Especificação do Projeto PBL – Jogo Genius

Post-processing

2025

Objetivos

Desenvolver um conjunto de soluções integradas de hardware e software para o jogo

Genius, utilizando a metodologia PBL (Problem-Based Learning) como instrumento de aprendizagem ativa. O projeto tem como finalidade promover o domínio progressivo de conceitos fundamentais, como a implementação concorrente em RTL (SystemVerilog) e linguagem de máquina (Assembly RISC-V).

Objetivos Específicos por Etapa (PBLs)

PBL01 – Implementação de FSM em RTL

- Desenvolver, utilizando linguagens de descrição de hardware (HDL), uma solução em RTL (Register Transfer Level) para o jogo Genius.
- Modelar uma Máquina de Estados Finitos (FSM) hardcoded, capaz de gerenciar a lógica do jogo.

PBL02 – Implementação em Assembly RISC-V

- Traduzir o funcionamento do jogo Genius para linguagem de máquina (Assembly), utilizando o conjunto de instruções da arquitetura RISC-V.
- Executar o código em um simulador RISC-V, reproduzindo os comportamentos definidos na FSM desenvolvida no PBL01.

PBL03 – Desenvolvimento e Verificação de Processador RISC-V

- Projetar um processador RISC-V simplificado, capaz de executar o código Assembly desenvolvido no PBL02.
- Realizar a verificação funcional do processador.
- A atividade será realizada em duas equipes: cada grupo será responsável por projetar seu próprio processador e, posteriormente, realizar a verificação cruzada, testando e

analisando o processador desenvolvido pelo grupo oposto.

PBL04 – Síntese, STA e Prototipagem em FPGA

- Realizar a síntese lógica do projeto RTL desenvolvido no PBL01, gerando o netlist necessário para implementação física.
- Aplicar técnicas de Análise Estática de Tempo (STA) para garantir que o circuito atenda aos requisitos.
- Montar um protótipo o design em uma FPGA, validando sua execução em hardware real.

Observações Importantes:

- Os módulos PBL01, PBL02 e PBL04 serão desenvolvidos de forma individual, permitindo o aprofundamento técnico em cada área. O módulo PBL03 será realizado de forma colaborativa, promovendo o trabalho em equipe e o entendimento aprofundado de arquitetura de processadores, além de introduzir práticas reais de verificação cruzada e revisão entre pares.

Requisitos

Requisitos funcionais

RF01 - O jogo Genius: conforme as regras do jogo, o usuário deve seguir uma sequência, replicando a sequência correta. A cada rodada, a sequência aumenta em um novo elemento.

RF02 – Níveis de dificuldade: o jogo deverá apresentar como requisito funcional três níveis

de dificuldade, cada um com características como apresentadas a seguir:

- **Fácil:** sequência de 8 luzes (0-7 bits)
- **Médio:** sequência de 16 luzes (0-15 bits)
- **Difícil:** sequência de 32 luzes (0-31 bits)

RF03 – Níveis de velocidade: requisito funcional requerido relacionado ao seu nível de velocidade, podendo ter dois modos distintos, sendo eles:

- **Lento (2s entre luzes)**
- **Rápido (1s entre luzes)**

RF04 – Modos de jogo: esse requisito irá definir qual o modo de jogo deverá ser selecionado pelo usuário, podendo ser:

- **Siga:** O aparelho irá gerar a sequência de cores de maneira aleatória e crescente, até o limite estabelecido pelo nível de dificuldade do jogo.
- **Mando eu**: O jogo começa gerando uma sequência inicial de uma cor. Em seguida, o primeiro jogador adiciona uma nova cor à sequência, desafiando o próximo jogador a reproduzir a sequência completa. A cada rodada, os jogadores continuam acrescentando uma cor, tornando o desafio progressivamente mais difícil. Nesse modo de jogo, não há níveis de dificuldade nem variação de velocidade, com suporte para sequências de até 32 luzes ou rodadas.

RF05 - Placar de Pontuação: Deve haver um sistema de pontuação através de um display

para acompanhar o desempenho dos jogadores.

RF06 - Geração Aleatória de Sequências: Não poderá haver o armazenamento prévio de sequências em memória, sendo as mesmas aleatórias (PRNG).

Requisitos Não Funcionais

RNF01 – Limitações de frequência: O sistema deve operar em uma frequência de 200 MHz.

RNF02 – Plataforma de desenvolvimento: O protótipo deve ser implementado em uma placa FPGA compatível com o Vivado.

RNF03 - Tecnologias e linguagens exigidas: As soluções devem ser implementadas com uso de SystemVerilog, Assembly RISC-V e ferramentas de simulação.

Restrições físicas

O projeto do jogo Genius será desenvolvido em um sistema embarcado baseado na FPGA RFSoC 4x2, selecionada por sua capacidade de desempenho, flexibilidade e integração com tecnologias como SystemVerilog e Assembly RISC-V. Essa plataforma permite a implementação eficiente de soluções hardware/software, essenciais para atender aos requisitos funcionais e não funcionais do jogo. A partir dessa plataforma serão utilizados os seguintes

recursos de hardware:

RFI01: Recursos de Hardware

- **Entradas:**
  - 4 botões GPIO para seleção das cores durante o jogo.
  - botões de controle (ON/OFF, Difficulty, Speed, Game Mode, Start) para configuração e operação do sistema.
  - Chaves deslizantes GPIO para ajuste de modo de jogo, velocidade e dificuldade.
- **Saídas:**
  - 4 LEDs para representação visual das cores da sequência.
  - 1 LED adicional para indicar o início da partida (PARTIDA).
  - Display LCD para exibição da pontuação em tempo real.

RFI02: Plataforma de Desenvolvimento

A síntese de hardware e a prototipagem serão realizadas no ambiente Xilinx Vivado, garantindo compatibilidade com as ferramentas de simulação, síntese e análise estática de tempo (STA). A escolha dessa plataforma assegura a integração fluida entre os módulos desenvolvidos em SystemVerilog (para a FSM e gerador de números aleatórios) e o software em Assembly RISC-V (para a lógica de controle do processador).

RFI03: Frequência de Operação

O sistema opera com um clock de 200 MHz, atendendo ao requisito de desempenho

definido para garantir respostas rápidas às interações do usuário e sincronização precisa entre os componentes.

RFI04: Integração Física

Os recursos de entrada/saída da FPGA (botões, LEDs, chaves e display) são fundamentais para a interação do jogador, permitindo a seleção de modos, ajustes de dificuldade e visualização imediata do progresso. A combinação desses elementos com a lógica programável da RFSoC 4x2 oferece um ambiente robusto para validação das funcionalidades do jogo, alinhando-se tanto às demandas técnicas quanto à experiência do usuário final.

Microarquitetura

Este documento descreve os módulos que compõem a microarquitetura do sistema de jogo baseado em FSM, especificando suas responsabilidades, interfaces e formas de integração.

O sistema é composto por cinco principais blocos modulares: Controller, GNR, RegisterBank, ScoreManager, e DisplayManager

Descrição dos Componentes

Controller — Módulo de Controle Principal

Função:

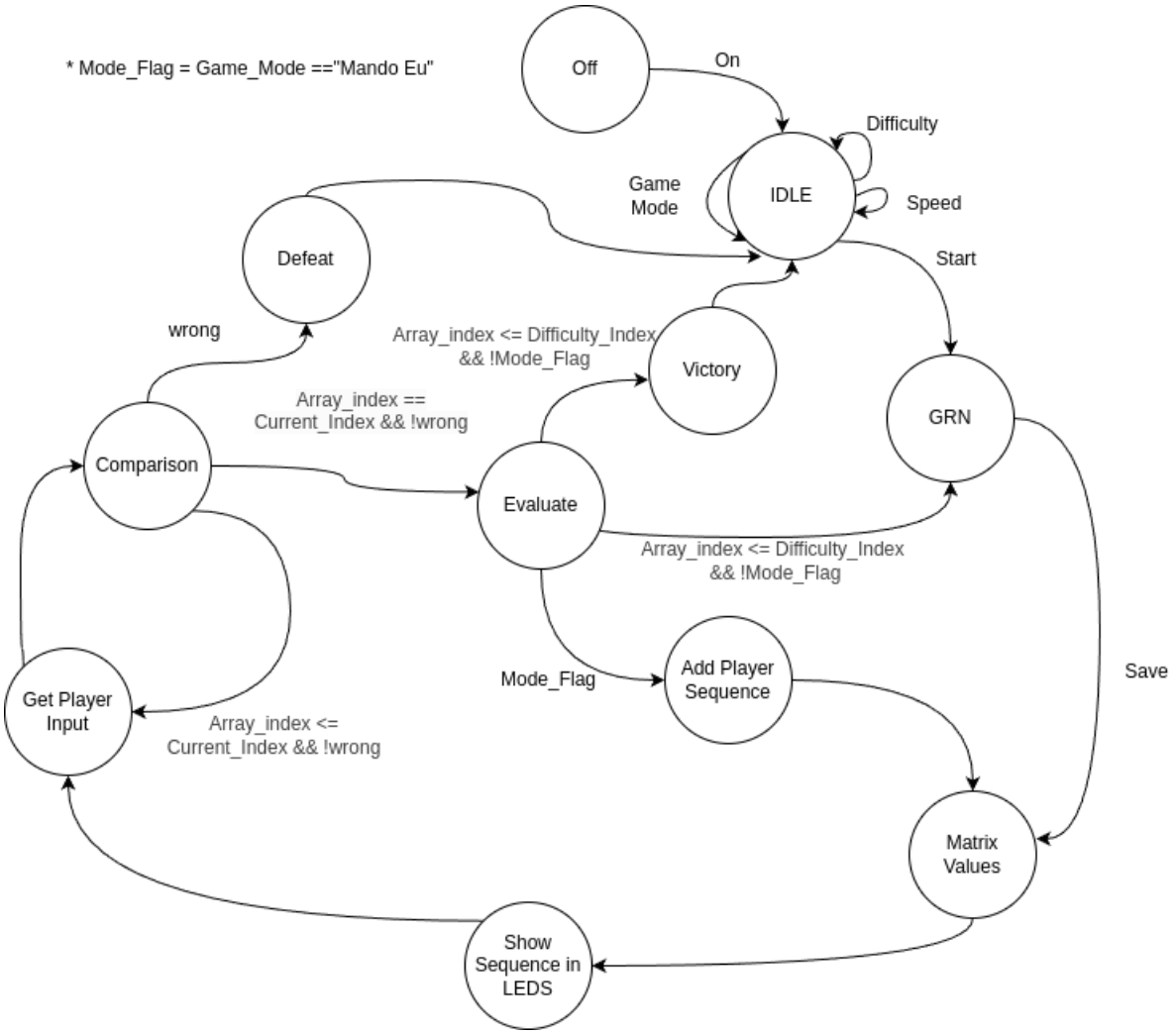
Implementa a máquina de estados finita (FSM) principal do sistema. Controla a execução do jogo, coordena os demais módulos e define os fluxos de dados e controle.

Responsabilidades:

- Gerenciar transições de estados com base em eventos externos e internos.
- Configurar e iniciar partidas.
- Controlar geração de números aleatórios, armazenamento de sequência e exibição.
- Verificar entradas do jogador e determinar resultado (acerto, erro, vitória).

Interface:

Sinal	Direção	Descrição
clk	input	Clock do sistema
botao_start	input	Início de partida
botao_velocidade	input	Alterna entre velocidades
botao_dificuldade	input	Alterna entre níveis de dificuldade
botao_modo_jogo	input	Alterna entre modo padrão e “Mando Eu”
entrada_jogador	input	Botão pressionado pelo jogador
numero_aleatorio	input	Número gerado pelo módulo GNR
score_atual	input	Valor atual do placar (do ScoreManager)
sinais_de_controle	output	Comandos para GNR, RegisterBank e Display



Estados da Máquina de Estado

Off:

Estado inicial da máquina de estados, no qual o componente encontra-se desligado. Ao pressionar o botão *On/Off*, a máquina transita para o estado **Idle**.

Idle:

Estado de configuração, onde o usuário pode ajustar os parâmetros de velocidade, dificuldade

e modo de jogo. Cada parâmetro possui um valor padrão inicial, podendo ser alterado via botões dedicados:

- **Velocidade:** Alterna entre *Lento* (2 segundos por cor) e *Rápido* (1 segundo por cor).
- **Modo de Jogo:** Alterna entre *Padrão* e *Mando Eu*. No modo *Mando Eu*, o jogador fornece a sequência manualmente.
- **Dificuldade:** Define o tamanho máximo da sequência no modo *Padrão*, podendo ser de 8, 16 ou 32 bits.

Ao pressionar o botão *Start*, os dados de configuração são armazenados nos registradores, e uma nova partida é iniciada. A máquina de estados então transita para o estado **GNR**.

GNR (Gerador de Número Aleatório):
Gera um número aleatório codificado entre 0 e 4, representando uma cor. Após a geração, a máquina avança para o estado **Matrix Values**.

Matrix Values:
O número gerado (representando a cor) é adicionado à última posição de um vetor, que representa a sequência de cores que será exibida e reproduzida pelo jogador. Após o armazenamento, a máquina avança para o estado **Show Sequence Values**.

Show Sequence Values:
Exibe a sequência de cores armazenada no vetor, acendendo os LEDs correspondentes. O tempo de exibição de cada cor é determinado pela velocidade configurada no estado **Idle**. Após a exibição completa da sequência, a máquina de estados avança para **Get Player Input**.

Get Player Input:
Aguarda que o jogador pressione os botões correspondentes às cores da sequência exibida,

na ordem correta. Após cada entrada do jogador, a máquina transita para o estado **Comparison**.

Comparison:

Compara a entrada do jogador com a cor esperada na sequência:

- Se a entrada estiver incorreta, a máquina transita para o estado **Defeat**.
- Se estiver correta:
    - E ainda houver elementos a serem verificados, o índice da sequência é atualizado e a máquina retorna para **Get Player Input**.
    - Se for o último elemento da sequência, a máquina transita para **Evaluate**.

Caso o jogador acerte, o placar é incrementado e atualizado no display de 7 segmentos.

Defeat:

Indica a derrota do jogador, acendendo todos os LEDs por 1 segundo e, em seguida, desligando-os. A máquina então retorna ao estado **Idle**, permitindo o reinício da partida.

Evaluate:

Estado acessado após o jogador acertar toda a sequência:

- Se estiver no modo *Mando Eu*, o próximo botão pressionado representa uma nova cor a ser adicionada à sequência, e a máquina retorna para **Matrix Values**.
- Caso contrário, verifica-se se a sequência atingiu o tamanho máximo definido pela dificuldade:
    - Se sim, transita para **Victory**.
    - Se não, gera uma nova cor aleatória (estado **GNR**) e a adiciona à sequência, retornando a **Matrix Values**.

Victory:

Indica que o jogador completou corretamente a sequência no nível máximo de dificuldade.

Os LEDs piscam duas vezes (1 segundo acesos, 1 segundo apagados, e novamente 1 segundo acesos) como forma de celebração. Após isso, a máquina retorna ao estado **Idle**.

Pressionar o botão On/Off a qualquer momento durante o funcionamento do sistema leva diretamente ao estado de Off.

GNR — Gerador de Número Aleatório

Função:

Gera números aleatórios codificados entre 0 e 4, representando diferentes cores do jogo.

Responsabilidades:

- Produzir número aleatório quando requisitado.
- Implementar algoritmo LFSR ou equivalente.

Interfaces:

Sinal	Direção	Descrição
clk	input	Clock do sistema
reset	input	Reset síncrono
enable	input	Habilitação do gerador
random_out	output	Saída do número aleatório [2:0]

2.3. RegisterBank — Banco de Registradores

Função:

Armazena configurações de jogo, sequência atual de cores e índice de progresso do jogador.

Responsabilidades:

- Armazenar e fornecer os parâmetros: velocidade, modo de jogo e dificuldade.
- Gerenciar vetor de sequência de cores geradas.
- Registrar e fornecer entrada atual do jogador.
- Armazenar estado intermediário da FSM, se necessário.

Interfaces:

Sinal	Direção	Descrição
clk	input	Clock do sistema
reset	input	Reset síncrono
config_write	input	Habilitação de escrita das configurações
sequencia_write	input	Escreve nova cor na sequência
sequencia_read	input	Leitura da sequência para exibição
indice_sequencia	input	Índice da cor atual a comparar
config_data	input	Dados de configuração
cor_input	input	Cor inserida ou gerada
sequencia_out	output	Saída da sequência armazenada
configs_out	output	Saída dos parâmetros

2.4. ScoreManager — Gerenciador de Placar

Função:

Controla a pontuação do jogador e atualiza o display de 7 segmentos.

Responsabilidades:

- Incrementar o placar em acertos.
- Reiniciar o placar no início de uma nova partida.
- Atualizar o display com valor atual.

Interfaces:

Sinal	Direção	Descrição
clk	input	Clock do sistema
reset	input	Reset síncrono
incrementar_score	input	Habilita incremento do score
zerar_score	input	Zera o placar
score_atual	output	Valor atual do placar [7:0]
display_out	output	Controle do display de 7 segmentos