RELATÓRIOS - EXERCÍCIOS

SD112 - Introdução a Verilog

DOCENTE Felipe Gustavo de Freitas **Rocha**

DISCENTE André Francisco Ribeiro **Bezerra**

DATA DE ENTREGA **07 de novembro de 2025 (prazo máximo)**

A-014 24 de outubro

Formulário para envio das Atividades

SUMÁRIO

REFERÊNCIAS

[1] Digital Systems ; Authors, Ronald Tocci, Neal Widmer, Greg Moss ; Edition, 12 ; Publisher, Pearson Education, 2016 ; ISBN, 0134220145, 9780134220147;

[2] Frank Vahid. 2010. Digital Design with RTL Design, Verilog and VHDL (2nd. ed.). Wiley Publishing.;

☐ A-014: Codificação de Síntese vs Simulação

Conceitual

1 Qual é a principal diferença entre a simulação e a síntese em Verilog?

2 Cite e explique dois dos principais objetivos da síntese.

3 Dê dois exemplos de estruturas ou construções em Verilog que não são suportadas pelas ferramentas de síntese.

4 O que ocorre quando há atribuições incompletas em um bloco combinacional?

5 O que é uma simulação em Verilog e qual é o seu principal objetivo? (page 330)

6 Quais recursos de Verilog podem ser usados em um testbench e não são sintetizáveis?

7 Qual é a diferença entre estímulos "em linha" e estímulos gerados por "arrays"?

8 Quais as vantagens de usar loops para gerar estímulos durante a simulação?

9 Explique o uso das palavras-chave "fork" e "join" em uma testbench.

Prático

1 Implemente um código em Verilog, que seja sintetizável, para realizar o cálculo de paridade. O resultado deve ser 1 se o número de bits 1 for ímpar, e 0 caso contrário.
Resolva este exercício utilizando uma função.

2 Implemente uma task que mapeie uma entrada de 3 bits em um barramento ativável de 8 bits, onde apenas uma entrada deve ser habilitada. Por exemplo, a entrada *d_in [3:0] = 3 ' b000* indica que apenas o valor *d_out [0]* deve ser igual a 1; já para *d_in [3:0] = 3 ' b111* apenas *d_out [7]* deve ser 1.

3 Implemente uma task que gere todos os padrões de teste para um multiplexador de 8 entradas, utilize delays para simulação. $2^8 = 256$

A-014: Codificação de Síntese vs Simulação (Conceitual)

1- A principal diferença reside no objetivo e no resultado de cada processo.

→ A SIMULAÇÃO tem como objetivo verificar o comportamento funcional do código. Ao simular o design, atrelado a um testbench, com foco na análise da lógica aplicada ao longo do tempo e ao gerar as formas de onda (ou dados) que permitem a depuração.

→ A SÍNTESE tem o objetivo definir um circuito físico. A ferramenta de síntese traduz o código em Verilog (RTL) em um netlist, descrição de componentes físicos conectados (portas lógicas e flip-flops) para ser implementado em FPGA ou ASIC.

2- Os principais objetivos da Síntese são: tradução para NETLIST, otimização sob restrições, timing, area e consumo.

I → Timing (velocidade): Garantir que o circuito opere na frequência de clock desejada.

II → Area: Minimizar a quantidade de recursos lógicos (silício) utilizados.

3- Duas construções em Verilog que não são suportadas na síntese: blocos INITIAL e DELAYS (#d).

verificação ← I — II

I → Apenas em simulação pela singularidade operacional.

II → Comportamento inerente aos componentes físicos, glitches, comportamento inesperado

4- Quando há atribuições incompletas em blocos combinacionais (exemplo: if sem else), a ferramenta de síntese infere um latch no circuito. (retem valor do estado)

continuação - página 330

5 - A simulação em Verilog é o processo de usar um software (simulador) para executar o código e modelar o comportamento funcional de um circuito digital por um período de tempo.
→ Objetivo principal é a validação do design (verificação).
→ aplicação virtual pré etapa de síntese e fabricação do hardware

6 - Recursos não sintetizáveis auxiliam na construção das testbenches: controle do fluxo na simulação.
Tarefas de sistema ($display, $monitor, $finish);
Tipos de dados (real e temporizadores - delays).

7 - A diferença fundamental está na organização e estabilidade dos vetores de teste.
→ Estímulo em linha (inline) - São valores codificados diretamente na lógica do testbench (bloco initial). A sequência de sinais e atrasos é descrita manualmente (a=1; #10; b=0). Essa abordagem é simples, mas rígida e difícil de manter para análises mais complexas.
→ Estímulos por "arrays" armazenam os vetores de teste em uma memória (reg array). Um laço

A-014: Codificação de Síntese vs Simulação (prático)
> código nos arquivos de suporte enviados no formulário

REPOSITÓRIO DO GITHUB

github.com/ci-digital-inatel/SD112-INTRO-VERILOG

A014 - Codificação de síntese vs simulação (for ou repeat) itera sobre o array, ao aplicar a cada vetor salvo ao design. Essa abordagem é escalável e flexível, ao permitir que N testes sejam facilmente adicionados ou lidos de um arquivo externo ($readmemh).

8 - As principais vantagens de usar loops functions (for, while, repeat...) para gerar estímulos:

→ AUTOMAÇÃO E CONCISÃO: Loops eliminam a necessidade de codificar manualmente N sentenças de avaliação (teste). Estratégia que fornece vetores ao testbench legibilidade e redução no tamanho (linhas de código).

→ ESCALABILIDADE: Permite testar um espaço de estados muito maior de forma configurável e conciso.

→ FLEXIBILIDADE E REUTILIZAÇÃO: Facilita a definição de testes parametrizáveis e são a base para abordagens orientadas a dados, onde os estímulos são lidos por arrays ou arquivos externos ($readmemh) ao separar os dados da lógica de teste.

9 - As palavras-chave "fork" e "join" são utilizadas para definição de blocos de execução paralela dentro de um processo sequencial (INITIAL) no testbench.

FORK - Marca o início de um bloco onde todas as instruções internas começam a ser executadas simultaneamente (concorrência). begin...end dentro do fork... Join é uma thread paralela.

JOIN - Atua com uma barreira de sincronização no fluxo principal da simulação ao pausar e só continuar após a conclusão de todos os processos paralelos iniciados pelo fork.

Comandos essenciais para modelar eventos concorrentes: gerar clock, aplicar reset e enviar dados ao mesmo tempo.