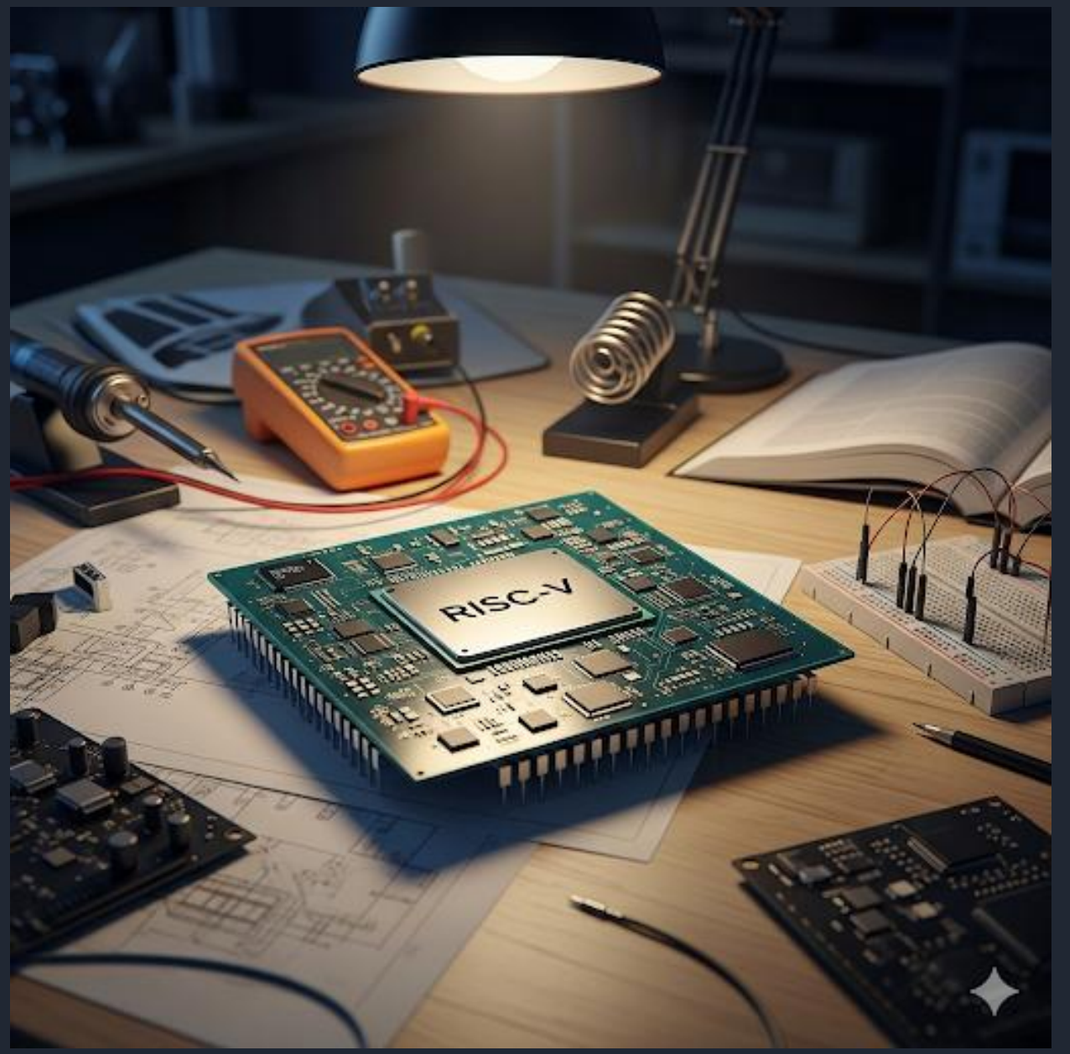
Atividade em Sala de Aula – Unid. 03 – Cap. 01

Subtítulo: Exercício – Código em VDHL – Decodificador binário de 3 para 8



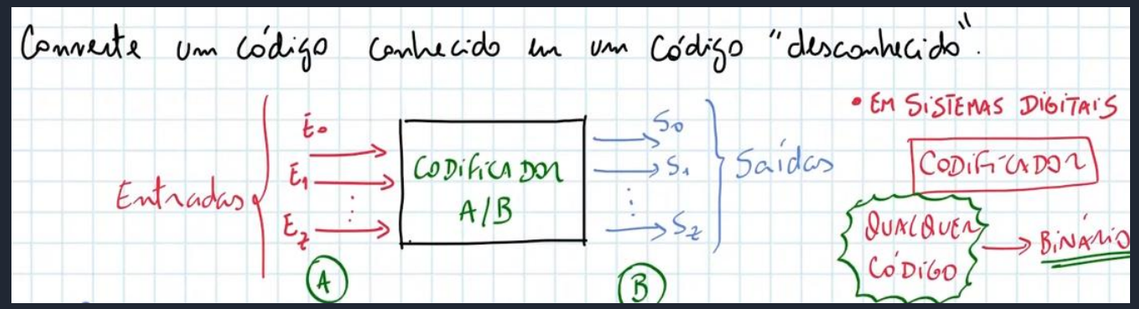
Aluno: Manoel Felipe Costa Furtado
Matrícula:20251RSE.MTC0086
Aluno: Yuri Gomes Candido
Matrícula: 20251RSE.MTC0067

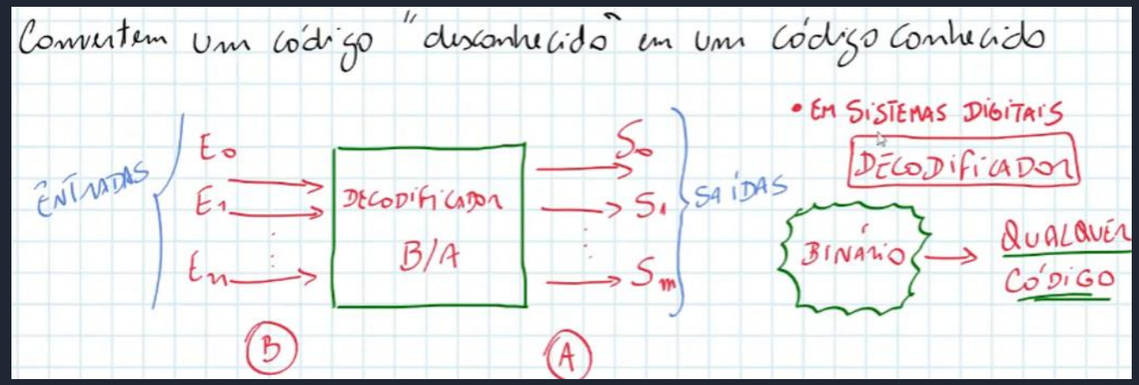
INSTITUTO FEDERAL Maranhão

embarca tech | Residência Tecnológica em Sistemas Embarcados

Codificador e Decodificador

Característica	Codificador (Encoder)	Decodificador (Decoder)
Propósito Principal	Compressão (Muitos -> Poucos)	Expansão (Poucos -> Muitos)
Nº de Entradas	2^N	N
Nº de Saídas	N	2^N
Lógica	Se a entrada `i` está ativa, a saída é o binário de `i`.	Se a entrada é o binário de `i`, a saída `i` é ativada.
Exemplo Comum	Codificador de 8 para 3	Decodificador de 3 para 8
Aplicação	Leitura de teclados, sistemas de prioridade.	Seleção de endereço de memória, controle de displays de 7 segmentos.





Codificadores

Existem diversos tipos de codificação, cada um otimizado para uma finalidade específica, como aritmética, detecção de erros, simplificação de lógica ou interface com sensores.

Tipo de Código	Característica Principal	Nº de Bits (n)	Nº de Estados Válidos	Aplicação Típica
Binário	Ponderado, bom para aritmética	n	2^n	Processamento e cálculo
Código Gray	Apenas 1 bit muda entre passos	n	2^n	Sensores de posição (encoders)
One-Hot	Apenas 1 bit é '1' por vez	n	n	Máquinas de estado em FPGAs
Código Johnson	1 bit muda por vez, fácil de decodificar	n	2n	Contadores e temporizadores
BCD	Codifica cada dígito decimal em 4 bits	4 por dígito	10	Displays e eletrônicos de consumo

3. Código One-Hot

Nesta codificação, apenas um bit do vetor de dados é '1' ("quente") em qualquer momento, enquanto todos os outros são '0' ("frios").

- **Conceito Principal:** Um único bit ativo para representar um estado.
- **Características:**
  - Simplifica muito a lógica de decodificação. Para saber se você está no estado 3, basta verificar se o bit 3 está em '1'.
  - É ineficiente em termos de armazenamento (usa N bits para representar N estados).
  - Muito seguro contra erros, pois qualquer estado com mais de um '1' é inválido.
- **Uso Comum:** Implementação de máquinas de estado em FPGAs (onde registradores são abundantes), seleção de multiplexadores e decodificação de saídas.

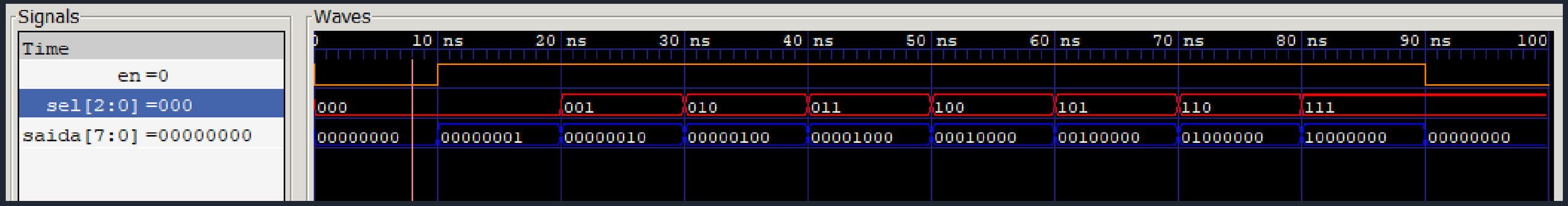
```vhdl
-- Verifica se o pino de habilitação 'en' está em nível lógico alto ('1').
if en = '1' then
    -- Se estiver habilitado, um CASE é usado para avaliar o valor da entrada 'sel'.
    case sel is
        -- Para cada valor possível de 'sel', uma única saída correspondente é ativada.
        when "000" => saida <= "00000001"; -- Ativa a saída 0
        when "001" => saida <= "00000010"; -- Ativa a saída 1
        when "010" => saida <= "00000100"; -- Ativa a saída 2
        when "011" => saida <= "00001000"; -- Ativa a saída 3
        when "100" => saida <= "00010000"; -- Ativa a saída 4
        when "101" => saida <= "00100000"; -- Ativa a saída 5
        when "110" => saida <= "01000000"; -- Ativa a saída 6
        when "111" => saida <= "10000000"; -- Ativa a saída 7
        -- 'when others' é uma cláusula de segurança para cobrir quaisquer outros
        -- casos possíveis (útil para tipos de dados com mais estados, como std_ulogic).
        when others => saida <= (others => '0');
    end case;
else
    -- Se o decodificador não estiver habilitado (en = '0'), todas as saídas
    -- são forçadas para o nível lógico baixo ('0').
    saida <= (others => '0');
end if;
```

```vhdl
stim_proc: process
begin
    -- 1. Inicia com o decodificador desabilitado para testar essa condição.
    en_tb <= '0';
    sel_tb <= "000";
    wait for 10 ns; -- Aguarda 10 nanosegundos.

    -- 2. Habilita o decodificador.
    en_tb <= '1';

    -- 3. Percorre todas as combinações de entrada possíveis para 'sel',
    --    mudando a cada 10 ns para que a mudança na saída possa ser observada.
    sel_tb <= "000"; wait for 10 ns;
    sel_tb <= "001"; wait for 10 ns;
    sel_tb <= "010"; wait for 10 ns;
    sel_tb <= "011"; wait for 10 ns;
    sel_tb <= "100"; wait for 10 ns;
    sel_tb <= "101"; wait for 10 ns;
    sel_tb <= "110"; wait for 10 ns;
    sel_tb <= "111"; wait for 10 ns;

    -- 4. Desabilita o decodificador novamente no final do teste.
    en_tb <= '0';
    wait for 10 ns;
```



1. O comportamento observado na simulação está de acordo com o esperado pela lógica descrita? Justifique com base em pelo menos um exemplo específico.

Sim, o comportamento que seria observado na simulação está perfeitamente de acordo com a lógica descrita no decodificador.
O testbench (tb_decodificador_3a8.vhd) foi projetado para testar sistematicamente todas as funcionalidades do componente decodificador_3a8.

Justificativa com Exemplo:
Lógica Esperada: No arquivo decodificador_3a8.vhd, a lógica para a entrada sel com valor "011" é que:
Se o pino de habilitação en estiver em '1', a saída deve ser "00001000".
Se en estiver em '0', a saída deve ser "00000000".

Comportamento na Simulação: O testbench primeiro desabilita o decodificador com en_tb <= '0'.
Em seguida, ele o habilita (en_tb <= '1') e, após alguns passos, aplica o valor sel_tb <= "011".
Durante os 40 ns seguintes, a simulação mostraria que o sinal saida_tb se tornaria "00001000", correspondendo exatamente ao comportamento esperado.
Por fim, ao desabilitar o componente novamente, a saída retornaria a "00000000", validando ambas as condições.

2. Quais sinais intermediários (não visíveis diretamente na saída final) são fundamentais para entender o funcionamento interno do projeto?

Neste projeto específico, devido à sua implementação com uma arquitetura puramente comportamental (behavioral), não existem sinais intermediários.

A lógica do decodificador é descrita dentro de um único bloco process. Este processo lê os sinais de entrada (sel e en) e atribui um valor diretamente ao sinal de saída (saida) com base em suas condições. Portanto, os sinais fundamentais para entender o funcionamento são as próprias entradas, sel e en, pois seus valores determinam o fluxo de controle dentro das estruturas if e case.

3. Quais construções VHDL (ex: if, case, with/select, process, port map) foram mais importantes neste projeto e por quê?

port map: É fundamental para a hierarquia e simulação do projeto.
É através do port map que o testbench se conecta à unidade sob teste (o decodificador),
mapeando os sinais de estímulo (sel_tb, en_tb) e de observação (saida_tb) às portas correspondentes do componente.

process: É a base da descrição de comportamento em ambos os arquivos.
No decodificador, o process contém toda a lógica combinacional que define como as saídas reagem às entradas.
No testbench, ele é usado para criar a sequência de estímulos ao longo do tempo.

case: Esta construção é o coração da lógica do decodificador.
Ela permite descrever de forma clara, legível e eficiente como cada valor da entrada

sel mapeia para uma saída única, que é a definição exata de um decodificador.

if: Foi crucial para implementar a funcionalidade do pino de habilitação (en).
A estrutura if en = '1' then ... else ... end if;
separa de forma limpa o estado habilitado do desabilitado, garantindo que o decodificador só funcione quando ativado.

4. Se este projeto fosse integrado a um sistema maior (como um controlador, processador ou periférico), quais sinais você exporia como interface e quais manteria internos?

Se este decodificador fosse integrado a um sistema maior, os sinais que seriam expostos como interface são exatamente aqueles definidos na sua porta (port):

- sel (in std_logic_vector(2 downto 0)):
  A entrada de seleção de 3 bits, que o sistema maior usaria para escolher qual linha de saída ativar.

- en (in std_logic): A entrada de habilitação, que o sistema maior usaria para ligar ou desligar o decodificador.

- saida (out std_logic_vector(7 downto 0)): A saída de 8 bits, que o sistema maior leria para obter o resultado da decodificação.

Conforme mencionado na pergunta 2, este projeto em sua forma atual não possui sinais internos. Todos os seus sinais são de interface. Se ele fosse reescrito de forma estrutural (usando portas lógicas), os fios conectando essas portas seriam sinais internos e não seriam expostos.

5. Qual parte da descrição ou simulação mais exigiu atenção para evitar erro lógico ou sintático? Como você validou que estava correta?

Ponto Crítico: A lista de sensibilidade process(sel, en) é crucial.

Se um sinal lido dentro do processo (como sel ou en) for omitido da lista, o simulador não atualizará a saída quando esse sinal mudar, criando um comportamento incorreto que não corresponde a um hardware combinacional real (resultando na inferência de um latch indesejado).
Além disso, era preciso garantir que a lógica if/case cobrisse todas as condições possíveis para evitar a mesma inferência de latch.

Validação: A correção foi validada através da simulação com o testbench.
O testbench foi cuidadosamente escrito para: Testar o estado desabilitado (en_tb <= '0') no início e no fim.

Percorrer todas as 8 combinações possíveis da entrada sel ("000" a "111") enquanto o decodificador está habilitado.

Ao executar a simulação e observar as formas de onda, poderíamos verificar visualmente se para cada estímulo de entrada gerado pelo testbench, a saída (saida_tb) correspondia exatamente ao valor esperado pela especificação lógica do decodificador. Essa cobertura completa garante a validação do componente.

Conclusão

GitHub:

https://github.com/ManoelFelipe/Embarcatech_37/tree/main/Segunda_Fase_FPGA/Unidade_03/