Fundação Universidade Federal do ABC
Pró reitoria de pesquisa
Av. dos Estados, 5001, Santa Terezinha, Santo André/SP, CEP 09210-580
Bloco L, 3ºAndar, Fone (11) 3356-7617
iniciacao@ufabc.edu.br

Relatório Final de Iniciação Científica referente ao Edital: (Nº 1/2024 - PROPESDAPIC (11.01.07.14))

Nome do aluno: Vitor Vieira Fernandes

Assinatura do aluno:

Nome do orientador: Mario Alexandre Gazziro

Assinatura do orientador:

Título do projeto: Explorando ferramentas modernas para desenvolvimento de jogos no NES: Um estudo da plataforma e aplicações práticas

Palavras-chave do projeto: NES, FPGA, desenvolvimento de jogos, programação embarcada, retrocomputação

Área do conhecimento do projeto: Computação e Engenharias

Bolsista: Sim (PIC - UFABC)

Santo André

2025

Sumário

1 - Resumo

Este trabalho explorou a arquitetura do Nintendo Entertainment System (NES) e as particularidades do desenvolvimento de jogos para a plataforma, utilizando ferramentas modernas como FPGAs e a linguagem C. O projeto culminou na criação de um demake do jogo Flappy Bird, adaptando suas mecânicas às restrições técnicas do NES, como limitações de memória e processamento gráfico. A metodologia combinou a análise documental da arquitetura do console com o desenvolvimento prático iterativo, empregando a FPGA Tang Nano 20K para emulação fiel do hardware e a biblioteca NESlib para a programação em C. Os resultados demonstraram a viabilidade de recriar um jogo moderno no NES, com a implementação bem-sucedida de mecânicas como scrolling horizontal, geração procedural de obstáculos e detecção de colisão, alcançando 60FPS estáveis no hardware. O principal desafio, a restrita capacidade de atualização da VRAM, foi solucionado com a divisão da renderização em múltiplos quadros. Conclui-se que o projeto não apenas valida a aplicação de ferramentas modernas no estudo de hardware clássico, mas também serve como um recurso didático aplicado de arquitetura de computadores. O repositório público com o código-fonte e a documentação detalhada do processo constitui uma contribuição valiosa para a comunidade de retrocomputação.

2 - Introdução

O Nintendo Entertainment System (NES) é um dos consoles mais icônicos da história dos videogames, desempenhando um papel fundamental na revitalização da indústria após a crise de 1983 (crise dos videogames, marcada pela saturação de títulos de baixa qualidade e queda nas vendas). Lançado no Japão como Family Computer (Famicom) em 1983 e posteriormente nos Estados Unidos em 1985, o NES popularizou franquias consagradas, como Super Mario Bros., The Legend of Zelda e Metroid. Seu sucesso deve-se não apenas à qualidade de seus jogos, mas também à sua arquitetura de hardware eficiente, que impôs desafios técnicos significativos aos desenvolvedores da época. O console era baseado no processador Ricoh 2A03 (uma variante do MOS 6502), com uma Picture Processing Unit (PPU) dedicada ao processamento gráfico e uma Audio Processing Unit (APU) para geração de som.

Nos últimos anos, o interesse por videogames clássicos tem ido além da nostalgia, abrangendo a preservação digital e o ensino de computação de baixo nível. Nesse contexto, as Field Programmable Gate Arrays (FPGAs), dispositivos eletrônicos reconfiguráveis que permitem a emulação precisa de hardware, surgem como uma alternativa avançada para a simulação de consoles antigos, oferecendo um ambiente de estudo próximo ao funcionamento do hardware original [Sieber, 2013]. Paralelamente, a programação para o NES em linguagem C proporciona uma oportunidade educacional única para explorar conceitos essenciais de otimização de código, gerenciamento de memória e processamento gráfico [HUGG, 2019].

Este estudo explora a arquitetura do NES e as particularidades do desenvolvimento de jogos para essa plataforma. Para isso, adota-se uma abordagem baseada na análise documental, experimentação prática com ferramentas contemporâneas e na implementação de um demake (uma versão simplificada e adaptada para hardware mais antigo) do jogo Flappy Bird, como estudo de

caso. O desenvolvimento respeita as restrições técnicas do console, empregando ferramentas como a biblioteca NESlib (uma biblioteca de desenvolvimento para o NES em linguagem C), o compilador cc65, e ferramentas especializadas como 8bitworkshop, NESst, FamiStudio e Mesen Emulator para programação, edição de gráficos, composição sonora e testes de código. O hardware do NES foi recriado em FPGA por meio da Tang Nano 20K, utilizando como base o projeto open-source NESTang.

A principal contribuição científica deste estudo reside na documentação detalhada da programação para um console clássico, promovendo o ensino de arquitetura de computadores e programação embarcada de forma aplicada. Os resultados obtidos também servirão de base para a segunda edição do livro Computadores e Videogames: Uma Abordagem Prática das Arquiteturas Clássicas [ASSUMPÇÃO JUNIOR et al., 2023], obra publicada pelo grupo de pesquisa deste projeto, ampliando sua relevância no contexto acadêmico e educacional.

Dessa forma, este projeto não apenas reforça a importância da preservação da história dos videogames, mas também demonstra como tecnologias modernas podem ser aplicadas ao estudo de arquiteturas computacionais clássicas. Espera-se que este trabalho inspire novos desenvolvedores e pesquisadores a explorar a interseção entre hardware retrô e computação moderna.

3 - Fundamentação Teórica

3.1 - FPGA e sua Aplicação na Emulação de Hardware

As Field Programmable Gate Arrays (FPGAs) são dispositivos eletrônicos reconfiguráveis que permitem a implementação direta de circuitos digitais. Diferentemente dos emuladores baseados em software, que dependem de interpretação dinâmica de código ou tradução binária, as FPGAs implementam fisicamente os circuitos do hardware original através de portas lógicas programáveis. Essa abordagem garante maior fidelidade na execução dos sistemas, preservando o comportamento temporal exato dos componentes. Por exemplo, a PPU (Picture Processing Unit) do NES exige sincronização precisa de ciclos para renderizar sprites sem glitches — algo trivialmente alcançado em FPGAs, mas desafiador em emuladores tradicionais [Sieber, 2013].

A principal vantagem das FPGAs sobre os emuladores tradicionais reside na redução da latência e na precisão dos ciclos de processamento. Em emuladores baseados em software, pequenas variações na execução das instruções podem causar incompatibilidades com jogos mais sensíveis ao tempo. Já em FPGAs, o hardware é descrito em linguagens de descrição de hardware (HDL), permitindo que o comportamento do NES seja recriado com sincronização ciclo a ciclo, o que garante uma experiência de jogo idêntica à do console original [Andreeti, 2015]. Além disso, as FPGAs possibilitam expansões e melhorias, como saída de vídeo em alta definição, sem comprometer a compatibilidade com o software original.

Uma das soluções mais promissoras para a emulação do NES via FPGA é o projeto open-source NESTang. O NESTang é uma implementação do NES para diversas placas FPGA, incluindo a Tang Nano 20K, utilizada neste estudo. Essa placa é baseada na FPGA GW2AR-18 QN88 e contém 20.736 células lógicas (LUT4) e 15.552 Flip-Flops, além de duas unidades PLL e blocos DSP para

operações matemáticas otimizadas. Ela suporta saída HDMI a 720p com áudio, leitura de ROMs via MicroSD e compatibilidade com controles NES, SNES e PS2 [NESTang, 2023].

A programação do NES na Tang Nano 20K via NESTang envolve a gravação de um firmware já pronto, disponibilizado pela Sipeed. Para isso, é necessário transferir a imagem compilada diretamente para a FPGA. O código-fonte do NESTang pode ser sintetizado do zero utilizando a IDE Gowin, permitindo ajustes no comportamento do console.

Sipeed FPGA Tang Nano 20k

3.2 - Arquitetura e Hardware do NES

A arquitetura do NES é baseada no processador Ricoh 2A03, uma versão customizada do MOS 6502, e inclui componentes dedicados para gráficos e som: a Picture Processing Unit (PPU) e a Audio Processing Unit (APU). Esses três elementos formam o núcleo do sistema e são essenciais para entender o funcionamento do console em nível de hardware.

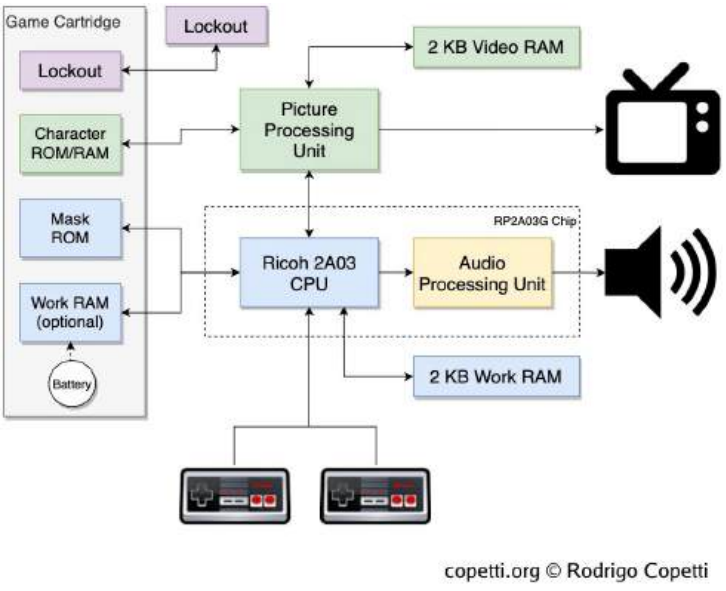


Diagrama de arquitetura principal do NES

3.2.1 - CPU – Ricoh 2A03

O processador central do NES, o Ricoh 2A03, é uma variante do MOS 6502, um dos processadores mais populares da época. Ele opera a 1,79 MHz em sistemas NTSC e a 1,66 MHz em sistemas PAL. Apesar de manter a maioria das instruções do 6502, o 2A03 não possui suporte ao modo BCD (Binary Coded Decimal), isso forçava programadores a implementar rotinas manuais para cálculos decimais, aumentando o uso de memória e ciclos de CPU.

A CPU utiliza um barramento de 8 bits e pode acessar até 64 KB de memória, embora parte desse espaço seja reservada para comunicação com outros componentes do sistema. O NES possui 2 KB de RAM interna (Work RAM - WRAM), usada para armazenar variáveis e a pilha de execução. Além disso, o console depende fortemente da memória presente nos cartuchos, que podem incluir ROMs de programa, RAM adicional e circuitos de mapeamento de memória (mappers) para estender suas capacidades.

A arquitetura do NES emprega um sistema de endereçamento mapeado em memória, permitindo que a CPU interaja diretamente com a PPU, a APU e outros periféricos através de registradores específicos. Isso possibilita o controle dos gráficos, som e entrada dos controles sem a necessidade de um barramento de comunicação separado.

3.2.2 - PPU – Picture Processing Unit

A PPU é o componente que define a identidade visual do NES. Operando de forma independente da CPU, ela é responsável por renderizar o quadro de 256×240 pixels. Sua arquitetura é baseada em *tiles*, o que impõe um conjunto único de limitações e possibilidades. Os conceitos gráficos fundamentais são:

- Pattern Tables e Tiles: Os gráficos são construídos a partir de tiles de 8×8 pixels. Esses tiles são armazenados na memória do cartucho (CHR ROM ou CHR RAM) em duas "tabelas de padrões" (*Pattern Tables*), que funcionam como o repositório de todos os gráficos disponíveis para o jogo [HUGG, 2019].

- Nametables e Background: O cenário de fundo (*background*) é construído organizando-se os tiles em um mapa de 32×30, chamado de *Nametable*. A VRAM interna do NES armazena duas *Nametables*, o que permite a implementação de *scrolling* através de uma técnica de espelhamento (*mirroring*) [COPETTI, 2025].

- Attribute Tables e Paletas: As cores são aplicadas em blocos de 2×2 tiles (16×16 pixels) por meio das *Attribute Tables*. Cada bloco compartilha uma mesma paleta, composta por até 3 cores mais uma cor transparente. No total, a PPU pode exibir 25 cores simultaneamente de uma paleta mestra de 64 cores, uma restrição que define a estética visual do console [HUGG, 2019].

- Sprites e OAM: Os elementos móveis (personagens, inimigos, projéteis) são chamados de *sprites*. Eles são gerenciados em uma memória interna à PPU de 256 bytes chamada OAM (*Object*

Attribute Memory). O sistema suporta até 64 sprites na tela, com um limite de 8 por linha horizontal, o que pode causar o conhecido efeito de *flickering* (piscamento) quando o limite é excedido [COPETTI, 2025].

3.2.3 - APU – Audio Processing Unit

A Audio Processing Unit (APU) do NES, integrada ao Ricoh 2A03, é responsável pela geração de som e conta com cinco canais de áudio:

1. Dois canais de onda quadrada (pulse wave) – Utilizados para melodias e efeitos sonoros, com suporte a duty cycles variáveis e controle de volume.

2. Um canal de onda triangular – Principalmente usado para linhas de baixo, mas também empregado na geração de efeitos percussivos.

3. Um canal de ruído (noise channel) – Produz sons de percussão e efeitos ambientais, como explosões e vento.

4. Um canal de amostra digital (DMC - Delta Modulation Channel) – Capaz de reproduzir amostras de áudio digitalizadas (como vozes e batidas), embora com qualidade limitada devido à baixa taxa de amostragem (~15,7 kHz) e resolução de 7 bits.

A APU do NES é um gerador de som programável (PSG), permitindo que os jogos sintetizem áudio diretamente por meio da CPU. Como não havia suporte nativo para reverberação ou eco, os compositores e programadores da época exploraram técnicas criativas para maximizar a qualidade sonora dentro das limitações do console.

3.3 - Desenvolvimento de Jogos para o NES em linguagem C

O desenvolvimento de jogos para o Nintendo Entertainment System (NES) envolve desafios técnicos significativos devido às limitações do hardware da plataforma. No entanto, ferramentas como a 8bitworkshop IDE e um pipeline de compilação estruturado simplificam esse processo, tornando possível programar para o NES utilizando linguagem C ao invés de Assembly. Essas ferramentas oferecem um ambiente mais acessível para desenvolvedores e pesquisadores interessados na programação para sistemas embarcados de 8 bits.

3.3.1 - 8bitworkshop IDE e seu Papel no Desenvolvimento

A 8bitworkshop IDE é uma plataforma de desenvolvimento integrada voltada para a criação de jogos em linguagens de baixo nível, como Assembly e C, para sistemas retrô como o NES. Seu diferencial está na capacidade de compilar e testar o código em tempo real, permitindo um ciclo de desenvolvimento mais ágil. Segundo “Making Games for the NES”, a IDE fornece um ambiente unificado, no qual os desenvolvedores podem escrever código, visualizar alterações instantaneamente e até realizar controle de versão via GitHub.

A interface simplificada da 8bitworkshop permite que desenvolvedores interajam diretamente com a PPU (Picture Processing Unit) e gerenciem os sprites e tiles do jogo sem necessidade de alternar entre múltiplas ferramentas externas. Além disso, o sistema integrado da IDE inclui funcionalidades de debugging, auxiliando na otimização da performance e na detecção de erros na renderização e lógica do jogo.

O uso da 8bitworkshop se mostra vantajoso para desenvolvedores iniciantes, pois elimina grande parte da complexidade envolvida na configuração manual do ambiente de desenvolvimento para o NES. Dessa forma, a IDE serve como uma ponte entre a programação moderna e os desafios técnicos do hardware de 8 bits, tornando a criação de jogos para o NES mais acessível e eficiente.

3.3.2 - Pipeline de Compilação e Geração de ROMs

Outro aspecto essencial no desenvolvimento para NES é o processo de compilação e construção da ROM. Diferente de plataformas modernas, onde o código é executado diretamente pelo sistema operacional, no NES o jogo precisa ser compilado para um formato que possa ser lido pelo hardware do console ou por um emulador. O livro Making Games for the NES detalha esse pipeline de compilação e linkagem, mostrando como transformar código-fonte em arquivos ROM compatíveis com o NES.

A compilação para NES utilizando linguagem C é viabilizada pelo cc65, um compilador otimizado para arquiteturas baseadas no MOS 6502. Esse compilador converte o código C para Assembly e, posteriormente, gera um binário que pode ser executado no NES. O pipeline típico envolve as seguintes etapas: (1) Compilação – O código-fonte em C é convertido para Assembly 6502. (2) Linkagem – O código Assembly é reunido em um único binário compatível com o NES. (3) Geração da ROM – O binário é formatado corretamente para ser carregado pelo console. (4) Testes em Emuladores – A ROM gerada é executada em um emulador ou diretamente na FPGA.

Esse fluxo de trabalho estruturado permite que desenvolvedores utilizem a linguagem C de maneira eficiente, aproveitando a portabilidade e a legibilidade do código sem comprometer a performance do NES. Além disso, a automação do processo de build reduz a chance de erros e facilita a implementação de atualizações e otimizações no jogo.

Assim, a combinação da 8bitworkshop IDE com um pipeline de compilação eficiente permite um desenvolvimento mais fluido e acessível para o NES, democratizando o acesso à programação para consoles clássicos e possibilitando a criação de jogos dentro das restrições do hardware original.

3.4 - As Técnicas de Desenvolvimento

3.4.1 - Construção dos Gráficos

A PPU do NES opera em um barramento separado da CPU e possui sua própria memória, chamada VRAM (Video RAM), onde são armazenados os dados gráficos do jogo. Em vez de utilizar bitmaps para representar imagens diretamente na tela, o NES organiza gráficos em tiles de 8×8

pixels, que podem ser combinados para formar cenários e personagens. Esses tiles são armazenados nas Pattern Tables, e sua disposição na tela é definida pelas Nametables.

Os gráficos são gerenciados e exibidos através de um ciclo de renderização sincronizado com a taxa de atualização da tela (60 Hz no padrão NTSC). Durante a execução do jogo, a VRAM só pode ser modificada em um curto intervalo de tempo conhecido como Vertical Blank (VBlank), exigindo que atualizações na tela sejam cuidadosamente planejadas para evitar artefatos visuais.

As Nametables são responsáveis pela organização dos tiles na tela, formando o cenário do jogo. Cada Nametable define uma matriz de 32×30 tiles, cobrindo a área visível do NES, que possui resolução de 256×240 pixels. Como a VRAM do NES tem espaço para armazenar apenas duas Nametables ao mesmo tempo, o console utiliza um sistema de mirroring, no qual as quatro Name Tables lógicas (A, B, C e D) são espelhadas conforme a necessidade do jogo.

Além de definir a disposição dos tiles na tela, o NES precisa gerenciar a coloração dos gráficos por meio das Attribute Tables. Cada Attribute Table controla as cores aplicadas aos tiles do background, agrupando-os em blocos de 2×2 tiles. Como consequência, todos os 4 tiles dentro desse bloco compartilham a mesma paleta de cores, o que pode limitar a flexibilidade artística dos jogos.

A PPU do NES suporta 64 cores, mas apenas 25 podem ser exibidas simultaneamente (13 para o background e 12 para os sprites). Para contornar essa limitação, os desenvolvedores planejam cuidadosamente a distribuição das paletas dentro da Attribute Table, garantindo que os gráficos tenham uma aparência coerente sem ultrapassar as restrições impostas pelo hardware.

Os sprites são os elementos gráficos móveis do NES, armazenados em uma área especial chamada Object Attribute Memory (OAM). Cada sprite individual ocupa um espaço de 8×8 pixels, podendo ser combinados para criar objetos maiores. A PPU consegue processar até 64 sprites simultaneamente, mas impõe um limite de 8 sprites por linha horizontal. Quando esse limite é ultrapassado, é utilizado o efeito de sprite flickering, no qual alguns sprites piscam na tela à medida que a PPU alterna entre eles para tentar exibir todos.

Os sprites no NES podem ser configurados para ficarem em primeiro plano (na frente do background) ou atrás do cenário, além de poderem ser espelhados horizontalmente e verticalmente (flip). No entanto, devido às restrições de paleta de cores da PPU, cada sprite pode utilizar apenas 3 cores mais a transparência, o que exige um planejamento cuidadoso na criação dos gráficos dos personagens e inimigos.

Para superar as limitações dos sprites individuais de 8×8 pixels, os desenvolvedores utilizam a técnica de metasprites, na qual múltiplos sprites são combinados para formar um único objeto maior. Essa abordagem permite a criação de personagens e inimigos mais detalhados, mantendo um controle eficiente sobre a alocação de sprites na OAM. A organização dos metasprites geralmente segue um padrão de 16×16 pixels ou 32×32 pixels, dependendo da complexidade do objeto, mas pode ser arranjada da maneira desejada pelo desenvolvedor.

3.4.2 - Técnicas de Scrolling

A implementação de scrolling no NES é essencial para jogos que apresentam mundos maiores do que a tela visível. Como a PPU só pode armazenar duas Nametables (tela completa de 256×240 pixels) ao mesmo tempo, a rolagem do cenário é realizada alternando entre elas, criando a ilusão de continuidade.

O NES permite dois tipos principais de scrolling: horizontal e vertical. Para scrolling horizontal, são utilizadas as Nametables A e B, com a técnica de vertical mirroring, que espelha automaticamente o cenário. Já no scrolling vertical, são usadas as Nametables A e C, com horizontal mirroring. Essa configuração permite que o jogo alterne entre diferentes seções do mundo sem precisar armazenar todas as informações simultaneamente na VRAM.

Para evitar que elementos gráficos sejam abruptamente removidos da tela, os jogos utilizam um sistema de "offscreen scrolling", onde partes do cenário são carregadas antes de se tornarem visíveis. Assim, ao avançar no jogo, novas seções do background são desenhadas na Nametable que está fora da tela, garantindo uma transição suave sem atrasos perceptíveis.

Muitos jogos do NES utilizam uma barra de status fixa na parte superior ou inferior da tela para exibir informações como pontuação, tempo e número de vidas. Entretanto, ao aplicar scrolling no cenário, essa barra também se moveria junto com o background. Para resolver esse problema, os desenvolvedores utilizam a técnica de Split Screen Status Bar, que divide a tela em duas seções com regras diferentes de scrolling.

A implementação desse recurso é feita utilizando a técnica do Sprite Zero, um sprite especial posicionado estrategicamente na tela para indicar à CPU o momento exato em que deve ocorrer a mudança no scrolling. Quando a PPU detecta a colisão entre o Sprite Zero e o background, o jogo pode modificar a posição do scrolling na linha seguinte, permitindo que a parte superior da tela permaneça estática enquanto o restante continua se movimentando.

3.4.3 - Atualizações Dinâmicas da Tela

Uma das principais limitações da PPU do NES é a impossibilidade de modificar a VRAM enquanto a tela está sendo desenhada. Todas as alterações nos gráficos precisam ser feitas durante o curto período do Vertical Blank (VBlank), que ocorre a cada 1/60 de segundo no padrão NTSC. Esse intervalo de tempo permite a escrita de no máximo 128 bytes por quadro, exigindo uma abordagem estratégica para atualizar os gráficos sem comprometer a taxa de quadros.

Para contornar essa limitação, os desenvolvedores utilizam buffers de VRAM, que armazenam todas as modificações a serem aplicadas na próxima oportunidade de VBlank. A NESlib fornece funções como vrambuf_put() e vrambuf_flush(), que adicionam dados ao buffer e os escrevem na VRAM de forma otimizada. Essa técnica é essencial para jogos que requerem atualização dinâmica do cenário, como aqueles que possuem scrolling procedural ou geração de elementos aleatórios.

O uso eficiente de buffers de VRAM em conjunto com offscreen scrolling permite que novos elementos do background sejam desenhados antes de entrarem na área visível, garantindo uma experiência fluida para o jogador. Essa abordagem foi amplamente utilizada em jogos de plataforma e shooters, onde a movimentação contínua do cenário é um aspecto central da jogabilidade.

3.4.4 - Geração de Números Aleatórios

A geração de números aleatórios no NES é baseada em algoritmos de números pseudo aleatórios, pois o console não possui um gerador de entropia real. O método mais comum é a utilização de um Linear Feedback Shift Register (LFSR), que gera sequências aparentemente aleatórias a partir de um valor inicial (seed).

A NESlib fornece funções como rand8() e rand16(), que geram valores de 8 e 16 bits, respectivamente. No entanto, como essas funções seguem um padrão determinístico, os valores gerados são sempre os mesmos se o jogo começar com a mesma seed. Para aumentar a aleatoriedade, os desenvolvedores utilizam eventos variáveis como entrada do usuário (por exemplo, o tempo que o jogador leva para pressionar Start) como base para inicializar a seed com set_rand(nesclock()), garantindo que cada partida tenha resultados distintos.

4 - Metodologia

4.1 - Materiais e Métodos

4.1.1 - Abordagem Geral

Esta pesquisa adota uma abordagem híbrida, integrando fundamentação teórica e desenvolvimento prático para criar um demake funcional de Flappy Bird que respeite integralmente as especificações do Nintendo Entertainment System (NES). O estudo foi estruturado em três eixos complementares, garantindo rigor acadêmico e aplicabilidade dos resultados.

A fundamentação teórica baseou-se no estudo aprofundado da arquitetura do NES por meio de fontes primárias. Foram analisados manuais técnicos do hardware, complementados pelo livro Making Games for the NES de Steven Hugg, que fornece diretrizes práticas para desenvolvimento na plataforma. A NES Dev Wiki também foi utilizada como referência para técnicas documentadas pela comunidade de desenvolvedores. Esse arcabouço teórico foi essencial para compreender as restrições técnicas do console e explorar soluções inovadoras dentro desses limites.

No desenvolvimento prático, adotou-se uma metodologia iterativa, com ciclos progressivos de prototipagem, teste e refinamento. O processo iniciou-se com uma prova de conceito envolvendo renderização de sprites e scrolling horizontal, evoluindo para a implementação de mecânicas mais complexas, como física de movimento, geração procedural de obstáculos e sistemas de colisão. Cada iteração foi submetida a testes de desempenho, assegurando conformidade com as especificações originais do console.

A escolha das ferramentas equilibrou a fidelidade histórica e eficiência no desenvolvimento contemporâneo. Para garantir a precisão na emulação, optou-se pela FPGA Tang Nano 20K com o projeto NESTang, que replica fielmente o hardware original ciclo a ciclo, mantendo-se uma solução de custo acessível e de código aberto. Paralelamente, a adoção da NESlib permitiu o uso da linguagem C, combinando produtividade com um aprendizado aprofundado de programação de baixo nível, tornando o processo mais acessível a novos desenvolvedores sem comprometer o entendimento da arquitetura.

A validação sistemática ocorreu em múltiplos níveis, desde testes funcionais no emulador Mesen – com seu avançado sistema de debugging – até a verificação final na FPGA, assegurando compatibilidade absoluta com o hardware original. Por fim, a documentação detalhada do desenvolvimento constitui uma das principais contribuições deste estudo. Todo o código-fonte e suas revisões iterativas foram registrados em um repositório público, garantindo transparência e possibilitando futuras análises e aprimoramentos por outros pesquisadores e desenvolvedores.

4.1.2 - Preparação do Ambiente

Para viabilizar o desenvolvimento e teste do jogo, foi necessário configurar um ambiente que permitisse tanto a emulação quanto a execução no hardware original. A configuração incluiu a instalação do NESTang na FPGA Tang Nano 20K, utilizando a Gowin IDE e os arquivos do repositório oficial do NESTang no GitHub.

Durante o desenvolvimento, o emulador Mesen foi utilizado como ferramenta primária de teste, pois permite depuração avançada, visualização da memória e das nametables, oferecendo um fluxo de iteração mais rápido do que testar diretamente na FPGA. Para edição e manipulação gráfica, foi utilizada a NES Screen Tool, e o FamiStudio foi empregado para a composição e inserção de áudio no jogo.

A plataforma 8bitworkshop foi escolhida como ambiente principal de desenvolvimento, pois oferece uma IDE integrada com suporte para escrita de código, testes em tempo real e controle de versionamento via GitHub. Essa abordagem permitiu uma iteração rápida entre desenvolvimento e testes, otimizando o fluxo de trabalho.

Para a validação do jogo no hardware real, a ROM foi exportada diretamente do 8bitworkshop, transferida para um cartão microSD e executada na FPGA, que realiza a leitura do arquivo ".nes" e gera a saída de vídeo via HDMI.

4.2 - Etapas da Pesquisa

4.2.1 - Escolha do Jogo

A escolha de Flappy Bird como base para este demake fundamenta-se em três principais motivos. Primeiramente, o projeto busca demonstrar a viabilidade de recriar um jogo de sucesso de

dispositivos móveis em um console de 1985, cujo hardware é significativamente mais limitado que o dos smartphones contemporâneos.

Além disso, apesar de sua aparente simplicidade, Flappy Bird engloba diversas funcionalidades essenciais do NES, tornando-se um estudo abrangente da plataforma. O jogo exige a implementação de scrolling horizontal para criar a ilusão de um mundo gerado proceduralmente, metasprites para a composição do personagem, uma barra de status baseada na técnica do "sprite zero", geração de números aleatórios para a variação dos obstáculos, e áudio sincronizado com a jogabilidade. Adicionalmente, a manipulação da VRAM via buffers é necessária para atualizar os elementos do cenário fora da tela (offscreen).

Por fim, a escolha também se justifica pela direção artística do jogo original, que utiliza pixel art—um requisito essencial para o NES, cuja resolução é limitada a 256×240 pixels. Isso possibilita uma recriação visual fiel, com adaptações mínimas na paleta de cores para se adequar às restrições do console.

4.2.2 - Os Gráficos

O desenvolvimento do jogo iniciou-se pela criação dos gráficos, garantindo que todos os elementos visuais respeitassem as limitações do NES. Para isso, utilizou-se a ferramenta NES Screen Tool, que permitiu a construção e organização das Pattern Tables, Nametables e paletas de cores de forma eficiente. Os gráficos foram recriados com base no jogo original, porém com algumas adaptações para atender às restrições do console. O NES utiliza tiles de 8×8 pixels, permitindo até 4 cores por tile, incluindo transparência. Como alguns elementos do jogo original possuíam mais cores ou não se encaixavam bem na grade 8×8, foi necessário ajustar a composição visual.

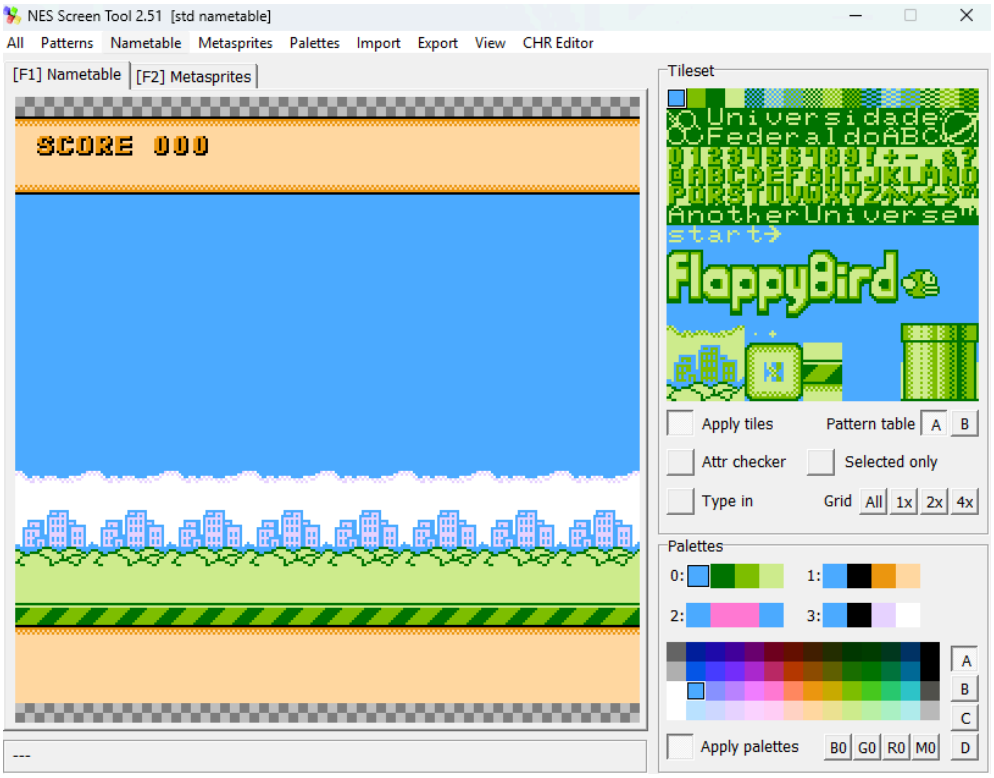
Para contornar essas limitações, os canos, arbustos e o chão passaram a compartilhar a mesma paleta de cores, reduzindo o número de paletas utilizadas. Além disso, o céu foi pintado com a cor transparente comum às quatro paletas disponíveis, otimizando o uso de cores no background.

Para otimizar o uso da memória gráfica, os elementos foram distribuídos nas duas Pattern Tables do NES. A Pattern Table de Background incluiu os elementos gráficos como chão, arbustos, prédios, nuvens, canos, título do jogo, números e letras. Já a Pattern Table de Sprites foi utilizada para armazenar os quatro tiles que compõem o pássaro, além do sprite zero, que será essencial para a implementação do split screen quando o scrolling for adicionado.

Como a implementação inclui scrolling horizontal, foram configuradas duas nametables idênticas para possibilitar uma transição suave do cenário. Além disso, cada nametable possui uma Attribute Table, que determina quais paletas de cores são atribuídas aos tiles. Como a Attribute Table opera em blocos de 2×2 tiles, foi necessário garantir que os elementos gráficos seguissem esse padrão para evitar problemas de coloração.

Após a definição dos gráficos, foi necessário carregá-los na VRAM da PPU. A biblioteca NESlib facilitou esse processo, fornecendo funções específicas para configurar as paletas de cores e escrever a nametable na VRAM. Como a VRAM só pode ser escrita quando a renderização da PPU está

desativada ou durante o período de Vertical Blank (VBlank), essas operações foram implementadas respeitando essa restrição, logo no início do código, quando a PPU está desligada, garantindo estabilidade na exibição dos gráficos.



Visualização da Nametable e Pattern Table do Background no NESst

A implementação dos sprites no jogo foi relativamente simples, pois apenas cinco sprites foram utilizados. O elemento principal é o pássaro, que se movimenta na tela e foi estruturado como um metasprite de 16×16 pixels, composto por quatro sprites de 8×8 pixels. O quinto sprite utilizado no jogo é o sprite zero, que desempenha um papel essencial na implementação do split screen.

Inicialmente, foi considerado utilizar sprites para os canos, visto que esses elementos se movem horizontalmente pela tela. No entanto, devido à limitação do NES de 64 sprites simultâneos e ao fato de que apenas 8 sprites podem ser renderizados na mesma linha horizontal, essa abordagem foi descartada. Como alternativa, os canos foram incorporados ao background, e seu movimento foi implementado através do scrolling, o que possibilitou uma renderização mais eficiente.

A gestão dos sprites na OAM foi facilitada pela biblioteca NESlib, que abstrai várias das complexidades inerentes ao baixo nível do hardware. A ferramenta NES Screen Tool foi empregada no design e posicionamento preciso dos metasprites, assegurando a correta atribuição de coordenadas e paletas de cores.

4.2.3 - Scrolling Horizontal da Tela

A mecânica de scrolling horizontal é um dos elementos essenciais na implementação do jogo, proporcionando a ilusão de um mundo infinito. O NES possui suporte a scrolling via hardware, permitindo mover o background sem a necessidade de redesenhá-lo completamente a cada quadro.

Uma nametable ocupa exatamente o espaço de uma tela completa, mas a RAM disponível permite armazenar apenas duas nametables simultaneamente. Para possibilitar o deslocamento horizontal do cenário, o jogo foi configurado com o vertical mirroring, que espelha as nametables A e B nas nametables C e D, garantindo uma continuidade visual no scrolling horizontal. Essa técnica possibilita que, ao mover a tela da direita para a esquerda, a nametable que sai da tela seja reutilizada, criando a ilusão de um ambiente infinito.

A biblioteca NESlib oferece funções que facilitam a implementação do scrolling. A função scroll() permite deslocar a tela em qualquer direção ao longo dos quadros, enquanto split() possibilita dividir a tela em seções com diferentes deslocamentos. No caso do Flappy Bird, a função scroll() foi utilizada inicialmente para mover o background horizontalmente, com o deslocamento sendo controlado pela variável scroll_x, que é incrementada a cada quadro para gerar o movimento contínuo.

No entanto, a implementação do scrolling traz um desafio adicional: ao mover a tela, todos os elementos gráficos se deslocam, incluindo qualquer barra de status ou placar de pontos que deva permanecer fixo. Para resolver esse problema, foi utilizada a técnica do Sprite Zero, um recurso da PPU que permite definir um ponto na tela onde a CPU pode detectar a colisão entre um sprite e o background, indicando o local exato onde a tela deve ser dividida. Com essa técnica, foi possível trocar a função scroll() pela função split() para garantir que a barra de status na parte superior da tela permanecesse estática, enquanto o restante do cenário continuava se movendo.

Essa abordagem permitiu recriar com fidelidade a mecânica de deslocamento do jogo original, garantindo um fluxo contínuo do cenário sem comprometer a exibição das informações essenciais na tela.

4.2.4 - Atualização Dinâmica dos Gráficos e Aleatoriedade

A atualização dinâmica dos gráficos no NES exige o uso de buffers de VRAM, uma vez que a VRAM só pode ser escrita durante o Vertical Blank (VBlank), o curto intervalo de tempo em que a PPU não está desenhando a tela. Para implementar a movimentação dos canos em Flappy Bird, foi necessário manipular a VRAM de forma eficiente, garantindo que os canos fossem gerados fora da área visível e atualizados corretamente sem comprometer a estabilidade do jogo.

Os canos foram implementados como elementos do background, desenhados diretamente na Nametable. Para evitar sobrecarga na escrita de dados na VRAM, foi utilizado um buffer, onde as modificações eram armazenadas durante o quadro e aplicadas no momento do VBlank. Como o NES permite a atualização de, no máximo, 140 bytes por quadro, o processo foi limitado a 128 bytes para

garantir estabilidade. O 8bitworkshop disponibiliza um módulo que facilita essa manipulação, oferecendo funções para armazenar, liberar e aplicar as alterações na VRAM.

A lógica de atualização dos canos seguiu uma abordagem baseada no deslocamento da tela. Quando o scrolling atingia um determinado ponto, um novo conjunto de canos era gerado fora da tela visível. Como a VRAM não pode ser atualizada instantaneamente com muitos dados, o desenho dos canos foi dividido em quatro colunas verticais, sendo cada uma escrita em um quadro diferente. Esse processo evitou falhas na exibição e garantiu que os canos fossem completamente desenhados antes de entrarem na tela.

Além disso, os canos precisavam ser gerados em alturas aleatórias para tornar o jogo dinâmico. Como o NES não possui um gerador de números verdadeiramente aleatórios, foi utilizada a função rand8() da NESlib, que gera um valor pseudo aleatório de 8 bits. Para limitar o intervalo de valores possíveis, a operação bitwise AND (&) foi aplicada, restringindo os números gerados a um conjunto específico de alturas predefinidas. A semente para o gerador de números aleatórios foi inicializada com set_rand(nesclock()), garantindo que os valores fossem diferentes a cada partida. Isso foi feito capturando o momento exato em que o jogador pressionava Start, garantindo que pequenas variações no tempo resultem em sequências distintas de números aleatórios.

4.2.5 - Implementação dos Controles e Movimentos

A implementação dos controles no Flappy Bird foi baseada na leitura dos comandos do joypad do NES, que possui 8 botões: direcionais (up, down, left, right), Select, Start, A e B. A NESlib oferece funções específicas para capturar essas entradas, permitindo verificar tanto quando um botão é pressionado pela primeira vez (trigger), quanto quando ele continua pressionado (hold).

No jogo, a mecânica de controle é simples e segue a estrutura do jogo original. O botão A é utilizado para fazer o pássaro pular e também para selecionar opções no menu, enquanto o botão Start é utilizado para iniciar o jogo. Inicialmente, o jogo foi implementado para um único jogador, mas futuras versões planejam incluir um modo para dois jogadores, aproveitando o suporte da NESlib para capturar entradas do segundo controle.

A leitura dos comandos foi feita utilizando a função pad_trigger(0), que detecta se o botão foi pressionado no frame atual, garantindo que o pulo ocorra apenas uma vez por acionamento do botão. Dessa forma, evita-se que um único pressionamento gere múltiplos saltos indesejados.

A movimentação do pássaro foi baseada em um modelo de física simplificado, que simula os efeitos de gravidade e impulso ao pular. Inicialmente, o sistema utilizava apenas movimentação baseada em pixels inteiros, mas os resultados não eram suficientemente fluidos. Para aprimorar a jogabilidade, foi adotado um sistema de subpixels, no qual um pixel é subdividido em 16 unidades menores. Dessa forma, a movimentação ocorre de maneira mais gradual, e a conversão para deslocamento real na tela só acontece após o acúmulo de 16 subpixels.

A mecânica de movimento segue a seguinte lógica: a cada frame, a gravidade é aplicada, aumentando progressivamente a velocidade de queda do pássaro. Para evitar que ele caia rápido

demais, foi estabelecido um limite de velocidade máxima. Quando o jogador pressiona o botão A, a função player_jump() é chamada, aplicando um impulso negativo na velocidade, fazendo com que o pássaro suba. Esse impulso diminui gradualmente à medida que a gravidade volta a atuar, criando um movimento característico do jogo original.

Essa abordagem garante que a movimentação do pássaro seja suave e responsiva. Além disso, a implementação baseada em subpixels permitiu um controle mais preciso da aceleração e desaceleração do personagem, contribuindo para uma experiência de jogo mais refinada.

4.2.6 - Detecção de Colisões

A detecção de colisão entre o pássaro e os canos é um elemento essencial para a jogabilidade. Como o NES não possui suporte nativo para detecção de colisão, foi necessário implementar um sistema baseado em coordenadas, comparando manualmente as posições dos objetos na tela.

A lógica de colisão funciona da seguinte forma: a cada quadro, as coordenadas do pássaro são comparadas com as coordenadas dos canos visíveis na tela. Se houver sobreposição entre os limites do pássaro e os canos, uma colisão é detectada e o jogo entra no estado de "Game Over", interrompendo a movimentação e ativando a tela de Game Over.

A implementação foi feita utilizando a técnica de bounding boxes, onde cada objeto é tratado como um retângulo e suas bordas são comparadas. Essa abordagem reduz a necessidade de cálculos complexos e garante um desempenho eficiente no NES. Para melhorar a jogabilidade, a bounding box do pássaro foi reduzida em relação ao seu tamanho visual, tornando a margem de erro mais tolerante e evitando que pequenas sobreposições causem colisões injustas.

5 - Resultados e Discussão dos Resultados

O desenvolvimento do *demake* de Flappy Bird para o NES resultou em um jogo funcional e performático, validando a metodologia empregada. O jogo opera a 60 FPS estáveis na FPGA Tang Nano 20K, confirmando que a emulação de hardware via NESTang oferece uma plataforma robusta e fiel ao console original.

O principal resultado técnico deste trabalho foi a superação das limitações de hardware da PPU, o que demonstra a aplicação prática dos conceitos teóricos. O maior desafio foi a restrição de escrita na VRAM, que só pode ocorrer durante o curto período do Vertical Blank (VBlank). Como a geração procedural dos canos exigia a atualização de mais de 128 bytes — o limite prático por quadro —, a solução foi implementar uma lógica que divide a renderização de um novo par de canos em múltiplos quadros. Essa abordagem, gerenciada por meio de buffers de VRAM, embora mais complexa, garantiu a atualização do cenário sem causar artefatos visuais ou queda na taxa de quadros.

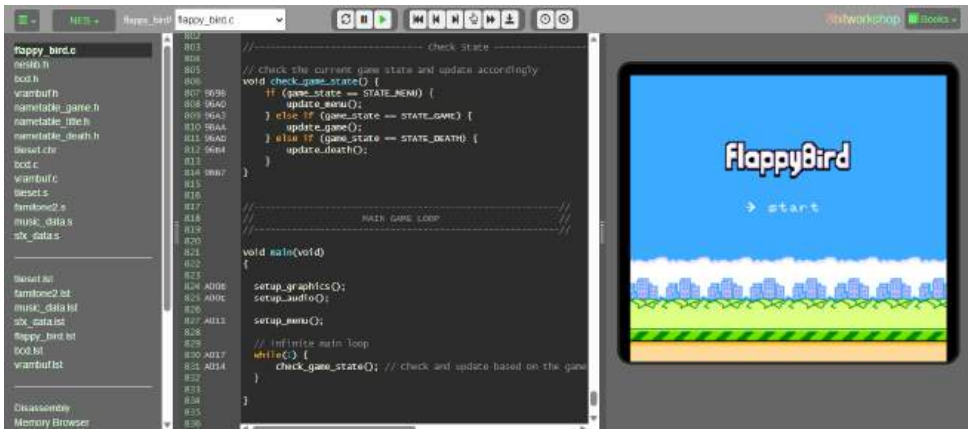
A biblioteca NESlib e o compilador cc65 mostraram-se ferramentas essenciais, provando que é viável desenvolver jogos complexos em linguagem C para o NES, mantendo um equilíbrio entre produtividade e desempenho. A abstração fornecida pela biblioteca para tarefas como a

manipulação da OAM, a configuração de paletas e a implementação do scrolling permitiu focar nas mecânicas do jogo, em vez de na complexa interação direta com os registradores de hardware.

Uma contribuição central do projeto é a documentação detalhada do processo de desenvolvimento, registrada em um repositório público no GitHub. O repositório não contém apenas o código-fonte final, mas também quatro capítulos de documentação que explicam, passo a passo, a implementação de gráficos, sprites e scrolling, conectando cada decisão de programação às restrições e funcionalidades da arquitetura do NES. Este material serve como um recurso didático prático, complementando a teoria e servindo como um guia para futuros desenvolvedores interessados na plataforma.

O código-fonte e a documentação completa do projeto estão disponíveis em:

- https://github.com/vitimbro/flappy_bird_nes
- https://github.com/vitimbro/dragons_leap/wiki



Visualização do Projeto do jogo no 8bitworkhop

ROM do jogo funcionando na FPGA via NESTang

6 - Conclusões e Perspectivas de Trabalhos Futuros

Este trabalho demonstrou com sucesso a viabilidade de utilizar ferramentas modernas para desenvolver um jogo para o Nintendo Entertainment System, unindo o estudo da arquitetura de hardware clássica com a prática da programação embarcada. A recriação do Flappy Bird em um hardware com mais de 30 anos de defasagem não apenas validou a metodologia, mas também serviu como um profundo exercício de otimização e gerenciamento de recursos limitados.

Conclui-se que a emulação via FPGA, especificamente com a placa Tang Nano 20K e o projeto NESTang, oferece uma plataforma de desenvolvimento e execução de altíssima fidelidade. Além disso, o uso da linguagem C, através da NESlib e do cc65, representa um ponto de entrada acessível e produtivo para a programação retrô, permitindo que desenvolvedores foquem na lógica do jogo ao mesmo tempo em que aprendem sobre as restrições de hardware. A principal contribuição do projeto, além do jogo funcional, é a documentação detalhada que serve como um recurso educacional, preenchendo a lacuna entre a teoria da arquitetura de computadores e sua aplicação prática.

Como perspectivas de trabalhos futuros, vislumbram-se as seguintes expansões e estudos:

- **Otimização com Módulos em Assembly (6502):** Um estudo futuro poderia identificar os gargalos de desempenho (como o loop de física, a detecção de colisão ou a atualização da VRAM) e reescrevê-los em Assembly 6502. Isso permitiria uma análise quantitativa do ganho de performance, medindo o uso de ciclos de CPU antes e depois da otimização.

- **Análise Comparativa:** Adaptar o projeto para outras plataformas de 8 ou 16 bits, como o Master System ou o Game Boy, para realizar um estudo comparativo entre as diferentes arquiteturas de hardware e seus respectivos desafios de desenvolvimento.

- **Criação de Material Didático Interativo:** Transformar a documentação do repositório criado em um tutorial completo ou uma série de videoaulas. O objetivo seria guiar outros estudantes de graduação no processo de criar um jogo simples para NES, usando seu projeto como base.

- **Análise de Desempenho e Profiling Aprofundado:** Utilizar os recursos avançados de emuladores como o Mesen para realizar um profiling detalhado do jogo. Medir exatamente quantos ciclos de CPU cada rotina do jogo consome, quanto tempo livre resta durante o VBlank e identificar os principais gargalos. O resultado seria um artigo técnico sobre a viabilidade e os custos de performance de cada mecânica implementada em C na arquitetura 6502.

Referências

HUGG, S. *Making Games for the NES*. 2022. Disponível em: https://github.com/sehugg/8bitworkshop.

COPETTI, R. *Nintendo Entertainment System (NES) Architecture - A Practical Analysis*. 2019. Última modificação em 2 de fevereiro de 2025. Disponível em: https://www.copetti.org/writings/consoles/nes/.

SIEBER, J. *Implementing the Nintendo Entertainment System on FPGA*. 2013. Disponível em: https://www.j-sieber.de/nes_on_fpga/nes_on_fpga.html.

ASSUMPÇÃO JUNIOR, J. M.; KLÜSER, C.; HERRERA, V. A. S.; CARMO, J. P. P.; GAZZIRO, M. A. *Computadores e Videogames - Uma Abordagem Prática das Arquiteturas Clássicas*. Editora da UFABC, Santo André, Brasil, 2023.

Making of Aspect Star N: Coding for the NES. 2019. Disponível em: https://aspect-star-n.blogspot.com/2019/03/making-of-aspect-star-n-coding-for-nes.html.

SIPEED. *Tang Nano 20K*. Disponível em: https://wiki.sipeed.com/hardware/en/tang/tang-nano-20k/nano-20k.html.

SEHUGG. *8bitworkshop*. Disponível em: https://github.com/sehugg/8bitworkshop.

CLBR. *neslib*. Disponível em: https://github.com/clbr/neslib.

NAND2MARIO. *nestang*. Disponível em: https://github.com/nand2mario/nestang.

CC65. *cc65*. Disponível em: https://www.cc65.org/oldindex.php.

Mesen Emulator. Disponível em: https://emulation.gametechwiki.com/index.php/Mesen.

BROKESTUDIO. *NES Screen Tool*. Disponível em: https://github.com/BrokeStudio/NES-Screen-Tool.

BLEUBLEU. *FamiStudio*. Disponível em: https://github.com/BleuBleu/FamiStudio.

Nesdev Wiki. Disponível em: https://www.nesdev.org/wiki/Nesdev_Wiki.

EMBARCAÇÕES. *nestang*. Disponível em: https://embarcacoes.ic.unicamp.br/posts/nestang/.

GOWIN SEMICONDUCTOR CORP. *GOWIN EDA*. Disponível em: https://www.gowinsemi.com/en/support/home/.