PBL – Genius

Especificação do Projeto

QuIIN / CIMATEC

Post-Processing

Salvador / BA, 2025

FICHA TÉCNICA

INFORMAÇÕES GERAIS			
DATA	06/05/2025	VERSÃO	1.0
COLABORADOR(ES)			
	David Machado Couto Bezerra		
REVISOR		DATA	VERSÃO
<revisor>		<data-analise>	<versao-analise>

DESENVOLVIMENTO INDIVIDUAL			
PESQUISADOR	David Machado Couto Bezerra	DATA	VERSÃO
PBL-1		<data-envio>	<versao-1>
PBL-2	<repositorio-branch>	<data-envio>	<versao-2>
PBL-3	<repositorio-branch>	<data-envio>	<versao-3>
PBL-4	<repositorio-branch>	<data-envio>	<versao-4>

SUMÁRIO

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

A. Desenvolver, utilizando linguagens de descrição de hardware (HDL), uma

solução em RTL (Register Transfer Level) para o jogo Genius.

B. Modelar uma Máquina de Estados Finitos (FSM) hardcoded, capaz de gerenciar a lógica do jogo.

PBL02 – Implementação em Assembly RISC-V

A. Traduzir o funcionamento do jogo Genius para linguagem de máquina (Assembly), utilizando o conjunto de instruções da arquitetura RISC-V.

B. 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

A. Projetar um processador RISC-V simplificado, capaz de executar o código Assembly desenvolvido no PBL02.

B. Realizar a verificação funcional do processador.

C. 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

A. Realizar a síntese lógica do projeto RTL desenvolvido no PBL01, gerando o netlist necessário para implementação física.

B. Aplicar técnicas de Análise Estática de Tempo (STA) para garantir que o circuito atenda aos requisitos.

C. Montar um protótipo o design em uma FPGA, validando sua execução em

hardware real.

Observações Importantes

A. 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:

a. **Fácil:** sequência de 8 luzes (0-7 bits)
b. **Médio:** sequência de 16 luzes (0-15 bits)
c. **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:

a. **Lento (2s entre luzes)**
b. **Rápido (1s entre luzes)**

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

a. **Siga:** O aparelho irá gerar a sequência de cores de maneira aleatória e crescente, até o limite estabelecido pelo nível de dificuldade do jogo.
b. **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

I. **Entradas:**
    a. 4 botões GPIO para seleção das cores durante o jogo.
    b. botões de controle (ON/OFF, Difficulty, Speed, Game Mode, Start) para configuração e operação do sistema.
    c. Chaves deslizantes GPIO para ajuste de modo de jogo, velocidade e

dificuldade.

II. **Saídas:**

a. 4 LEDs para representação visual das cores da sequência.

b. 1 LED adicional para indicar o início da partida (PARTIDA).

c. 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: Genius, PRNG(LFSR), ButtonEdgeDectetor, Counter e LedDriver.

Descrição dos Componentes

Genius— 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:

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

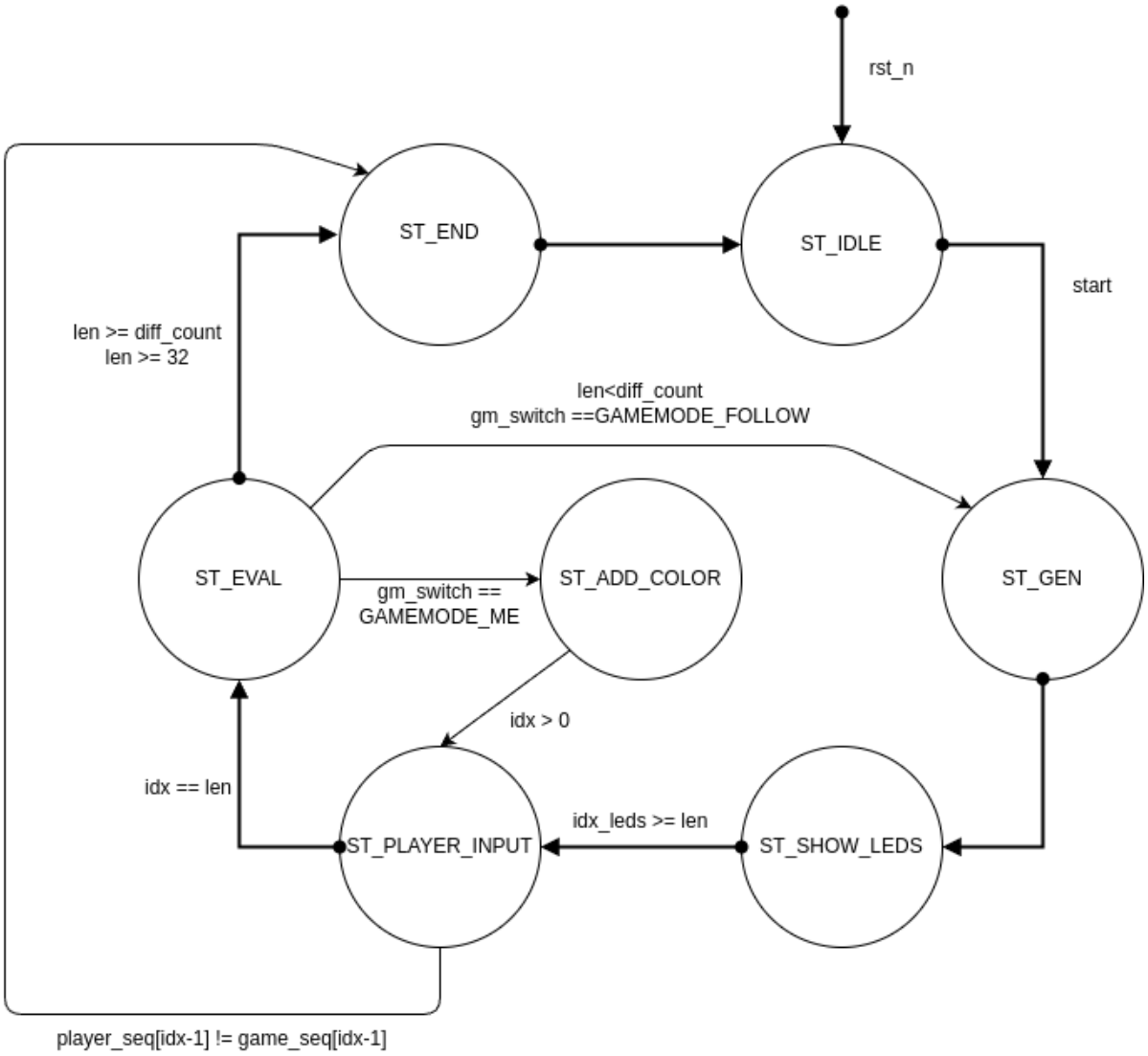
Interface:

SINAL	DIREÇÃO	DESCRIÇÃO
clk	input	Clock do sistema
rst_n	input	Reset

start	input	Início de partida
speed_switch	input	Alterna entre velocidades
diff_switch	input	Alterna entre níveis de dificuldade
gm_switch	input	Alterna entre modo padrão e “Mando Eu”
btn_green	input	Botão verde pressionado pelo jogador
btn_red	input	Botão vermelho pressionado pelo jogador
btn_blue	input	Botão azul pressionado pelo jogador
btn_yellow	input	Botão amarelo pressionado pelo jogador
win	output	Flag de vitória
lost	output	Flag de derrota

Estados da Máquina de Estado

A máquina de estados finita apresentada foi desenvolvida para controlar por meio dos estados: ST_IDLE, ST_GEN, ST_ADD_COLOR, ST_SHOW_LEDS, ST_PLAYER_INPUT, ST_EVAL e ST_END, conforme condições de entrada e variáveis de controle. Trata-se de uma máquina do tipo Mealy, pois suas transições e saídas são influenciadas tanto pelo estado atual quanto pelas entradas recebidas em tempo real.



Estados da Máquina de Estado

ST_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:

a. **Velocidade:** Alterna entre *Lento* (2 segundos por cor) e *Rápido* (1 segundo por cor).
b. **Modo de Jogo:** Alterna entre *Padrão* e *Mando Eu*. No modo *Mando Eu*, o jogador fornece a sequência manualmente.
c. **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 **ST_GEN**.

ST_GEN (Gerador de Número Pseudo-Aleatório):
Gera um número aleatório de 4 bits e pegando seus bits 0 e 1 para representar uma cor. Após a geração, a máquina avança para o estado ST_SHOW_LEDS.

ST_SHOW_LEDS:
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 ST_IDLE. Após a exibição completa da sequência, a máquina de estados avança para ST_PLAYER_INPUT.

ST_PLAYER_INPUT:
Aguarda que o jogador pressione os botões correspondentes às cores da sequência exibida, na ordem correta. Após finalizar com todos os inputs do jogador e sem erros,

a máquina transita para o estado ST_EVAL, caso tenha erros, a máquina pula diretamente para o estado ST_END com a flag de derrota.

ST_EVAL:

Estado acessado após o jogador acertar toda a sequência atual:

a. Se estiver no modo *Mando Eu*, a máquina vai para ST_ADD_COLOR.
b. Se estiver no modo siga, a máquina vai para ST_GEN.
c. Caso a sequência atingiu o tamanho máximo definido pela dificuldade. Se sim, transita para **ST_END,** com a flag de win, win_player1 ou win_player2.

ST_END:

Estado final que representa o fim de uma partida e assim os jogadores podem iniciar uma nova partida.

LFSR— Gerador de Número Pseudo-Aleatório

Função:

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

Responsabilidades:

a. Fornecer número aleatório quando requisitado.

Interfaces:

SINAL	DIREÇÃO	DESCRIÇÃO
clk	input	Clock do sistema
rst_n	input	Reset síncrono
load	input	Bit para passar a seed para o LFSR
data_in	input	Seed inicial que vai ser carregado

rnd	output	Saída do número aleatório [1:0]

LedDriver— Controlador de LEDs

Função:

Controla a ativação dos LEDs de acordo com a cor informada e um sinal de habilitação.

Responsabilidades:

a. Ativar o LED correspondente à cor recebida quando o sinal enable está ativo.
b. Garantir que apenas um LED seja aceso por vez, conforme o valor do sinal color.
c. Desligar todos os LEDs caso o sinal enable esteja desativado.

Interfaces:

SINAL	DIREÇÃO	DESCRIÇÃO
Enable	input	Sinal que habilita o acionamento do LEDs
color	input	Cor a ser exibida
led_red	output	Sinal de controle do LED vermelho
led_blue	output	Sinal de controle do LED azul
led_green	output	Sinal de controle do LED verde
led_yellow	output	Sinal de controle do LED amarelo

counter— Contador Temporizador

Função:

Gera um pulso periódico de controle (flag) com base em um número fixo de ciclos de clock, funcionando como um temporizador de retardo ou sincronização.

Responsabilidades:

a. Contar ciclos de clock até atingir um valor limite definido.

b. Gerar um pulso alto em flag por um único ciclo de clock quando o contador atinge o valor alvo.
c. Reniciar o contador automaticamente após gerar o pulso.
d. Suportar reset síncrono ativo em nível baixo.

Interfaces:

SINAL	DIREÇÃO	DESCRIÇÃO
clk	input	Clock do sistema
rst_n	input	Reset assíncrono em nível baixo
flag	input	Pulso de controle gerado após tempo definido

counter— Contador Temporizador

Função:
Detecta a borda de subida (rising edge) dos sinais dos botões, gerando pulsos únicos que indicam o momento exato em que cada botão foi pressionado.

Responsabilidades:

- Registrar o estado anterior de cada botão (verde, vermelho, azul e amarelo).
- Identificar uma transição de 0 para 1 (borda de subida) em cada botão.
- Gerar um sinal de pulso (btn_*_edge) correspondente apenas no ciclo em que ocorre a transição.

SINAL	DIREÇÃO	DESCRIÇÃO
clk	input	Clock do sistema
rst_n	input	Reset assíncrono ativo em nível baixo
btn_green	input	Entrada do botão verde
btn_red	input	Entrada do botão vermelho
btn_blue	input	Entrada do botão azul
btn_yellow	input	Entrada do botão amarelo
btn_green_edge	output	Pulso gerado na borda de subida do botão verde
btn_red_edge	output	Pulso gerado na borda de subida do botão vermelho
btn_blue_edge	output	Pulso gerado na borda de subida do botão azul
btn_yellow_edge	output	Pulso gerado na borda de subida do botão amarelo