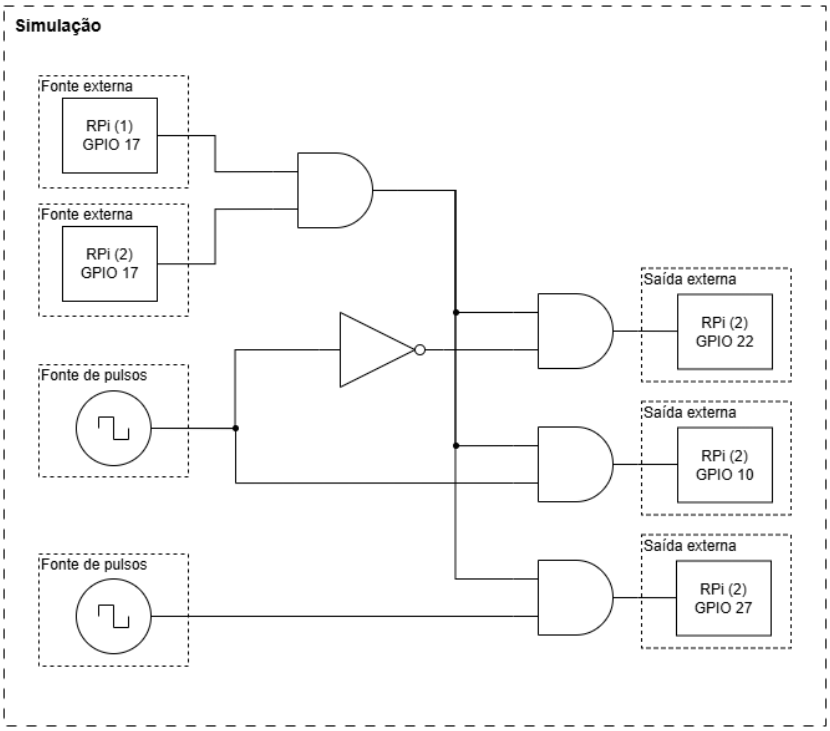
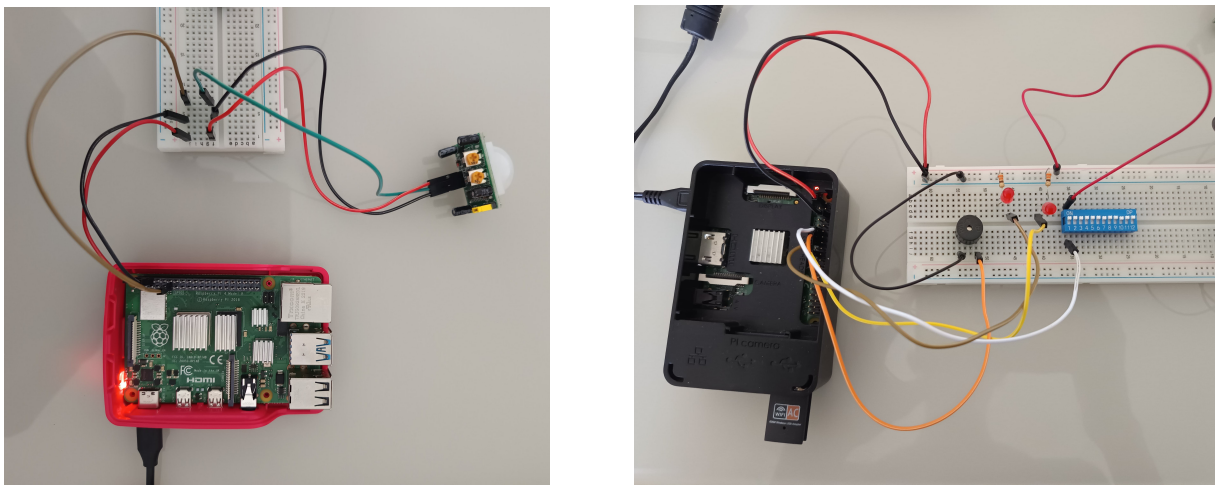


Figura 50: Diagrama de circuito de alarme: trecho simulado.

A partir dos diagramas, os circuitos foram montados conforme a figura 51.



(a) **(b)**

Figura 51: Circuitos montados para o acionamento do alarme. (a) Dispositivo Raspberry conectado ao sensor de movimento PIR. (b) Segundo dispositivo Raspberry conectado aos LEDs, *buzzer* e chave de ativação do alarme.

Por sua vez, para a simulação no SimClick, os componentes foram inicialmente adicionados e as conexões feitas entre eles. Posteriormente, com os dispositivos conectados, foram feitas as associações das entradas e saídas. Além disso, o *clock* refente à ativação dos LEDs foi posto em $2s$, fazendo com que cada LED fique acesso

por $1s$, alternadamente. Também, o *clock* do *buzzer* foi posto em $500ms$, fazendo com que toque a cada $250ms$.

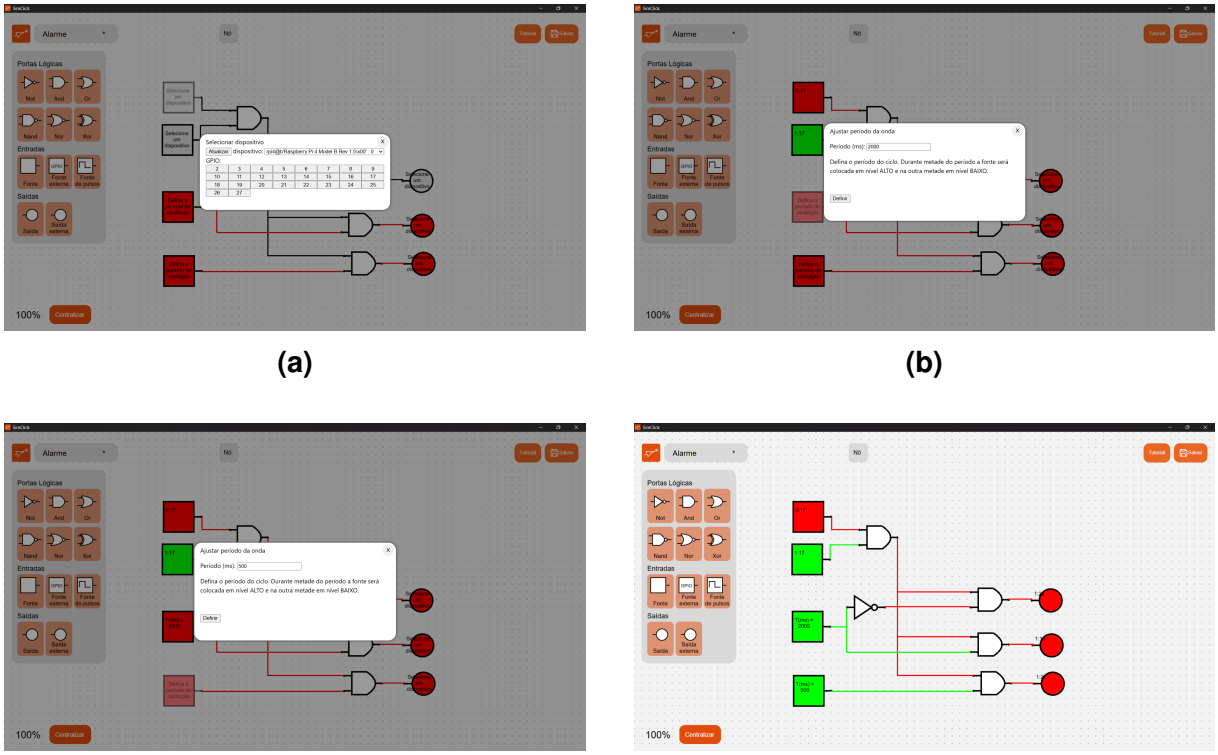


Figura 52: Simulação para acionamento de alarme através da leitura do sensor de presença. (a) Mostra o processo de associação dos elementos da simulação com os dispositivos. (b) e (c) Mostram a configuração dos tempos de *clock*. (d) Mostra o circuito completamento.

Capítulo 7

Conclusões e Desenvolvimentos Futuros

No presente trabalho foi discutido o desenvolvimento do simulador de circuitos digitais SimClick, que foi desenvolvido como uma aplicação disponível para instalação em desktop. Também foi desenvolvido, em conjunto com o programa, um módulo de conexão para permitir a conexão de dispositivos Raspberry Pi com o programa.

O programa permitiu propor, por meio de seus recursos implementados, uma nova abordagem para o aprendizado de eletrônica digital que combina a utilização de software e hardware para os circuitos que são explorados.

Devido ao escopo limitado do projeto, alguns recursos que foram pensados como de potencial utilidade, ou soluções mais elaboradas para limitações encontradas durante o período de desenvolvimento do programa, não puderam ser implementadas no momento. Ainda assim, todas esses possíveis pontos de alteração foram anotados e incluídos como possibilidades de atualização para o desenvolvimento de futuras versões. A seguir, segue uma listagem não exaustiva:

- Desenvolvimento de módulos de comunicação para outras famílias de dispositivos além do Raspberry Pi, como ESP e Arduino;
- Possibilitar a utilização de outras funcionalidades das placas de desenvolvimento, além do uso dos pinos da GPIO como entradas ou saídas simples, como o acesso a recursos de PWM, I2C, e outros;
- Inclusão de outras funções para interação que facilitem a montagem e modificação de circuitos, como por exemplo um histórico de ações (para desfazer e refazer), copiar e colar elementos do circuito e atalhos de teclado;
- Criação de uma paleta de ferramentas e modos de ferramentas para separar as ações do usuário. Ferramentas seriam itens específicos para ações como a seleção de componentes, movimentação pelo circuito, criação de fios, entre outras;

- Adicionar uma função para a descoberta de dispositivos na rede e opção para conexão de dispositivos selecionados, diferente da abordagem atual de conexão automática dos módulos. Isso poderia ser possível com a aplicação de algum protocolo de descoberta de serviços como o mDNS — Sistema de Nomes de Domínio Multicast (inglês: *Multicast Domain Name System*) (mDNS);
- Adição de novos componentes para funções mais complexas como o armazenamento de estados no circuito (flip-flops), blocos para a abstração de máquinas de estado;
- Sintetização de circuitos com seus componentes a partir de fórmulas lógicas inseridas pelo usuário;
- Ampliar a validação dos arquivos de circuitos carregados;
- Acrescentar uma função para permitir a reutilização de circuitos salvos como componentes de blocos em novos circuitos.

Referências

ABELSON, Harold; SUSSMAN, Gerald Jay. **Structure and interpretation of computer programs**. [S.l.]: The MIT Press, 1996.

FRANCO, Fernando Henrique Alves. **Ferramenta Didática WEB Colaborativa para Criação e Simulação de Circuitos Lógicos Digitais**. 2016. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípedes de Marília, Marília. Disponível em: <http://hdl.handle.net/11077/1592>. Acesso em: 19 out. 2024.

GROUP, Khronos WebGL Working. **WebGL Specification**. Disponível em: <https://registry.khronos.org/webgl/specs/1.0/#1>. Acesso em: 30 jan. 2025.

GUPTA, Poonam; M, Indhra Om Prabha. A Survey of Application Layer Protocols for Internet of Things. In: 2021 INTERNATIONAL CONFERENCE ON COMMUNICATION INFORMATION AND COMPUTING TECHNOLOGY (ICCICT). **2021 International Conference on Communication information and Computing Technology (ICCICT)**. Mumbai, India: IEEE, 25 jun. 2021. P. 1–6. ISBN 978-1-6654-0430-3. DOI: 10.1109/ICCICT50803.2021.9510140. Disponível em: <https://ieeexplore.ieee.org/document/9510140/>. Acesso em: 28 out. 2024.

MACMANUS, Richard. **1995: The Birth of JavaScript**. Disponível em: <https://cybercultural.com/p/1995-the-birth-of-javascript/>. Acesso em: 30 jan. 2025.

MDN. **How browsers load websites**. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn_web_development/Getting_started/Web_standards/How_browsers_load_websites>. Acesso em: 30 jan. 2025.

_____. **JavaScript technologies overview**. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/JavaScript_technologies_overview>. Acesso em: 30 jan. 2025.

PIXI.JS. Disponível em: <https://pixijs.com/>. Acesso em: 30 jan. 2025.

RICE, Adam. **WebSockets Living Standard**. Disponível em: <https://websockets.spec.whatwg.org/>. Acesso em: 30 jan. 2025.

SANTOS, Igor Marques Espinosa dos. **Desenvolvimento de simulador WEB didático de circuitos lógicos digitais para estudantes e entusiastas de eletrônica**. 2023. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) – Departamento de Educação Superior, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, Rio de Janeiro.

SOUZA DE LUCENA, Pedro Cezar. **Desenvolvimento de uma Ferramenta Computacional para Modelagem e Simulação de Circuitos Eletrônicos Digitais**. 2013. Trabalho de Conclusão de Graduação (Bacharelado em Engenharia Elétrica) – Escola Politécnica, Universidade Federal do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://hdl.handle.net/11422/11542>. Acesso em: 19 out. 2024.

SVELTE Overview. Disponível em: <https://svelte.dev/docs/svelte/overview>. Acesso em: 30 jan. 2025.

TOCCI, Ronald J; WIDMER, Neal S; MOSS, Gregory L. **Sistemas digitais: princípios e aplicações**. 12a ed. São Paulo: Pearson Education do Brasil, 2018.

TYPESCRIPT for JavaScript Programmers. Disponível em: <https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html>. Acesso em: 30 jan. 2025.

USING Go at Google. Disponível em: <https://go.dev/solutions/google/>. Acesso em: 30 jan. 2025.

WAILS Documentation. Disponível em: <https://wails.io/docs/introduction/>. Acesso em: 30 jan. 2025.