Relatório referente ao trabalho final de sistemas digitais

Nome: Bruna Izabel da Silva Pereira - 15635441
Camila Piscioneri Magalhães - 15697249

Introdução

O objetivo é desenvolver em VHDL um processador digital simples segundo as especificações do processador, sendo elas baseadas em uma cpu monocíclica, como a *figura 1*. Para realizá-lo foram usadas as seguintes unidades em formato vhdl:

- alu (arithmetic logic unit);
- Registrador (unidade de registrador padrão);
- banco_de_reg(unidade para a instanciação e escolha de qual registrador usar);
- mux_eight_two (multiplexador 8x2);
- mux_eight_four (multiplexador 8x4);
- demux (desmultiplexador);
- contador (Process counter(Pc);
- unid_controle (unidade de controle);
- cpu_offic(montagem do processador);
- memoria_ram(mem256x17)

Arithmetic Logic Unit (Alu)

Funcionalidades:

A ALU possui as seguintes funcionalidades:

- **Operações Aritméticas**: Adição e Subtração;
- **Operações Lógicas**: AND, OR, e NOT;
- **Geração de Flags**: Zero, Sinal, Carry e Overflow.

Entradas:

- A (8 bits): Primeiro operando;
- B (8 bits): Segundo operando;
- Op (4 bits): Código da operação a ser realizada.

Saídas:

- Result (8 bits): Resultado da operação;
- Zero (1 bit): Indica se o resultado é zero;
- Sinal (1 bit): Indica o sinal do resultado (bit mais significativo);
- Carry (1 bit): Indica um transporte na adição ou subtração;
- Overflow (1 bit): Indica se ocorreu overflow durante a operação.

Operações:

Código (op)	Operação	Descrição
0000	ADD	Soma dos operandos A e B. Gera as flags Carry e Overflow.
0001	SUB	Subtração de A por B. Gera as flags Carry, Sinal e Overflow.
0010	AND	Operação lógica **AND** bit a bit entre A e B.
0011	OR	Operação lógica **OR** bit a bit entre A e B.
0100	NOT	Operação lógica **NOT** bit a bit sobre o operando A.
others	DEFAULT	Saída zerada para valores de Op não especificados.

Detalhes de implementação:

Aritmética com Carry:

- A soma e subtração são realizadas com extensão de sinal, adicionando um bit extra ('0' & A, '0' & B).
- O bit mais significativo do resultado estendido (temp_result(8)) é usado para definir o **Carry**.

Cálculo de Overflow:

- O **Overflow** é calculado comparando os sinais dos operandos e o sinal do resultado:
    - Soma: Ocorre overflow se os sinais de A e B forem iguais, mas diferentes do resultado.
    - Subtração: Ocorre overflow se o sinal de A for diferente de B, mas o resultado for inconsistente.

Operações Lógicas:

- São realizadas diretamente sobre os operandos sem necessidade de cálculos adicionais.

Flags:

- **Zero**: Ativado ('1') se o resultado for 00000000.
- **Sinal**: Indica o bit mais significativo do resultado (signo em complemento de dois).
- **Carry**: Indica transporte em operações aritméticas.
- **Overflow**: Indica inconsistência de sinal em operações aritméticas.

Resultado Padrão:

- Para operações não definidas (when others), o resultado e todas as flags são zerados.

Registrador

Entradas

- clk: Sinal de clock para sincronização.
- reset: Sinal de reset (ativo alto) para zerar o registrador.
- enable: Sinal de controle para habilitar a carga de dados.
- d_in: Vetor de 8 bits representando os dados de entrada.

Saídas

- d_out: Vetor de 8 bits contendo o valor armazenado no registrador.

Funcionamento

Reset:

- Quando reset = '1', o registrador é zerado (d_out <= (others => '0')).

Sincronismo:

- Em uma borda de subida do clock (rising_edge(clk)), verifica-se o valor de enable:
  - Se enable = '1', os dados de entrada (d_in) são carregados no registrador e disponibilizados na saída (d_out).
  - Caso contrário, o valor armazenado permanece inalterado.

Banco_de_reg (Banco de registradores)

Resumo

Este código implementa um **banco de registradores de 8 bits** com 4 registradores e um mecanismo de seleção para leitura e escrita. O design permite armazenar, ler e selecionar dados em múltiplos registradores controlados por sinais de entrada.

Descrição Geral

Componentes Principais:

- **Registradores**: Armazenam valores de 8 bits. São instâncias do componente Registrador.
- **Demultiplexador (Demux)**: Direcciona o sinal de escrita (reg_write) ao registrador correspondente, baseado no endereço de escrita (WR).
- **Multiplexadores (Mux)**: Selecionam os dados de saída dos registradores com base nos endereços de leitura (end_RA e end_RB).

Funcionamento Geral:

- O sinal de escrita (reg_write) e o endereço (WR) habilitam o registrador apropriado para armazenar os dados de entrada (WD).
- Os dados podem ser lidos de dois registradores ao mesmo tempo, controlados pelos endereços de leitura (end_RA e end_RB).
- Sinais de reset (reset) são usados para limpar os valores armazenados.

Entradas:

- **clk**: Sinal de clock para sincronização.
- **reset**: Sinal de reset (ativo alto) para limpar todos os registradores.
- **reg_write**: Sinal de habilitação para escrita.
- **WR**: Endereço (2 bits) para seleção do registrador de escrita.
- **WD**: Dados de entrada (8 bits) a serem escritos.
- **end_RA e end_RB**: Endereços (2 bits cada) para seleção dos registradores de leitura.

Saídas:

- **RA e RB**: Dados de saída (8 bits) dos registradores selecionados para leitura.

Funcionamento Interno

Escrita

- O Demux direciona o sinal de escrita (reg_write) ao registrador correspondente com base no endereço (WR).
- Apenas o registrador selecionado é habilitado para armazenar o valor de entrada (WD).

Leitura

- Dois multiplexadores (Mux1 e Mux2) são usados para selecionar os dados de saída dos registradores, com base nos endereços de leitura (end_RA e end_RB).

Reset

- Quando reset = '1', todos os registradores são zerados.

Mux_eight_two

Componentes do Código

Entidade (entity)

Define a interface do módulo com as entradas e saídas:

- **A**: Entrada de 8 bits.
- **B**: Outra entrada de 8 bits.
- **Sel**: Sinal de seleção ('0' ou '1') para escolher entre as entradas.
- **Y**: Saída de 8 bits que recebe o valor de A ou B.

Arquitetura (architecture)

Descreve o comportamento do multiplexador:

- Um **processo sensível às mudanças em A, B e Sel** implementa a lógica de seleção:

- Se Sel = '0': A saída Y recebe o valor de A.
- Se Sel = '1': A saída Y recebe o valor de B.

Funcionamento do Multiplexador

- **Entrada**: Duas palavras de 8 bits (A e B) são disponibilizadas para o módulo.
- **Seleção**: O sinal Sel determina qual das entradas será passada para a saída:
- **Sel = '0'**: A entrada A é direcionada para a saída Y.
- **Sel = '1'**: A entrada B é direcionada para a saída Y.
- **Saída**: A saída Y é atualizada dinamicamente com base nas entradas e no sinal de seleção.

Mux_eight_four

Este código implementa um multiplexador (MUX) **4:1** com entradas de 8 bits (A, B, C e D) e uma saída (Y). O multiplexador seleciona uma das entradas com base no valor do sinal de seleção de 2 bits (Sel).

Componentes do Código

Entidade (entity)

Define a interface do módulo:

- **Entradas**:
  - **A, B, C, D**: Entradas de 8 bits.
  - **Sel**: Sinal de seleção de 2 bits para determinar qual entrada será passada para a saída.
- **Saída**:
  - **Y**: Vetor de 8 bits que contém o valor da entrada selecionada.

Arquitetura (architecture)

Implementa o comportamento funcional do MUX:

- Um **processo sensível às entradas (A, B, C, D) e ao sinal de seleção (Sel)** decide qual entrada será atribuída à saída:
  - **Sel = "00"**: Y recebe o valor de A.
  - **Sel = "01"**: Y recebe o valor de B.
  - **Sel = "10"**: Y recebe o valor de C.
  - **Sel = "11"**: Y recebe o valor de D.
  - **others**: Por segurança, a saída Y é zerada.

Funcionamento do Multiplexador

Entrada:

- O multiplexador recebe quatro palavras de 8 bits (A, B, C, D) como entradas.
- Um sinal de seleção de 2 bits (Sel) é usado para determinar qual entrada será direcionada à saída.

Seleção:

- O valor do sinal de seleção (Sel) é analisado em um bloco **case**:
  - "00" seleciona a entrada A.
  - "01" seleciona a entrada B.
  - "10" seleciona a entrada C.
  - "11" seleciona a entrada D.

Saída:

- A saída Y é atribuída dinamicamente com base no valor do sinal de seleção.

Demux

Componentes do Código

Entidade (entity)

Define a interface do módulo:

- **Entradas**:
  - **input_data**: Sinal de entrada que será direcionado a uma das saídas.
  - **Sel**: Sinal de seleção de 2 bits que determina para qual saída o sinal será encaminhado.
- **Saídas**:
  - **output_0, output_1, output_2, output_3**: Saídas individuais, cada uma controlada pelo sinal de seleção.

Arquitetura (architecture)

Implementa o comportamento funcional do Demux:

- Um **processo** sensível às entradas (input_data, Sel) realiza o seguinte:
  1. Se **input_data = '0'**, todas as saídas são definidas como '0'.
  2. Caso contrário, o valor de input_data é direcionado para uma das saídas, com base no valor de Sel:

        - **Sel = "00"**: input_data é atribuído a output_0.
        - **Sel = "01"**: input_data é atribuído a output_1.
        - **Sel = "10"**: input_data é atribuído a output_2.
        - **Sel = "11"**: input_data é atribuído a output_3.
    3. As demais saídas são sempre zeradas para evitar interferência.

Funcionamento do Demultiplexador

- **Entrada Única**:
- O demultiplexador recebe um único sinal binário (input_data) como entrada.
- **Seleção da Saída**:
- O sinal de seleção de 2 bits (Sel) é usado para determinar qual saída será ativada.
- **Saída Controlada**:
- Dependendo de Sel, apenas uma das saídas (output_0 a output_3) recebe o valor de input_data. As outras saídas permanecem zeradas.

Contador(PC)

O código implementa um **contador programável (Program Counter, PC)** que pode:

- Incrementar o valor atual (próxima instrução),
- Saltar para um endereço específico (Branch),
- Permanecer no mesmo endereço (NOP),
- Ou reiniciar (Reset).

Componentes do Código

Entradas

- **clk_in**: Sinal de clock que sincroniza as operações.
- **reset_in**: Sinal de reset para reiniciar o contador.
- **pc_op_in**: Sinal de 2 bits que define a operação do contador:
    - "00": Incrementa o contador.
    - "01": Salta para o endereço fornecido em pc_in.
    - "10": NOP (não realiza nenhuma alteração no valor do contador).
    - "others": Caso padrão, mantém o valor do contador.
- **pc_in**: Endereço de entrada para operações de salto (Branch).

Saída

- **pc_out**: Representa o valor atual do contador (endereço).

Sinal Interno

- **pc**: Sinal de 8 bits que armazena o estado atual do contador.

Arquitetura e Funcionamento

Reset

- Se o sinal reset_in for '1', o contador é reiniciado para 0.

Operação no Clock

- Com a borda de subida de clk_in, o comportamento do contador depende do valor de pc_op_in:
  - "00": Incrementa o valor atual (pc).
  - "01": Carrega o endereço fornecido em pc_in (Branch).
  - "10": Mantém o valor atual (NOP).
  - **Default**: Caso nenhuma operação válida seja especificada, o contador mantém seu valor.

Saída

- O valor do contador (pc) é continuamente direcionado à saída pc_out.

Unid_controle

Nome da Entidade: unid_controle
A entidade implementa uma unidade de controle para um processador, gerando sinais de controle baseados em um código de entrada (code) de 17 bits.

Entradas:

- code (17 bits): Contém a instrução codificada que define as operações da unidade de controle.

Saídas:

- **WR (2 bits):** Seleciona o registrador de destino.
- **end_RA (2 bits):** Seleciona o endereço do registrador A.
- **end_RB (2 bits):** Seleciona o endereço do registrador B.
- **Imm (8 bits):** Contém o valor imediato para operações.
- **RegWrite (1 bit):** Indica se o registrador será escrito.
- **ALUOp (4 bits):** Determina a operação da ALU.
- **ALUSrc (1 bit):** Define se o valor imediato será usado.
- **PCSrc (2 bits):** Controla o fluxo do programa (desvios/jumps).
- **cmp (1 bit):** Sinal utilizado na instrução de comparação.

Funcionamento

O código implementa uma máquina de controle combinacional baseada em um processo sensível à entrada code. Ele utiliza a segmentação dos bits do código para determinar os sinais de controle.

- **Bits Segmentados do Código**:
  - **code(16 downto 13)**: Define o tipo de instrução (opcode).
  - **code(12)**: Sinaliza se o operando imediato será utilizado (ALUSrc).
  - **code(11 downto 10)**: Determina o registrador de destino (WR).
  - **code(9 downto 8)**: Define o registrador A (end_RA).
  - **code(1 downto 0)**: Define o registrador B (end_RB).
  - **code(7 downto 0)**: Valor imediato (Imm).

Descrição das Instruções

Opcode (Bits 16:13)	Instrução	Ações
0000	ADD	Soma entre registradores. Ativa RegWrite, define ALUOp.
0001	SUB	Subtração entre registradores.
0010	AND	Operação lógica AND.
0011	OR	Operação lógica OR.
0100	NOT	Operação lógica NOT.
0101	CMP	Comparação. Ativa cmp, desativa RegWrite.
0110	JMP	Salto incondicional. Atualiza PCSrc.
0111	JEQ	Salto condicional (igualdade).
1000	JGR	Salto condicional (maior).
1001	LOAD	Carrega valor da memória.
1010	STORE	Armazena valor na memória.
1011	MOV	Move dados entre registradores.
1100	IN	Entrada de dados.
1101	OUT	Saída de dados.
1110	WAIT	Instrução de espera.

memoria_ram(mem256x17)

A memória foi gerada primeiramente demtro do quartus e depois foi feito um aquivo.mif com as instruções a partir de código feito em linguagem python.

cpu_offic