Universidade Federal de Mato Grosso do Sul

Laboratório de Hardware

System on a Chip

Alunos: Rafael Torres Nantes
Sarah Merigue Baraldi
Professor: Renan Albuquerque Marks

20 de Novembro de 2023

1 Introdução

O trabalho desenvolvido tem como objetivo modelar um hardware denominado *System on a Chip*. O sistema foi desenvolvido em **VHDL** e consiste nos seguintes subcomponentes: um **Processador**, duas **Memórias Principais** e um **Codec**.

O intuito do desenvolvimento é o software executar um conjunto de instruções *RISC* para verificar uma futura viabilidade em desempenho, na redução no consumo de recursos físicos.

2 Desenvolvimento do Projeto

2.1 Processador

2.1.1 Proposta

O processador proposto possui características de uma arquitetura **MISC** (*Minimal Instruction Set Computer*), consistindo em poucas e simples instruções. Além disso, manipula dados de 1 *byte*, suporta complemento de 2 e acessa duas memórias principais:

- **IMEM**: Memória de **instruções**;
- **DMEM**: Memória de **dados**.

Todas as instruções presentes no processador operam os dados como uma arquitetura de pilha e estão descritas na tabela abaixo:

Opcode	Mneumônico	Significado
`0x0`	`HLT`	Interrompe execução indefinidamente.
`0x1`	`IN`	Empilha um byte recebido do codec.
`0x2`	`OUT`	Desempilha um byte e o envia para o codec.
`0x3`	`PUSHIP`	Empilha o endereço armazenado no registrador **`IP`**(2 bytes, primeiro MSB[2]e depois LSB[3]).
`0x4`	`PUSH imm`	Empilha um byte contendo imediato (armazenado nos 4 bits menos significativos da instrução)
`0x5`	`DROP`	Elimina um elemento da pilha.
`0x6`	`DUP`	Reempilha o elemento no topo da pilha.
`0x8`	`ADD`	Desempilha Op1 e Op2 e empilha (Op1 + Op2).
`0x9`	`SUB`	Desempilha Op1 e Op2 e empilha (Op1 − Op2).
`0xA`	`NAND`	Desempilha Op1 e Op2 e empilha `NAND`(Op1, Op2).
`0xB`	`SLT`	Desempilha Op1 e Op2 e empilha (Op1 < Op2).
`0xC`	`SHL`	Desempilha Op1 e Op2 e empilha (Op1 ≪ Op2).
`0xD`	`SHR`	Desempilha Op1 e Op2 e empilha (Op1 ≫ Op2).
`0xE`	`JEQ`	Desempilha Op1(1 byte), Op2(1 byte) e Op3(2 bytes); Verifica se (Op1 = Op2), caso positivo soma Op3 no registrador **`IP`**.
`0xF`	`JMP`	Desempilha Op1(2 bytes) e o atribui no registrador **`IP`**.

Figura 1: Tabela de Instruções

2.1.2 Funções Implementadas

O processador desenvolvido funciona como uma máquina de estados, sendo:

1. **Halted**: CPU permanece **parada** por tempo indeterminado sem executar qualquer instrução;

2. **Fetch Instruction**: CPU **busca** o endereço da próxima instrução;
3. **Decode Instruction**: CPU **decodifica** a instrução recebida pela **IMEM** e carrega os operandos necessários para a execução;
4. **Execute Instruction**: CPU **executa** a instrução;
5. **Modify Ip**: CPU **altera** o endereço no registrador **IP**.

Além disso, o processador segue a proposta, portanto, trabalha com dados de 1 *byte*, suporta complemento de 2 (números negativos), acessa as duas memórias (**IMEM** e **DMEM**) em arquitetura de pilha e executa as seguintes instruções descritas no **firmware**:

firmware	mneumonico
00000000	HLT
00010000	IN
00100000	OUT
00110000	PUSHIP
01001100	PUSH -4
01001101	PUSH -3
01001110	PUSH -2
01001111	PUSH -1
01000000	PUSH 0
01000001	PUSH 1
01000010	PUSH 2
01000011	PUSH 3
01000100	PUSH 4
01010000	DROP
01100000	DUP
10000000	ADD
10010000	SUB
10100000	NAND
10110000	SLT
11000000	SHL
11010000	SHR
11100000	JEQ
11110000	JMP

O processador possui somente dois registradores:

- **IP** (Instruction Pointer): Armazena o endereço da **instrução** a ser executada, ou seja, aponta para a **IMEM**.
- **SP** (Stack Pointer): Armazena o endereço do topo da pilha da memória de **dados**, ou seja, para **DMEM**.

2.2 Memória

2.2.1 Proposta

A memória proposta possui os mesmos parâmetros **genéricos** da entidade **CPU**:

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

- **addr_width** : natural := 16 sendo tamanho do endereço da memória (16 *bits*);
- **data_width** : natural := 8 sendo tamanho do dado da memória (8 *bits*).

Esta entidade é instanciada duas vezes dentro da entidade **SoC** representando as memórias **IMEM** e **DMEM** do processador.

2.2.2 Funções Implementadas

A memória implementada segue as mesmas diretrizes da proposta acima. Sendo implementada como uma arquitetura comportamental (*behavioral*). Por padrão, o barramento de dados lidos representado pelo sinal **data_out** retorna 4 *bytes* lidos a partir do endereçoo enviado no sinal **data_addr**.

```
architecture behavioral of memory is
    subtype instruction is std_logic_vector((data_width-1) downto 0);
    type mem_type is array ((2 ** data_width - 1) downto 0) of instruction;
    signal mem : mem_type := (others => (others => '0'));

begin
    process(clock)
    begin

        data_out <= std_logic_vector(to_unsigned(0, data_width*4));

        if(data_write = '1' and falling_edge(clock)) then
                mem(to_integer(unsigned(data_addr))) <= data_in(data_width-1 downto 0); -- Instruction (OpCode + Immediate)
                mem(to_integer(unsigned(data_addr)) + 1) <= data_in(2*data_width-1 downto data_width); -- Instruction : PUSHIP, JEQ e JMP
        end if;

        if(data_read = '1') then
            data_out <= mem(to_integer(unsigned(data_addr)) + 3) &
                        mem(to_integer(unsigned(data_addr)) + 2) &
                        mem(to_integer(unsigned(data_addr)) + 1) &
                        mem(to_integer(unsigned(data_addr)) + 0);
        end if;

    end process;

end behavioral;
```

Figura 2: Código da Memória

2.3 Codec

2.3.1 Proposta

O Codec proposto é a entidade que faz a comunicação de entrada e saída com a **CPU**. Ela trabalha com *bytes* lidos de um arquivo de texto e os transmite à **CPU**, assim como recebe *bytes* da CPU que são **impressos** em um arquivo de saída.

A entidade Codec é ser **instanciada** dentro da entidade **SoC** e conectda à **CPU**. Sua funcionalidade gira em torno de duas intruções, IN e OUT.

1. Quando executada a instrução **IN**:
   - O processador **suspende** a execução;
   - Atribui **1** ao **read_signal**;
   - Atribui **0** ao **write_signal**;
   - Emite um pulso no sinal **interrupt**

2. Quando executada a instrução **OUT**:
   - O processador **suspende** a execução;
   - Atribui **0** ao **read_signal**;
   - Atribui **1** ao **write_signal**;
   - Emite um pulso no sinal **interrupt**.

Ao final de cada um das intruções, o **Codec** gera um pulso no sinal **valid**.

2.3.2 Funções Implementadas

A entidade Codec implementada segue as mesmas diretrizes da proposta acima. Sendo implementada como uma arquitetura de fluxo de dados (*dataflow*). Lê o arquivo de entrada "input.txt"e escreve no arquivo de saída "output.txt".

```
architecture dataflow of codec is

   file file_input : text open read_mode is "codec/input.txt";
   file file_output : text open write_mode is "codec/output.txt";

   signal aux_data : std_logic_vector(7 downto 0);
```

Figura 3: Código do Codec

2.4 SoC

2.4.1 Proposta

O SoC proposto é a entidade de mais alto nível da hierarquia e **contém** todas as outras entidades, ou seja, faz o **encapsulamento** e ligação entre elas.

A entidade possui somente dois sinais de entrada:

- **Clock**: Recebe o **pulso** de clock gerado por um circuito auxiliar **externo**;
- **Started**: Iniciar a execução quando colocado no valor **1**.

2.4.2 Funções Implementadas

A entidade SoC implementada segue as mesmas diretrizes da proposta acima. Sendo implementada como uma arquitetura de fluxo de dados (*dataflow*). Lê o arquivo de entrada "**firmware.bin**"e mapeia as entidades **Codec**, **IMEM**, **DMEM** e **CPU**.

```
codec : entity work.codec(dataflow)
    port map(
        interrupt => codec_interrupt,
        read_signal => codec_read,
        write_signal => codec_write,
        valid => codec_valid,
        codec_data_in => codec_data_in,
        codec_data_out => codec_data_out
    );
```

Figura 4: Mapeamento do Codec

```
imem : entity work.memory(behavioral)
    generic map(
        addr_width => addr_width,
        data_width => data_width
    )
    port map(
        clock => clock,
        data_read => imem_data_read,
        data_write => imem_data_write,
        data_addr => imem_data_addr,
        data_in => imem_data_in,
        data_out => imem_data_out
    );
```

Figura 5: Mapeamento da IMEM

```
dmem : entity work.memory(behavioral)
    generic map(
        addr_width => addr_width,
        data_width => data_width
    )
    port map(
        clock => clock,
        data_read => dmem_data_read,
        data_write => dmem_data_write,
        data_addr => dmem_data_addr,
        data_in => dmem_data_in,
        data_out => dmem_data_out
    );
```

Figura 6: Mapeamento da DMEM

```vhdl
cpu : entity work.cpu(behavioral)
    generic map(
        addr_width => addr_width,
        data_width => data_width
    )
    port map(
        clock => clock,
        halt => cpu_halt,
        instruction_in => cpu_instruction_in,
        instruction_addr => cpu_instruction_addr,
        mem_data_read => dmem_data_read,
        mem_data_write => dmem_data_write,
        mem_data_addr => dmem_data_addr,
        mem_data_in => dmem_data_in,
        mem_data_out => dmem_data_out,
        codec_interrupt => codec_interrupt,
        codec_read => codec_read,
        codec_write => codec_write,
        codec_valid => codec_valid,
        codec_data_out => codec_data_out,
        codec_data_in => codec_data_in
    );
```

Figura 7: Mapeamento da CPU

3 Problemas Encontrados

3.1 Dificuldades

Os autores desse trabalho encontraram dificuldades ao compreender e definir as entradas e saídas do testbench da CPU, porém acredita-se que conseguiram sanar as dúvidas durante a elaboração do trabalho.

3.2 Problemas Solucionados

Os autores desse trabalho tiveram problemas ao fazer conversões de dados:

- De positivo (unsigned) para negativo (signed);
- Do tipo natural para o tipo std_vector.

Outro problema encontrado foi no *testbench* da **CPU** onde o mesmo tentava executar novamente a si próprio. O problema foi resolvido usando uma instrução de wait ao final do código.

3.3 Problemas Não-Solucionados

Ao executar o testbench do **SoC** há o seguinte warning que não se sabe ainda como removê-lo:

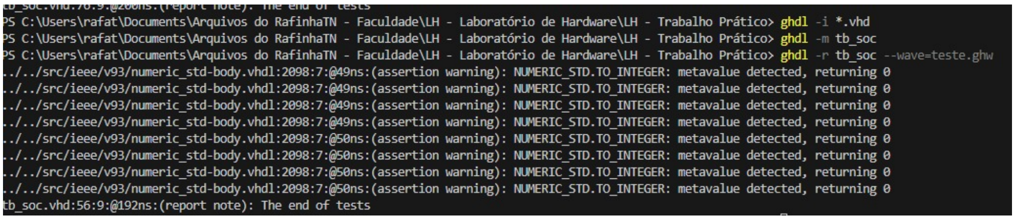

Figura 8: Warning do tb_soc.vhd

Além disso, os autores não têm certeza de que **todas** as funcionalidades da entidade **SoC** estão operando corretamente.