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-001	A-002	A-003
A-004	A-005	A-006
A-007	A-008	A-009
A-010	A-011	A-012
A-013	A-014	

Formulário para envio das Atividades

SUMÁRIO

REFERÊNCIAS 1
ANOTAÇÕES 2
A-001: Álgebra Booleana 3
A-002: Mintermos, Maxtermos e Mapas de Karnaugh 11
A-003: O inversor 16
A-004: Half Adder 20
A-005: Full Bit Adder 22
A-006: Declarações Processuais e Contínuas 25
A-007: Circuito simples de debounce 27
A-008: Reset Síncrono e Assíncrono 29
A-009: Estilos de Código 31
A-010: Descrição RTL 33
A-011: Descrição Comportamental 35
A-012: Descrição Estrutural 37
A-013: Primitivas 39
A-014: Codificação de Síntese vs Simulação 41

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-007: Circuito simples de debounce pts

Conceitual

1 Qual é a principal diferença entre atribuições blocking e non-blocking?

2 Onde você deve usar atribuições non-blocking?

3 Onde você deve usar atribuições blocking?

Prático

1 Compile o código fonte e a testbench fornecida e execute a simulação do circuito de debounce.

2 Analise o circuito apresentado e explique o funcionamento do código debounce fornecido.

3 Quais tipos de declaração são utilizados no código (contínua ou processual)? Aponte os estilos utilizados em cada parte do código. Qual estilo de implementação (RTL, comportamental ou estrutural) visto no código?

4 Após simular o código, descreva o comportamento observado na simulação.

5 Como o valor de clean_out varia ao longo do tempo quando o botão é pressionado e solto?

6 O filtro de debounce apresenta algum atraso? Qual o impacto desse atraso no sistema?

7 Qual o valor do atraso observado e como controlar esse atraso? Altere o código de forma a simular um atraso de 64 pulsos de clock.

A-007: Circuito simples de debounce (Conceitual)

1- A principal diferença está no momento da atualização da variável.

BLOCKING (=) - Atribuição é executada imediatamente e em sequência. A execução do código é "bloqueada" até que a atribuição seja concluída. É usada para modelar lógica combinacional.

NON-BLOCKING (<=): A avaliação da expressão é efeita imediatamente, mas a atribuição é agendada para ocorrer no final do passo de tempo atual. Todas as atribuições non-blocking em um bloco always ocorrem "em paralelo". É usada para modelar lógica sequencial (registradores), ao evitar race conditions.

2- Atribuições non-blocking (<=) devem ser usadas para modelar lógica sequencial, ou seja, hardware que possui memória e é controlado por um clock.

A regra principal é sempre use atribuições non-blocking dentro de blocos always que são sensíveis a uma borda de clock (exemplo: always @(posedge clk)).

No geral, descreve o comportamento de registradores, →

A-007: Circuito simples de debounce (prático)

> código nos arquivos de suporte enviados no formulário

REPOSITÓRIO DO GITHUB

(imagens e gravações)

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

A-DQ7 - circuito simples de debounce (continuação)

flip-flops e máquinas de estado. O uso de '<=' garante que todas as saídas de registradores em um mesmo bloco sejam atualizados "simultaneamente" após a borda do clock (RACE CONDITIONS) e garantir que a simulação corresponda ao hardware sintetizado.

→ 3 - Atribuições blocking (=) devem ser usadas para modelar lógica combinacional pura.
> O local ideal é dentro de blocos always com lista de sensibilidade de asterisco (exemplo: always @(*)). Como a atribuição é imediata, ela descreve corretamente como os sinais se propagam através de uma cadeia de portas lógicas.
> São a única opção permitida em functions, que por definição devem executar em tempo zero e retornar um valor, ao representar um comportamento puramente combinacional.

Usar para lógica sequencial (em blocos always) é uma má prática que pode levar a race conditions e resultados de simulação diferente do hardware real.