Eduardo Hiroshi Ito - N° USP: 11806868
Henrique D'Amaral Matheus - Nº USP: 11345513
Luiz Guilherme Budeu - Nº USP: 11821639

NANDesis - Simulador didático de Circuitos Digitais

São Paulo, SP

2024

Eduardo Hiroshi Ito - N° USP: 11806868
Henrique D'Amaral Matheus - Nº USP: 11345513
Luiz Guilherme Budeu - Nº USP: 11821639

NANDesis - Simulador didático de Circuitos Digitais

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Edson Toshimi Midorikawa

São Paulo, SP

2024

Resumo

O NANDesis consiste em uma aplicação didática voltada para o ensino de Sistemas Digitais. É composto por um simulador de portas lógicas, que permite aos usuários construírem livremente seus componentes, além de um ambiente de professores, que os permite gerenciar turmas e atividades para os alunos. Elaborada como uma ferramenta educacional baseada em navegador, o simulador atua como uma boa alternativa a softwares instalados, sendo assim, de fácil acesso ao usuário, garantindo uma experiência democratizada a estudantes e indivíduos buscando um aprendizado de metodologia prática e interativa.

Palavras-chave: simulador. sistemas digitais. ferramenta didática. navegador. portas lógicas.

Abstract

NANDesis is a didactic application designed for teaching Digital Systems. It includes a logic gates simulator that allows users to freely build their own components, as well as a teacher environment that enables classroom and activity management for students. Developed as a browser-based educational tool, the simulator serves as a convenient alternative to installed software, offering easy access to users and ensuring a democratized experience for students and individuals seeking to learn through a practical and interactive methodology.

Keywords: simulator. digital systems. didatic tool. browser. logic gates.

Lista de ilustrações

Sumário

1 Introdução

1.1 Motivação

O aprendizado de construção de circuitos digitais preza pelo ensino incremental dos possíveis circuitos e portas lógicas de forma a ser um processo gradual, desde componentes mais simples e básicos até módulos mais complexos. Wakerly (WAKERLY, 2006) elenca as portas NOT, NAND e NOR como básicas para a construção de qualquer sistema digital, devido, principalmente, à facilidade de se obter portas inversoras com menos transistores CMOS. Nisan e Schocken (NISAN; SCHOCKEN, 2005), por sua vez, reduzem a lista de portas lógicas básicas apenas para a porta NAND, descrevendo o processo de construção, a partir dessa única porta, de lógica e aritmética booleana e dispositivos lógicos sequenciais.

As obras desses autores são base para o aprendizado de alunos de áreas vinculadas à computação a respeito de Sistemas Digitais e é necessário o oferecimento de ferramentas interativas que possam complementar e auxiliar nesse processo. Atualmente, há no mercado inúmeras ferramentas para simulação de circuitos digitais, mas nenhuma apresenta um foco didático, desenvolvimento procedural, uso prático e alto desempenho por alunos e professores, evidenciando uma oportunidade de preenchimento dessa lacuna.

1.2 Objetivos

O objetivo principal do projeto consiste em desenvolver um sistema educacional acessível com um gerenciamento de turmas e atividades e um simulador de portas lógicas, inspirado nos tópicos introduzidos nas disciplinas de Sistemas Digitais. Seu prático acesso é um contraponto à softwares concorrentes instaláveis e é obtido por ser elaborada como uma ferramenta didática baseada em navegador, garantindo uma experiência democratizada a estudantes e indivíduos buscando um aprendizado de metodologia prática e interativa. Além disso, é proposto um ambiente que permita professores e educadores da área usufruir e realizar atividades com suas turmas de forma prática que auxilie em seu ensino, também apresentando uma base teórica suficiente para cada exercício proposto.

1.3 Justificativa

A argumentação se baseia na seguinte tese proposta: O projeto NANDesis é uma ferramenta para sanar uma atual defasagem no âmbito educacional de sistemas digitais, de forma a melhor democratizar o aprendizado da criação e desenvolvimento acerca de circuitos digitais.

As quatro defasagens principais podem ser vistas ao estudar os serviços de simuladores competidores, sendo elas:

- Muitos simuladores atuais necessitam de instalação em uma máquina pessoal, sendo assim uma barreira no uso, por via de uso de computadores com limitações técnicas (performance) ou operacionais (instalações bloqueadas, preço, etc.);
- Interfaces complexas, dificultando o uso de iniciantes via a introdução de elementos desnecessários e/ou fora do escopo;
- Ausência de contexto, sendo inapropriadas para uso educacional, priorizando aspectos de criação livre, sem oferecer um direcionamento ao usuário;
- Ausência de ambientes para educadores organizarem e avaliarem atividades sobre circuitos digitais em conjunto com uma simulação para os alunos.

A proposta de trabalho, portanto, servirá como solução para os quatro problemas encontrados via: um serviço acessível via navegador (não baseado na instalação de softwares) para prover a socialização do acesso a ferramentas de estudo/simulação; uma interface simplificada visando eliminar fatores de *overchoice*; um serviço educacional gamificado que oferece uma escala de dificuldade compatível com o nível do estudante; e um ambiente prático em que professores e alunos poderão interagir para a proposição e resolução de atividades.

Alguns trabalhos e recursos consultados foram mais contextualizados com seus temas aprofundados nas seguintes seções do trabalho:

- Estudo de competidores e simuladores alternativos *Nand Game* (KJæR, 2022), LogiJS (BUCHHOLZ, 2023), CircuitVerse (KONWAR, 2024), Linkuit Studio (BUCHHOLZ, 2024), Digital (NEEMANN, 2024), LogiSim (BURCH, 2011) e Logic.ly (HAT, 2023) na seção 2.1;
- Estudo de *overchoice* (CHERNEV; BöCKENHOLT; GOODMAN, 2015) na seção 5.2.2.

1.4 Organização do Trabalho

O trabalho é dividido em 5 partes principais. O Capítulo 2 apresenta os conceitos explorados pela plataforma e lista alguns simuladores presentes no mercado com funcionalidades ou implementações semelhantes ao projeto proposto, descrevendo suas características e diferenças para o NANDesis. No Capítulo 3, é apresentado o método empregado, listando brevemente as atividades realizadas no desenvolvimento e planejamento do projeto, assim

como as atividades futuras planejadas. O Capítulo 4, por sua vez, apresenta de forma detalhada todos requisitos de sistema elencados para o projeto, enquanto o Capítulo 5 apresenta todo planejamento do projeto baseados nesse requisitos e os resultados das atividades realizadas até o presente momento. Por fim, o Capítulo 6 apresenta as consideração finais do grupos sobre o planejamento e desenvolvimento do sistema proposto.

2 Aspectos Conceituais

2.1 Fundamentação de 'NAND' como ponto de partida

Conforme mencionado na seção 1.1 do trabalho, a porta NAND permite o processo de descrição de qualquer componente/dispositivo lógico booleano. A escolha se dá com base na conceituação de um grupo de operadores booleanos com completude funcional, em que um dado conjunto poderia ser utilizado para formar qualquer tabela verdade via a manipulação dos membros deste grupo via uma expressão booleana. Motivado por sua simplicidade, o grupo escolhido para ser aplicado é o conjunto unitário {NAND}, contudo, conceitualmente é possível que o conjunto inicial seja {AND, NOT}, {NOR} ou {OR, NOT}, pois todos eles são funcionalmente completos (NOLT; ROHATYN; VARZI, 1998) (ENDERTON, 2001).

Sendo assim, a porta NAND foi definida no escopo do trabalho como a primeira e única porta lógica oferecida inicialmente ao usuário do simulador. Ademais, de forma a atuar como prova de conceito da progressão proposta pelo aspecto didático do simulador, o trabalho se propõe a guiar um usuário a contruir/simular um processador a partir desta primeira e única porta lógica (HORTON, 2012). O ponto de partida escolhido, portanto, compactua com a fundamentação teórica, enfatizando o aspecto educacional e crescimento linear lógico das funcionalidades do simulador.

2.2 Simuladores alternativos

As versões alternativas que mais se aproximam do trabalho proposto são os projetos *open source* LogiJS (BUCHHOLZ, 2023), por Simon Buchholz, e o simulador de circuitos lógicos do serviço CircuitVerse (KONWAR, 2024), e o projeto *The Nand Game* (KJæR, 2022), por Olav Junker Kjær. Eles funcionam via navegador, com funcionalidades mais vertentes a interação sandbox, com destaque ao serviço focado em aspectos educacionais do segundo e terceiro.

Com relação ao desenvolvimento, o projeto do LogiJS em si aparenta estar em hiato, com a última atualização do código fonte ocorrendo em 2021, em devido ao desenvolvimento de seu sucessor espiritual, Linkuit Studio (BUCHHOLZ, 2024). O novo projeto possui características similares, mas é um instalável do Windows, estando indisponível para uso em um navegador. O CircuitVerse, por sua vez, aparenta ainda estar sendo mantido por seu time principal e por colaboradores voluntários.

O *The Nand Game* possui muitas similaridades com a proposta do projeto NAN-

Desis, pois fornece uma gameficação bem estabelecida onde há fases graduais para a construção de componentes até o desenvolvimento do processador, apresentando material de apoio e validação das respostas fornecidas. Além disso, apresenta uma área para simulação lógica que só é disponível ao completar o desafio de construir o processador.

Outros simuladores de circuitos lógicos digitais alternativos são o Digital (NEEMANN, 2024), LogiSim (BURCH, 2011) e Logic.ly (HAT, 2023):

- O Digital, por Helmut Neemann, é um bom simulador, contudo necessita de instalação para seu funcionamento, o que difere do objetivo de acesso livre e universal;
- O LogiSim, por Carl Burch, apresenta os mesmos problemas do Digital, com o adendo de que ele é um projeto antigo, cujo principal desenvolvimento foi interrompido (mas "continuado" não oficialmente por outros indivíduos);
- Por fim, o Logic.ly apresenta funcionamento em navegador como o desejado, contudo, possui limitações em sua versão gratuita, com custo inviável para estudantes brasileiros.

Para comparação, alguns dos principais elementos que compõem o NANDesis foram buscados nos concorrentes, como mostra a Tabela 1. As perguntas foram:

- **É um serviço web?**
  O acesso simples e prático dos usuários à plataforma é um dos pilares do projeto e, para isso, foi buscada a existência de uma versão *online* das plataformas, que não requerem que o cliente baixe uma aplicação em seus dispositivos.
- **É gratuito?**
  Ainda vinculado ao acesso, é importante que todos os recursos estejam disponíveis aos usuários, sem a necessidade de pagamento. Para plataformas que disponibilizam uma versão gratuita com uma quantidade restrita de recursos, por bloqueio de compra da versão *premium*, também foram consideradas como não gratuitas.
- **Possui manutenções recentes?**
  Apesar do NANDesis ainda estar em desenvolvimento, o histórico de atualiações de um projeto é um indicativo de seu uso e conformidade com implementações mais recentes, assim, foram buscados no histórico de lançamento e *commits* no repositório Github oficial das plataformas por atividades recentes, há menos de 1 ano, ou indicativos de descontinuidade das plataformas.
- **Ausência de componentes complexos pré-prontos?**
  Outro foco e diferencial da plataforma é a criação dos componentes pelo próprio usuário, para serem usados em circuitos mais complexos, desde uma porta básica,

como o NAND. Assim, foi considerada a ausência desses componentes como algo positivo.

- **Permite salvar e utilizar componentes criados?**
  Como o desenvolvimento gradual é incentivado no escopo do projeto, é importante que circuitos criados pelo usuário possam ser utilizados posteriormente no desenvolvimento de componentes mais complexos, fortalecendo a criação passo-a-passo do circuito. Por isso, é importante o sistema possuir a mecânica de salvar um circuito e permitir que ele seja usado como um componente.

- **Tem foco educacional?**
  Além do simulador, o NANDesis conta com uma base teórica e um passo-a-passo da criação de cada componente de forma gradual até a construção de um processador simples. Assim, foram buscados plataformas que fornecem, além de tutoriais de uso da plataforma, um guia para a construção de elementos famosos dentro da ferramenta, mesmo que não apresentem todos os passos.

- **Gera tabela verdade?**
  Uma implementação, que não é o foco da ferramenta, mas auxilia na visualização e validação das implementações é a geração de tabela verdade. Então essa opção foi buscada nas plataformas estudadas, tendo uma aplicação limitada a circuitos não cíclicos.

- **Corrige o componente criado?**
  Para facilitar a criação e auxiliar o aluno a desenvolver o componente solicitado, o simulador deve fornecer uma forma de validar a implementação do componente.

- **Possui ambiente aluno-professor?**
  Uma ferramenta importante da plataforma é a possibilidade de professores utilizarem a plataforma para auxiliar seu ensino, possuindo um ambiente separado do simulador para a manutenção de sua classe dentro da plataforma.

Com os resultados levantados, nota-se que o *Nand Game* é a ferramenta que mais se aproxima do escopo proposto ao NANDesis. Sua principal diferença é a ausência da relação aluno-professor dentro da plataforma e uma base teórica mais superficial, que configura um grande diferencial do NANDesis. Além disso, o *Nand Game* possui um foco maior nas atividades a serem resolvidas do que a criação livre de componentes, possuindo a ferramenta, mas fornece todos os componentes pré-prontos, enquanto o NANDesis permite a opção livre em conjunto com as atividades.

O segundo simulador que mais se aproxima com o escopo do projeto é o CircuitVerse, no entanto, a falta de geração de tabela verdade, a existência de circuitos pré prontos e a falta de correção dos componentes criados o afastam dessa premissa. Entretanto, diferente

Tabela 1 – Comparação de Simuladores Lógicos

	NANDesis	Nand Game	LogiJs	Linkuit Studio	Circuit Verse	Digital	LogiSim	Logic.ly
É um serviço web?	Sim	Sim	Sim	Não	Sim	Não	Não	Sim
É gratuito?	Sim	Sim	Sim	Sim	Sim	Sim	Sim	Não
Possui manutenções recentes?	Em desenvolvimento	Não	Não	Sim	Sim	Sim	Não	–
Ausência de componentes complexos prés-prontos?	Sim	Sim	Não	Não	Não	Não	Não	Não
Permite salvar e utilizar componentes criados?	Sim	Sim	Não	Não	Sim	Sim	Sim	Não
Tem foco educacional?	Sim	Sim	Não	Não	Sim	Não	Não	Não
Gera tabela verdade?	Sim	Sim	Não	Não	Não	Sim	Não	Sim
Corrige o componente criado?	Sim	Sim	Não	Não	Não	Não	Não	Não
Possui ambiente aluno-professor?	Sim	Não	Não	Não	Sim	Não	Não	Não

dos demais simulares, essa ferramenta possui a interface aluno-professor, mas apenas permite que o professor crie novas atividades com correção manual, não permitindo a adição de *testbenches* para automatizar a correção.

Portanto, o NANDesis se destaca dos demais por apresentar ferramentas que carecem em outras plataformas, ao unificar um simulador de construção livre, um guia teórico de construção de circuitos e validações automáticas dos componentes criados pelos os alunos, com um ambiente educacional prático para professores, permitindo ser utilizado em sua didática e na geração de atividades aos estudantes.

3 Método do trabalho

3.1 Estruturação de progressão do projeto

O projeto foi estruturado aos moldes do aprendizado sobre os processos de desenvolvimento de *software*, com um planejamento de diferentes etapas principais, com as fases essenciais sendo listadas conforme um plano que favorece um bom fluxo de trabalho. As principais atividades desenvolvidas estão listadas a seguir.

Especificação e planejamento:

- Definição da proposta inicial;
- Levantamento de requisitos funcionais e não funcionais;
- Estabelecimento de especificações;
- Definição de funcionalidades;
- Definição da arquitetura;
- Definição de tecnologias utilizadas;
- Definição de plano de testes;
- Protótipos de telas.

Implementação:

- Implementação da prova de conceito/protótipo.
- Desenvolvimento da interface funcional do simulador;
- Integração do banco de dados;
- Criação do ambiente do professor e aluno;
- Elaboração da documentação para guia do usuário;
- Criação do corretor automático de atividades.

Testes:

- Testes e validações internas;

- *Playtest* com usuários de teste;

A partir das atividades citadas, a figura 1 indica o cronograma de expectativa e desenvolvimento das tarefas, que foi alterado durante o desenvolvimento devido a alterações de escopo, detalhadas na seção 5.2.6.

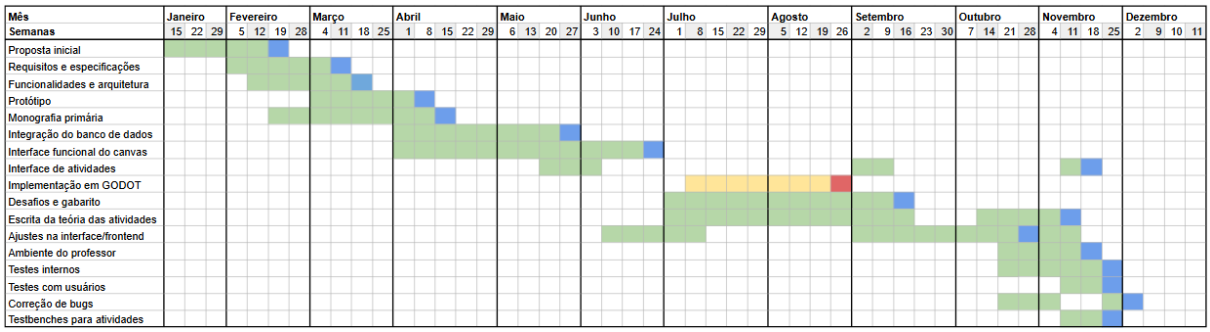


Figura 1 – Cronograma do desenvolvimento janeiro > dezembro

Os preenchimento em verde simbolizam o desenvolvimento da atividade no período correspondente, já os preenchimentos em azul simbolizam as *deadlines* e expectativas acerca de quando serão finalizadas as tarefas descritas em cada linha. Os itens marcados em amarelo e vermelho indicam um progresso de uma *feature* descartada e o seu *deadline*, respectivamente.

Novos itens referentes à autocorreção de circuitos e criação de ambientes para aluno e professor foram inseridos no cronograma no mês de outubro, assim como atividade referentes a testes foram adiadas para incluir as novas implementações.

3.2 Fundamentos de implementação/metodologia do projeto

De maneira a agir como guia para a implementação do projeto, foi priorizado o estudo de referências recomendadas pelo professor orientador e material consultado independentemente pelos integrantes do grupo, com tais sendo mencionados nos capítulos prévios e posteriores da monografia.

Ademais, como forma de acompanhamento, foi desenvolvido e implementado um protótipo funcional como uma prova de conceito a pedido do professor orientador. Tal protótipo busca, via a capacidade de simulação básica de alguns circuitos lógicos, exemplificar a operação do aspecto funcional do simulador.

Para o protótipo, aspectos referentes às atividades e acompanhamento de progresso do estudante/usuário não foram implementados, sendo mais essenciais em etapas futuras não relacionadas ao aspecto funcional da simulação.

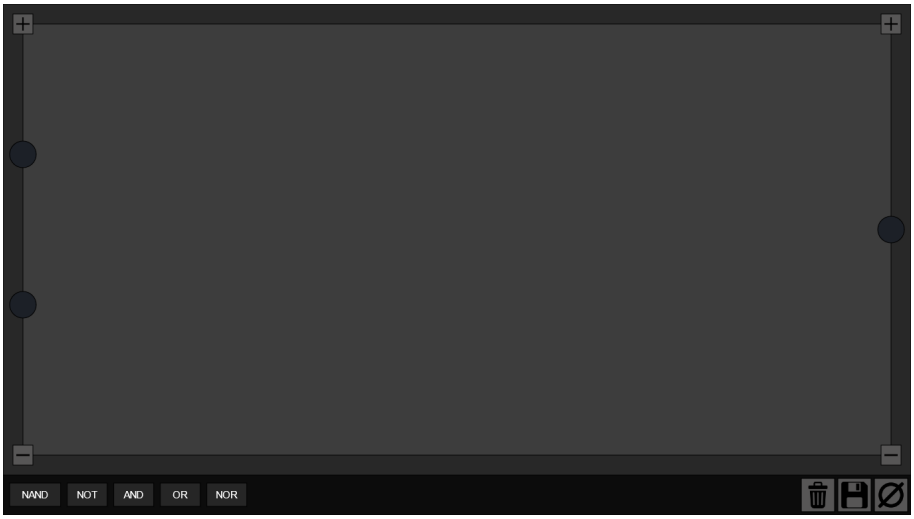


Figura 2 – Interface do protótipo realizado

Detalhamentos mais aprofundados acerca do desenvolvimento em si e das tecnologias empregadas estão disponíveis no capítulo 5.

3.3 Testes

A realização de testes do funcionamento são planejadas e recomendadas para averiguar a proposta e o progresso das funcionalidades essenciais do projeto.

Outro objetivo é o acompanhamento de possíveis desafios de implementação e a preparação para o desenvolvimento de futuras funcionalidades nos moldes estabelecidos do simulador.

Correções para garantir uma boa experiência de uso também estão planejadas para serem realizadas em tempo hábil, garantindo uma construção funcional estável.

4 Especificação de Requisitos

De forma a caracterizar mais objetivamente as funcionalidades do projeto e suas características, é possível mencionar os requisitos funcionais e não funcionais, com suas definições e detalhamento.

4.1 Requisitos

Para melhor organizar os requisitos, eles foram separados em categorias envolvendo sua principal aplicação.

4.1.1 Funções gerais

Código: ACESSO_NAVEGACAO	**Tipo:** Funcional	Contemplado
Requisito: Acesso à plataforma via navegador		
Descrição: Aplicação Web, sem necessidade de download do usuário final		
Prioridade: Alta	**Estabilidade:** Média	
***Rationale*:** Permitir aplicação mobile (foco na versão de navegador, com a implementação mobile sendo averiguada em caso da disponibilidade de tempo útil). Implementação será validada a partir do acesso e uso da plataforma em dispositivos diversos e distintos dos usados para desenvolvimento.		
Requisitos associados:		

Código: CADASTRO_LOGIN	**Tipo:** Funcional	Contemplado
Requisito: Realizar cadastro e login na plataforma		
Descrição: O sistema deve identificar um usuário por seu cadastro na plataforma.		
Prioridade: Médio	**Estabilidade:** Alta	
***Rationale*:** O cadastro deverá incluir se o usuário é professor ou aluno. O sistema de cadastramento deve permitir o armazenamento de portas/circuitos, informações do usuário, progresso dos desafios, e gerenciamento de turmas e atividades. Implementação será validada a partir do acesso registro e acesso de múltiplos usuários de teste.		
Requisitos associados:		

Código: INTERFACE	**Tipo:** Não funcional	Contemplado
Requisito: Interface simples e intuitiva		
Descrição: Possuir uma interface simples, intuitiva, leve e com alta responsividade, focando em UX para um estudante iniciante.		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** Os componentes devem estar acessíveis para inclusão do dashboard e operações de manipulação de objetos deve ser intuitiva e seguir mecânicas semelhantes no mercado. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados:		

4.1.2 Simulador

Código: INTERAGIR_INPUTS	**Tipo:** Funcional	Contemplado
Requisito: Interagir com *inputs* do circuito		
Descrição: O simulador apresenta *inputs* interativos do circuito, permitindo ao usuário ligar ou desligar o sinal dos *inputs* livremente.		
Prioridade: Alta	**Estabilidade:** Alta	
***Rationale*:** Elementos de *input* devem ser aparentes, distinguíveis e de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados:		

Código: MANUSEAR_PORTA_LOGICA	**Tipo:** Funcional	Contemplado
Requisito: Adicionar ou remover qualquer porta lógica previamente salva do circuito		
Descrição: O simulador apresenta *inputs* interativos do circuito, permitindo ao usuário ligar ou desligar o sinal dos *inputs* livremente.		
Prioridade: Alta	**Estabilidade:** Alta	
***Rationale*:** Elementos de porta lógica ou circuito pré-criado devem ser aparentes, distinguíveis e de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: CADASTRO_LOGIN, INTERAGIR_INPUTS		

Código: REGENCIAR_IOS	**Tipo:** Funcional	Contemplado
Requisito: Aumentar/Diminuir número de *inputs*/*outputs* do circuito		
Descrição: O simulador permite que o usuário altere o número de *inputs*/*outputs* do circuito livremente, para que tenha a quantidade de entradas e saídas necessárias para sua implementação		
Prioridade: Alta	**Estabilidade:** Alta	
***Rationale*:** Elementos de ajuste na quantidade de *input* e *output* devem ser aparentes, distinguíveis e de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: INTERAGIR_INPUTS		

Código: CONECTAR_IOS	**Tipo:** Funcional	Contemplado
Requisito: Conectar I/O's via cabos		
Descrição: O simulador deve permitir que o usuário possa adicionar ou remover conexões entre os I/O's das portas lógica livremente, afim de realizar a propagação de sinais desejados		
Prioridade: Alta	**Estabilidade:** Alta	
***Rationale*:** Elementos de conexão entre I/O's devem ser aparentes, distinguíveis e de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: INTERAGIR_INPUTS, MANUSEAR_PORTA_LOGICA		

Código: SALVAR_PORTA_LOGICA	**Tipo:** Funcional	Contemplado
Requisito: Salvar circuito atual como porta lógica própria, com um nome escolhido pelo usuário		
Descrição: O simulador deve permitir que o usuário possa adicionar ou remover conexões entre os I/O's das portas lógica livremente, afim de realizar a propagação de sinais desejados		
Prioridade: Alta	**Estabilidade:** Alta	
***Rationale*:** Elementos de conexão entre I/O's devem ser aparentes, distinguíveis e de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: MANUSEAR_PORTA_LOGICA		

Código: EXPORTAR_CIRCUITO	**Tipo:** Funcional	Não contemplado
Requisito: Exportar e importar um arquivo ".json"de um componente criado		
Descrição: O sistema contará com um opção de exportar o circuito desenvolvido		
Prioridade: Baixa	**Estabilidade:** Média	
***Rationale*:** Permite ao usuário compartilhar um circuito desenvolvido de forma prática. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados:		

Código: TABELA_VERDADE	**Tipo:** Funcional	Não contemplado
Requisito: Exibir tabela verdade de um circuito simples salvo		
Descrição: O sistema contará com um opção de exportar o circuito desenvolvido		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** Essa funcionalidade é focada em circuitos básicos por fim didático. Para circuitos que dependem de entradas anteriores, essa opção não estará disponível. Implementação será validada a partir de testes internos com comparação de resultados obtidos com os esperados.		
Requisitos associados:		

Código: SEGURANCA	**Tipo:** Não funcional	Não contemplado
Requisito: Possuir alta segurança de dados		
Descrição: Aplicação de criptografia em dados sigilosos.		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** Dados sigilosos, como senhas, deverão ser criptografadas ao serem salvas no banco de dados. Implementação será validada ao verificar o efeito da criptografia.		
Requisitos associados: CADASTRO_LOGIN		

Código: TEMPO_RESPOSTA	**Tipo:** Não funcional	Contemplado
Requisito: Consistência do tempo de resposta entre múltiplos dispositivos com diferentes configurações		
Descrição: O sistema deve ter uma implementação leve capaz de manter a mesma performance em dispositivos de configurações e recursos variados, mantendo uma resposta equivalente entre eles.		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** Implementação será validada por meio de testes com dispositivos com configurações de hardware variados.		
Requisitos associados:		

Código: CONFIABILIDADE	**Tipo:** Não funcional	Contemplado
Requisito: Exibir alta confiabilidade da simulação do circuito montado e de seus componentes (+99%)		
Descrição: O circuito montado deve se comportar conforme é esperado, segundo implementações físicas.		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** Implementação será validada por meio de testes comparativos com modelos físicos.		
Requisitos associados:		

4.1.3 Área do educador

Código: GERENCIAMENTO_TURMAS	**Tipo:** Funcional	Contemplado
Requisito: Gerenciar turmas		
Descrição: O sistema permitirá professores gerenciar turmas		
Prioridade: Alta	**Estabilidade:** Média	
***Rationale*:** Essa funcionalidade permite aos professores criar, visualizar, editar e apagar turmas, além de convidar alunos para elas. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: CADASTRO_LOGIN		

Código: GERENCIAMENTO_ATIVIDADES	**Tipo:** Funcional	Contemplado
Requisito: Gerenciar atividades		
Descrição: O sistema permitirá professores a gerenciar atividades		
Prioridade: Alta	**Estabilidade:** Média	
***Rationale*:** Essa funcionalidade permite aos professores editar quais atividades estão disponíveis para seus alunos e conferir o desempenho deles por tarefa. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados: CADASTRO_LOGIN		

4.1.4 Atividades

Código: PROGRESSAO_APRENDIZADO	**Tipo:** Funcional	Contemplado
Requisito: Propor desafios com progressão de complexidade linear de forma a induzir o aprendizado		
Descrição: O sistem apresentará um guia de construção de circuitos ordenados do mais específico ao mais geral. A ordem segue a evolução de complexidade dos circuitos criados		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** O sistema deve apresentar uma ordem de criação de elementos, de forma a progredir logicamente com o conteúdo, com dificuldade crescente de uso intuitivo. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados:		

Código: RESPOSTAS_CIRCUITOS	**Tipo:** Funcional	Contemplado
Requisito: Apresentar as respostas dos desafios de forma opcional		
Descrição: O sistema contará com um guia com explicação e resolução dos componentes propostos para criação		
Prioridade: Média	**Estabilidade:** Média	
***Rationale*:** O sistema deve permitir opcionalmente a visualização das respostas, de forma que estudantes frustrados com os desafios propostos não fiquem estagnados, podendo também aprender ao estudar uma resolução alternativa. Implementação será validada a partir de testes internos e do uso da plataforma por usuário de testes e seu feedback.		
Requisitos associados:		

4.2 Aplicação Web

A aplicação web tem o foco em ser o portal central de todas interações dos usuários. Nela os estudantes poderão ter acesso ao simulador e atividades, enquanto o professor terá acesso às turmas e ao gerenciamento de atividades.

O sistema possui uma arquitetura simples cliente servidor, como ilustra a figura 3. A interface do programa realiza requisições HTTP conforme a demanda do usuário e o servidor *backend* é responsável por acessar o banco de dados para obter e tratar as informações, realizando o controle de acesso também.

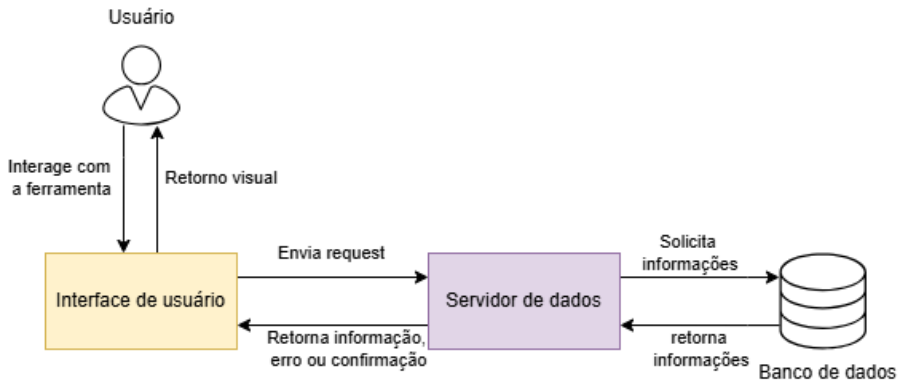


Figura 3 – Esquemático da arquitetura do sistema NANDesis

4.2.1 Frontend

O *Frontend* será o responsável pela interação do usuário e apresentação das informações. Seus elementos são descritos nas sub-seções a seguir.

4.2.1.1 Simulador

O simulador apresenta uma interface simples, fornecendo uma porta lógica inicial (NAND), e funcionalidades básicas para a criação de circuitos (adição de *inputs* e *outputs*,

salvamento, deleção e um menu de atividades). É possível inserir elementos previamente criados e salvos para montar circuitos mais complexos, conectando os elementos internos via fios para a realização de fluxo de dados/sinais. Ademais, o serviço é leve e com performance apropriada para elementos mais complexos. No simulador, as seguintes ferramentas são esperadas:

- **Gerenciamento de portas:** Usuário pode salvar o circuito criado em forma de porta, utilizar essa porta como componente para outro circuito e apagar uma porta salva.
- **Manipulação de portas:** No *dashboard*, o usuário poderá selecionar as portas de sua lista de portas e manipular elas no *dashboard*, posicionando onde desejar e removendo elas, caso necessário.
- **Manipulação de *inputs* e *outputs*:** Para o circuito criado, o usuário pode adicionar ou remover *inputs* e *outputs* como desejar, com um mínimo de 1 unidade de cada. Além disso, pode nomear eles conforme desejado e ativar e desativar *inputs*, observando o funcionamento.
- **Conexão de componentes:** O usuário poderá estabelecer conexão entre os *inputs* e *outputs* globais e das portas presentes no *dashboard*, além de deletar essas conexões quando desejado.
- **Listagem de atividades:** Conforme a turma cadastrada, o aluno tem acesso às atividades disponíveis, observando: sua maior nota em cada uma delas, a teoria vinculada, uma possível solução, quando a atividade estiver finalizada; e a possibilidade de enviar um circuito desenvolvido para a correção automática.
- **Importar e exportar circuitos:** Uma forma de facilitar o compartilhamento de circuitos entre os usuários, é proposta a opção de exportar e importar um circuito no formato JSON. A aplicação seria responsável de montar o arquivo JSON e interpretar um recebido, informando se é válido ou não.
- **Gerador de tabela verdade:** Uma ferramenta simples que auxiliaria os alunos na criação de circuitos combinatórios seria a geração de tabela verdade. Isso permitiria validar de forma prática se o componente construído tem comportamento de suas saídas conforme o esperado na documentação.

4.2.1.2 Área do professor

Na área exclusiva para professores, as seguintes atividades são esperadas:

- **Gerenciamento de turmas:** Interface onde professor pode acessar uma lista com todas suas turmas e cadastrar ou editar uma turma.

- **Gerenciamento de alunos:** Nos detalhes de cada turma, o professor, além de listar os alunos cadastrados na turma, deve ser capaz de convidar os alunos a entrarem na turma.
- **Gerenciamento de atividades:** Para cada turma, o professor é capaz de listar as atividades disponíveis, configurando as datas de início e finalização da submissão dessas tarefas pelos alunos.
- **Gerenciamento de notas:** Em cada atividade, o professor tem disponível a lista de notas dos alunos, incluindo tanto a melhor nota de cada um, quanto todas submissões realizadas por eles. Além disso, é possível exportar as notas da atividade para uma planilha.

4.2.2 Backend

O *backend* tem um caráter passivo, aguardando as requisições do *frontend*. As principais atividades definidas para ele são:

- **Gerenciamento de perfis:** O servidor autentica os usuários e as requisições recebidas, permitindo algumas ações de serem realizada exclusivamente por determinados tipos de perfis. Além disso, conta com ações básicas de cadastro e login, incluindo o registro de alunos vinculados a uma turma.
- **Gerenciamento de turmas:** A api permite o professor listar todas suas turmas e cadastrar ou editar uma turma. Além disso, o aluno poderá acessar informações de sua turma.
- **Gerenciamento de atividades:** O servidor deve ser capaz de prover a possibilidade de configuração das atividades pelo professor e fornecer listagem personalizada para os alunos, dependendo dessa configuração de sua turma. Além disso conta também com a função de salvar as notas obtidas pelos usuários.
- **Gerenciamento de circuitos:** Vinculado com o simulador, esses *endpoints* são os responsáveis por permitir o cadastro, listagem e deleção de portas lógicas criadas pelos usuários.

4.3 Orientação educacional

Uma proposta que diferencia o sistema de outros simuladores *sandbox* é o aspecto de educação gamificada concomitante a uma menor quantidade de elementos iniciais, ao qual poderia minimizar os sintomas de *overchoice* (CHERNEV; BöCKENHOLT; GOODMAN, 2015) que são presentes em simuladores modernos. A redução dos elementos iniciais

atuaria de forma a reduzir o número de escolhas dispostas ao usuário em sua introdução, amenizando o impacto de paralisia de escolhas comumente encontrado em ambientes com abundância em excesso de opções; e a gamificação seria um direcionamento com uma progressão lógica de complexidade.

De forma a melhor detalhar a gamificação, ela consiste em atividades acompanhadas por introduções teóricas dos elementos a serem construídos, sendo assim proposto ao estudante tentar formá-los com base nos elementos básicos iniciais. Esta lista de componentes será gerenciada pelo professor, permitindo a ele apresentar aos alunos conforme o andamento de sua disciplina.

Seria assim formada uma sequência lógica de desenvolvimento, com um nível crescente de dificuldade de forma a melhor introduzir o aluno a conceitos de forma organizada, reduzindo eventuais dificuldades provenientes de uma sobrecarga de escolhas. Além disso, é promovida uma visão vertical do funcionamento de componentes mais complexos.

Como forma de dar suporte aos usuários do sistema, uma base teórica também acompanha cada desafio, informando sobre os detalhes funcionais de cada componente, além de também fornecer uma introdução concisa acerca dos tópicos de álgebra booleana, mapas de Karnaugh com o intuito de melhor conduzir o estudante aos conceitos requeridos pelo sistema.

Além disso, para cada atividade, é apresentado um corretor que verificaria o circuito criado pelo estudante, validando seu conteúdo por meio da comparação das entradas e saídas da tabela verdade, averiguando se o circuito criado possui as mesmas características do proposto ou passando o componente por um *testbench*, ou seja, um script que executa o componente e valida suas saídas em diversas etapas. Os resultados provenientes dessa correção estariam disponíveis para o aluno e o professor, permitindo seu uso para atividades avaliativas.

Por fim, após a data de finalização da atividade configurada pelo professor, as submissões para correção não são mais habilitadas, mas fica disponibilizado ao aluno um gabarito informando uma forma de solucionar o problema da atividade.

O sistema permite que alunos sejam cadastrados sem vínculo a nenhuma turma, uma vez que se propõe a ser um sistema de aprendizado livre, que permite que todos possam usar. Nesse caso, todas atividades da lista ficam disponíveis, além de poder acessar a teoria, solução e enviar ao corretor a qualquer momento.

4.4 Testes com usuários

Para a validação de aspectos, como usabilidade, intuitividade, entre outros, será realizado testes com usuários com indivíduos não relacionados com o desenvolvimento do projeto. Para isso, foi preparado um formulário do Google com as perguntas apresentadas a seguir. Todas perguntas, exceto a última, são de pontuação que variam numa escala de 1 a 5, sendo 1 uma nota ruim e 5 uma nota ótima. Cada pergunta de nota é acompanhada com uma pergunta textual optativa em que a pessoa pode incluir mais detalhes daquele assunto, ou vertentes não perguntadas.

- **"Sobre o simulador, quanto você avalia cada aspecto":**

  - **"Visual":** Relacionado à estética do simulador, se o visual dele é agradável e está coerente com a proposta de um simulador de circuitos digitais;
  - **"Intuitividade":** Busca avaliar se as possíveis ações são claras e objetivas com o usuário;
  - **"Usabilidade do dashboard":** Referente a ações como adicionar componente, arrastar componente e conectar *inputs*;
  - **"Usabilidade das ferramentas (Barra inferior)":** Referente à barra de ferramentas do simulador, que envolve a lista de componentes, o botão de deletar, salvar e limpar o *"dashboard"*. Deseja medir a qualidade de uso delas;
  - **Funcionamento do circuito:** Referente ao comportamento do circuito montado, se ao alterar um *input* ele se comporta como esperado;
  - **"Acessibilidade":** Busca avaliar se as ferramenta são facilmente acessíveis ao usuário.

- **"Sobre as atividades, quanto você avalia cada aspecto":**

  - **"Visual":** Relacionado à estética do simulador, se o visual dele é agradável e está coerente com a proposta de um simulador de circuitos digitais;
  - **"Usabilidade":** Clareza nas ações e informações da lista de atividades;
  - **"Intuitividade":** Busca avaliar se as possíveis ações são claras e objetivas com o usuário;
  - **"Teoria da atividade":** Clareza do texto teórico;
  - **"Solução da atividade":** Clareza da solução da atividade;
  - **"Correção da atividade":** Clareza na ação de corrigir o circuito montado e na nota obtida;

  - **"Acessibilidade":** Busca avaliar se as ferramenta são facilmente acessíveis ao usuário.

- **"Sobre o gerenciamento de turmas (Criação, listagem, edição e convite de alunos), quanto você avalia cada aspecto":**

  - **"Visual":** Relacionado à estética do simulador, se o visual dele é agradável e está coerente com a proposta de um simulador de circuitos digitais;
  - **"Usabilidade":** Clareza nas ações e informações da lista de atividades;
  - **"Intuitividade":** Busca avaliar se as possíveis ações são claras e objetivas com o usuário;
  - **"Acessibilidade":** Busca avaliar se as ferramenta são facilmente acessíveis ao usuário.

- **"Sobre o gerenciamento de atividades (Atribuição de datas e listagem e exportação de notas), quanto você avalia cada aspecto":**

  - **"Visual":** Relacionado à estética do simulador, se o visual dele é agradável e está coerente com a proposta de um simulador de circuitos digitais;
  - **"Usabilidade":** Clareza nas ações e informações da lista de atividades;
  - **"Intuitividade":** Busca avaliar se as possíveis ações são claras e objetivas com o usuário;
  - **"Acessibilidade":** Busca avaliar se as ferramenta são facilmente acessíveis ao usuário.

- **"Use esse espaço para relatar bugs, problemas não discutidos nas questões anteriores, ou sugestões gerais":**
  Pergunta de texto aberta para quaisquer feedback do usuário sobre a plataforma. tem o intuit de capturar algum ponto não abordado no escope de perguntas anteriores

Para esse formulário, as perguntas voltadas para o ambiente do professor são mandatórias caso o usuário tenha testado a funcionalidade, não sendo obrigatória.

5 Desenvolvimento do Trabalho

5.1 Tecnologias Utilizadas

A aplicação desenvolvida é dividida em duas partes: o *backend* e o *frontend*. O *backend* é composto por uma aplicação Node.js (DAHL, 2024), utilizando o *framework* Django (KIRBY, 2024). O servidor web servirá o *frontend* criado com React (META, 2024), utilizando principalmente o elemento *canvas* para a exibição do simulador.

5.2 Projeto e Implementação

O projeto é denominado "NANDesis" proveniente da junção das palavras NAND e Gênesis, que representa a porta NAND como porta universal fazendo alusão ao início de tudo apresentado em Gênesis.

5.2.1 Prova de conceito

Para validar a viabilidade e eficácia do simulador de portas lógicas proposto neste projeto, foi definida uma prova de conceito consistente na simulação de qualquer circuito combinatório. Ela visa demonstrar a capacidade do simulador em permitir a construção e simulação de circuitos digitais complexos, culminando em uma etapa fundamental para processamento de informações.

A simulação de circuitos combinatórios é o primeiro e mais essencial passo no desenvolvimento do simulador, uma vez que é a base do funcionamento dos circuitos mais complexos. Esses são construídos utilizando apenas portas lógicas e componentes básicos, sem elementos de armazenamento. Isso significa que o resultado de uma saída é determinado exclusivamente pelas entradas no momento da avaliação.

A partir das portas lógicas básicas fornecidas pelo simulador, os usuários têm a capacidade de combinar e interconectar as portas lógicas salvas para construir quaisquer circuitos combinatórios, como somadores, subtratores, multiplexadores e a Unidade Lógica e Aritmética (ULA).

A ULA é uma parte essencial em muitos processadores e sistemas digitais, sendo responsável por realizar operações aritméticas e lógicas. Portanto, atingir a simulação de uma ULA demonstra a capacidade do simulador em lidar com circuitos digitais altamente complexos e funcionais.

Para simular uma ULA, é necessário que o usuário tenha previamente desbloqueado

as portas lógicas básicas, tais como as portas NOT, AND e OR, além dos circuitos de soma e subtração, se necessário. Adicionalmente, é necessário a construção de um multiplexador para a seleção da entrada a ser refletida na saída. Todos esses componentes requerem que o simulador seja capaz de salvar circuitos com qualquer número de entradas e saídas, e realizar a interconexão adequada entre eles para formar a ULA.

Ao alcançar com sucesso a simulação de uma ULA, é possível confirmar a robustez e utilidade do simulador de portas lógicas proposto, validando assim a sua viabilidade como uma ferramenta educacional e de desenvolvimento de sistemas digitais.

5.2.1.1 Protótipo para prova de conceito

Como mencionado no capítulo 3, um protótipo foi desenvolvido para atuar como método de acompanhamento do professor orientador Edson Midorikawa.

O sistema apresenta grande parte das funcionalidades de um simulador, podendo representar e simular um grande número de portas lógicas e circuitos digitais, além de salvar diferentes componentes para serem utilizados em outros circuitos sendo construídos.

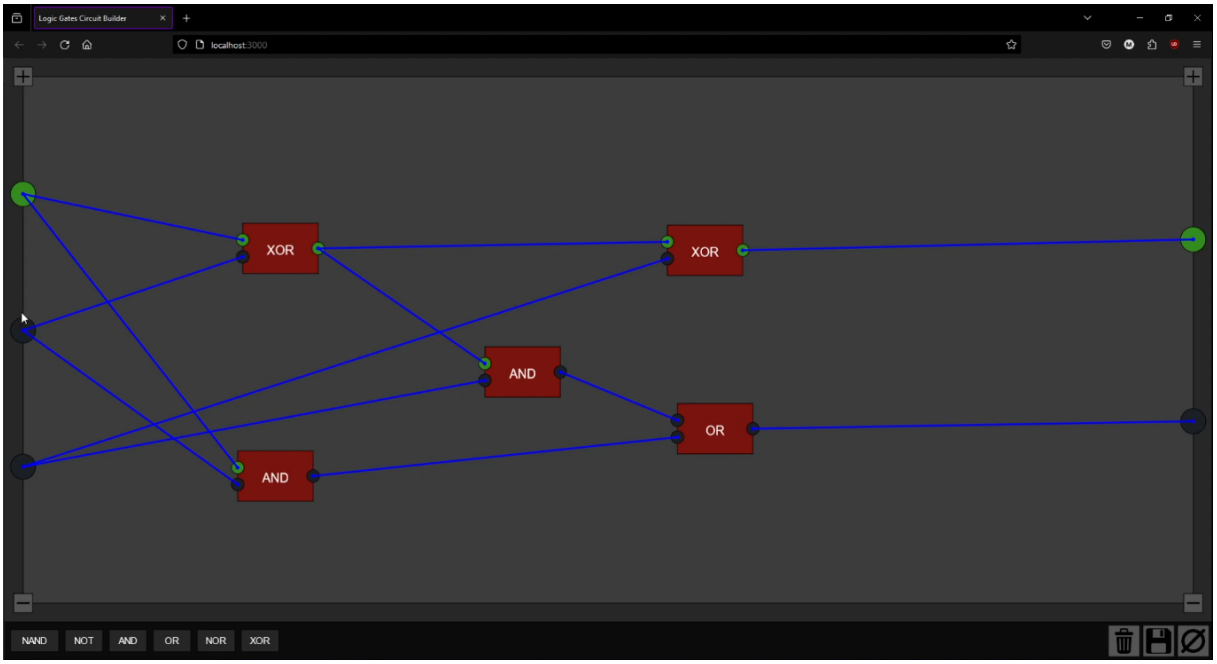


Figura 4 – Implementação de um *adder* no protótipo

A figura 4 ilustra a implementação de um *adder* binário utilizando portas XOR, AND e OR construídas anteriormente e acessíveis por meio do menu inferior da tela. Foi realizado seu *deployment* por meio de ferramentas do GitHub, com o acesso estando disponível para testes e simulações básicas.

Seu foco [e na interface e ferramentas do simulador, enquanto as funcionalidades de desafio e guias didáticos não foram implementadas neste primeiro momento, sendo

secundárias aos aspectos de simulação. Outros exemplos no simulador são exemplificadas a seguir:

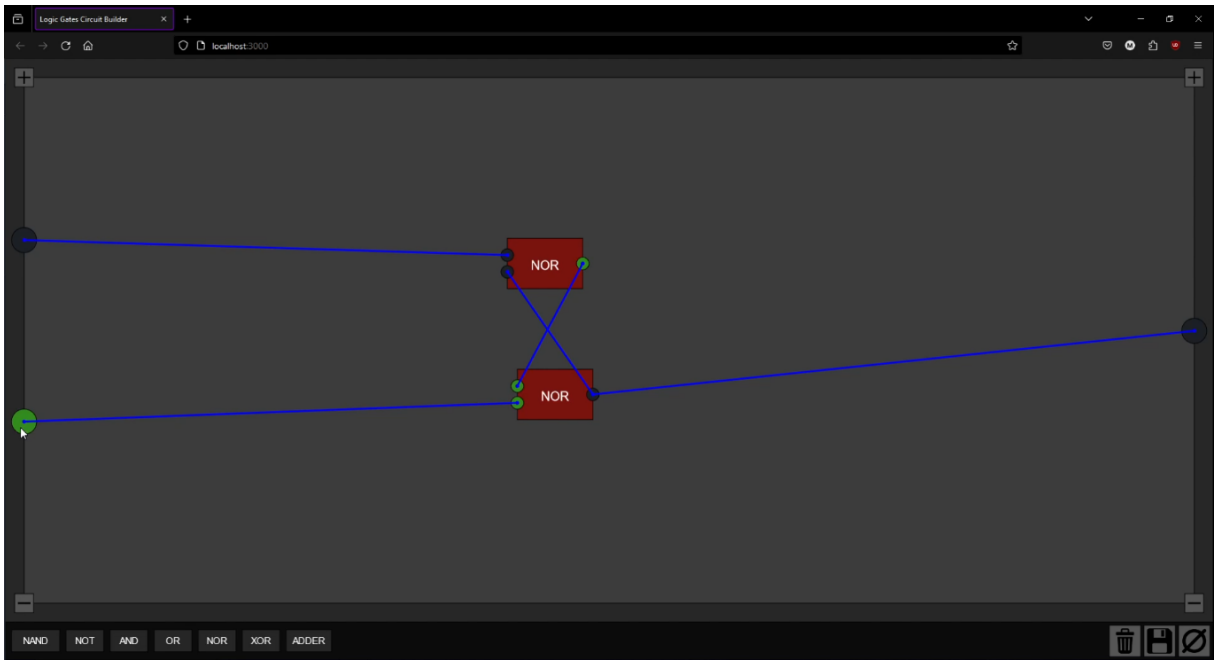


Figura 5 – Implementação de uma SR *Latch* no protótipo

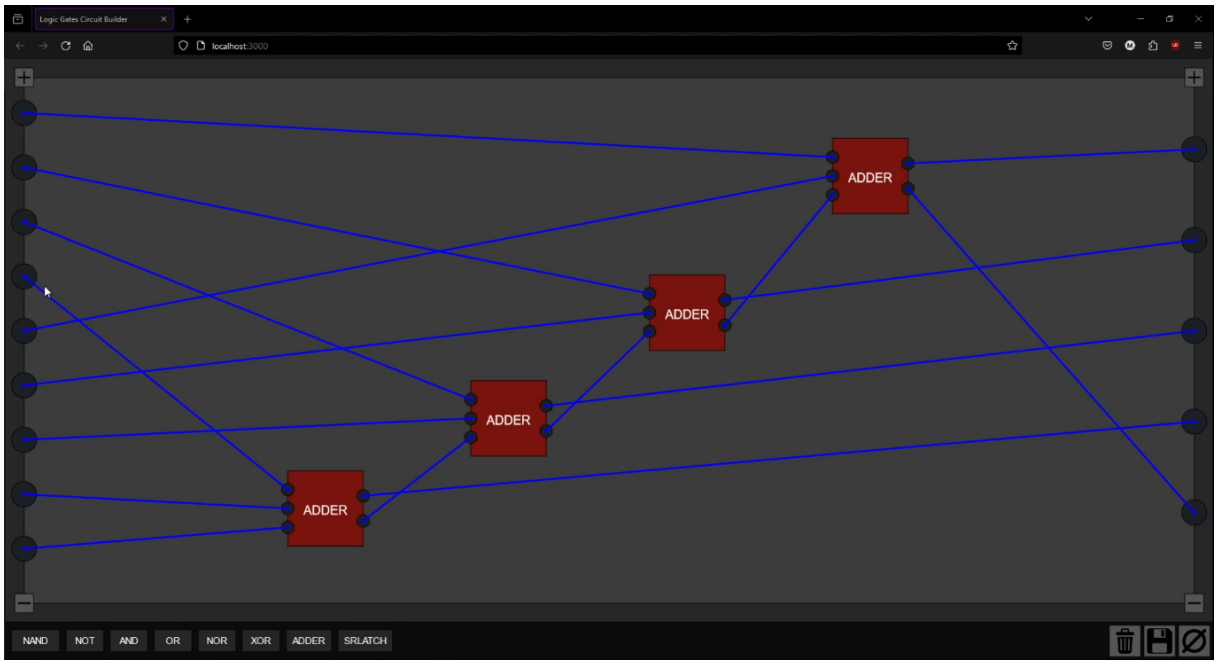


Figura 6 – Implementação de um 4 bit *adder* no protótipo

5.2.2 Simulador

5.2.2.1 Lógica e funcionamento

O projeto consiste num simulador para a construção e desenvolvimento de circuitos digitais. A ideia é que a partir de portas lógicas e outros componentes básicos, outros circuitos possam ser construídos progressivamente até o desenvolvimento de sistemas mais complexos, tal como um processador, por exemplo.

Seu funcionamento é semelhante ao de ferramentas como o Digital (NEEMANN, 2024), que apresenta um *dashboard* com entradas e saídas, onde o usuário poderá posicionar os componentes e realizar conexão entre eles. Cada circuito montado poderá ser salvo e usado como uma porta lógica em componentes mais complexos.

Inicialmente será fornecido apenas a porta lógica NAND. A partir dela é possível criar todas outras portas lógicas principais e circuitos mais complexos (NISAN; SCHOCKEN, 2017). Há outros elementos mínimos não lógicos, como clock, que podem entrar nos recursos básicos, mas a ideia é fornecer o mínimo de circuitos prontos possíveis, para incentivar a construção pelo usuário.

Para ilustração, um exemplo de uso é a criação das portas NOT e OR. A primeira pode ser desenvolvida utilizando apenas a porta NAND, enquanto a segunda utilizaria a porta NOT desenvolvida e a NAND. Os circuitos estão apresentados nas figuras 7 e 8.

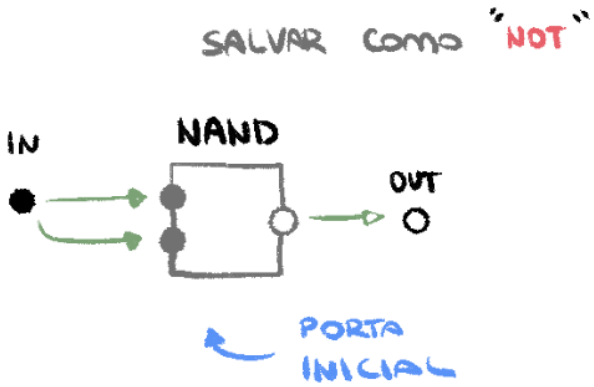


Figura 7 – Esquemático do circuito de uma porta NOT

5.2.2.2 Arquitetura

A arquitetura da aplicação será dividida em duas partes: o *backend* e o *frontend*. O simulador rodará no *browser* do cliente, fazendo requisições para o *backend* para salvar ou carregar circuitos, assegurando a persistência de dados do usuário.

A arquitetura do simulador envolve paradigmas de programação como o Entity Component System (ECS) (TERRA, 2023), onde cada objeto em uma cena é uma entidade,

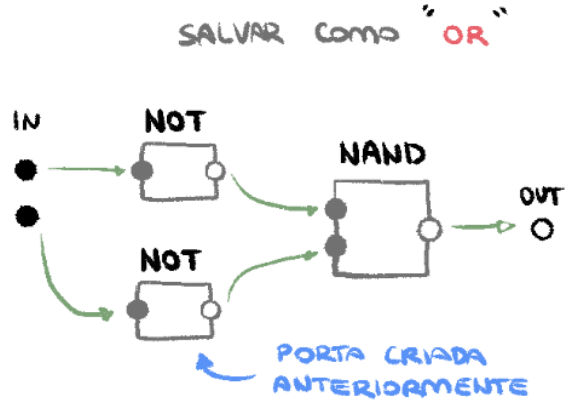


Figura 8 – Esquemático do circuito de uma porta OR

que consiste de um ou mais componentes que adicionam comportamento ou funcionalidade. Portas lógicas e I/O's são configurados como entidades, possuindo componentes como formas básicas visuais (retângulos, círculos, etc.), função lógica e *inputs* e *outputs*, para porta lógica. Além disso, o simulador usufrui de paradigmas como Facade (GURU, 2014–2024a) e Mediator (GURU, 2014–2024b), simplificando a lógica do programa e facilitando a comunicação entre os componentes do circuito. A propagação de sinais, cálculos de portas lógicas, e a solução do circuito em geral é realizada por *managers* (mediadores), enquanto o manejo de eventos de entrada (como movimentos do *mouse*) é gerenciado por classes especializadas, evitando acoplamento apertado do código.

5.2.3 Estudo de viabilidade de alternativa de ferramenta para a simulação

Como alternativa para a simulação, verificou-se a viabilidade de usar a ferramenta Godot, um *game engine* de código aberto feito para o desenvolvimento de jogos. Essa ferramenta tem como vantagem ser um projeto grande e de código aberto, com bastantes recursos online, podendo compilar os projetos desenvolvidos para diversas plataformas como para *web*, *Windows*, *Linux*, etc.

Essa decisão foi tomada com o propósito de testar se o simulador seria mais apropriado na forma de uma aplicação executável, em vez de uma aplicação web, na qual a performance pode ser menor. No entanto, após algumas semanas de experimentação, decidiu-se por não seguir adiante com a implementação do simulador em Godot. Múltiplas razões contribuíram para essa decisão, começando pela curva de aprendizado envolvida. Desenvolver o simulador do zero em *JavaScript*, uma linguagem amplamente conhecida e dominada pela equipe, revelou-se significativamente mais prático e direto do que aprender uma ferramenta completamente nova como Godot em um curto espaço de tempo. Embora Godot ofereça uma estrutura robusta para jogos, suas particularidades, como a arquitetura de nós e cenas, e o uso do *GDScript*, uma linguagem própria da *engine*, exigiriam um esforço extra considerável para adaptar o simulador.

Adicionalmente, a aplicação web original foi planejada para se conectar a um *backend* com banco de dados para salvar e carregar circuitos, uma funcionalidade que é bem mais simples de se implementar em uma aplicação web pura. Com *JavaScript* e bibliotecas já conhecidas, como *Axios* ou *Fetch API*, a comunicação entre o *frontend* e o *backend* é trivial. Já na Godot, a implementação dessa comunicação envolve mais complexidade, sendo necessário lidar com a biblioteca de requisições HTTP e personalizações adicionais, além de aumentar o esforço de integração com serviços externos.

Outro fator decisivo foi a facilidade na publicação de código novo. Em uma aplicação web tradicional, o processo de modificar o código e aplicá-lo em produção é mais direto. Com ferramentas de controle de versão e automação como *Git* e *CI/CD pipelines*, as alterações podem ser rapidamente testadas e publicadas. Em contraste, a Godot, por ser uma *game engine*, tem um fluxo de publicação mais pesado, principalmente quando se trata de *deploy* contínuo em ambientes web, exigindo compilações e testes mais demorados, o que não se alinha com a necessidade de iterações rápidas no desenvolvimento do simulador.

Por essas razões, a escolha mais eficiente foi continuar o desenvolvimento com a tecnologia já familiar e amplamente utilizada no projeto original, garantindo tanto o progresso quanto a simplicidade da implementação.

Deve-se notar que a linguagem também permitiria a implementação do simulador em forma de aplicativo *mobile*, como sugerido pelo professor orientador. Contudo, pelos motivos listados anteriormente, a implementação não seria coerente com o espaço de tempo disponível para finalização do projeto.

5.2.4 Estrutura do banco de dados

O banco de dados é responsável por armazenar as informações de usuários, turmas, atividades e circuitos construídos, além de estabelecer a relação entre eles. Sua estrutura foi desenvolvida através da ferramenta Django e SQlite e sua arquitetura, com as tabelas, colunas e suas relações, está apresentada na figura 9.

5.2.5 Implementação da Aplicação Web

5.2.5.1 Frontend

As figuras apresentadas nas subseções a seguir ilustram o resultado obtido para o visual da aplicação. O guia completo por todas as principais funcionalidades implementadas no site pode ser vista no *storyboard* presente no apêndice A.

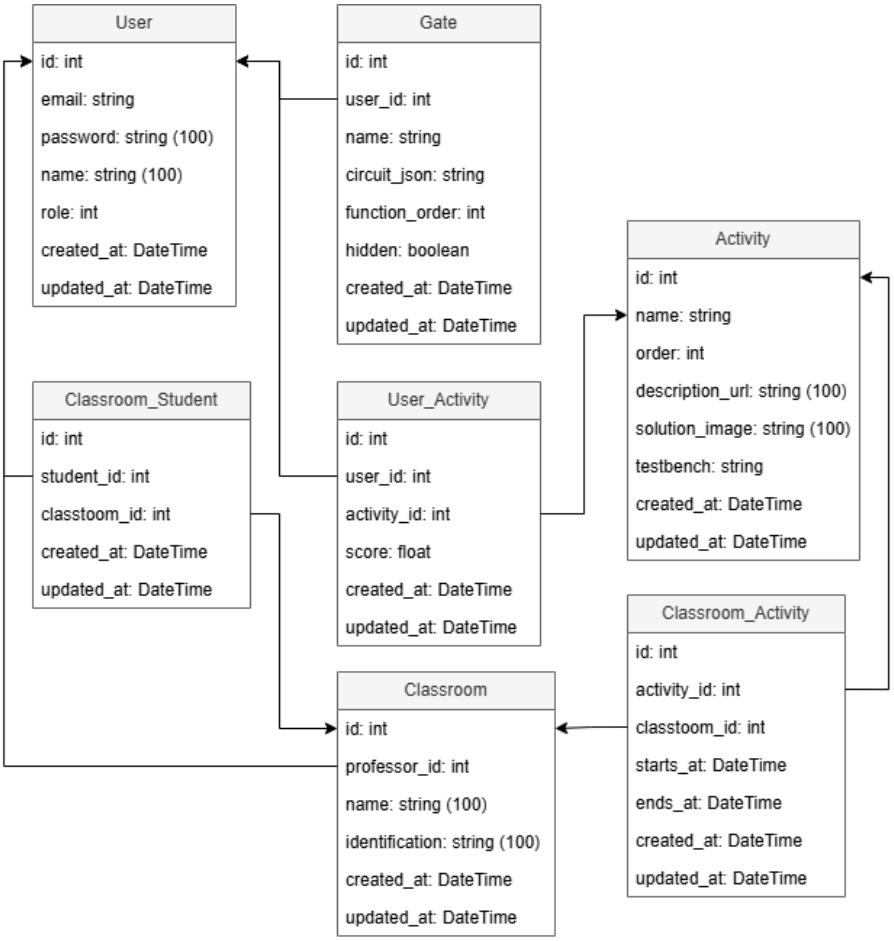


Figura 9 – Estrutura do banco de dados

5.2.5.1.1 Simulador

A parte do simulador possui um *design* simples, apresentando um *dashboard* para a construção do circuito, um menu superior e uma barra de utilidades inferior. No *Dashboard*, do lado esquerdo são apresentadas as entradas e do lado esquerdo as saídas. Na barra de utilidades é encontrada uma lista dos componentes disponíveis para o usuário, que podem ser arrastados para o *dashboard* para incluir eles no circuito. Além disso, há um botão para salvar o circuito como um componente e realizar um "*clear*" no *dashboard*, apagando todo circuito montado. A figura 10 ilustra esse conceito.

Figura 10 – Tela do simulador

5.2.5.1.2 Atividades na visão do aluno

A figura 11 ilustra o visual da lista de atividades. Nela, o cabeçalho possui a opção "Atividades", que apresenta a lista de atividades da turma, assim com a melhora nota do aluno nelas, caso tenha . Na lista é apresentado, para cada atividade seu *status*, sendo "Finalizada" ou "Em andamento", pois outros estados não são apresentados aos alunos. As opções de ver a teoria, ver a solução e enviar circuito ao corretor aparecem conforme o *status* da atividade.

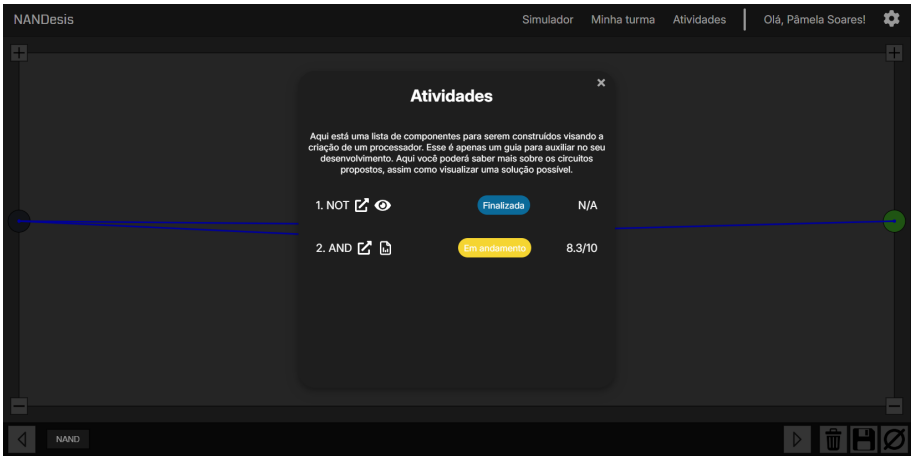


Figura 11 – Tela com lista de atividades na visão do aluno

5.2.5.1.3 Turma na visão do aluno

A figura 12 ilustra o visual das informações da turma para o aluno, que pode ser acessado pela opção "Minha turma" do cabeçalho. Ela conta com dados básicos, como nome da turma e nome e e-mail do professor responsável. Caso o aluno não esteja vinculado a uma turma, é apresentado uma aviso sobre isso.

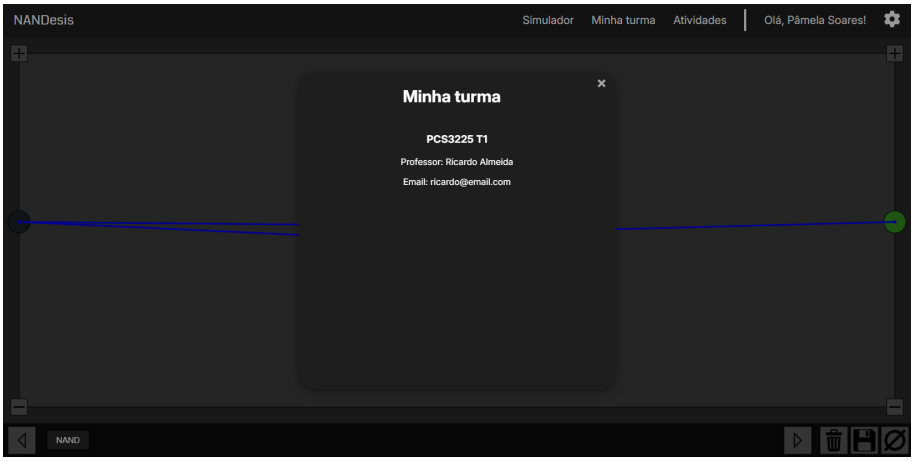


Figura 12 – Tela com informações da turma do aluno

5.2.5.1.4 Lista de turmas

A fugura 13 demonstra a tela principal de listagem de turmas. Ela pode ser acessada apenas pelo professor através da opção "Minhas turmas" do cabeçalho. Nela é encontrada a lista de turmas do professor, assim como a opção de criar e editar suas turmas.

Figura 13 – Tela com lista de turmas do professor

5.2.5.1.5 Detalhe da turma

Nos detalhes da turma apresentados na figura 14, o professor tem acesso completo à lista de alunos e de atividades, podendo adicionar estudantes à turma via link de convite e gerenciar as atividade, definindo data e horário de início e fim da atividade e conferir a lista de notas dos alunos, podendo exportar os resultados em uma planilha Excel.

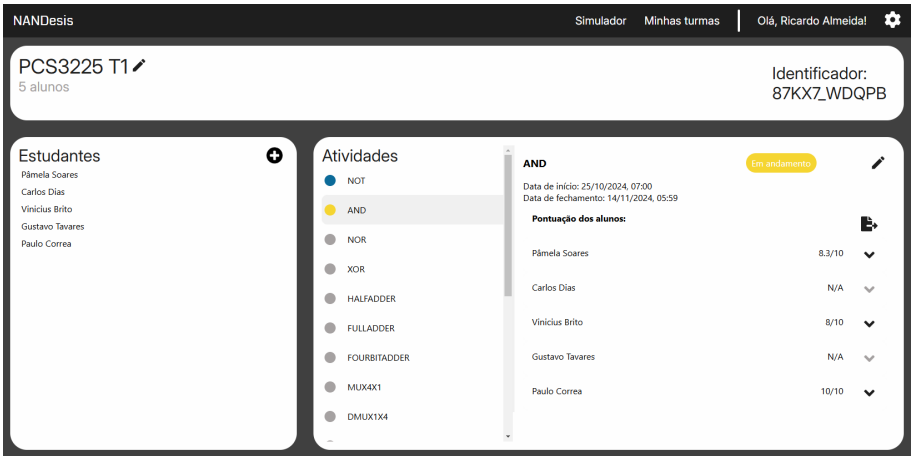


Figura 14 – Tela com informações detalhadas da turma

5.2.5.2 Backend

A seguir são apresentados os endpoints finais configurados no *backend* da aplicação. Todos servem como APIs internas usadas exclusivamente pelo *frontend*.

5.2.5.2.1 Usuário

- **'/register':** Recebe os dados cadastrais e realiza o cadastro do usuário gerando um *token* de acesso. Caso o usuário seja um aluno a o código da turma foi informado, o aluno é vinculado àquela turma;
- **'/login':** Recebe email e senha do usuário validando sua existência no banco e retornando um *token* de acesso;
- **'/userInfo':** Requer autenticação. Baseado no *token* enviado, fornece informações básicas a respeito do usuário.

5.2.5.2.2 Atividade

- **'/listActivities':** Requer autenticação. Lista atividades na visão do estudante, onde é fornecido informações baseadas na turma vinculada. Para usuários sem vínculo com uma turma, a lista de atividades é completa;
- **'/saveActivity':** Requer autenticação e é exclusiva de usuários professores. Permite cadastrar informações de data de início e fim de uma atividade para uma determinada turma;
- **'/activityDetails/<int:classroom_id>/<int:activity_id>':** Requer autenticação e é exclusiva de usuários professores. Baseado nos parâmetros fornecidos pela

url, é fornecido detalhes da atividade, tais como nome, data de inicio e fim e lista de notas dos alunos;

- **'/activitySolution/<int:activity_id>':** Requer autenticação. Baseado no parâmetro fornecido pela url, retorna a imagem de solução da atividade, validando se o usuário possui acesso a ela.

5.2.5.2.3 Circuito

- **'/listCircuits':** Requer autenticação. Lista os circuitos salvos pelo usuário adicionando o circuito NAND básico;
- **'/saveCircuit':** Requer autenticação. Armazena o circuito enviado validando que o nome não exista para aquele usuário;
- **'/deleteCircuit/<int:gate_id>':** Requer autenticação. Altera o *status* do circuito para escondido, não sendo mais apresentado visualmente para o usuário, mas podendo ser usado em outro circuitos que dependem dele.
- **'/judgeCircuit':** Requer autenticação. Salva no banco a nota obtida por um usuário em uma atividade.

5.2.5.2.4 Turma

- **'/listClassrooms':** Requer autenticação e é exclusiva do usuário professor. Lista as turmas ministradas pelo usuário;
- **'/classroomInfo/<int:classroom_id>':** Requer autenticação e é exclusiva do usuário professor. Retorna detalhes básicos e editáveis da turma;
- **'/saveClassroom':** Requer autenticação e é exclusiva do usuário professor. Permite ao usuário alterar informações de uma turma, como o nome;
- **'/classroomDetails/<int:classroom_id>':** Requer autenticação e é exclusiva do usuário professor. Retorna detalhes da turma, incluindo lista de alunos vinculados e lista básica de configuração de atividade;
- **'/classroomStudentInfo':** Requer autenticação. Usada na visão do aluno, retorna informações básicas, como nome da turma e informações do professor, a respeito da turma vinculada ao usuário.

5.2.6 Modificações e refatoração do escopo inicial

Durante o desenvolvimento do projeto, novas ideias surgiram e foram encontrados novos concorrentes com funcionalidades semelhantes ao projeto NANDesis. Dessa forma, o escopo original do projeto sofreu alterações relevantes, sendo as principais delas listadas nessa seção.

5.2.6.1 Ajuste das tecnologias utilizadas

Originalmente, o escopo consistia apenas no simulador, com uma lista de atividades fixa, controlada apenas pelo usuário, sem a mecânica de turmas e área do professor. Nesse caso, o principal elemento de complexidade do projeto era o simulador, desenvolvido usando a biblioteca canvas, que facilita na diagramação visual do simulador, com Javascript Vanilla (WASTL, 2024) no *frontend* e Express.js (OPENJS, 2024) para o *backend* e definição de rotas.

No entanto, analisando o concorrente mais semelhante, o NAND game, tornou-se necessário definir um complemento ao escopo para ser o diferencial do NANDesis e, para isso, as ferramentas do professor foram adicionadas ao plano do projeto. Com o desdobramento delas, aumentaram a complexidade da plataforma que não envolvia o simulador e do banco de dados. Assim, novas tecnologias foram escolhidas para auxiliar e facilitar o desenvolvimento, sendo React (META, 2024) para o *frontend*, devido a familiaridade dos integrantes do grupo ao *framework* e habilidade da ferramenta de componentizar elementos para reaproveitamento; E Django (KIRBY, 2024) para o *backend*, também devida à familiaridade com o produto, mas principalmente à facilidade de manutenção de banco de dados fornecida por ela. Dessa forma, parte do trabalho foi voltado para a refatoração e adaptação do código já existente para utilizar as novas ferramentas.

5.3 Testes e Avaliação

5.3.1 Validação do texto teórico

O texto teórico completo pode ser encontrado no apêndice B. Sua validação consistiu na revisão do professor orientador sobre o material, diante de várias interações e ajustes com o andamento do desenvolvimento do trabalho.

5.3.2 Testes de circuitos

Os resultados completos podem ser vistos no apêndice C. Como forma de validação do funcionamento do simulador, foram testados componentes com diversas características, como circuitos combinatórios simples (portas lógicas NOT, AND, OR, NOR, XOR, etc.),

circuitos combinatórios complexos (multiplexadores e demultiplexadores), circuitos aritméticos (*half-adders* e *full-adders*) e circuitos sequenciais (SR *latch*, D *latch*, D *flip-flop*, registradores, contadores, uma ULA, etc.).

Assim, com base nos resultados observados, percebe-se um funcionamento adequado da ferramenta de simulação, sendo possível construir diversos elementos lógicos comumente utilizados. Ademais, outros elementos não registrados podem ser construídos, como codificadores, *n-bit adders*, diferentes *latches* e *flip-flops*, máquinas de estados, entre outros.

5.3.3 Testes com usuários

Os resultados do formulário foram captados entre os dias 29 de novembro e 2 de dezembro e os dados completos podem ser vistos no apêndice D. No geral, é possível concluir que a ferramenta atende aos parâmetros analisados, visto que a maioria dos critérios apresentaram notas entre 4 e 5. O quesito mais dissonante é o de "acessibilidade", que recebeu nota 3 para o simulador. O comentário recebido, *"Creio que no estado atual de desenvolvimento, é justificável o fator de acessibilidade não estar evoluído."*, indica que o termo "acessibilidade" pode ter sido interpretado como a condição de permitir que toda e qualquer pessoa consiga participar da plataforma de forma igualitária e autônoma, destoando do significado original pensado durante a elaboração do questionário, culminando em uma falha de comunicação no desenvolvimento do formulário. Sob a perceptiva da interpretação conforme o esperado pelo usuário de teste sobe o termo, é importante rever as ações e disposição dos elementos no simulador.

Além disso, foram relatados alguns *bugs* durante os testes, especialmente na parte do simulador. Alguns deles já haviam sido identificados previamente pelos membros do grupo e a lista completa está detalhada na seção 5.4. Por fim, uma sugestão levantada foi a construção de um tutorial inicial para o uso do simulador. Tal ferramenta é amplamente utilizadas em diversos softwares com *dashboard* como uma forma prática de introduzir as ações e possibilidades da plataforma. Por ser uma implementação inicial da plataforma, o desenvolvimento atual não possui tal ferramenta, sendo incluída nos planos de expansão e melhorias futuras da aplicação.

5.4 Bugs e problemas conhecidos

Com a realização de testes internos e com usuários, foram detectados alguns *bugs* com a ferramenta, que não puderam ser corrigidos ou investigados no desenvolvimento do projeto. São eles:

- **Simulador permite múltiplas conexões entre os mesmos inputs:** A prática

leva a alguns *bugs* visuais. O correto seria limitar a apenas uma conexão entre os mesmos inputs;

- **Simulador permite que *outputs* recebam sinas de 2 ou mais inputs:** A prática leva a um comportamento incorreto dos sinais do componente simulado. O correto seria impedir que um *output* pudesse receber mais de um *input*;
- **Simulador impede a seleção de um *input*, mesmo com nenhum outro selecionado:** É apresentado ao usuário a mensagem que dois inputs não podem ser conectados entre si ao selecionar um *input*, mesmo que um nenhum outro *input* esteja selecionado;
- ***Hitbox* dos elementos do *dashboard* incorretos:** O *hitbox* de elementos, como os *inputs*, estão deslocados levemente para direita, equivalendo seu tamanho e incluindo a metade direita do elemento.

6 Considerações Finais

6.1 Conclusões do Projeto de Formatura

O NANDesis, como um projeto de simulador de circuitos lógicos, seria uma alternativa viável para um estudante de computação aos simuladores já existentes, de forma que o sistema seria não apenas uma ferramenta leve e intuitiva, mas também apresentaria funcionalidades didáticas guiadas de forma organizada, diferenciado-se por seus teor mais educacional e gamificado, com diferentes etapas de progressão dos componentes lógicos, podendo evoluir até elementos mais completos, como no caso proposto, o de um processador simples. Aliado a isso, ele fornece um ambiente didático prático e de fácil uso para professores e educadores que podem utilizar a ferramenta como forma de ensino interativo se utilizando dos mecanismos avaliativos oferecidos pelo NANDesis.

6.2 Contribuições

O sistema do NANDesis, apensar de possuir funcionalidades existentes em outros produtos, busca reunir todas em um único serviço, elaborado de seu próprio jeito, sem a busca por implementações de concorrentes. Todo o sistema foi desenvolvido pelos próprios integrantes do grupo, partindo de ideias autênticas para toda organização e arquitetura se inspirando em implementações semelhantes na internet e trabalhos acadêmicos e profissionais já realizados. Como comentado na seção 5.1, as tecnologias escolhidas foram baseadas, em parte, na familiaridade dos integrantes a elas, configurando a base técnica obtida na experiência dos integrantes. Além disso, o material teórico, apresentado de forma integral no apêndice B, foi desenvolvido pelos integrantes, obtendo o conteúdo da literatura, e serve como uma alternativa pública de aprendizado do conteúdo de sistemas digitais.

6.3 Perspectivas de Continuidade

O NANDesis apresenta um esquema funcional, mas básico quanto a ferramentas de ensino dentro da plataforma. Ele permite a implementação de diversas melhorias e ideias descartadas anteriormente, tais como:

- **Edição de circuitos:** Permitir ao usuário altere um circuito já criado, influenciando ou não no funcionamento de circuitos que dependem/usam ele em sua implementação. Seria necessário estabelecer regras sobre alteração de *inputs* e *outputs*, assim como

definir se essa mudança seria de fato propagada aos outros componentes dependentes. É uma função de alta complexidade de implementação, mas com potencial de melhorar a experiência do usuário;

- **Adição de tabelas verdades:** Permite ao estudante gerar uma tabela verdade do circuito construído, facilitando, assim, a comparação com uma tabela verdade esperada, ou simples análise de resultados. Essa ferramenta seria disponibilizada apenas para circuitos abertos;

- **Guia introdutório pela aplicação:** Uma das sugestões levantadas pelo usuários de teste foi a adição de um guia inicial para introduzir o usuário ao simulador e suas ferramentas. Sua implementação é comum no mercado e ensina o usuário a utilizar a aplicação;

- **Criação de atividades personalizadas:** Permite ao professor criar sua própria atividade, fornecendo a teoria ou enunciado, a imagem de gabarito e o *testbench* para a correção automática. Boa parte do *layout* e estrutura pode ser reaproveitado das atividades base atualmente implementados e parte do trabalho será proveniente da validação e padronização dos *scripts* de *testbench* aceitos;

- **Complementar teoria:** Além dos aspectos já presentes na base teórica (abrange componentes essenciais para a criação de um processador), poderiam ser mais detalhados componentes extras e circuitos lógicos adicionais visando expandir o conhecimento dos estudantes. Ademais, existem certos componentes que atualmente não podem ser construídos por limitações técnicas do simulador (ex: inputs de múltiplos bits que extrapolam o limite da área atual, *grids* para a construção de memórias, etc), em que em desenvolvimentos futuros poderiam ser mais detalhados;

- **Adição de *inputs* e *outputs* em cadeia:** Permite ao usuário habilitar um *input* ou *output* para funcionar como uma lista de bits ao invés de ser uma entrada binária. Assim, o usuário poderia penas inserir uma cadeia de bits ou um número hexadecimal na entrada e ler nesse formato numa possível saída. Isso é muito útil para melhorar a experiência do usuário na construção de circuitos mais complexos, como o próprio processador;

- **Aprimoramento da interface e manuseio em dispositivos *touch*:** O modelo atualmente implementado possui melhor desempenho e usabilidade quando utilizado em um dispositivo não *touch*. Como é comum que alunos utilizem dispositivos móveis como *smartphones* e *tablets*, para permitir o acesso à ferramenta, é necessário realizar algumas alterações na captura dos gestos dentro do simulador, pois, atualmente depende do botão direito do mouse, que não possui equivalente em dispostivos *touch*;

- **Apresentar saídas via forma de onda:** Uma ferramenta que possui em simuladores instalados e que poderia auxiliar no ensino e aprendizagem dos alunos é apresentar entradas e saídas via forma de onda;
- **Correção dos *bugs* conhecidos:** A seção 5.4 apresenta a lista de problemas detectados durante o desenvolvimento ou teste da aplicação. Alguns impactam o bom uso do simulador e precisariam ser corrigidos.

Referências

BUCHHOLZ, S. *LogiJS*. 2023. Disponível em: <https://logijs.com/#about>. Acesso em: 15 fev 2024. Citado 2 vezes nas páginas 14 e 17.

BUCHHOLZ, S. *Linkuit Studio*. 2024. Disponível em: <https://linkuit.com/#about>. Acesso em: 24 set 2024. Citado 2 vezes nas páginas 14 e 17.

BURCH, C. *LogiSim*. 2011. Disponível em: <http://www.cburch.com/logisim/>. Acesso em: 24 set 2024. Citado 2 vezes nas páginas 14 e 18.

CHERNEV, A.; BöCKENHOLT, U.; GOODMAN, J. *Choice overload: A conceptual review and meta-analysis*. Journal of Consumer Psychology, 2015. v. 2. 333-358 p. Disponível em: <https://doi.org/10.1016/j.jcps.2014.08.002>. Acesso em: 16 fev 2024. Citado 2 vezes nas páginas 14 e 32.

DAHL, R. *Node.js*. 2024. Disponível em: <https://nodejs.org/en/about>. Acesso em: 28 out 2024. Citado na página 37.

ENDERTON, H. B. *A Mathematical Introduction to Logic*. 2. ed. [S.l.]: Harcourt Academic Press, 2001. Acesso em: 12 nov 2024. Citado na página 17.

GURU, R. *Facade*. 2014–2024. Disponível em: <https://refactoring.guru/design-patterns/facade>. Acesso em: 05 fev 2024. Citado na página 41.

GURU, R. *Mediator*. 2014–2024. Disponível em: <https://refactoring.guru/design-patterns/mediator>. Acesso em: 02 fev 2024. Citado na página 41.

HAT, B. *Logic.ly*. 2023. Disponível em: <https://logic.ly/>. Acesso em: 24 set 2024. Citado 2 vezes nas páginas 14 e 18.

HORTON, K. *NANDputer running its first program - kevtris*. 2012. Disponível em: <https://www.youtube.com/watch?v=zIOiiTpCiwM>. Acesso em: 15 mar 2024. Citado na página 17.

KIRBY, C. *Django*. 2024. Disponível em: <https://www.djangoproject.com/foundation/>. Acesso em: 28 out 2024. Citado 2 vezes nas páginas 37 e 48.

KJæR, O. J. *The Nand Game*. 2022. Disponível em: <https://nandgame.com/about>. Acesso em: 13 out 2024. Citado 2 vezes nas páginas 14 e 17.

KONWAR, S. *CircuitVerse*. 2024. Disponível em: <https://circuitverse.org/>. Acesso em: 16 fev 2024. Citado 2 vezes nas páginas 14 e 17.

META. *React*. 2024. Disponível em: <https://react.dev/>. Acesso em: 28 out 2024. Citado 2 vezes nas páginas 37 e 48.

NEEMANN, H. *Digital*. 2024. Disponível em: <https://github.com/hneemann/Digital>. Acesso em: 04 fev 2024. Citado 3 vezes nas páginas 14, 18 e 40.

NISAN, N.; SCHOCKEN, S. *The Elements of Computing Systems: Building a Modern Computer from First Principles*. [S.l.]: MIT press, 2005. Acesso em: 26 fev 2024. Citado na página 13.

NISAN, N.; SCHOCKEN, S. *From Nand to Tetris*. 2017. Disponível em: <https://www.nand2tetris.org/course>. Acesso em: 21 fev 2024. Citado na página 40.

NOLT, J.; ROHATYN, D.; VARZI, A. *Schaum's outline of theory and problems of logic*. 2. ed. [S.l.]: McGraw-Hill, 1998. Acesso em: 15 mar 2024. Citado na página 17.

OPENJS. *Express.js*. 2024. Disponível em: <https://expressjs.com/pt-br/>. Acesso em: 28 out 2024. Citado na página 48.

PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design RISC-V Edition*. 2. ed. [S.l.]: Morgan Kaufmann, 2014. Acesso em: 11 nov 2024. Citado 2 vezes nas páginas 99 e 107.

PLANTZ, R. G. *Introduction to Computer Organization with x86-64 Assembly Language*. 2019. Disponível em: <https://bob.cs.sonoma.edu/IntroCompOrg-x64/bookch5.html>. Acesso em: 29 out 2024. Citado 2 vezes nas páginas 97 e 98.

TERRA, J. *Entity Component System: An Introductory Guide*. 2023. Disponível em: <https://www.simplilearn.com/entity-component-system-introductory-guide-article>. Acesso em: 30 jan 2024. Citado na página 40.

WAKERLY, J. F. *Digital Design Principles And Practices*. 4. ed. [S.l.]: Prentice Hall, 2006. Acesso em: 26 fev 2024. Citado na página 13.

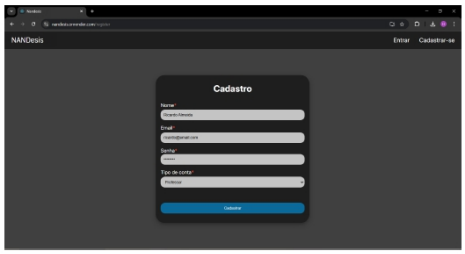
WASTL, E. *Javascript Vanilla*. 2024. Disponível em: <http://vanilla-js.com/>. Acesso em: 28 out 2024. Citado na página 48.

Apêndices

APÊNDICE A – NANDesis StoryBoard

Para ilustrar o funcionamento e as ferramentas presentes no NANDesis, é apresentado seus principais fluxos no formato de *StoryBoard*.

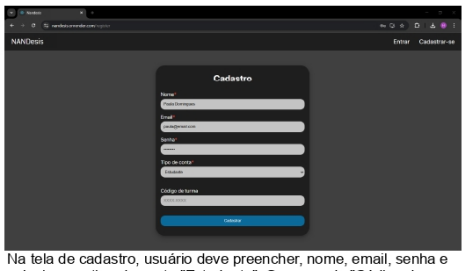
Cadastro de professor na plataforma



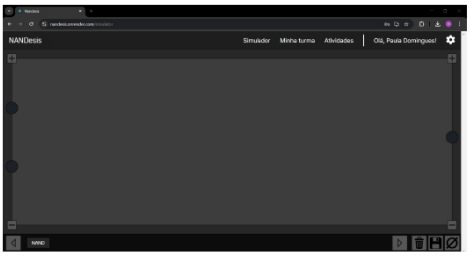
Na tela de cadastro, usuário deve preencher, nome, email, senha e selecionar o tipo de conta "Professor". Então, deve selecionar o botão "Cadastrar".

Após o cadastro, ele será redirecionado a tela "minhas turmas", onde está listado todas as turmas daquele professor.

Cadastro de aluno na plataforma

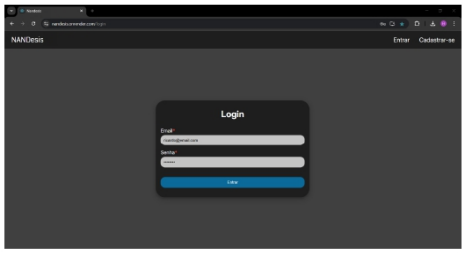


Na tela de cadastro, usuário deve preencher, nome, email, senha e selecionar o tipo de conta "Estudante". O campo de "Código de turma" não é obrigatório e é usado para vincular o novo aluno a uma turma. Então, deve selecionar o botão "Cadastrar".



Após o cadastro, ele será redirecionado a tela de do simulador, onde o aluno pode iniciar o desenvolvimento de seu circuito.

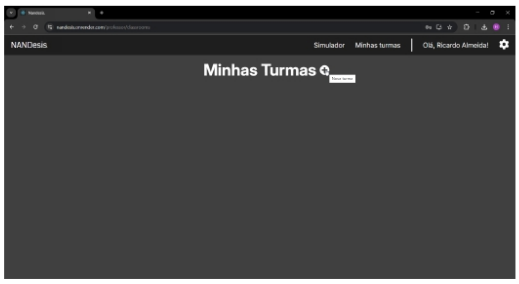
Login



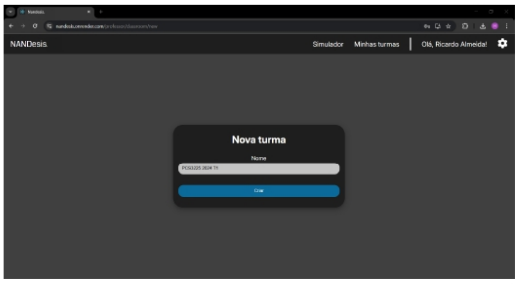
Na tela de login, usuário deve preencher, seu email e senha previamente cadastrados. Então, deve selecionar o botão "Entrar".

Após o cadastro, caso seja um professor, ele será redirecionado a tela de "minhas turmas", e, caso seja aluno, será direcionado ao simulador.

Cadastro e edição de turmas

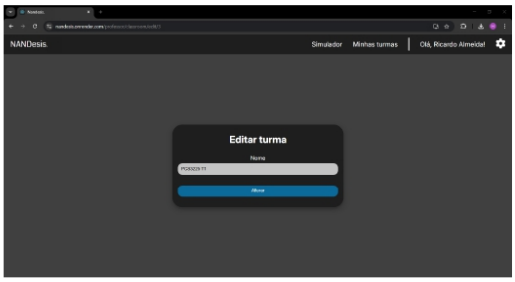


Na tela de "Minhas turmas", ao lado do título, ao selecionar o botão +, o professor será redirecinado para a tela de nova turma.

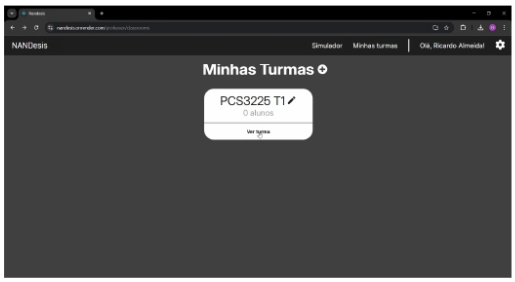


Na tela de nova turma, o professor pode preencher o nome da nova turma e seleciona o botão "Criar" para cadastrar ela.

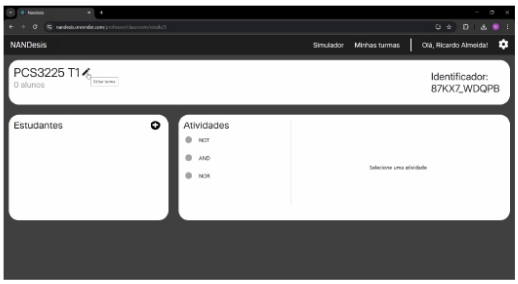
Após o cadastro, o usuário retorna a lista de turmas atualizada com o novo registro. Para editar uma turma, basta selecionar o ícone do lápis.



Na tela de edição de turma, o professor pode alterar o valor do campo de nome e seleciona o botão "Alterar" para editar a turma.

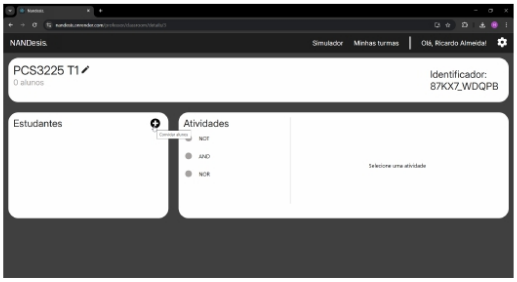


Após a edição, o usuário retorna a lista de turmas atualizada com o ajuste feito. O usuário pode, então, ver detalhes da turma em "Ver turma".

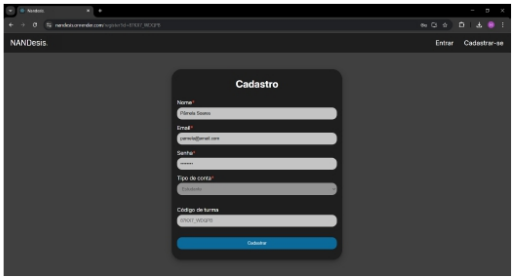


Nos detalhes da turma, o usuário pode clicar no ícone de lápis para editar a turma e ser redirecionado à tela de edição apresentada anteriormente. O botão "Minhas turmas", no caabeçãlho faz o usuário retornar à lista de turmas.

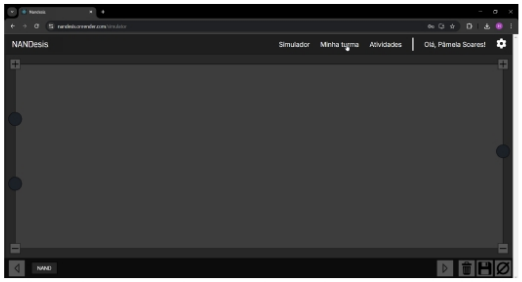
Convite de aluno para participar da turma



O professor, na tela de detalhes da turma, deve selecionar o botão + ao lado da lista de estudantes, no card inferior esquerdo. Assim, o link de convite é copiado para o *clipboard* do professor.

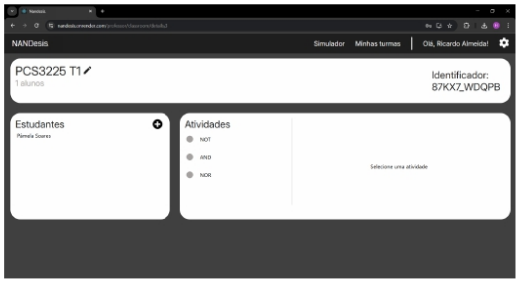


O aluno pode acessar o link recebido e preencher suas informações de nome, email e senha para o cadastro. O campo de tipo de conta e o código da turma já vem prepreenchido. Ao selecionar o botão "Cadastrar", uma nova conta para esse aluno é criada.



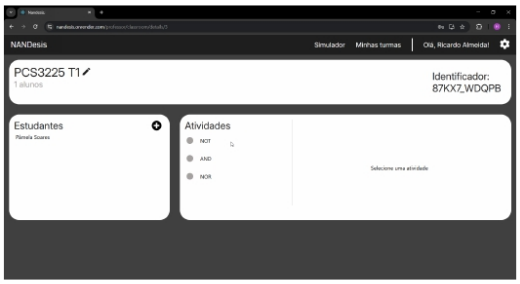
Após o cadastro, o estudante é redirecionado à tela do simulador. No cabeçalho da página, a opção "Minha turma" irá fornecer informações da turma vinculada a ele.

Um modal será aberto com informações da turma do aluno. Para fechar, basta clicar fora do modal ou selecionar o botão X no canto superior direito dele.

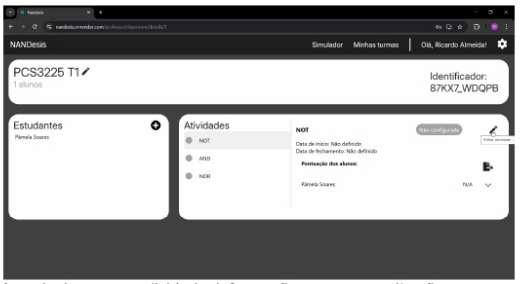


Após o cadastro do aluno, a lista de estudantes é atualizada para o professor, informando o novo aluno.

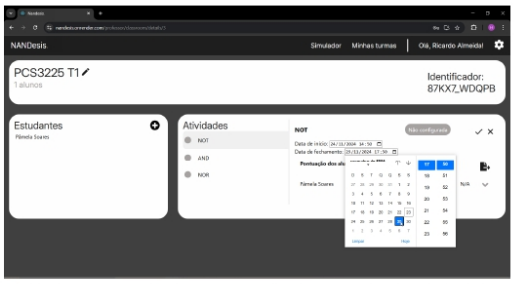
Gerenciamento de atividades



O professor, na tela de detalhes da turma, deve selecionar uma atividade na lista de atividades do card inferior direito.

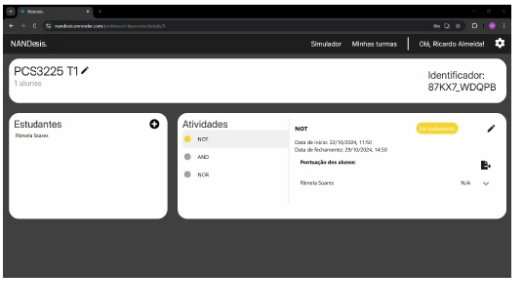


Ao selecionar uma atividade, informações a seu respeito são apresentadas na tela. Ao clicar no icone de lápis, o professor pode alterar alguma delas. Como não há datas de início e fim definidas, o status é "Não configurado".

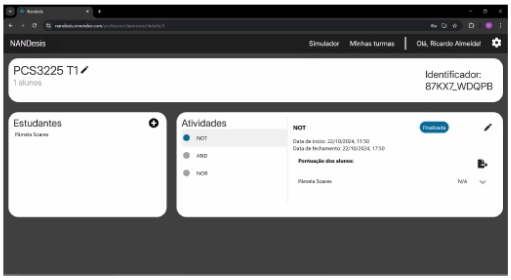


O professor pode, então, inserir a data de início e fim da atividade. Para salvar as alterações, basta clicar no botão de check no canto superior direito, e, caso queira cancelar, basta clicar no X.

Após salvar, o status da atividade para aquela turma é alterado dependendo das datas inseridas. No exemplo, visto que a data inicial é posterior à data atual, o status é "Indisponível".

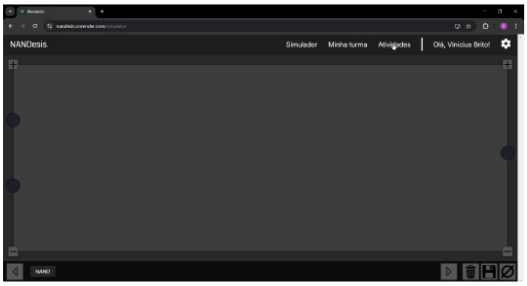


Exemplo onde a data atual está entre a data inicial e final da atividade. Neste caso o status é "Em andamento".

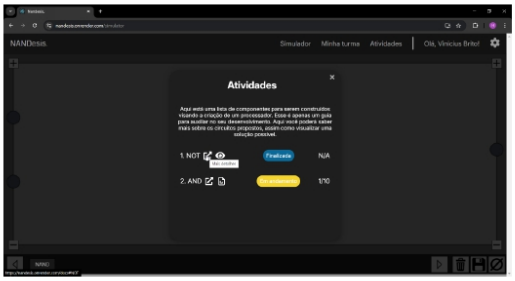


Exemplo onde a data atual é posterior à final da atividade. Neste caso o status é "Finalizado".

Acesso à teoria da atividade



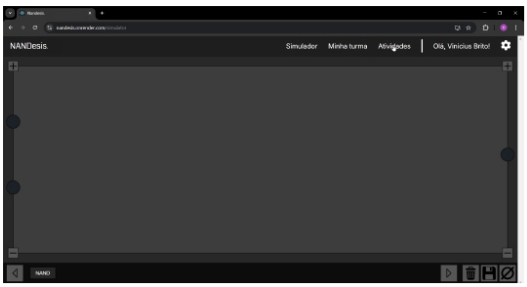
Na tela do simulador, o usuário pode acessar as atividades disponíveis no botão "Atividades" do cabeçalho da página.



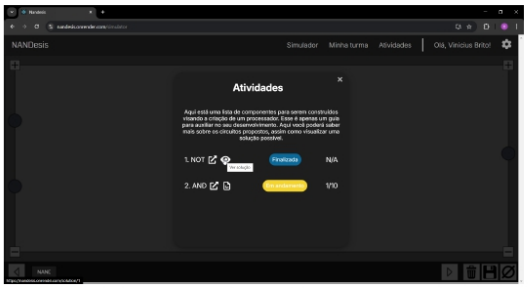
Na lista de atividades, o usuário pode clicar no ícone de externo para ser redirecionado à seção daquele componente na página de teoria.

Na página de teoria, o usuário pode encontrar uma descrição mais detalhada do componente solicitado.

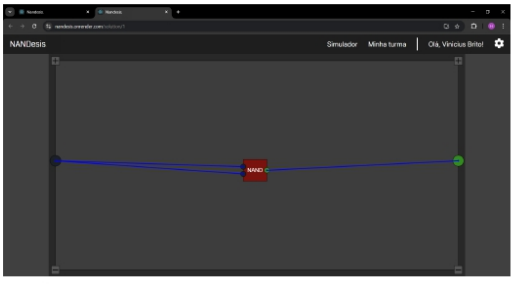
Acesso à solução da atividade



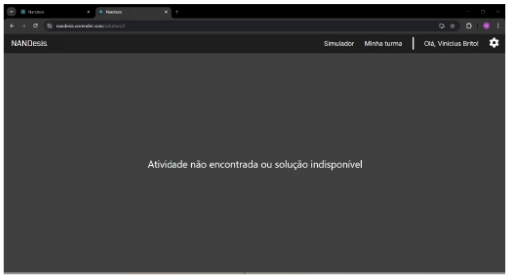
Na tela do simulador, o usuário pode acessar as atividades disponíveis no botão "Atividades" do cabeçalho da página.



Na lista de atividades, para atividades finalizadas, o usuário pode clicar no ícone do olho para ser redirecionado á página com a solução da atividade.

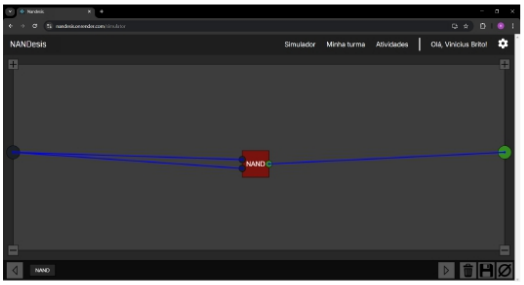


Na página de solução, caso o usuário tenha acesso, uma imagem do circuito será apresentado.

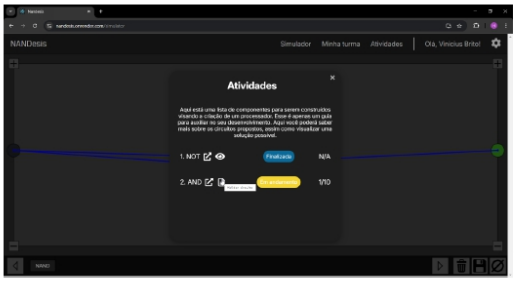


Na página de solução, caso a atividade não esteja finalizada ou seja inválida, uma aviso será apresentado.

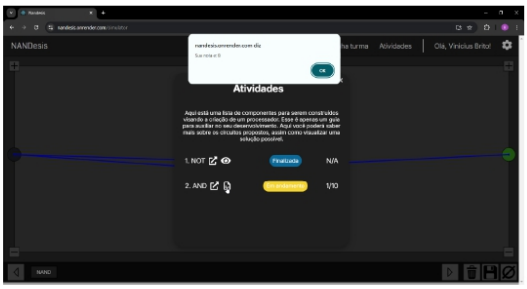
Envio de componente para correção



Na tela do simulador, com um circuito montado, o usuário pode acessar as atividades disponíveis no botão "Atividades" do cabeçalho da página.

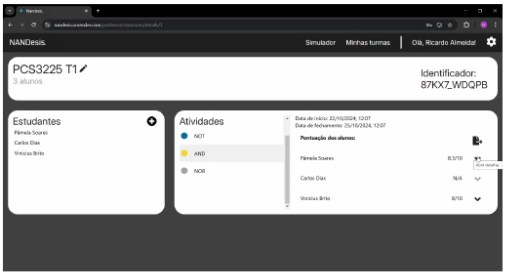


Na lista de atividades, para atividades em andamento, o usuário pode clicar no ícone da folha para enviar o circuito atual aberto no simulador para a correção.

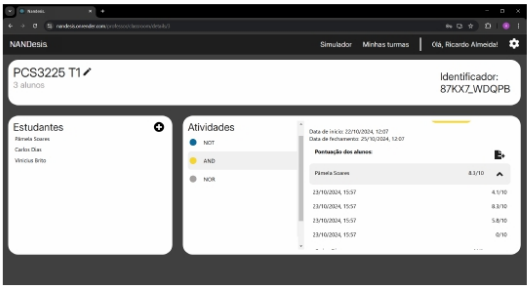


Após o processamento, a nota obtida é informada ao usuário, e sua maior nota é atualizada na listagem.

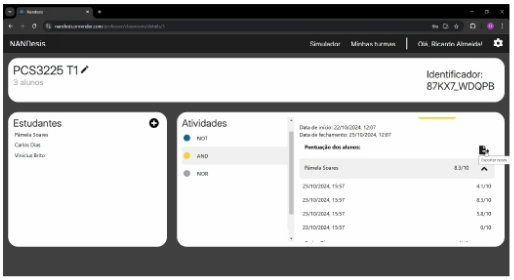
Lista de notas dos alunos de uma atividade



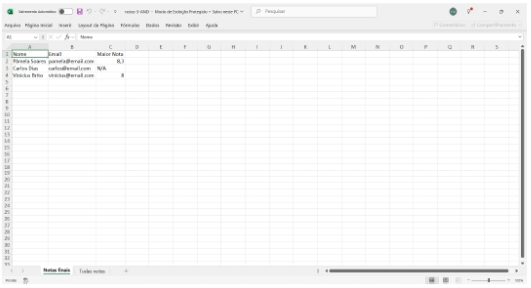
Na tela de detalhes da turma, nos detalhes das atividades, o professor encontra uma lista de alunos informando a maior nota de cada aluno.



O porfessor pode expandir os alunos para ver detalhes da nota de cada envio realizado por eles.

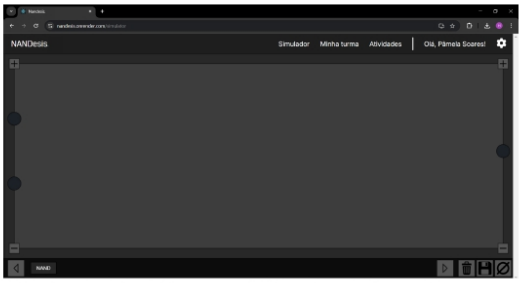


O ícone de arquivo permite ao professor exportar as notas dos alunos em uma planilha .xlsx.

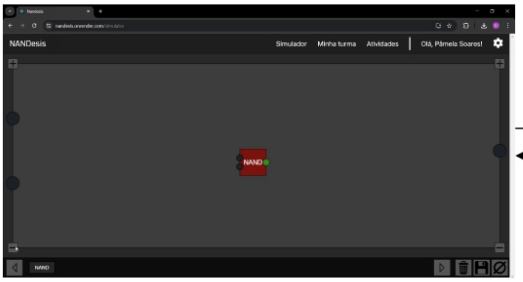


A planilha é composta por 2 páginas, a "Notas Finais", com as maiores notas de cada aluno, e a "Todas Notas", com todo histórico de envios daquela atividade.

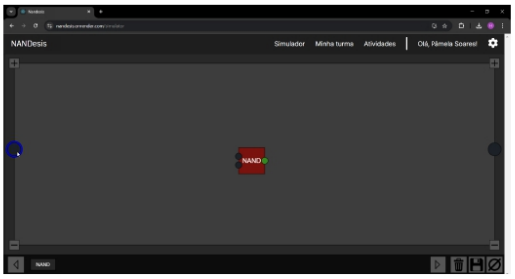
Uso do simulador



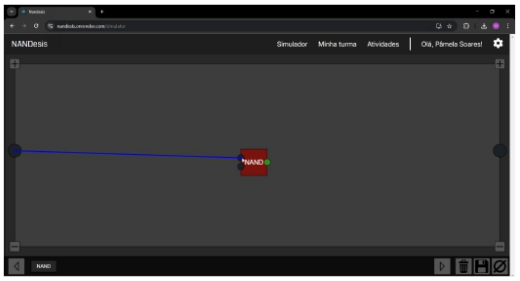
A tela do simulador inicia com o *dashboard* vazio. Para adicionar um componente, o usuário seleciona ele na barra inferior da tela, onde temos a lista de componentes, no caso o NAND.



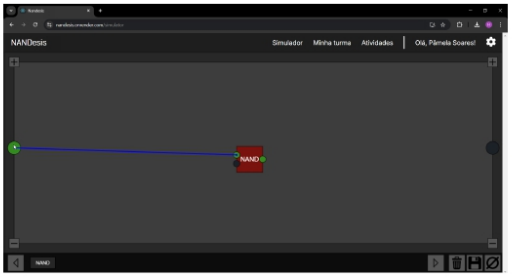
É possível configurar o número de *inputs* e *outputs* por meio dos símbulos + (Adicionar *input/output*) e - (Remover *input/output*) nos cantos do *dashboard*.



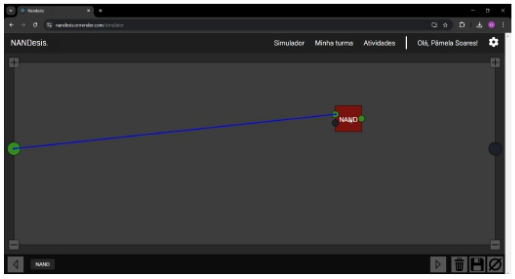
Para realizar uma conexão é necessário selecionar 2 entradas. Com o botão direito do mouse, ao selecionar uma entrada, um *highlight* azul aparece sobre ela.



Clicando com o botão direito do mouse sob outra entrada, uma conexão é estabelecida entre elas.

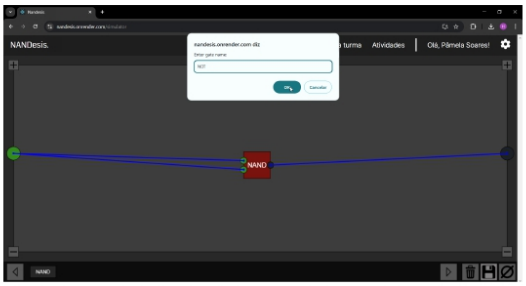


Para ativar ou desativar um *input*, basta clicar com o botão esquerdo do mouse sobre ele. A cor cinza indica ele estar desativado (Valor lógico 0) e verde, ativado (Valor lógico 1)

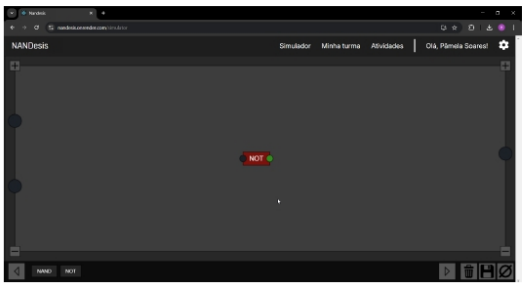


É possível alterar a posição de um componente no *dashboard* selecionando, segurando e arrastando ele pela tela, permitindo o usuário posicionar-lo no local desejado.

Salvando um circuito

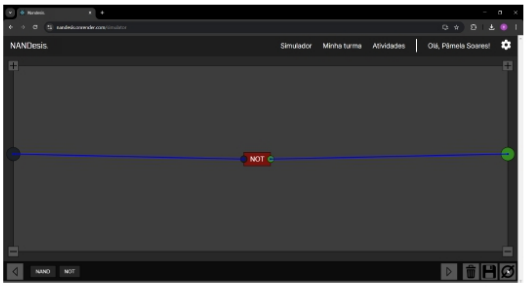


Com o circuito desenvolvido, para salvar ele, basta selecionar o botão com o disquete no canto inferior direito da tela, informar o nome do componente e selecionar "Ok".

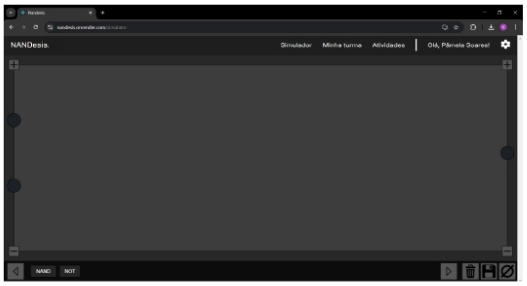


Após salvar a nova porta, a tela será recarregada e o novo componente aparecerá na lista inferior e poderá ser colocado no *dashboard* para uso.

Limpando o dashboard



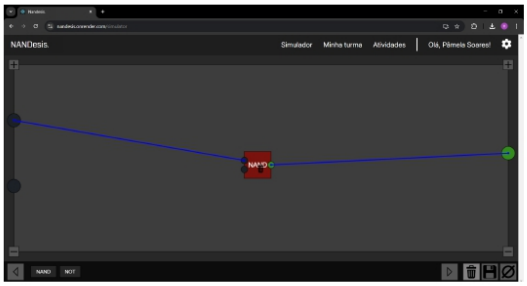
Para limpar o *dashboard* completo, basta selecionar o botão de *clear* no canto inferior direito da tela e confirmar a ação.



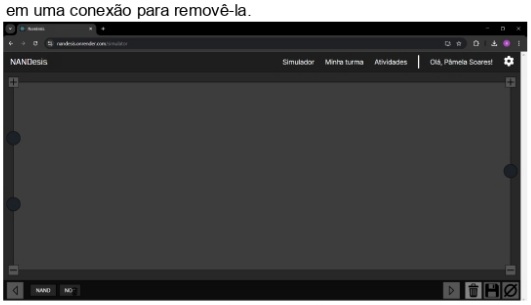
Isso forçará um reload da tela, retornando à configuração inicial do dashboard.

Apagando elementos do dashboard

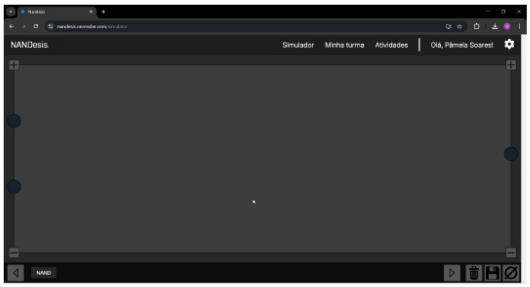
Para habilitar o modo de deleção, basta selecionar o botão de lixeira no canto inferior direito. Durante esse modo, o cursor se tornará uma lixeira e tudo que for tocado será apagado. No caso, é possível clicar em uma conexão para removê-la.



É possível remover componentes do dashboard clicando neles nesse modo.

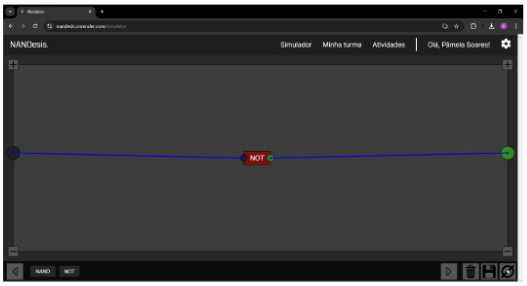


Para apagar um componente de sua lista de componentes, basta clicar sobre eles no modo de deleção. A porta NAND original não pode ser apagada.

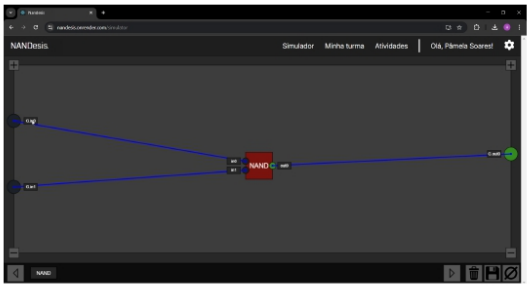


Após apagar uma porta, a tela é recarregada e a lista de componentes é atualizada. Para sair do modo de deleção, basta selecionar o botão com a lixeira novamente.

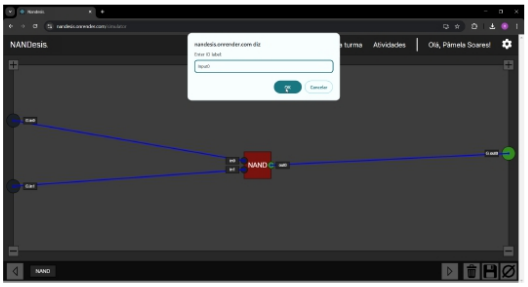
Utilizando labels nos inputs



Para apresentar ou esconder as *labels* no simulador, basta clicar na tecla "H" do teclado.



Nome dos *inputs* aparentes.



Selecionando um *input* é possivel alterar seu nome.

Nome do *input* "G.in0" após atualização.

APÊNDICE B – Teoria fornecida ao aluno

B.1 Algebra Booleana

É uma forma de representar equações cujas variáveis apenas podem assumir valores binários (ex: 1 ou 0, verdadeiro ou falso, etc.), fazendo uso de operações lógicas simples para manipular estes valores. Essas operações são a negação (NÃO ou $\sim$), a conjunção (E ou $+$) e a disjunção (OU).

A aplicação dela se dará no contexto de circuitos lógicos, em que a partir de estruturas lógicas básicas, é possível construir elementos lógicos mais complexos, como processadores. Essas portas serão estudadas mais detalhadamente na seção de circuitos combinatórios.

Um dos aspectos fundamentais para compreender a álgebra booleana é via a aplicação de identidades, similarmente ao que é observado na matemática. As principais para o entendimento são as propriedades aditivas e as multiplicativas, referentes aos conceitos de disjunção e conjunção, respectivamente.

Identidades aditivas	A + 1 = 1	A OR 1 = 1
	A + 0 = 0	A OR 0 = 0
	A + A = 0	A OR A = A
	A + ~ A = 0	A OR NOT A = 0
Identidades multiplicativas	A × 1 = 1	A AND 1 = 1
	A × 0 = 0	A AND 0 = 0
	A × A = A	A AND A = A
	A × ~ A = 0	A AND NOT A = 0
	~ (~ A) = A	NOT (NOT A) = A

OBS: Existem diversas formas de escrever equações booleanas, por exemplo, a representação A $\times$ A também pode ser escrita como AA, com $\sim$A também podendo ser equivalente a $\bar{A}$.

Outro aspecto fundamental a ser aprendido são as leis relativas às propriedades de equações lógicas: comutatividade, associatividade e distributividade.

Por fim, a última ferramenta a ser apresentada é o teorema de De Morgan: A

Comutatividade	A+B = B+A	A OR B = B OR A
	AB = BA	A AND B = B AND A
Associatividade	A+(B+C) = (A+B)+C	A OR (B OR C) = (A OR B) OR C
	A(BC) = (AB)C	A AND (B AND C) = (A AND B) AND C
Distributividade	A(B+C) = AB+AC	A AND (B OR C) = (A AND B) OR (A AND C)

negação (ou complemento) de uma conjunção (produto - E) de variáveis é igual a disjunção (soma - OU) dos complementos das variáveis, assim como o complemento ou negação de uma disjunção de variáveis é igual a conjunção dos complementos das variáveis.

Teorema de De Morgan	NOT (A AND B) = (NOT A) OR (NOT B)
	NOT (A OR B) = (NOT A) AND (NOT B)

Com esse conhecimento, é possível aplicar as regras de identidade junto a essas propriedades de forma a simplificar equações booleanas, para assim facilitar a construção de futuros circuitos lógicos.

B.1.1 Tabela verdade e expressões booleanas

O próximo passo para a construção de circuitos digitais é a transformação de uma tabela verdade em uma expressão booleana.

Essa representação pode ser efetuada de diversas formas, mas uma metodologia simples de ser feita é via a transformação dos inputs de cada linha da tabela verdade em um fragmento de uma expressão booleana, assim representando todos os possíveis outputs do circuito.

Para essa resolução, geralmente estes fragmentos são os mintermos e maxtermos, que serão comentados a seguir:

B.1.2 Mintermos e Maxtermos

Uma forma eficiente de converter uma tabela verdade em uma equação lógica simplificada é via o uso da soma de mintermos e do produto de maxtermos. Para isso, uma função/equação lógica é reescrita de forma a destacar as variáveis no formato de soma de produtos (mintermos) ou no produto de somas (maxtermos) via o estudo dos possíveis valores de input e outputs.

B.1.2.1 Mintermos

Para o caso de mintermos, cada linha deve ser transformada em uma multiplicação de variáveis de entrada (ou seja, E), em que caso o valor de entrada seja 1, o valor da variável de entrada não deve ser negada, e caso o valor de entrada seja 0 ela deve ser invertida.

A partir disso, cada um desses mintermos com valor de saída equivalente a 1 deve ser somado (ou seja, OU) para então formar uma equação de álgebra booleana que poderia ser consequentemente simplificada via os métodos estudados previamente.

B.1.2.2 Maxtermos

Para o caso de maxtermos, o inverso deve ser feito, onde cada linha deve ser transformada em uma soma de variáveis de entrada (ou seja, OU), em que caso o valor de entrada seja 0, o valor da variável de entrada não deve ser negada, e caso o valor de entrada seja 1 ela deve ser invertida.

A partir disso, cada um desses mintermos com valor de saída equivalente a 0 deve ser multiplicado (ou seja, E) para então formar uma equação de álgebra booleana que poderia ser consequentemente simplificada via os métodos estudados previamente.

B.1.3 Exemplo Mintermo e Maxtermo

Um exemplo de mintermos e maxtermos pode ser observado na seguinte tabela:

A	B	C	Maxtermo	Mintermo
0	0	0	~ A × ~ B × ~ C	A + B + C
0	0	1	~ A × ~ B × C	A + B + ~ C
0	1	0	~ A × B × ~ C	A + ~ B + C
0	1	1	~ A × B × C	A + ~ B + ~ C
1	0	0	A × ~ B × ~ C	~ A + B + C
1	0	1	A × ~ B × C	~ A + B + ~ C
1	1	0	A × B × ~ C	~ A + ~ B + C
1	1	1	A × B × C	~ A + ~ B + ~ C

Assim, a tabela verdade pode ser representada pela soma dos mintermos ou produto dos maxtermos.

B.1.4 Exemplo Mintermo e Maxtermo

Os métodos mencionados previamente são interessantes por um ponto de vista teórico, mas podem não ser adequados para circuitos mais complexos. Assim pode ser estudado um método alternativo: Mapas de Karnaugh.

Também chamados de K-maps, são diagramas utilizados para converter uma tabela verdade em sua representação de circuito lógico ou simplificar uma equação lógica. Para isso, será necessário um conhecimento prévio sobre o tema de conjuntos, haja vista que a mesma lógica aplicada a diagramas de Venn é observada para a álgebra booleana.

Uma figura ilustrativa de um mapa de Karnaugh pode ser observada abaixo.

B / A	NOT B	B
NOT A	(NOT A) AND (NOT B)	(NOT A) AND B
A	A AND (NOT B)	A AND B

A sua construção se baseia nos possíveis valores obtidos das variáveis da função booleana, com construções com mais de duas variáveis sendo também possíveis (ex: com as variáveis A, B, C e D, as colunas são relativas às entradas A e B, e as linhas são equivalentes as possíveis permutações de C e D).

AB / CD	00	01	11	10
00				
01				
11				
10				

As células que compõem as tabelas devem ser preenchidas com o valor lógico de cada linha da tabela verdade referente a função a ser simplificada (0 e 1).

Uma vez com o mapa construído, é possível agrupar valores equivalentes a 1, gerando uma soma de elementos (ex: soma de produtos), possibilitando identificar outputs

A	B	Saída
0	0	W
0	1	X
1	0	Y
1	1	Z

A \ B	0	1
0	W	X
1	Y	Z

que resultam na equação booleana mais simples possível referente a função inicial, com a mesma lógica se aplicando ao produto de somas (mas selecionando células com 0). Ademais, como princípio, quanto maior for o agrupamento, maior será a simplificação obtida.

Uma propriedade interessante dos mapas de Karnaugh é a capacidade de "unir" arestas opostas de um mapa, possibilitando que laterais opostas possam ser agrupadas.

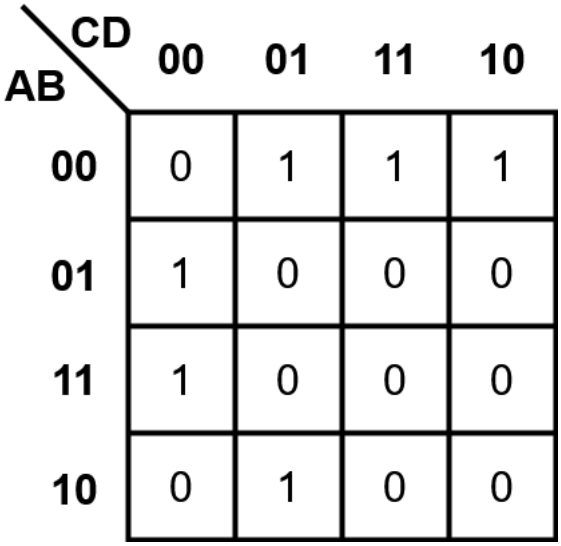
AB \ CD	00	01	11	10
00	0	0	1	0
01	1	0	1	1
11	1	0	1	1
10	0	0	1	0

Saída = C AND D + NOT D AND B

Uma vez com os maxtermos (ou mintermos) escolhidos, é relativamente mais simples realizar a simplificação da equação booleana para então construir um circuito lógico.

B.1.5 Exercício Mintermo e Maxtermo

Minimize o seguinte mapa de Karnaugh de forma a encontrar a melhor simplificação, explicitando se a forma encontrada foi via mintermos ou maxtermos:



Solução:

A simplificação foi encontrada via a soma de produtos, sendo ela:

$$Saída = (B \cdot \neg C \cdot \neg D) + (\neg B \cdot \neg C \cdot D) + (\neg A \cdot \neg B \cdot C)$$

Deve-se notar que a utilização do produto de somas resultaria um uma simplificação inferior a encontrada via soma de produtos.

B.2 Circuitos Combinatórios

OBS: Com relação às terminologias utilizadas nas seguintes descrições, podem ser consideradas equivalentes neste contexto:

Verdadeiro/True	Falso/False
1	0
High	Low
Ativo	Inativo

B.2.1 NAND

A porta NAND é a porta universal inicial do NANDesis, essa porta lógica já foi fornecida no simulador e com ela é possível construir todos os componentes necessários para se criar um processador.

Seu funcionamento é simples: ela possui duas (ou mais) entradas e uma saída, com o valor da saída sendo representada por "NOT (**(entrada A)** AND **(entrada B)**)", ou seja, a saída possui valor falso (0) quando ambas as suas entradas A e B forem verdadeiras (1), e quaisquer outras entradas retornará uma saída verdadeira.

Entrada A	Entrada B	Saída
0	0	1
0	1	1
1	0	1
1	1	0

É possível representar sua tabela verdade da seguinte forma:

A partir deste conhecimento, é possível construir outras portas lógicas e componentes. Assim, os itens seguintes vão atuar como introduções teóricas para aumentar o seu repertório e permitir a criação de diversos circuitos lógicos, permitindo até mesmo atingir o nível de um processador.

B.2.2 NOT

A porta lógica NOT é de grande utilidade, podendo, em essência, inverter um sinal lógico. Ou seja, uma entrada verdadeira (1) terá uma saída falsa (0), com a mesma lógica sendo aplicada para uma entrada falsa.

É possível representar sua tabela verdade da seguinte forma:

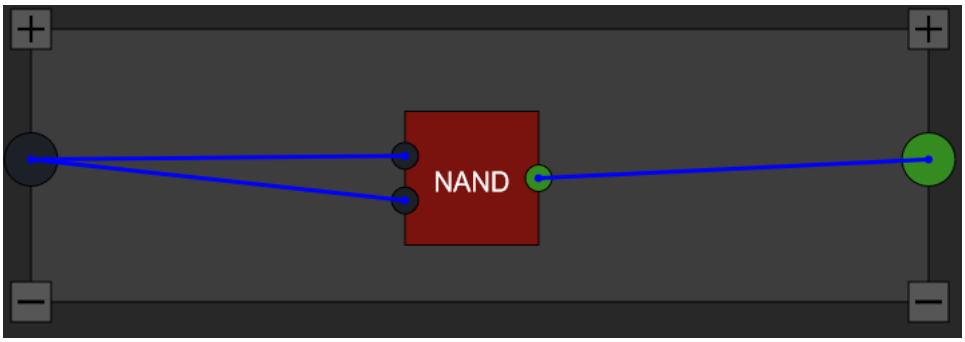
Entrada A	Saída
0	1
1	0

Com base nisso, já é possível começar a imaginar como seria a criação da porta NOT com uma NAND atuando como sua base. Observa-se na tabela verdade do item anterior um comportamento similar ao desejado:

Entrada A	Entrada B	Saída
0	0	1
0	1	1
1	0	1
1	1	0

Ao alimentar simultaneamente um mesmo sinal nas entradas A e B do elemento NAND, a saída encontrada é o inverso do sinal inicial. Sendo possível, portanto, criar a porta NOT via a manipulação das entradas de uma porta NAND.

Como exemplo, a criação no simulador seria executada da seguinte forma:



Agora contendo duas portas lógicas ao seu dispor, este mesmo tipo de exercício lógico pode ser aplicado para a criação de outros componentes.

B.2.3 AND

A porta AND pode ser considerada o inversa da porta NAND: ela também possui duas (ou mais) entradas e uma saída, com o valor da saída sendo representada por "**(entrada A)** E **(entrada B)**", ou seja, a saída possui valor verdadeiro (1) quando todas as entradas forem verdadeiras (1), e quaisquer outras variações de inputs retornarão uma saída falsa.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Observa-se, portanto, que o comportamento desejado pode ser obtido via manipulação de múltiplas portas NAND concatenadas, com a saída da primeira alimentando as duas entradas da segunda porta (ou seja, o equivalente a adição de uma porta NOT na saída de uma NAND).

```
A AND B = NOT (NAND (A, B)) , sendo que: NAND (A, B) = NOT (A AND B)

Portanto,

A AND B = NOT (NOT (A AND B))
```

B.2.4 OR

A porta OR é construída com duas ou mais entradas, em que caso pelo menos uma entrada seja verdadeira, a saída também retornará 1. Ou seja, o valor da saída é definido via "**(entrada A)** OU **(entrada B)**" no caso de uma porta com duas entradas.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Saída
0	0	0
0	1	1
1	0	1
1	1	1

Para induzir o seu desenvolvimento a partir de uma porta NAND, é possível estudar as propriedades de sua tabela:

Entrada A	Entrada B	Saída
0	0	1
0	1	1
1	0	1
1	1	0

Com base nisso, observa-se que a inversão dos sinais de entrada seria uma das possíveis soluções para a criação de uma porta OR.

B.2.5 NOR

Similarmente a relação de uma porta AND com o seu inverso (NAND), o mesmo comportamento é observado com a porta OR e o seu inverso (NOR). O valor da saída é

definido via "NOT (**(entrada A)** OU **(entrada B)**)" no caso de uma porta com duas entradas.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Saída
0	0	1
0	1	0
1	0	0
1	1	0

B.2.6 XOR

O funcionamento de uma porta OU Exclusiva pode ser descrita como um elemento de duas ou mais entradas em que caso o número de entradas verdadeiras for ímpar, a saída será verdadeira, e caso o número for ímpar, a saída será falsa.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

A porta lógica XOR é mais interessante no quesito didático, pois requer maior conhecimento de álgebra booleana quando comparada com os itens anteriormente discutidos. Como aspecto introdutório, estão listadas a seguir três possíveis representações de uma porta XOR com 2 inputs fazendo uso apenas de portas AND, OR e NOT provenientes da análise da tabela verdade anterior:

OBS: Existem outras possíveis formas de montar uma porta XOR, inclusive com apenas elementos NAND (é recomendado tentar diversas combinações).

Fora de um contexto apropriado, seu uso pode parecer pouco intuitivo, mas ao estudar elementos futuros como ADDERS pode ajudar a melhor compreender sua funcionalidade.

Entrada A	Entrada B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

B.3 Circuitos combinatórios complexos

B.3.1 Multiplexadores - MUX nx1 (Multiplexers)

Multiplexadores são circuitos combinatórios que, em geral, permitem a seleção de uma entrada entre diversos inputs baseados em um ou mais sinais de controle. Mais especificamente, para cada **n** possíveis entradas, $\log_2(n)$ seletores são necessários para realizar a escolha.

Iniciando de forma simples, um MUX 2x1 possui dois bits de entrada, um bit seletor e uma saída. Seu funcionamento se baseia no sinal de controle permitindo que um dos dois bits de entrada tenha seu valor selecionado (ex: 0 seleciona a entrada A, 1 seleciona a entrada B).

Eliminando valores irrelevantes É possível representar sua tabela verdade da seguinte forma:

Bit seletor/controle	Entrada A	Entrada B	Saída
0	0	valor irrelevante	0
0	1		1
1	valor irrelevante	0	1
1		1	0

Percebe-se então que a tabela pode ser representada pela seguinte equação booleana, permitindo também que seja desenvolvida no simulador com componentes simples previamente criados.

```
Saída = (NOT seletor AND A) OR (seletor AND B)
```

Assim, é possível simplificar a tabela para melhor representar suas propriedades, destacando o valor dos bits de entrada:

Bit seletor/controle	Entrada A	Entrada B	Saída
0	valor bit A	-	**valor bit A**
1	-	valor bit B	**valor bit B**

O mesmo se aplica ao MUX 4x1 ou qualquer outro multiplexador mais complexo.

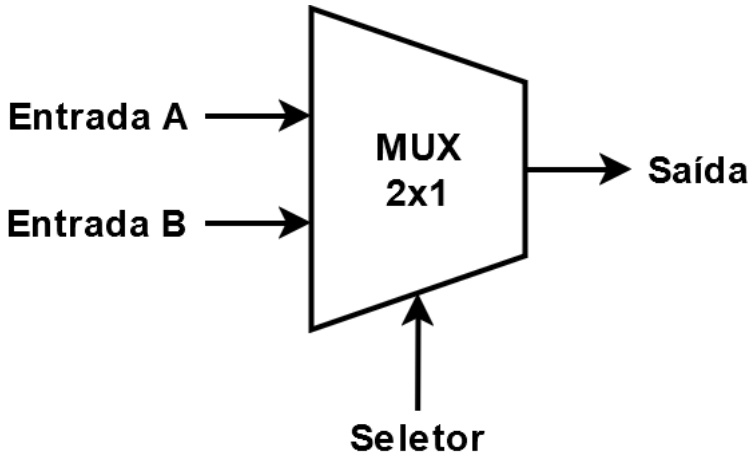
Seletor 1	Seletor 2	Entrada A	Entrada B	Entrada C	Entrada D	Saída
0	0	valor bit A	-	-	-	**valor bit A**
0	1	-	valor bit B	-	-	**valor bit B**
1	0	-	-	valor bit C	-	**valor bit C**
1	1	-	-	-	valor bit D	**valor bit D**

A equação booleana se torna:

```
Saída = (NOT S1 AND NOT S2 AND A) OR
        (NOT S2 AND     S2 AND B) OR
        (    S2 AND NOT S2 AND C) OR
        (    S2 AND     S2 AND D)
```

Uma informação importante que deve ser adicionada sobre o uso de multiplexadores é que eles podem ser utilizados em conjunto para criar MUXs de múltiplos bits.

Uma representação de um mux 2x1 pode ser observada a seguir:



B.3.2 Portas lógicas a partir de multiplexadores

Um aspecto interessante sobre esse componente é o fato que a partir de um ou mais multiplexador 2X1 é possível criar diversas portas lógicas.

B.3.2.1 Porta AND

Para a construção de uma porta AND via o uso de um multiplexador é possível analisar o comportamento geral de um MUX 2x1, em que um input seletor permite escolher qual de duas entradas irá corresponder a saída.

Entrada A	Entrada B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Revisando a tabela verdade de um componente AND, percebe-se que a uma vez com a entrada A sendo verdadeira, o valor da saída é correspondente ao valor da entrada B. Assim, é possível construir uma porta AND via o uso de A como o input seletor, com as duas entradas do MUX sendo uma entrada nula 0 e uma entrada B, dessa forma a saída default (seletor não ativo) é sempre 0 e uma vez com seletor ativo a entrada B define o valor da saída.

B.3.2.2 Porta NOT

A porta NOT pode ser construída de forma análoga, em que basta que a entrada a ser negada seja o input seletor do MUX, com as entradas sendo escolhidas de forma inversa: primeira entrada 1 (escolhida quando seletor é falso) e segunda entrada 0 (escolhida quando seletor é verdadeiro).

B.3.2.3 Porta NAND

Relembrando as propriedades de uma porta NAND, ela pode ser representada por NAND = NOT (A AND B) com a seguinte tabela verdade:

Entrada A	Entrada B	Saída
0	0	1
0	1	1
1	0	1
1	1	0

A tabela teve seus valores de output destacados pois apresentam um comportamento interessante para os propósitos do exercício: quando o valor da entrada A é falso, a saída é sempre 1, mas quando o valor de A é verdadeiro, o output observado é equivalente ao inverso da entrada de B. Assim, é possível construir uma porta lógica NAND via dois multiplexadores 2x1, um para inverter o sinal de entrada B, transformando-o em um seletor de entradas 1 e 0 (ou seja, uma porta NOT), e outro utilizando o sinal A como seletor de uma entrada 1 quando A = 0 e NOT B quando A = 1 persistida do MUX anterior.

B.3.2.4 Porta OR

Observando-se a tabela verdade:

Entrada A	Entrada B	Saída
0	0	0
0	1	1
1	0	1
1	1	1

Percebe-se que a forma de obter a porta OR é análoga a construção da uma porta AND, com a diferença sendo em qual caso uma certa entrada é selecionada. Uma vez com a entrada A sendo verdadeira, o valor da saída é sempre 1, mas quando é falsa a saída é correspondente ao valor da entrada B. Assim, é possível construir uma porta OR via o uso de A como o input seletor, com as duas entradas do MUX sendo uma entrada 1 e uma entrada B, dessa forma a saída default (seletor não ativo) é definida pela entrada B e uma vez com seletor ativo saída é sempre 1.

B.3.2.5 Porta NOR

Entendendo a lógica empregada para a criação das portas lógicas NAND e OR, a criação de uma porta NOR ocorre de forma similar. Assim, deve-se empregar o uso de dois multiplexadores, um para a realização da inversão do sinal, e outro para empregar a lógica OR.

A seguinte tabela verdade representa seu comportamento:

Entrada A	Entrada B	Saída
0	0	1
0	1	0
1	0	0
1	1	0

B.3.2.6 Porta XOR

Uma porta XOR segue a seguinte tabela verdade:

Entrada A	Entrada B	Saída
0	0	0
0	1	1
1	0	1
1	1	0

Percebe-se então que essa porta possui uma lógica similar à empregada pelas portas NAND e NOR, mas com o detalhe diferencial sendo o fato que a entrada A faz a escolha se o valor da saída final será a entrada B natural ou B invertida. Ou seja, o primeiro MUX é uma porta NOT e o segundo faz a escolha entre B e NOT B.

B.3.2.7 Porta XNOR

A porta XNOR demonstra o inverso do funcionamento da porta NOR, com sua tabela verdade sendo observada abaixo:

Entrada A	Entrada B	Saída
0	0	1
0	1	0
1	0	0
1	1	1

Analogamente ao observado em XOR, uma mesma lógica pode ser empregada para a criação da porta via o uso de dois multiplexadores: a entrada A faz a escolha se o valor

da saída final será a entrada B invertida ou B natural. Ou seja, o primeiro MUX é uma porta NOT e o segundo faz a escolha entre NOT B e B.

B.3.3 Codificadores (Encoders)

Um codificador é um componente responsável por codificar um número binário de 2^n bits em uma informação de **n** bits (**OBS:** valores de input e output também são comumente referidos como linhas).

Algumas tabelas verdades podem ser disponibilizadas para facilitar a compreensão de seu funcionamento:

- **n = 1:** Para o caso, observa-se que a primeira linha de entrada resulta na saída 0, ou seja, ele codificou o valor de 01 no valor 0;

E1	E0	S
0	1	0
1	0	1

- **n = 2:** Analogamente, o mesmo ocorre para o caso de 2 outputs;

E3	E2	E1	E0	S1	S0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- **n = 3:** Seguindo a mesma lógica, um caso de uso comum é a transformação de um sinal de entrada específico em sua representação análoga em binário, ou seja, a entrada 5 representada pela entrada com sinal E5 ativa tem como codificação o número 5 em binário (101).

E7	E6	E5	E4	E3	E2	E1	E0	S2	S1	S0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

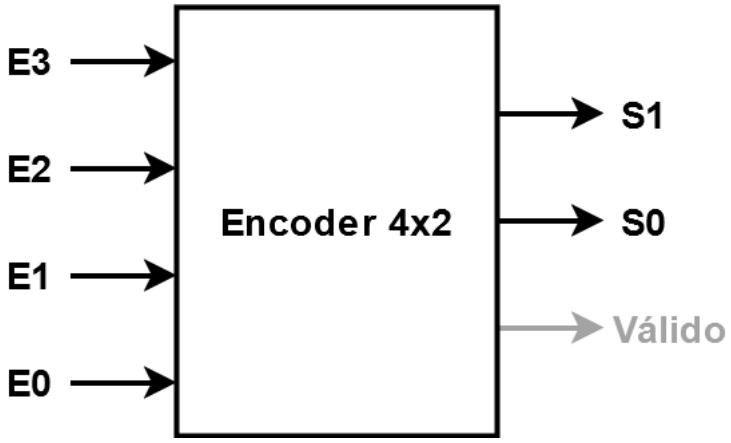
Em seu uso é geralmente considerado que apenas um dos bits de entrada é ativo para cada input, mas existem alternativas que podem considerar mais de um input como ativo, escolhendo assim o bit mais significativo como prioritário na escolha de output, com os demais menos significativos sendo desconsiderados (*don't care*).

Um exemplo de tabela verdade de um codificador com prioridade é observado a seguir, com x representando a condição de don't care. Nota-se também a presença de uma saída adicional que define se o input é válido ou não.

E3	E2	E1	E0	S1	S0	Válido
0	0	0	0	x		0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

A construção do circuito referente ao componente pode ser realizada via a análise da tabela verdade, destacando os valores relativos a 1, ou seja, realizando a soma de mintermos. Com mais entradas é recomendado a realização de um mapa de Karnaugh.

Uma representação do elemento pode ser observada a seguir:



B.3.4 Demultiplexadores DMUX (Demultiplexers)

Demultiplexadores fazem o inverso de um multiplexador, em que os bits seletores escolhem a partir de uma única entrada o valor da saída.

A tabela verdade de um DEMUX 1x2 pode ser observada a seguir:

Entrada	Bit seletor/controle	Saída A	Saída B
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

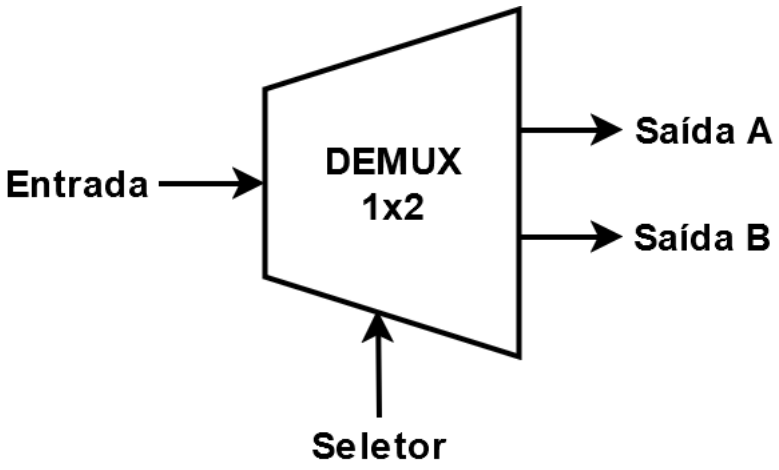
Observa-se que o bit seletor define qual das saídas recebe o valor do input de entrada. Assim, a tabela verdade pode ser simplificada da seguinte forma:

Bit seletor/controle	Saída A	Saída B
0	Entrada	0
1	0	Entrada

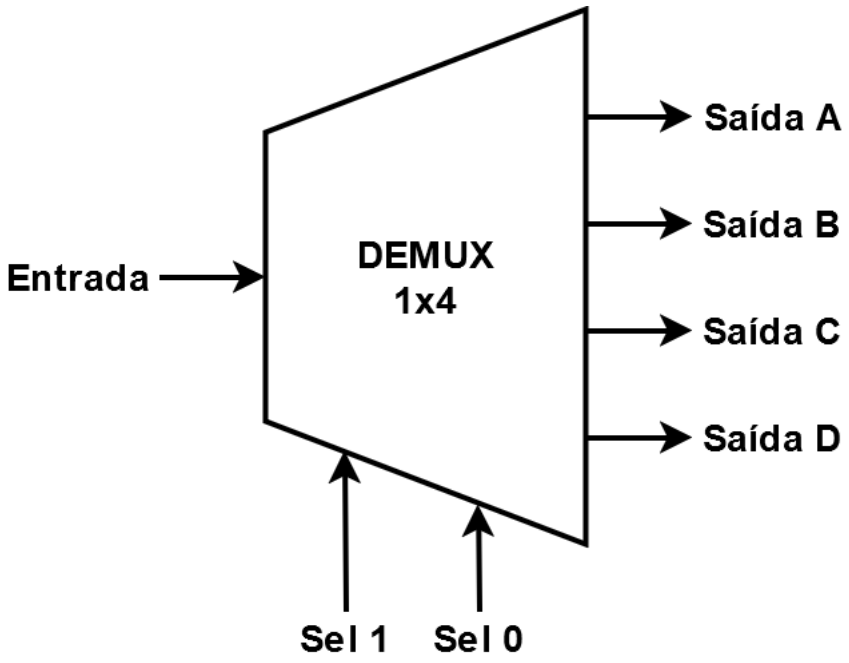
Analogamente, um DEMUX 1x4 poderia ser representado da seguinte forma, com os demais demultiplexadores seguindo o mesmo padrão.

Seletor 1	Seletor 0	Saída 3	Saída 2	Saída 1	Saída 0
0	0	0	0	0	Entrada
0	1	0	0	Entrada	0
1	0	0	Entrada	0	0
1	1	Entrada	0	0	0

Uma representação de um demultiplexador 1x2 pode ser observada a seguir:



Uma representação de um demultiplexador 1x4 pode ser observada a seguir:



B.3.5 Decodificadores (Decoders)

Com funcionamento inverso a um codificador, decodificadores são componentes cujo objetivo é decodificar um número binário de **n** bits em um de 2^n.

Duas tabelas verdades de casos comuns podem ser disponibilizadas para facilitar a compreensão de seu funcionamento:

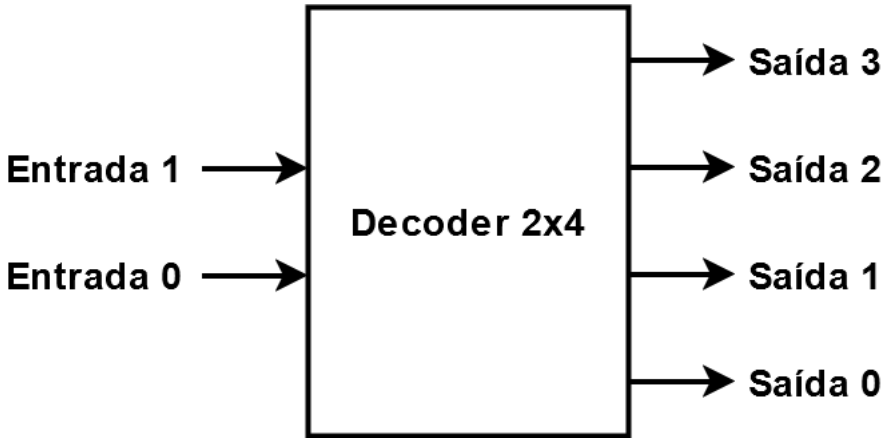
- **n = 1:** Com apenas uma linha de entrada, observa-se que existem duas linhas de saída (uma para cada input possível);

Entrada	Saída 1	Saída 0
0	0	1
1	1	0

- **n = 2:** No caso de duas entradas, existem quatro linhas de saída, em que cada combinação de entradas fornece como output uma informação (ex: 01 → 0100);

Entrada 1	Entrada 0	Saída 3	Saída 2	Saída 1	Saída 0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Uma representação do elemento pode ser observada a seguir:



B.4 Circuitos aritméticos

B.4.1 HALF ADDER

Uma funcionalidade que vai se tornar essencial na criação de um processador é a capacidade de realizar operações aritméticas, para isso é necessário desenvolver os componentes individuais que irão permitir a construção de uma Unidade Lógica Aritmética (ALU).

Com esse intuito, é possível iniciar com um HALF ADDER, um elemento capaz de realizar a adição de bits. Ele possui duas entradas e duas saídas, sendo os inputs os bits a serem somados e os outputs o resultado da soma e o bit de carry-out, respectivamente. Em suma, o resultado é o bit menos significativo de uma soma, e o carry-out é o bit mais significativo.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Carry-out	Soma
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Observa-se que a tabela de fato representa a realidade, uma vez que ao somar um bit 1 com um bit 0, o resultado é 1, e ao somar dois bits 1, o resultado é 2 (em binário: 10). Percebe-se também que os resultados obtidos são equivalentes a duas portas lógicas previamente estudadas (AND e XOR).

Uma representação do componente pode ser exemplificada abaixo:

B.4.2 FULL ADDER

Como progressão lógica da funcionalidade de adição de bits, é possível agora estudar um pouco sobre os FULL ADDERS: componentes de soma que possuem o input de carry-in, permitindo que o bit de carry-out de uma etapa de uma soma anterior possa influenciar

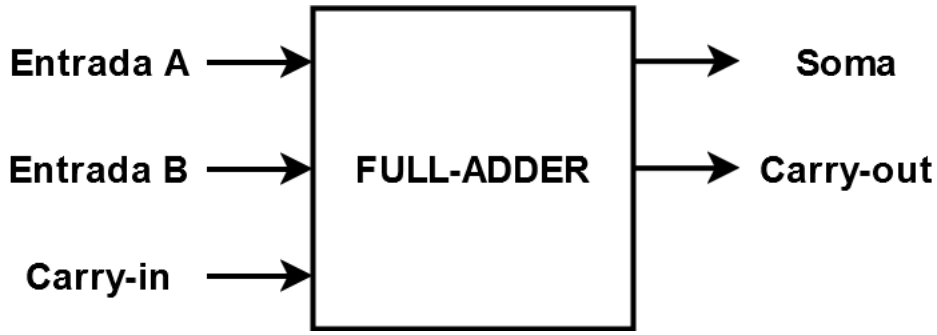
em uma próxima etapa (ex: somador de múltiplos bits via cascata). No caso, o bit de carry-in é somado ao resultado da soma das entradas A e B, com o resultado também influenciando no carry out do somador.

É possível representar sua tabela verdade da seguinte forma:

Entrada A	Entrada B	Carry-in	Carry-out	Soma
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Para o seu desenvolvimento no simulador, é possível tanto fazer uso de dois half adder desenvolvidos anteriormente com uma porta OU, quanto via portas simples diretamente (XOR, AND e OR).

Uma representação do componente pode ser exemplificada abaixo:

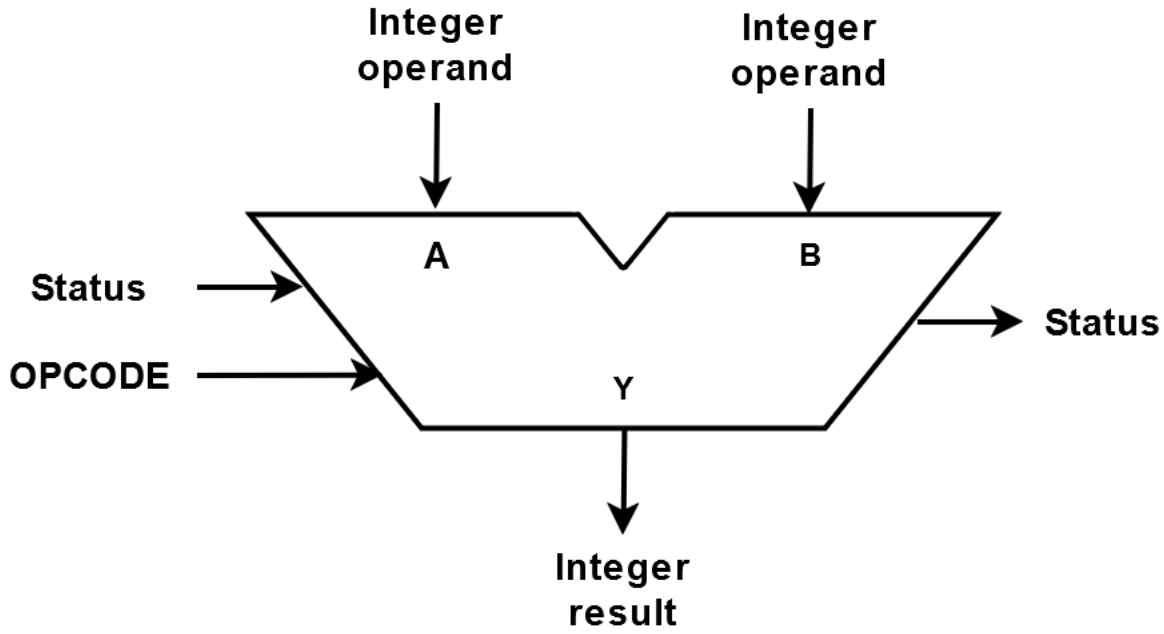


B.4.3 n BIT ADDER

Com os FULL ADDERS em mãos, é possível manipulá-los para criar um componentes que possa atuar como somador de não apenas dois bits, mas de quaisquer n bits. Para isso, basta que eles sejam concatenados em cascata de forma que os sinais de carry-out de somas de bits menos significativos enviem os sinais para os carry-in de bits mais significativos.

B.4.4 ALU (Arithmetic Logic Unit)

Uma Unidade Lógica Aritmética é um circuito digital de lógica combinatória capaz de realizar operações lógicas (AND, OR) e operações aritméticas (adição, subtração).



As operações a serem realizadas são decididas via o valor de OPCODE, que junto com as informações provenientes do status conseguem enfim manipular os operandos para obter os resultados desejados (caso de carry-out, valores nulos ou negativos, etc.).

Para isso, ela faz uso de diversos componentes vistos previamente para conseguir estruturar as operações e os seus sinais de entrada, como multiplexadores, somadores e outros componentes lógicos.

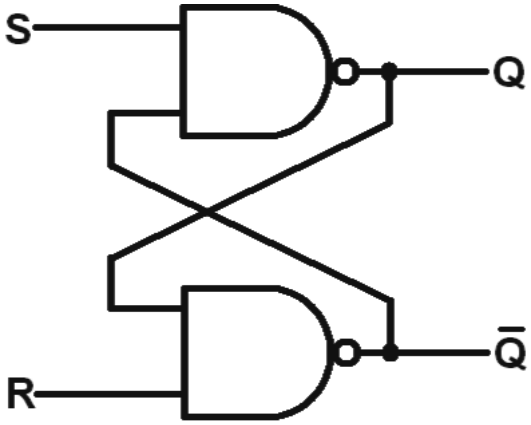
B.5 Circuitos sequenciais

B.5.1 LATCH

Latches são circuitos digitais utilizados para armazenar uma informação binária, sendo muito utilizadas para a construção de componentes observados em partes futuras da teoria estudada (ex: FLIP-FLOPs). A implementação delas é, portanto, essencial no desenvolvimento de circuitos sequenciais como memórias e máquinas de estados. Em geral, existem diversos tipo de LATCHES, com eles também podendo possuir um sinal de ENABLE para controlar ativações, como:

B.5.1.1 SR Latches (Set-Reset)

Uma representação didática de um LATCH SET-RESET pode ser observada no diagrama abaixo:



A realimentação nos componentes é o que permite o armazenamento de estados, com a estrutura deste LATCH sendo o tipo mais simples e que serve como base para todas as outras variações e tipos.

B.5.1.2 D Latches (Data ou Transparente)

São LATCHES cujo status é definido pelo input único D, mas que somente atua quando o sinal de ENABLE estiver ativado.

B.5.1.3 JK Latches (Inverte quando ambos 1)

Similar ao SR, com o detalhe que o status é invertido quando ambas as entradas J e K são ativas.

B.5.1.4 T Latches (Toggle)

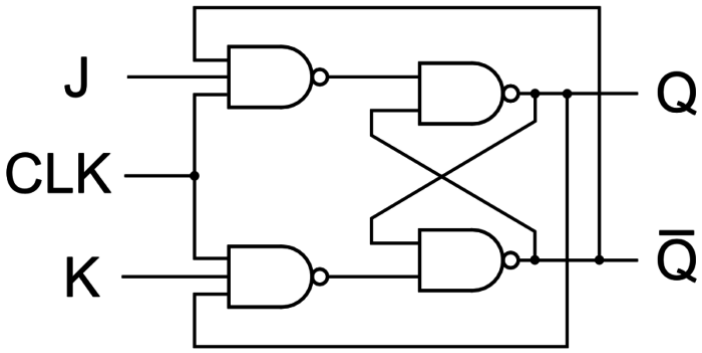
Possui apenas o input T que inverte o estado do sinal toda vez que é ativado

B.5.2 FLIP-FLOP

Similarmente a LATCHES, atuam como armazenamento de memória binária, com a diferença essencial entre ambos é que LATCHES independentemente de timings, já FLIP-FLOPs dependem do sinal de um clock para serem alteradas.

Em essência, sua construção é similar, possuindo também as variações observadas em LATCHES (SR, D, JK, T), mas geralmente contém uma porta lógica alimentada por sinal de clock (CLK) que limita a atuação do componente.

O diagrama do JK FLIP-FLOP abaixo serve como exemplo de composição deste tipo de componente, com este padrão de composição sendo repetido para as demais variações:



B.5.3 REGISTER

Registradores são formados por dois ou mais FLIP-FLOPs associados a um sinal de clock, permitindo assim armazenar um conjunto de bits (8 flip-flops formam um registrador de um byte). Dentre suas propriedades, eles podem ser síncronos ou assíncronos, podendo também estar interligados entre si (SHIFT-REGISTER).

Alguns de seus sinais de controle possíveis são o clock, enable, set, reset, shift, etc. Com relação aos seus sinais, podem ter entradas em série com saídas em paralelo, ou entradas e saídas em paralelo.

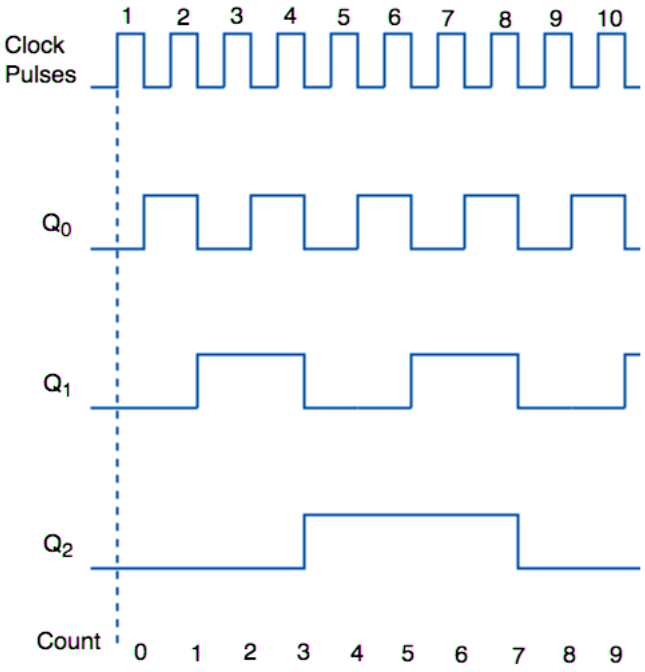
Ademais, dependendo de sua complexidade, podem conter em si multiplexadores como no caso de registradores universais.

B.5.4 COUNTER

Um contador é um componente sequencial utilizado para armazenar a quantidade de vezes que um evento ocorreu (pulsos de input), sendo um grupo de flip-flops associados a um clock.

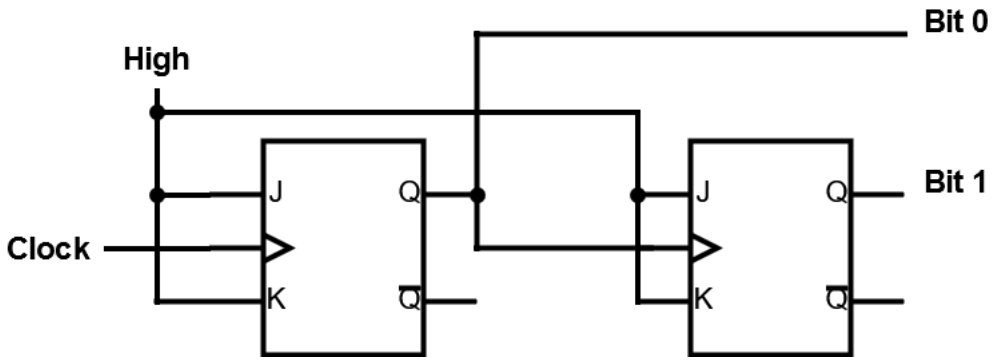
Vale ressaltar que os sinais de clock de input podem ser relativos tanto a subida do sinal de clock quanto a sua descida.

Como forma de melhor detalhar estes componentes, podemos separá-los em dois tipos principais: assíncronos ou síncronos, sendo descritos a seguir:



B.5.4.1 Contadores assíncronos

Também chamado de *ripple counter*, um contador assíncrono não faz uso de um clock universal, ou seja, apenas o primeiro flip-flop é conectado a um clock, com todos os flip-flops subsequentes sendo controlados pelo output do flip-flop anterior, ou seja, como "cascata".



Supondo que o evento contado é possui sempre valor 1 (high), a tabela verdade do comportamento do contador seria a seguinte:

Clock	Bit 1	Bit 0	Contagem
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3
4	0	0	4

B.5.4.2 Contadores síncronos

Também chamado de *parallel counter*, um contador síncrono possui um clock universal conectando todos os flip-flops do componente, de forma que as mudanças ocorram em paralelo, não possuindo a limitação de frequência observada em um contador assíncrono. Sua construção tem como base os flip-flops necessários para cada bit e portas AND para unir os sinais dos componentes.

B.5.5 SHIFT-REGISTERS (Deslocadores)

Shift-registers são registradores cujo conteúdo armazenado (n bits) pode ser deslocado via o input de pulsos, movendo o dado binário de um flip-flop para outro.

Existem diversos tipos diferentes de deslocadores, com os principais podendo ser mencionados a seguir:

B.5.5.1 Serial-in serial-out

A entrada de dados ocorre em série, com o valor menos significativo do dado entrando primeiro, deslocando os valores previamente armazenados entre os flip-flops, e resultando em um output de dados também em série.

Ou seja, com um shift-register inicialmente armazenando 000, ao receber um input 111 o valor armazenado vai sendo alterado a cada ciclo de clock ($100 > 110 > 111$), com o output sendo sempre o valor que foi deslocado obtido pelo ultimo flip-flop.

Sua construção se baseia em posicionar os flip-flops em série conectados a um único sinal de clock.

B.5.5.2 Serial-in parallel-out

Construído de forma similar ao serial-in serial-out, mas a diferença ocorrendo no fato que o output é obtido em paralelo via o output de cada flip-flop que compõe o componente.

B.5.5.3 Parallel-in serial-out

O dado de input é inserido de forma paralela via uma mesma lógica combinatória, com o output de cada flip-flop sendo usado como input do próximo flip-flop fazendo o uso deste mesmo circuito.

Ele possui duas funções, o modo de deslocamento (shift) e o modo de carregamento (load):

- **SHIFT:** os valores são inseridos bit a bit nos flip-flops similarmente ao observado em serial-in serial-out, com um deslocamento ocorrendo a cada pulso de clock;
- **LOAD:** os valores do dado de input são carregados paralelamente nos flip-flops.

A lógica combinatória utilizada em sua construção são multiplexadores que recebem um dos dados a serem inseridos, o sinal indicando a operação (shift ou load) e o output do flip-flop anterior, com o output do mux sendo o input do próximo flip-flop.

B.5.5.4 Parallel-in parallel-out

Ambos os dados de input e de output são paralelos, com os flip-flops estando conectados a um mesmo clock, mas com seus sinais de entrada e saída não interagindo entre si. Assim, os dados de entrada são fornecidos individualmente assim como os seus sinais de saída.

B.5.5.5 Bidirectional Shift Register

Este componente deve ser capaz de deslocar os dados para qualquer direção desejada (para esquerda ou para direita), com este deslocamento sendo definido pelo modo escolhido. Sua construção é similar a um parallel-in serial-out, com a diferença sendo nos inputs das conexões entre os flip-flops.

Com relação a sua utilidade, um deslocador bidirecional pode ser utilizado para representar a multiplicação ou divisão de um número binário por 2.

B.5.5.6 Universal Shift Register

Um deslocador universal é um componente capaz de tanto deslocar bidirecionalmente os dados armazenados quanto também realizar a operação de load paralelamente.

B.5.5.7 Shift Register Counter

Uma de suas principais implementações é um contador em anel, ao qual o output do último flip-flop está conectado ao input do primeiro, assim formando um ciclo que se

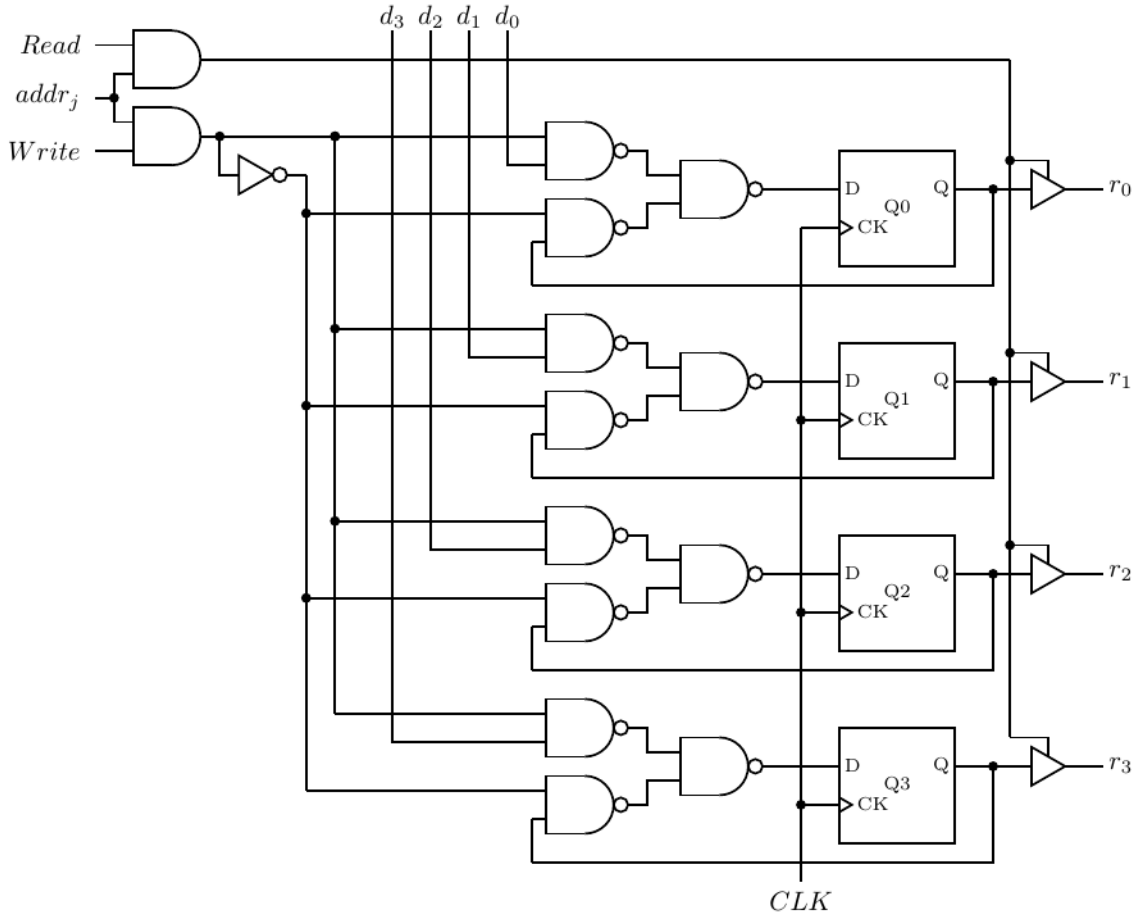
repete a cada n ciclos de clock (com n sendo relacionado ao número de flip-flops usados para a construção do componente).

B.6 Memórias

B.6.1 Memória RAM

Uma memória RAM é um componente de memória volátil (ou seja, de armazenamento temporário, precisa de energia para guardar dados) que permite escrita e leitura de dados. Exemplos comumente observados de memórias RAM são DRAM e SRAM.

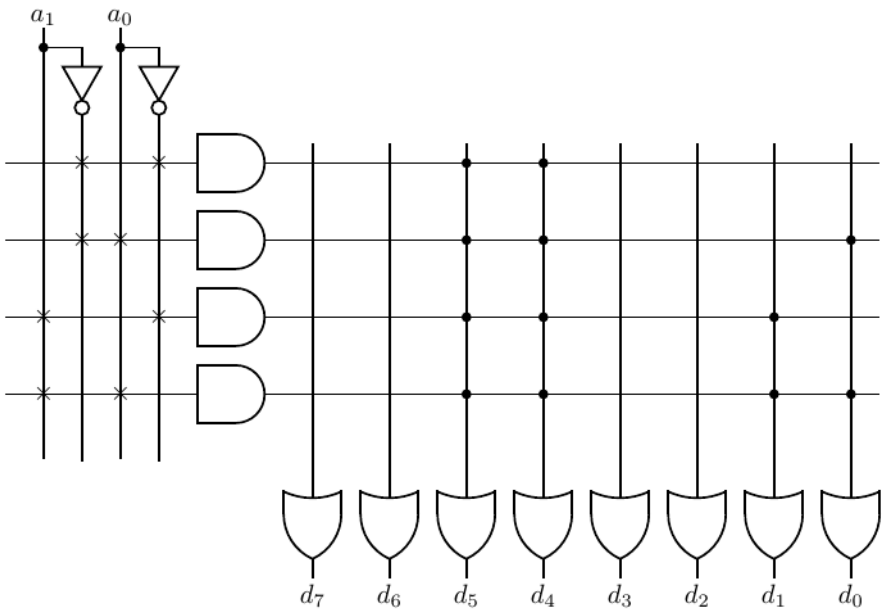
Elas são compostas, essencialmente, por flip-flops (para a construção de registradores), decodificadores e multiplexadores. Em tal, os registradores são escolhidos para serem lidos ou escritos via um endereço especificado, e estes bits de endereço podem ser escolhidos via decodificadores ou via multiplexadores. Uma representação de uma célula de memória de 4 bits pode ser observada abaixo, em que addr é um input de um decodificador para a escolha de um endereço da memória. A seguinte figura representa uma célula de memória (PLANTZ, 2019).



B.6.2 Memória ROM

Uma memória ROM é um componente de memória não volátil, sendo usada para leitura de dados previamente escritos/programados. Exemplos comumente observados de memórias ROM são PROM, EPROM e EEPROM, com alguns dos modelos permitindo a reescrita de valores.

Na imagem abaixo é possível observar uma representação de uma memória ROM, em que os inputs definem o endereço da informação via as portas AND (essencialmente um decoder), e as conexões no plano das portas OR definem os padrões da informação armazenada. A seguinte figura representa uma seção de uma memória ROM (PLANTZ, 2019):



Assim, pode-se dizer que a estrutura interna de uma ROM é composta por decodificadores e portas OU, em que os decoders atuam para transformar uma informação em binário para sua forma decimal, e essa informação é o input das portas OR da ROM, aos quais são organizadas em grade potencialmente conectadas, permitindo a passagem de energia e consequentemente o output da informação desejada.

B.6.3 Aplicações de memórias

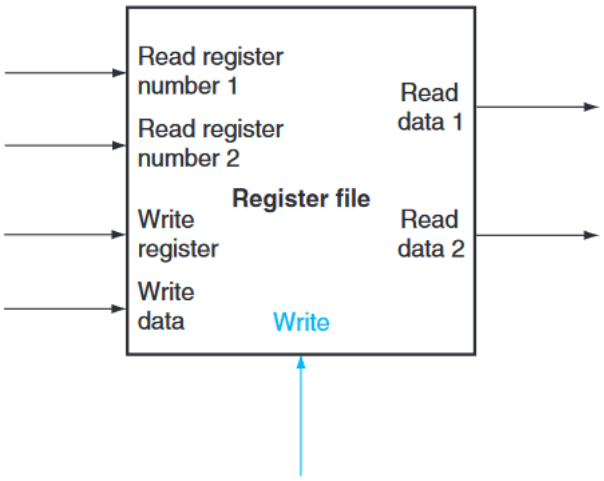
Memórias digitais podem possuir muitas aplicações em computadores, algumas delas estão listadas nas seções seguintes.

B.6.3.1 Banco de registradores

Um banco de registradores é um componente importante para um processador, uma vez que permite o fluxo de dados importantes para seu funcionamento, sendo formado

por múltiplos registradores diferentes que podem ser acessados para a escrita ou leitura de dados. Eles geralmente são operados via bits que indicam o endereço/identificação de um determinado registrador, uma via de dados para a escrita, um sinal que decide se algo será escrito ou lido, e um sinal de clock para a escrita.

A seguinte figura representa o componente: (PATTERSON; HENNESSY, 2014)



B.6.3.2 Memória principal

São as memórias primárias de um sistema computacional, geralmente utilizadas para armazenamento de instruções de programas e seus dados. No contexto da engenharia de computação elas são as memórias que o processador consegue acessar, com os exemplos sendo a memória RAM e a ROM.

Memórias secundárias possuem um papel auxiliar, podendo armazenar uma grande quantidade de informações quando comparado com a memória principal, mas em contra-partida é consideravelmente mais lenta, também não podendo ser acessada diretamente pela CPU. Exemplos de memória secundárias são observadas em discos magnéticos (HDs, SSDs, etc.).

B.6.3.3 Cache

Por fim, a memória cache é um tipo de memória que atua como buffer, sendo mais rápida que a memória RAM. Ela fica entre a CPU e a memória principal, armazenando informações utilizadas com maior frequência, agilizando a busca por informações.

Ela geralmente possui múltiplos níveis (L1, L2, L3, etc.), aos quais são organizados de forma a ficar progressivamente maiores, aumentando sua capacidade, mas diminuindo sua velocidade.

B.7 Máquinas de Estado

Com o conhecimento obtido até o momento, é possível realizar a construção de circuitos sequenciais mais avançados. Para isso, é geralmente empregado o desenvolvimento de um diagrama de estados para representar de forma visual o funcionamento do circuito. Essas representações são denominadas máquinas de estados finitas e podem ser apresentadas de duas formas principais: máquinas de Moore e máquinas de Mealy. Máquinas de estado de Moore determinam o seu output com base apenas no estado atual da máquina, já os outputs das máquinas Mealy dependem tanto da transição quanto do estado atual.

Nos diagramas, os estados são representados por círculos e transições por setas. Os círculos geralmente possuem informações referentes a descrição do estado e o output/resultado referente, já as setas informam a transição de um estado a outro baseado no input detectado. Ademais, deve-se mencionar que as transições de estado ocorrem a cada ciclo de clock.

Uma vez definido o diagrama, é possível construir o circuito desejado ao organizar as informações referentes aos inputs, outputs, estados e transições.

1. **1)** Para a construção dos circuitos, um primeiro passo é a representação dos estados como informação binária, uma vez que devem ser utilizados flip-flips para armazenar informações referentes ao estado atual da máquina;

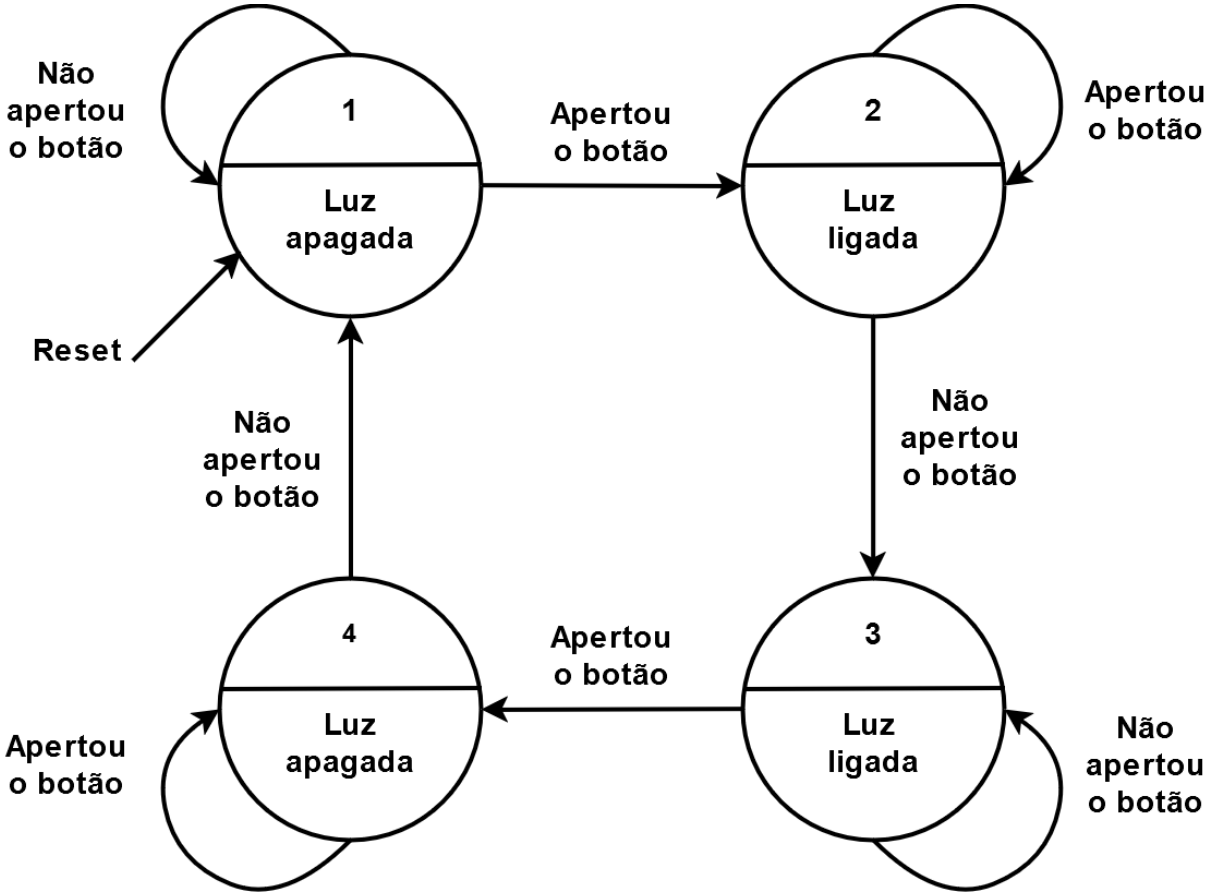
   1

2. **2)** Em seguida, é necessário a construção de tabelas verdade para cada um dos estados, informando aspectos como outputs e próximos estados a partir de determinados inputs;

3. **3)** Essas tabelas verdades podem ser transformadas em mapas de Karnaugh, e uma vez simplificadas, elas informam os componentes necessários para construir o circuito lógico desejado.

B.7.1 Exemplo

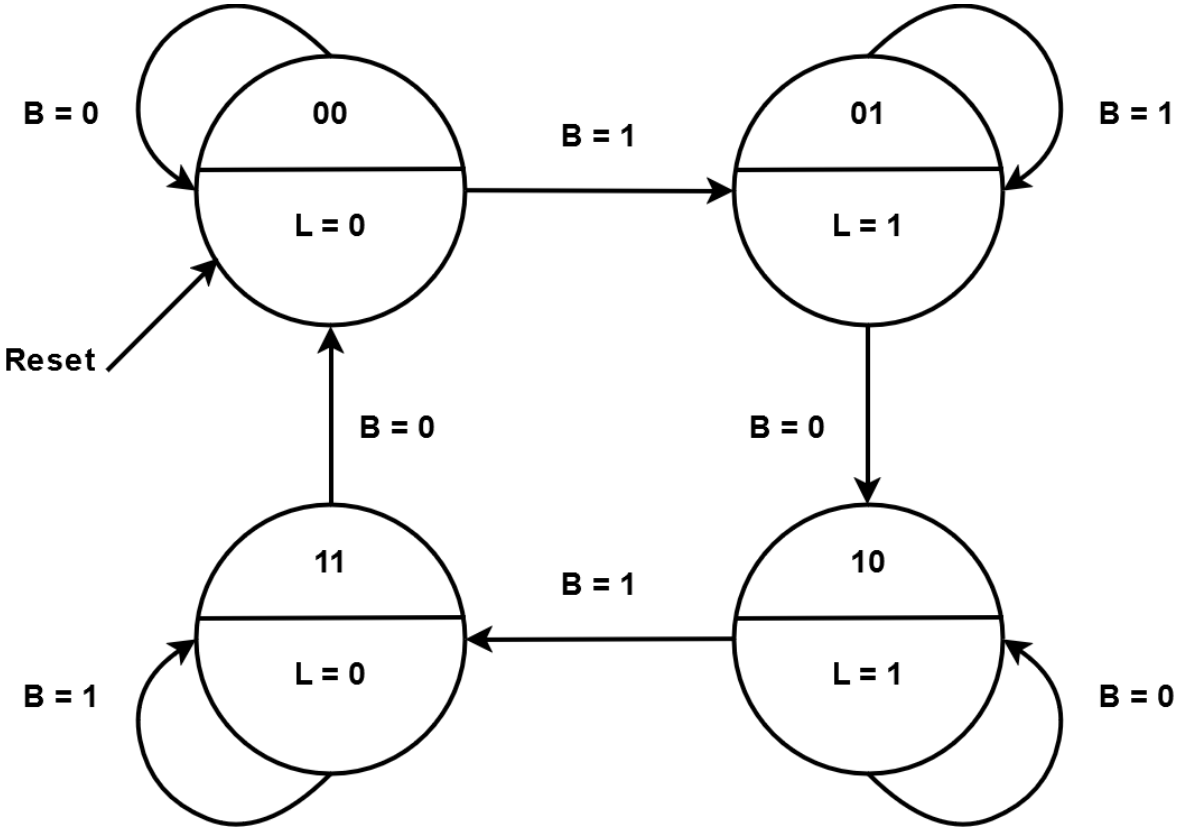
Um exemplo de construção de uma máquina de estados a partir de um diagrama pode ser observado a seguir. Adaptado de David L. Tarnoff da *East Tennessee State University*.



A máquina representa um sistema de ligar e apagar luzes, cujo funcionamento é o seguinte:

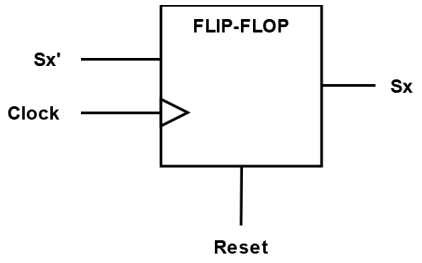
- A partir de seu estado inicial desligado (1) ele espera o pressionar do botão a cada ciclo de clock;
- Uma vez apertando o botão o estado muda para a luz ligada (2), ao qual espera que o botão seja solto;
- Uma vez solto, o é efetuada a transição de estado (3) e a luz continua ligada (3), mas espera o pressionar do botão para apagar a luz;
- Apertando o botão novamente, ele transiciona para o próximo estado (4), apagando a luz e esperando o botão ser solto para retornar ao estado inicial (1);
- **OBS:** Uma vez ativando o reset, é sempre retornado ao estado inicial.

Para facilitar a construção do circuito e das tabelas verdade, é possível reescrever o circuito para uma representação a partir de bits.



Os estados 00, 01, 10 e 11 (antes 1, 2, 3 e 4, respectivamente) podem ser representados de forma binária com dois bits, em que cada um dos dois bits que representam cada estado é armazenado por um flip flop.

Estado		S1 (bit 1 do estado)	S0 (bit 0 do estado)
1	00	0	0
2	01	0	1
3	10	1	0
4	11	1	1



A partir das informações referentes aos estados, inputs do botão, e output da luz, é possível criar uma tabela verdade de transição de estados.

Estado atual		Botão (B)	Próximo estado		Luz (L)
S1	S0		S1	S0	
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	0

Cada uma das colunas destacadas deve ser utilizada para construir um mapa de Karnaugh referente ao valor de input do flip-flop de S1, flip-flop de S0 e do output da luz, pois são valores importantes na construção do circuito lógico. A construção é similar ao processo visto na seção 1.4 do material teórico.

Próximo estado S1

S1 S0 \ B	0	1
00	0	0
01	1	0
11	0	1
10	1	1

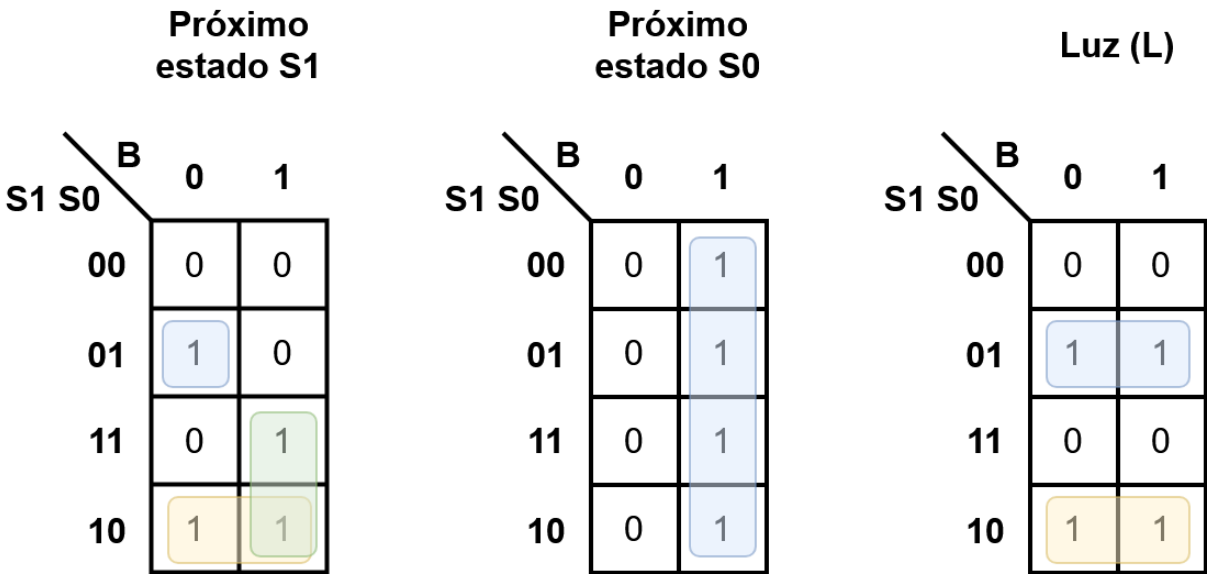
Próximo estado S0

S1 S0 \ B	0	1
00	0	1
01	0	1
11	0	1
10	0	1

Luz (L)

S1 S0 \ B	0	1
00	0	0
01	1	1
11	0	0
10	1	1

Uma vez construídos, é necessário realizar a simplificação dos mapas a fim de obter as funções booleanas referente aos sinais de input.



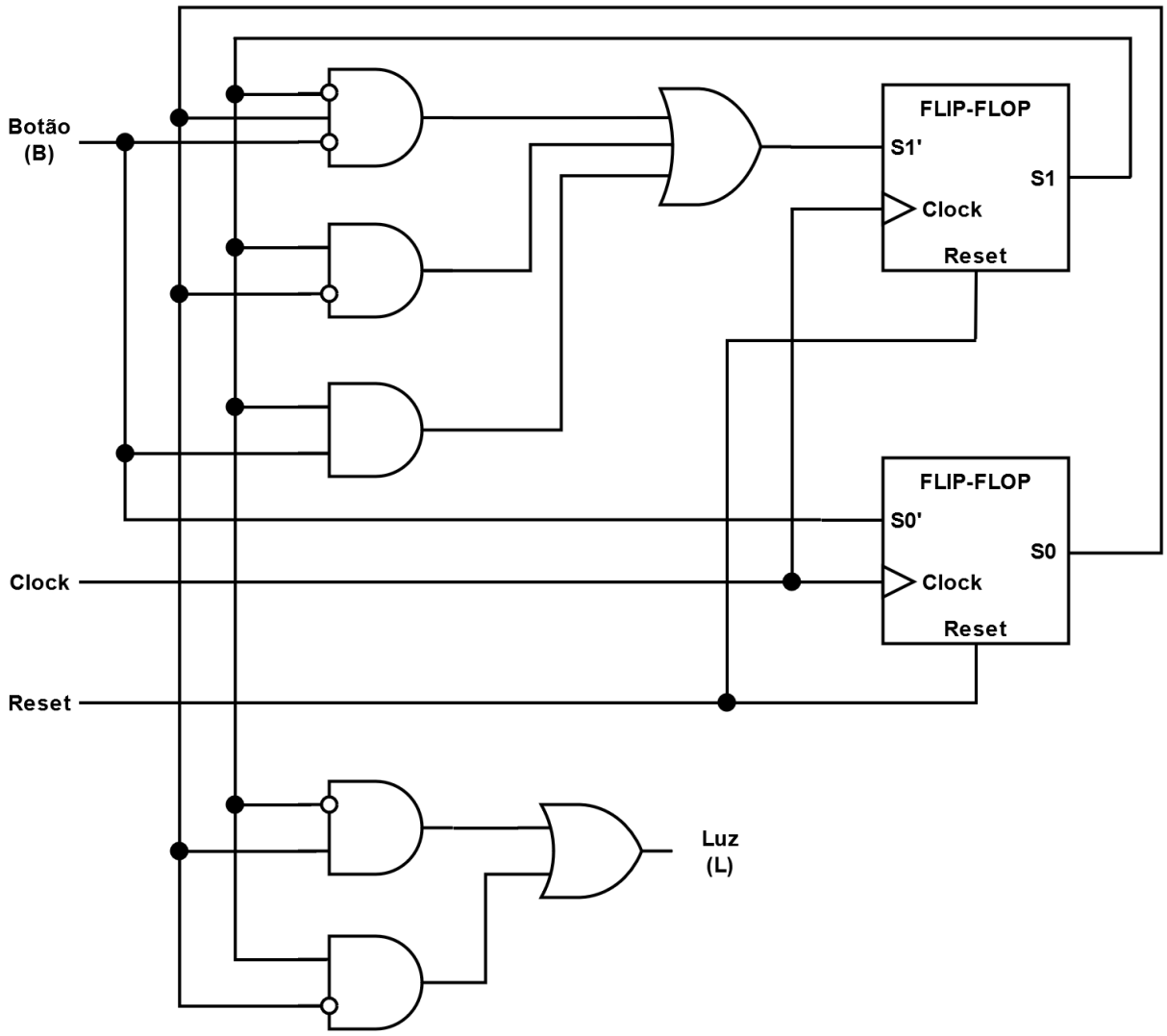
Obtendo assim a soma de produtos (mintermos) de cada um dos mapas, temos os valores referentes aos sinais de próximo estado S1', próximo estado S0' e L.

$$S_1' = (\neg S_1 \cdot S_0 \cdot B) + (S_1 \cdot \neg S_0) + (S_1 \cdot B)$$

$$S_0' = B$$

$$L = (\neg S_1 \cdot S_0) + (S_1 \cdot \neg S_0)$$

Com isso, se torna possível montar o circuito lógico.



B.8 Fluxo de dados & Unidade de controle

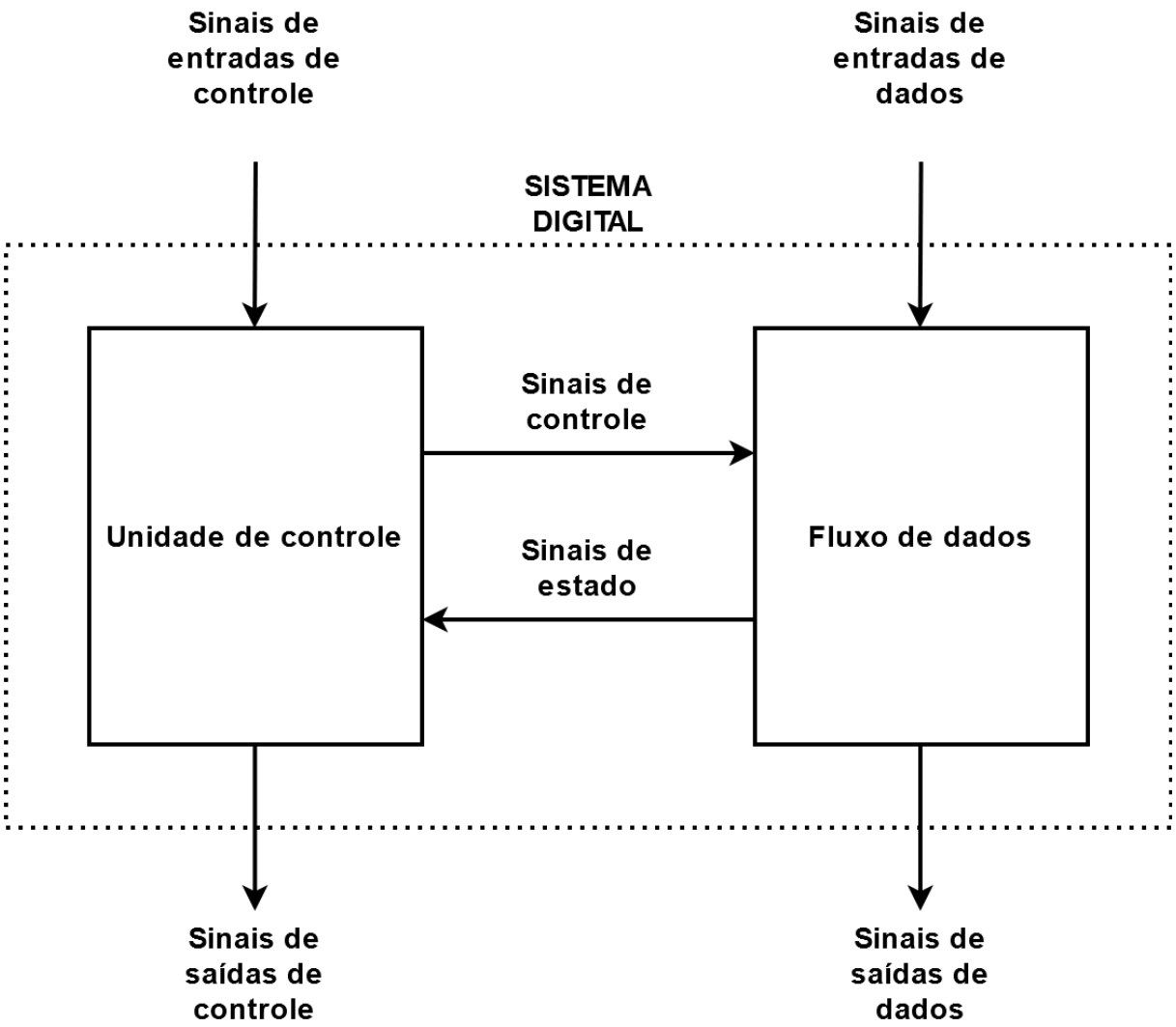
Um sistema digital pode ser dito como a união entre o fluxo de dados com a unidade de controle. Assim, é necessário compreender o papel destes dois elementos para melhor compreender como construir circuitos e sistemas digitais.

Fluxo de dados: ele é responsável por armazenar, rotear, combinação e processar dos dados de um circuito lógico, necessitando ser construído como um circuito no nível de transferência de registradores para que os dados possam ser usados pelos componentes combinatórios internos ao circuito lógico criado, com as informações fluindo através de componentes de memória e registradores.

Ele recebe comandos da unidade de controle, sendo composto por componentes como registradores, memórias, elementos funcionais, etc.

Unidade de controle: ele é responsável pelo controle e sequenciamento das operações realizadas pelo circuito lógico criado, comunicando-se com o fluxo de dados para o envio de sinais de tomada de decisões, definindo o comportamento de elementos como componentes combinatórios do fluxo de dados e elementos registradores. Em geral, a unidade de controle é desenvolvida a partir de um circuito sequencial fundamentado em uma máquina de estados finita.

A seguir pode ser observada uma figura exemplificando uma unidade de controle e um fluxo de dados:



B.9 Estratégias de projeto de sistemas

A estratégia de modularização refere-se a fragmentação de um sistema mais complexo em partes mais simples que podem ser projetadas e testadas individualmente, sem precisar integrá-las ao circuito completo, facilitando aspectos essenciais como otimização e depuração. Ou seja, no contexto observado, torna-se possível o desenvolvimento

de componentes de unidades funcionais para então se construir um elemento complexo posteriormente. Assim, a modularização se torna uma estratégia essencial para a projeção de sistemas como processadores.

Ademais, com relação a paradigmas de desenvolvimento, existem duas abordagens relevantes que devem ser comentadas: a top-down e a bottom-up.

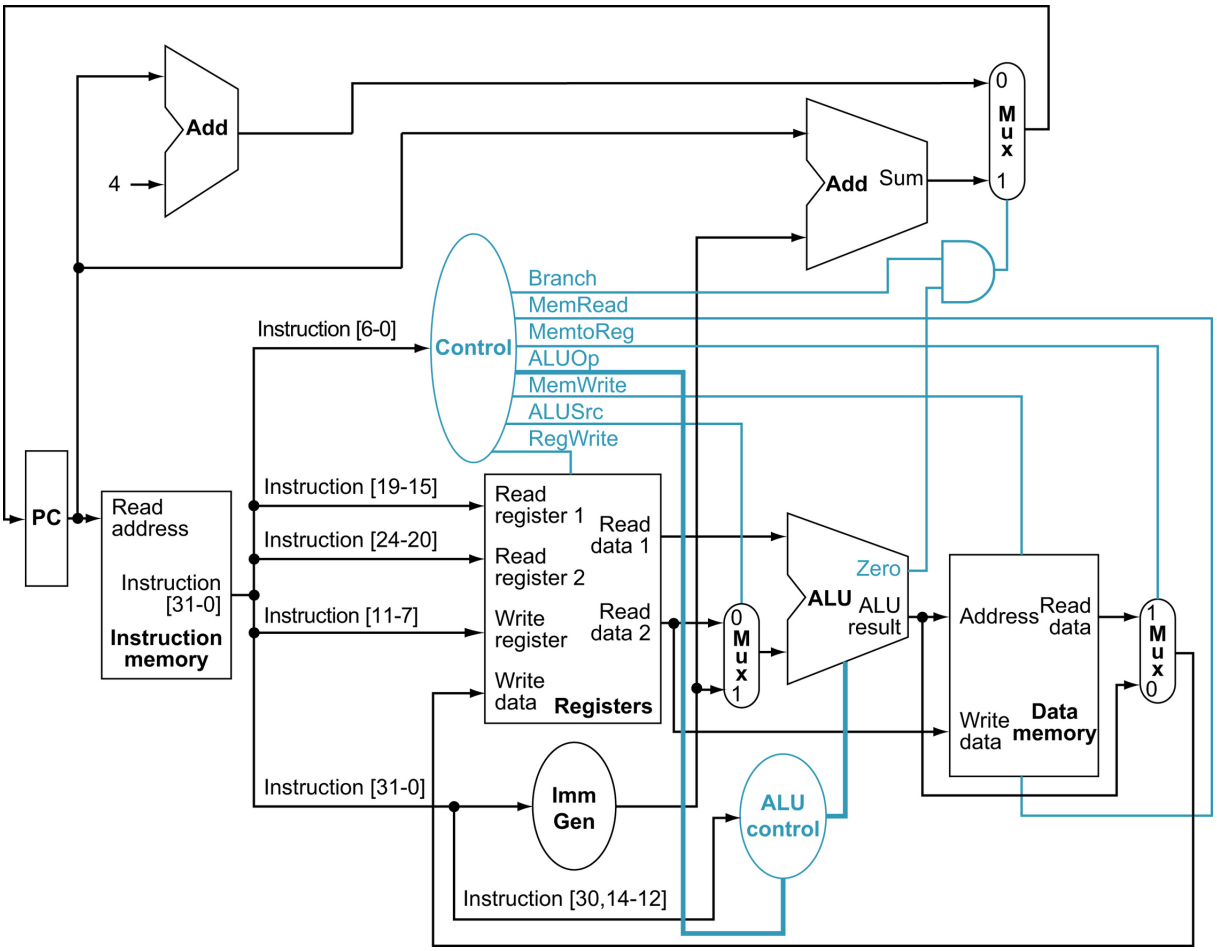
A **top-down** é uma abordagem que inicia com uma visão geral do sistema a ser desenvolvido, partindo do alto-nível e decompondo-o então em partes menores para então identificar seus requisitos e melhor implementar componentes e circuitos lógicos de seus módulos funcionais. Dessa forma, o aspecto de modularização previamente estudado permite que estas partes menores sejam melhor gerenciadas, desenvolvendo estes módulos de maneira independente.

Já a **bottom-up** é fundamentada no desenvolvimento de um sistema a partir de seus elementos mais simples, criando inicialmente os componentes individuais para que eles sejam posteriormente integrados para formar o sistema final, com a visão global do sistema não sendo definida no início do processo. Deve-se mencionar, contudo, que a ausência de uma visão global pode dificultar a integração final quando comparada com a metodologia empregada em top-down.

Destarte, estratégias que melhor gerenciam a hierarquia de componentes podem facilitar o desenvolvimento de módulos complexos, uma vez que podem ser implementados a partir de módulos mais básicos.

B.10 Arquitetura de um processador

Por fim, como um exemplo de um processador RISC-V, a seguinte imagem detalha seu fluxo de dados e unidade de controle (PATTERSON; HENNESSY, 2014):


Add
4
Add
Sum
0
M
u
x
1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6-0]
PC
Read
address
Instruction
[31-0]
Instruction
memory
Instruction [19-15]
Instruction [24-20]
Instruction [11-7]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0
Instruction [31-0]
Imm
Gen
ALU
control
Instruction [30,14-12]

APÊNDICE C – Resultado da construção de circuitos no NANDesis

A seguir são apresentados *prints* dos principais circuitos que podem ser montados utilizando o simulador NANDesis.

C.1 NOT

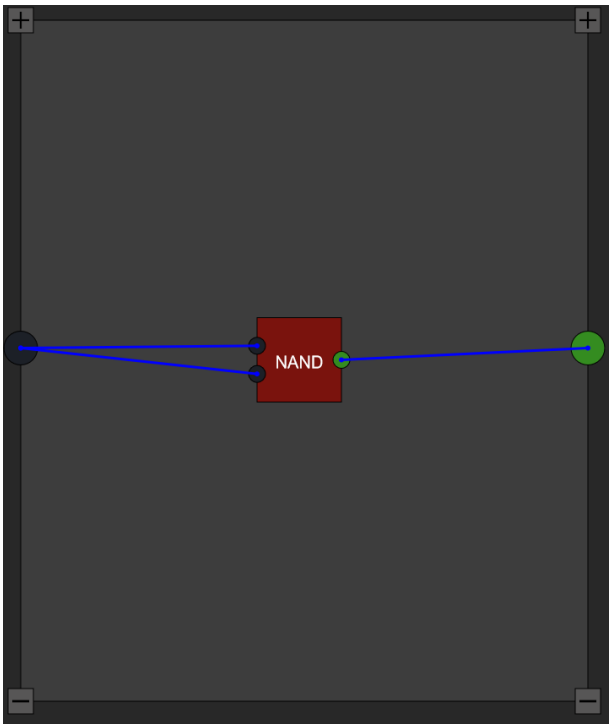


Figura 15 – Circuito NOT

C.2 AND

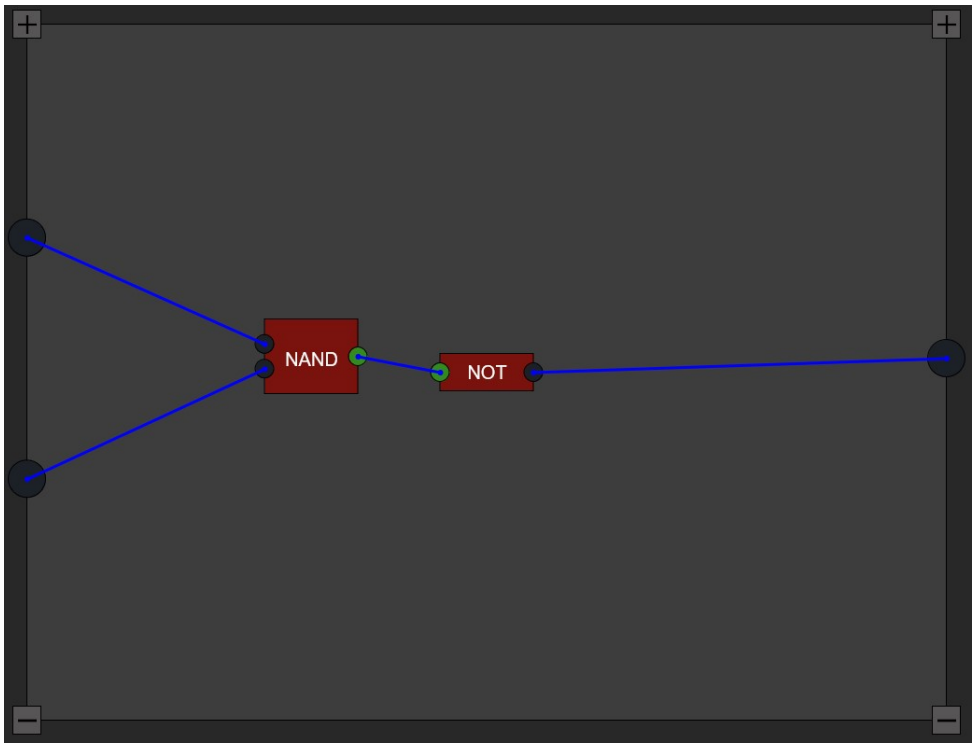


Figura 16 – Circuito AND

C.3 OR

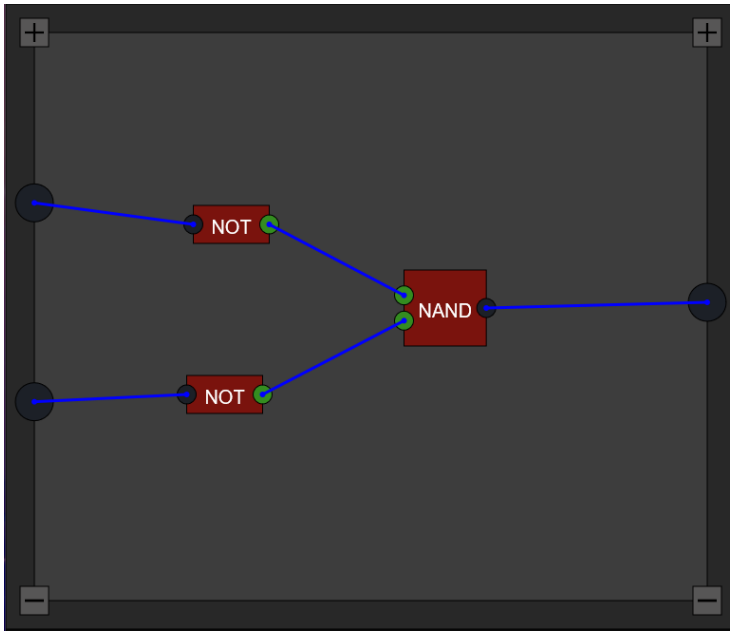


Figura 17 – Circuito OR

C.4 XOR

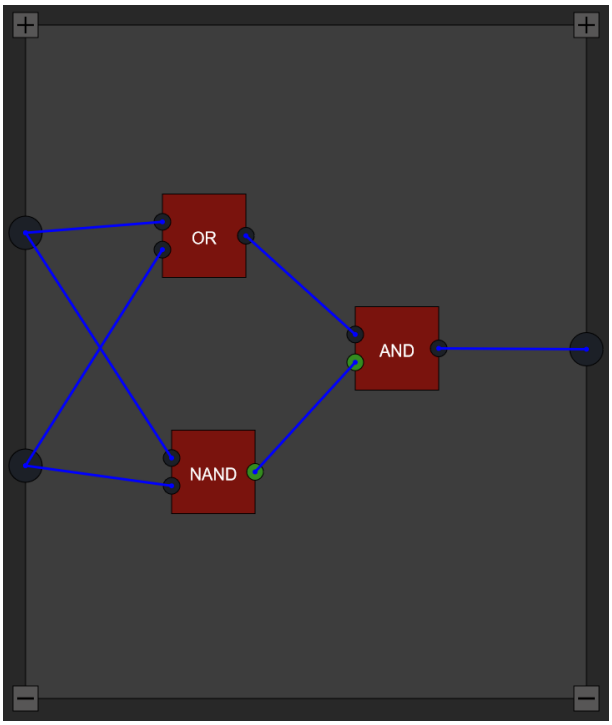


Figura 18 – Circuito XOR

C.5 Half Adder

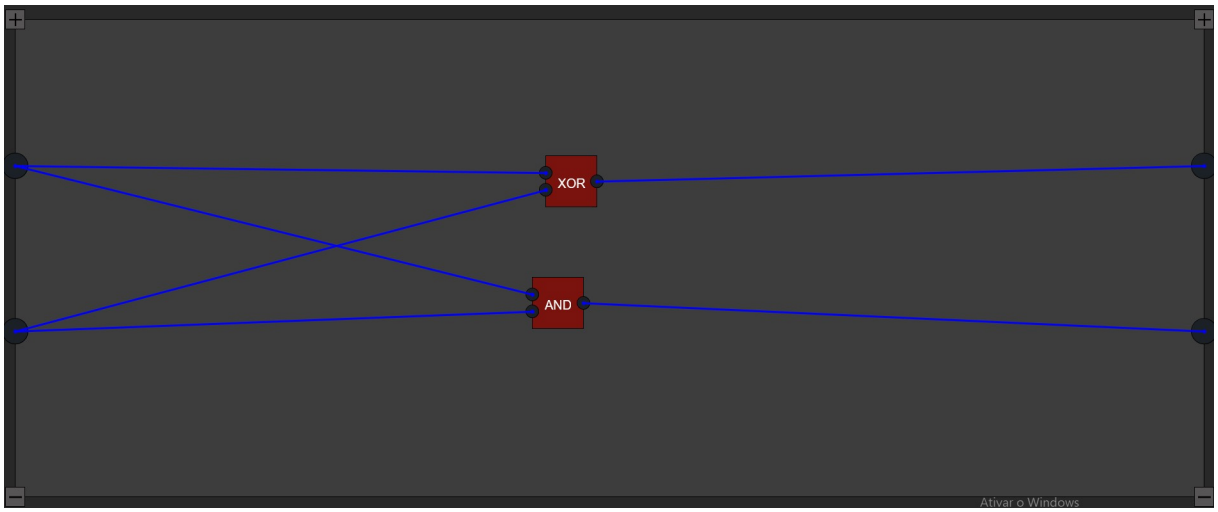


Figura 19 – Circuito Half Adder

C.6 Full Adder

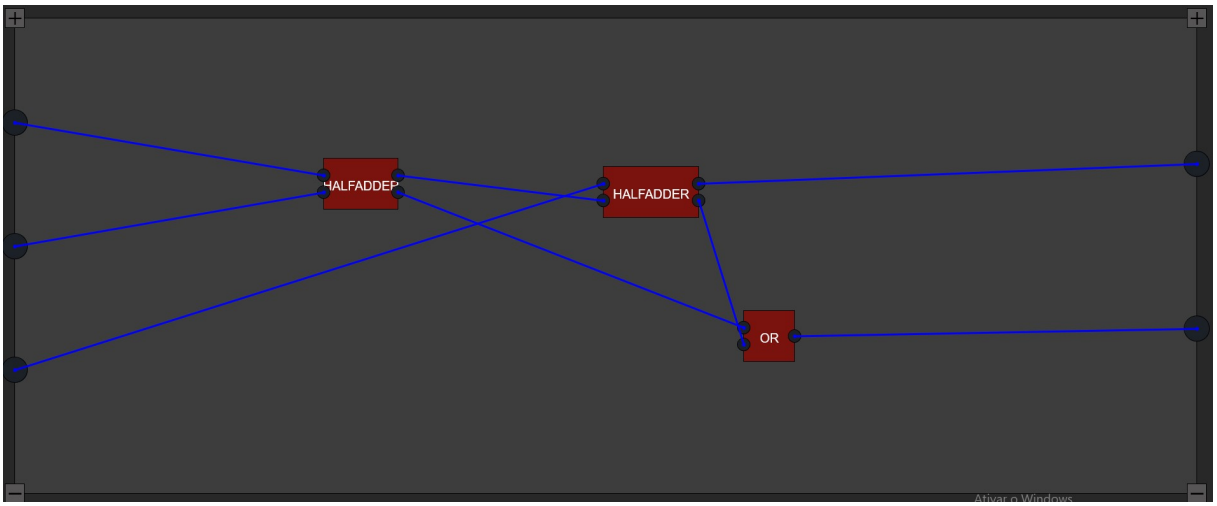


Figura 20 – Circuito Full Adder

C.7 Multiplexador 2X1

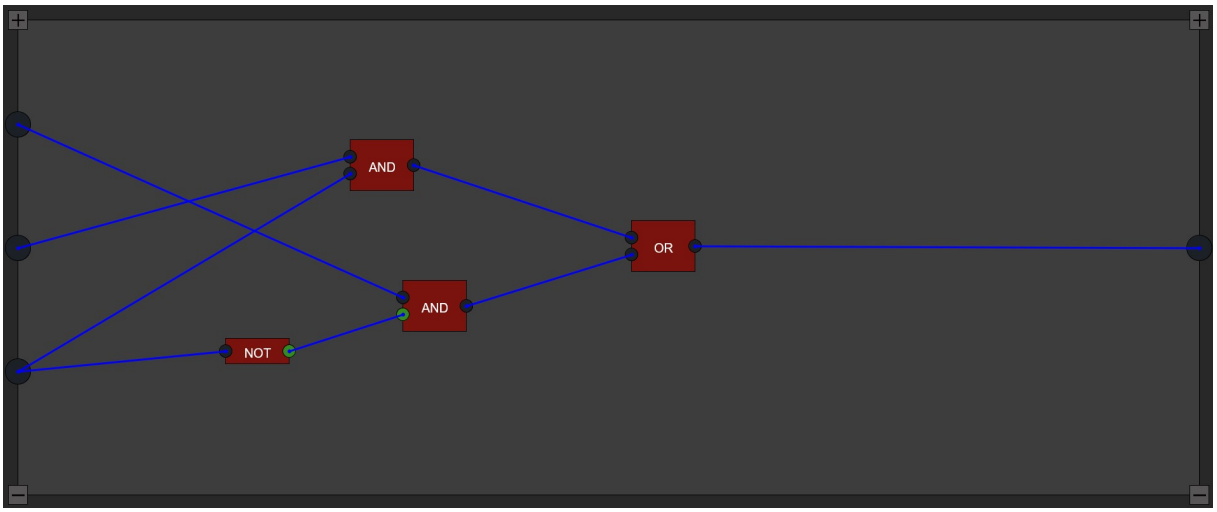


Figura 21 – Circuito Multiplexador 2x1

C.8 Demultiplexador 1X2

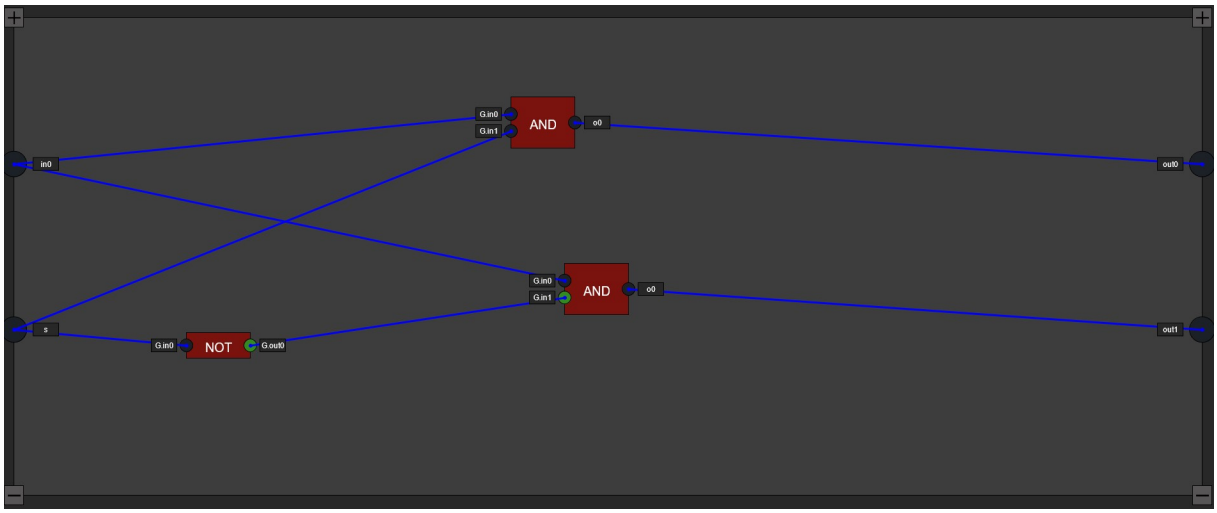


Figura 22 – Circuito Demultiplexador 1x2

C.9 Set Reset Latch

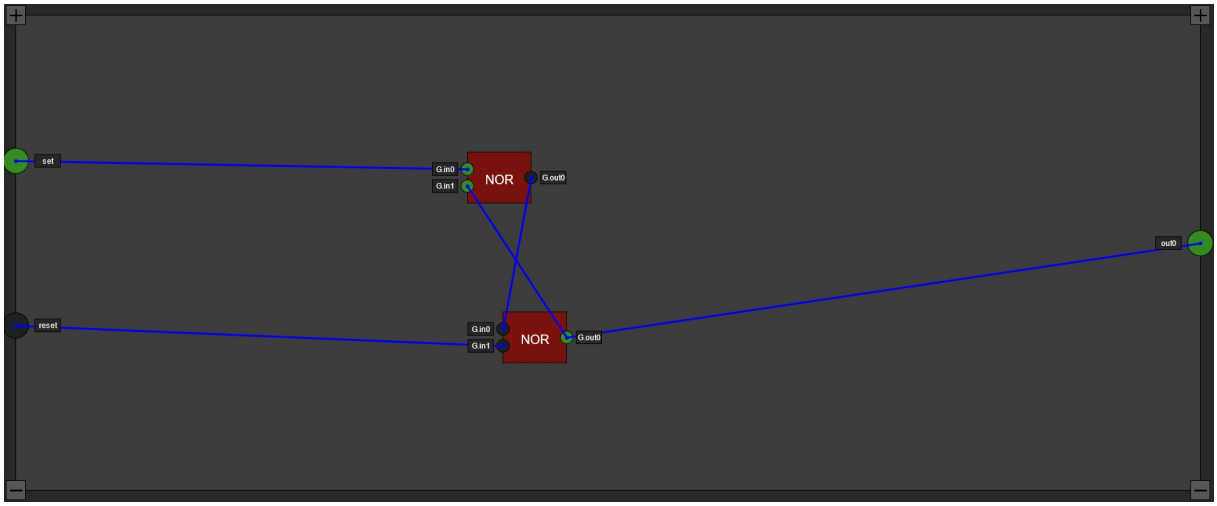


Figura 23 – Circuito Set Reset Latch

C.10 Data Latch

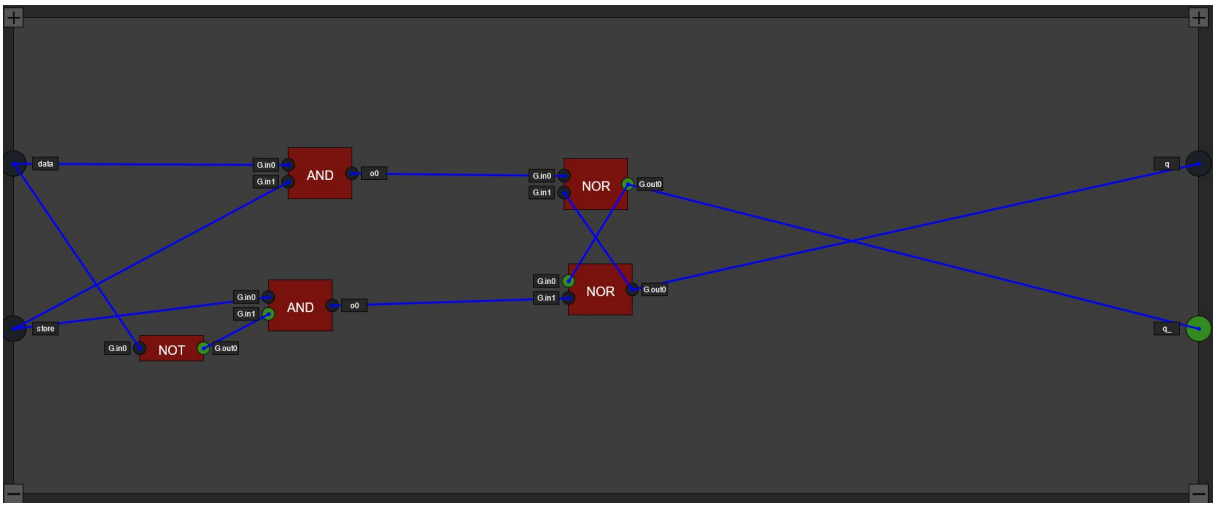


Figura 24 – Circuito Data Latch

C.11 Data Flip Flop

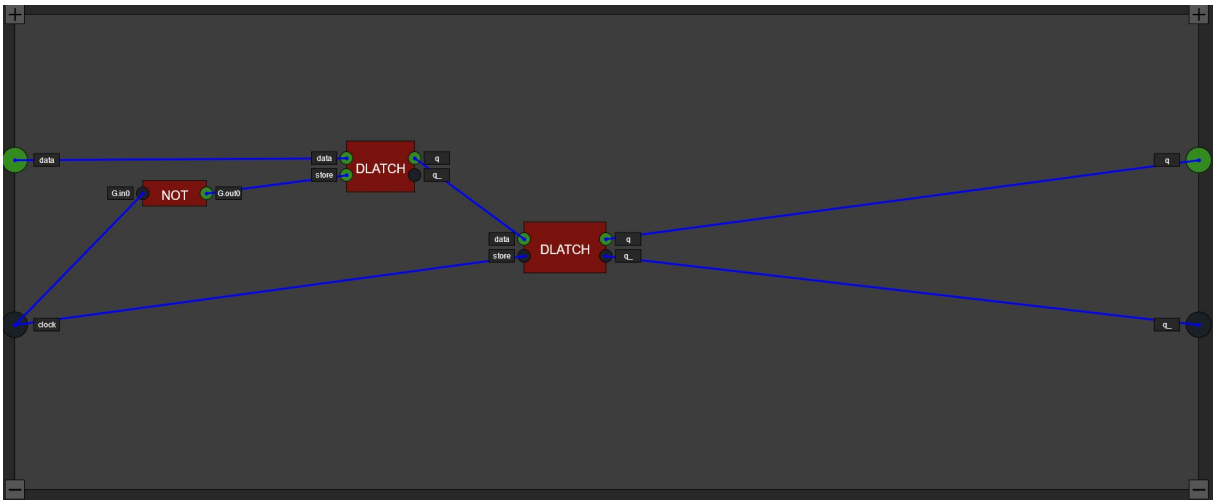


Figura 25 – Circuito Data Flip Flop

C.12 Registrador de 1 bit

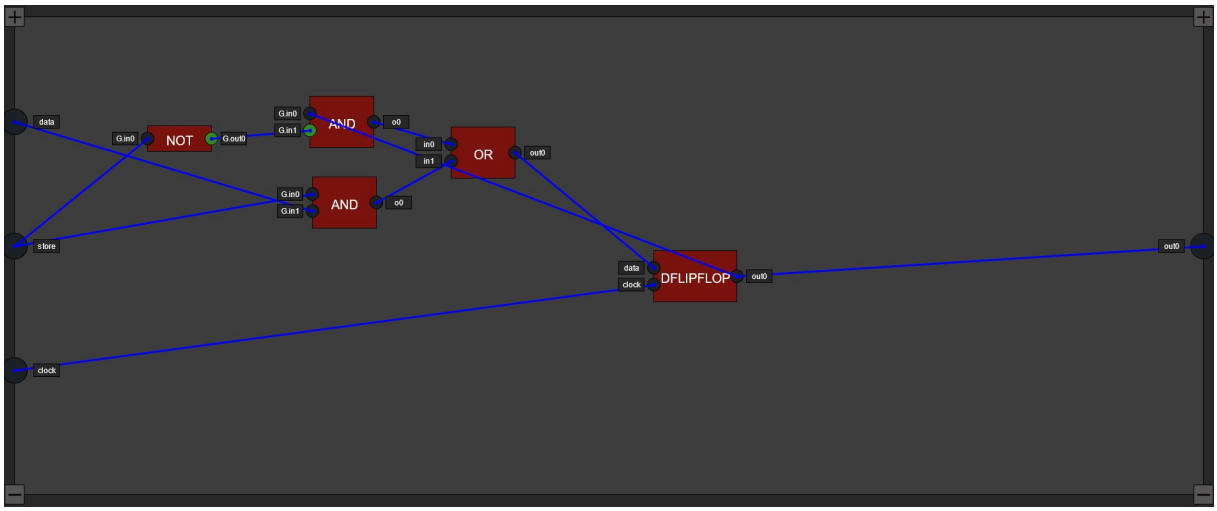


Figura 26 – Circuito Registrador de 1 bit

C.13 Contador

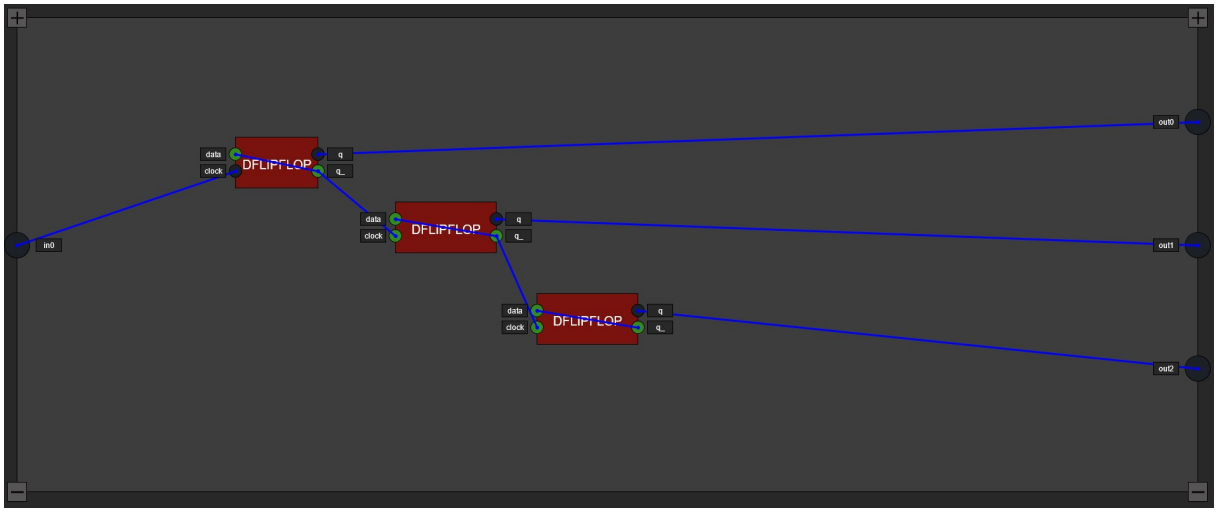


Figura 27 – Circuito Contador

C.14 Unidade Lógica Aritmética (ULA)

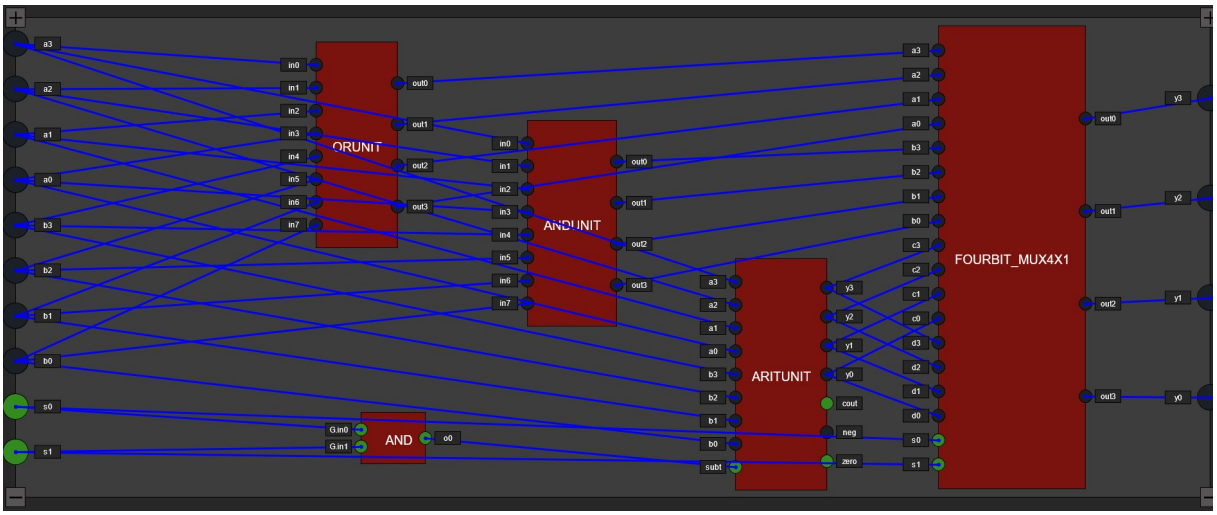


Figura 28 – Circuito Unidade Lógica Aritmética (ULA)

C.15 Memória RAM 4x1

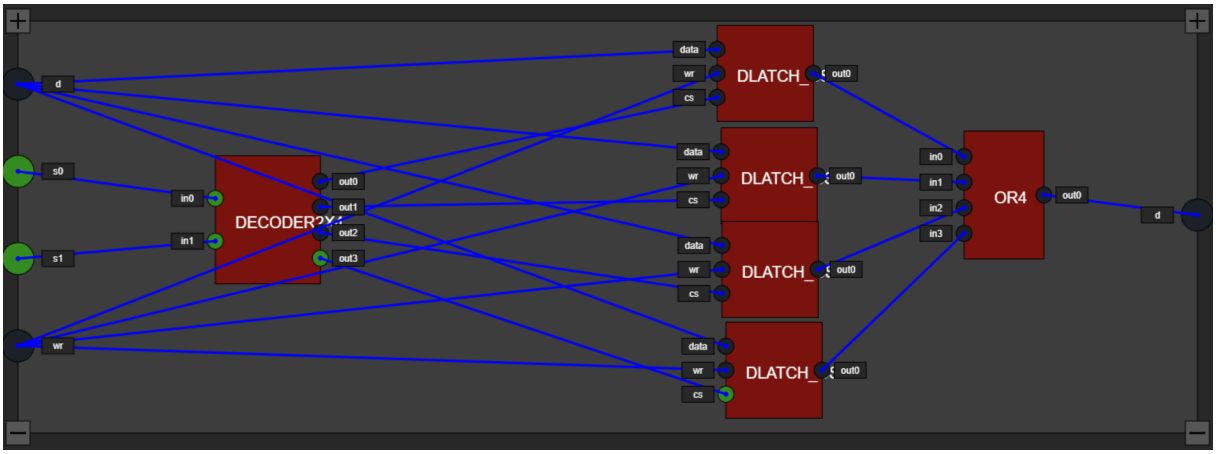


Figura 29 – Circuito Memória RAM 4x1

APÊNDICE D – Resultado do formulário de *feedback* do NANDesis

A seguir estão apresentados todos os *feedback* coletados pelo formulário. Ele inclui 2 respostas, sendo apenas uma delas contendo a área do professor.

- **Sobre o simulador, quanto você avalia cada aspecto (Seja 1 uma nota ruim e 5 uma nota ótima)**

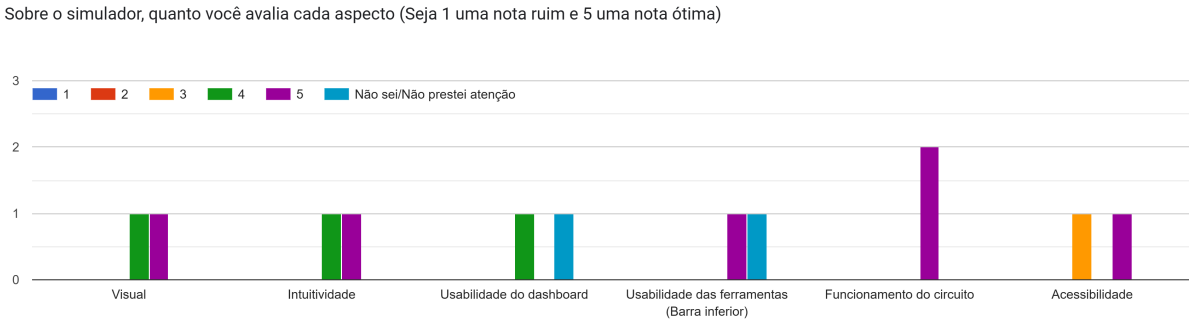


Figura 30 – Respostas da pergunta sobre o simulador

- **Use esse espaço para complementar sua resposta, falar sobre outro ponto que julgar válido ou qualquer sugestão sobre o simulador. (Opcional)**
  Resposta obtida: *"Creio que no estado atual de desenvolvimento, é justificável o fator de acessibilidade não estar evoluído."*

- **Sobre as atividades, quanto você avalia cada aspecto (Seja 1 uma nota ruim e 5 uma nota ótima)**

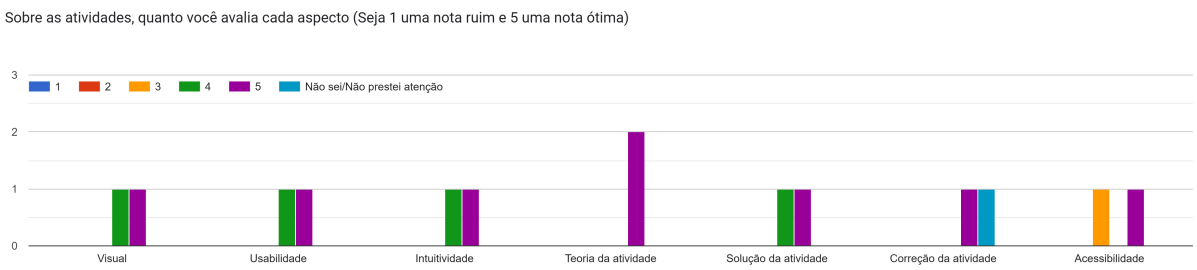


Figura 31 – Respostas da pergunta sobre as atividades a visão do aluno

- **Use esse espaço para complementar sua resposta, falar sobre outro ponto que julgar válido ou qualquer sugestão sobre as atividades. (Opcional)**
  Não há respostas para esta pergunta.
- **Testou a visão do professor?**

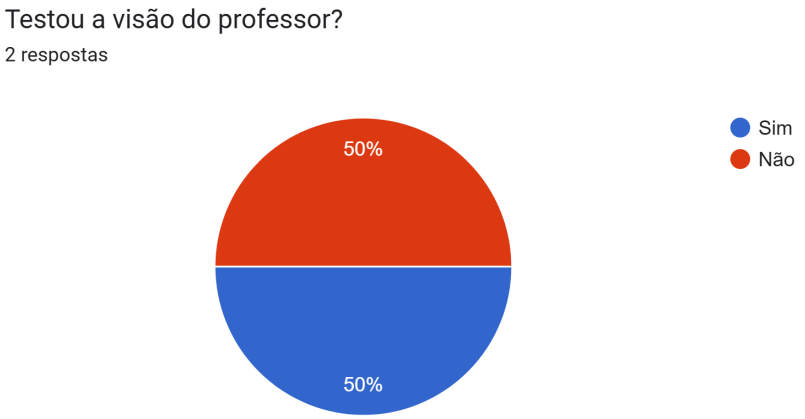


Figura 32 – Respostas validado se o usuário de teste utilizou o ambiente do professor

- **Sobre o gerenciamento de turmas (Criação, listagem, edição e convite de alunos), quanto você avalia cada aspecto (Seja 1 uma nota ruim e 5 uma nota ótima)**

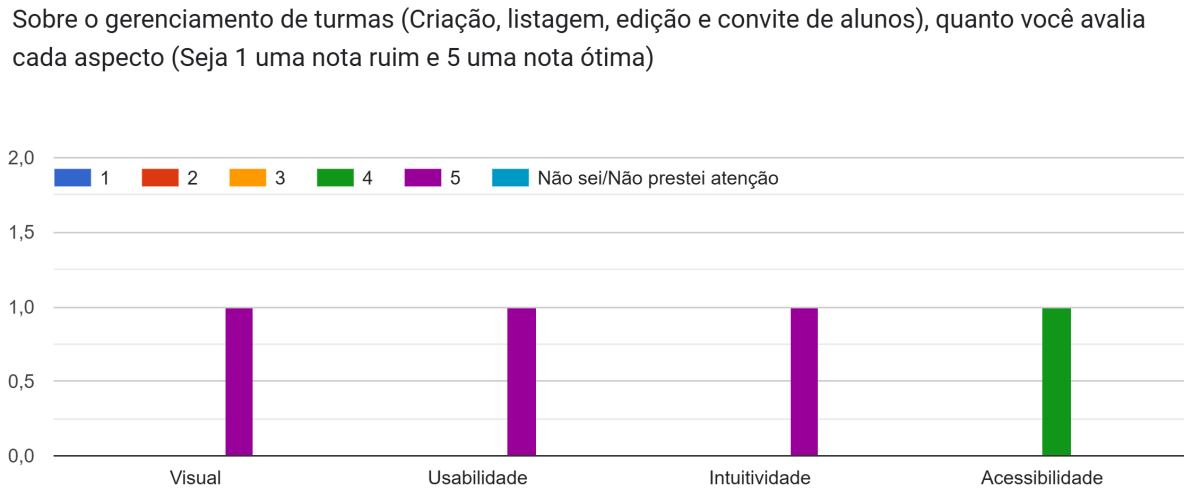


Figura 33 – Respostas da pergunta sobre o gerenciamento de turmas

- **Use esse espaço para complementar sua resposta, falar sobre outro ponto que julgar válido ou qualquer sugestão sobre o gerenciamento de turmas. (Opcional)**
  Resposta obtida: *"Nessa parte, creio que os tipos de interações diferentes que o usuário pode ter afetam menos o fator de acessibilidade"*

- **Sobre o gerenciamento de atividades (Atribuição de datas e listagem e exportação de notas), quanto você avalia cada aspecto (Seja 1 uma nota ruim e 5 uma nota ótima)**

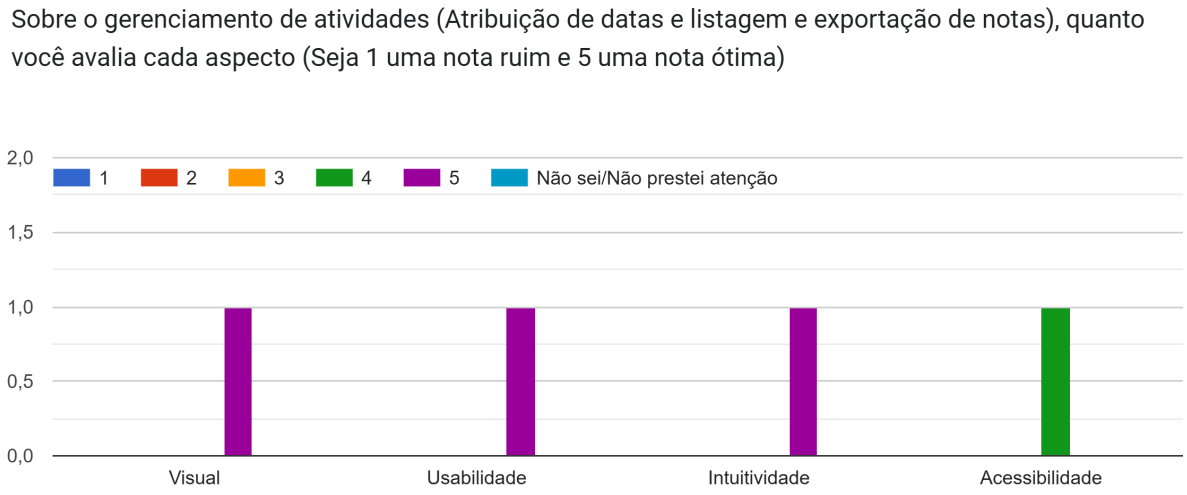


Figura 34 – Respostas da pergunta sobre as atividades a visão do professor

- **Use esse espaço para complementar sua resposta, falar sobre outro ponto que julgar válido ou qualquer sugestão sobre o gerenciamento de atividades. (Opcional)**
  Não há respostas para esta pergunta.

- **Use esse espaço para relatar bugs, problemas não discutidos nas questões anteriores, ou sugestões gerais. (Opcional)**
  Respostas obtidas:

  – *"Houve algumas dificuldades ao conectar entradas e saídas dos circuitos, mas pode ter sido problema de duplo clique do mouse, que poderia causar o que houve em meu teste."*

  – *"Algumas vezes o simulador apresentava pequenos travamentos, ou simplesmente precisava ser reinicializado para que algumas funções voltassem a operar. A área de clique para realizar a interconexão nos circuitos.*
  No mais, é uma aplicação muito interessante, e como sugestão eu deixaria a inserção de um pequeno tutorial ensinando detalhadamente como a ferramenta funciona."