Sistemas Reconfiguráveis – Eng. de Computação
Especificações para o primeiro trabalho

2º semestre de 2024

1. Objetivo

Nesse trabalho serão feitos dois projetos independentes: **addr_mux**, especificado no item 2 e **alu**, no item 3. Cada projeto consiste em descrever em linguagem VHDL, comentar, simular o funcionamento e comentar os resultados da simulação, de acordo com as especificações apresentadas. Deverá ser usado exclusivamente **código concorrente**, e todas as entradas e saídas deverão ser do tipo **std_logic** ou **std_logic_vector**. Os dois projetos são totalmente combinacionais, ou seja, não têm *latches* nem *flip-flops*.

Deverá ser entregue um relatório do trabalho na forma de um documento padrão ABNT para trabalhos acadêmicos (Capa, folha de rosto, índice de figuras, etc, etc) em um arquivo no formato pdf, via Canvas. Nesse relatório, cada projeto deverá estar em um capítulo próprio. Além do relatório, deverá ser entregue um arquivo compactado (.zip ou .rar), com todos os arquivos dos projetos gerados no ambiente Quartus. Nesse arquivo, cada projeto deverá estar em uma pasta própria. Obrigatoriamente deverá ser usada a **versão 9.1sp2** do *software* Quartus. Essa versão poderá ser baixada do *link*:

https://1drv.ms/u/s!AvS7tfohiU-IgZJ4cWPDZ1UQexI0fw

Para a avaliação, a primeira parte (item 2 – **addr_mux**) valerá 15% do total de pontos do projeto, e a segunda parte (item 3 – **ALU**) os outros 85 %.

2. Addr_mux

Multiplexador com saída de 9 bits. A saída deve ser igual à concatenação das entradas irp_in e ind_addr_in (nessa ordem, do mais significativo para o menos significativo) quando todos os bits de dir_addr_in forem iguais a ‘0’. Caso contrário, a saída deve ser deve ser igual à concatenação das entradas rp_in e dir_addr_in (nessa ordem, do mais significativo para o menos significativo).

Obs.: Os nomes das entradas e da saída farão sentido no desenvolvimento do quarto projeto.

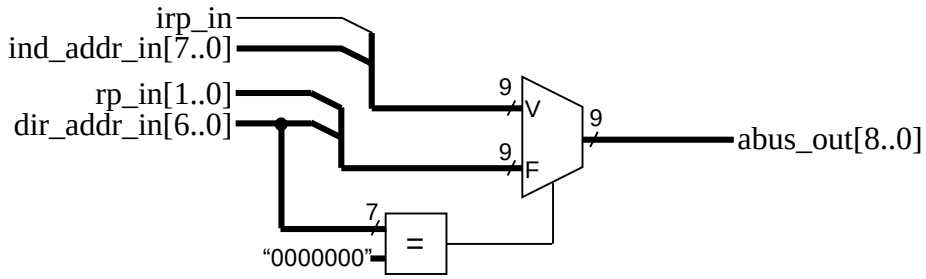
2.1. Entradas

rp_in[1..0]	Entrada de seleção de banco para endereçamento direto.
dir_addr_in[6..0]	Entrada para endereçamento direto.
irp_in	Entrada de seleção de banco para endereçamento indireto.
ind_addr_in[7..0]	Entrada para endereçamento indireto.

2.2. Saídas

abus_out[8..0]	Saída de endereço.

2.3. Diagrama equivalente



3. ALU

Unidade lógica e aritmética (ALU na sigla em inglês) que faz operações lógicas e aritméticas em palavras de 8 bits. Realiza 16 operações diferentes, entre operações lógicas, aritméticas, de rotação e de manipulação de bit, conforme especificado a seguir.

3.1. Entradas

a_in[7..0]	Entrada "a" de dados.
b_in[7..0]	Entrada "b" de dados. Usada nas operações que envolvem dois operandos.
c_in	Entrada de *carry*. Usada nas operações de rotação (RR e RL).
op_sel[3..0]	Entrada de seleção da operação a ser realizada.
bit_sel[2..0]	Entrada de seleção de bit. Usada apenas nas operações de manipulação de bit (BC e BS)

3.2. Saídas

r_out[7..0]	Saída do resultado.
c_out	Saída de *carry/borrow*. Nas operações aritméticas de soma, este sinal é o *carry out* (vai um) no bit mais significativo. Nas operações de subtração, este sinal é o *borrow out* (empréstimo). Para *borrow*, a polaridade é invertida (ver exemplos ao final do item 2.3). Este sinal também é usado nas operações de rotação.
dc_out	Saída de *digit carry/borrow*. Nas operações aritméticas de soma, este sinal é o *carry out* (vai um) do bit 3 para o bit 4, ou seja, do primeiro *nibble* para o segundo. Nas operações de subtração, este sinal é o *borrow out* (empréstimo). Para *borrow*, a polaridade é invertida.
z_out	Saída de zero. Na maior parte das operações, sinaliza quando o resultado é igual a zero. Nas operações BC e BS, corresponde ao bit selecionado pela entrada bit_sel.

3.3. Operações

op_sel[3..0]	Mnemônico	Operação
0000	XOR	XOR lógico: r_out = a_in XOR b_in (bit a bit) z_out = '1' se o resultado for igual a 0
0001	OR	OR lógico: r_out = a_in OR b_in (bit a bit) z_out = '1' se o resultado for igual a 0
0010	AND	AND lógico: r_out = a_in AND b_in z_out = '1' se o resultado for igual a 0
0011	CLR	Limpa: r_out = "00000000" z_out = '1'

0100	ADD	Soma: r_out = a_in + b_in c_out = ‘1’ se houver *carry* no bit mais significativo dc_out = ‘1’ se houver *carry* no primeiro *nibble* z_out = ‘1’ se o resultado for igual a 0
0101	SUB	Subtração: r_out = a_in – b_in c_out = ‘0’ se houver *borrow* no bit mais significativo dc_out = ‘0’ se houver *borrow* no primeiro *nibble* z_out = ‘1’ se o resultado for igual a 0
0110	INC	Incremento: r_out = a_in + 1 z_out = ‘1’ se o resultado for igual a 0
0111	DEC	Decremento: r_out = a_in – 1 z_out = 1 se o resultado for igual a 0
1000	PASS_A	Passa A: r_out = a _in z_out = ‘1’ se o resultado for igual a 0
1001	PASS_B	Passa B: r_out = b_in z_out = ‘1’ se o resultado for igual a 0
1010	COM	Complemento (inverte todos os bits): r_out = NOT a_in z_out = ‘1’ se o resultado for igual a 0
1011	SWAP	Permuta *nibbles* r_out = a_in[3..0], a_in[7..4]
1100	BS	Ajusta em ‘1’ o bit apontado por bit_sel: r_out = a_in, exceto bit apontado por bit_sel, igual a ‘1’ z_out = a_in(N), onde a_in(N) = bit apontado por bit_sel.
1101	BC	Limpa o bit apontado por bit_sel: r_out = a_in, exceto bit apontado por bit_sel, igual a ‘0’ z_out = a_in(N), onde a_in(N) = bit apontado por bit_sel.
1110	RR	Rotação para direita, passando pelo *carry*: r_out = cin, a_in[7..1] c_out = a_in[0]
1111	RL	Rotação para esquerda, passando pelo *carry*: r_out = a_in[6..0], c_in c_out = a_in[7]

Obs.: Nas instruções onde as saídas z_out, c_out e dc_out não estão especificadas, o resultado não importa (*don’t care*)

Exemplos para a operação ADD:

1) Entradas: a_in = 37h, b_in = 10h; Saídas: r_out = 47h, c_out = ‘0’, dc_out = ‘0’, z_out = ‘0’

2) Entradas: a_in = 10h, b_in = F7h; Saídas: r_out = 07h, c_out = ‘1’, dc_out = ‘0’, z_out = ‘0’

3) Entradas: a_in = 70h, b_in = 90h; Saídas: r_out = 00h, c_out = ‘1’, dc_out = ‘0’, z_out = ‘1’

4) Entradas: a_in = 17h, b_in = 3Ah; Saídas: r_out = 51h, c_out = ‘0’, dc_out = ‘1’, z_out = ‘0’

Exemplos para a operação SUB:

1) Entradas: a_in = 02h, b_in = 01h; Saídas: r_out = 01h, c_out = ‘1’, dc_out = ‘1’, z_out = ‘0’

2) Entradas: a_in = 02h, b_in = 02h; Saídas: r_out = 00h, c_out = ‘1’, dc_out = ‘1’, z_out = ‘1’

3) Entradas: a_in = 02h, b_in = 03h; Saídas: r_out = FFh, c_out = ‘0’, dc_out = ‘0’, z_out = ‘0’

Exemplos para a operação BC (bit clear):

1) Entradas: a_in = “01**1**10110”, bit_sel = “101” ; Saídas: r_out = “01**0**10110”, z_out = ‘1’

   Na saída r_out, o bit 5 (bit_sel = “101”) foi ajustado para ‘0’.
   Como na entrada a_in o bit 5 é igual a ‘1’, a saída z_out recebe ‘1’.

2) Entradas: a_in = “01**0**10110”, bit_sel = “101” ; Saídas: r_out = “01**0**10110”, z_out = ‘0’

   Na saída r_out, o bit 5 (bit_sel = “101”) foi ajustado para ‘0’.
   Como na entrada a_in o bit 5 é igual a ‘0’, a saída z_out recebe ‘0’.

Exemplos para a operação BS (bit set):

1) Entradas: a_in = “0111**0**110”, bit_sel = “011” ; Saídas: r_out = “0111**1**110”, z_out = ‘0’

   Na saída r_out, o bit 3 (bit_sel = “011”) foi ajustado para ‘1’.
   Como na entrada a_in o bit 3 é igual a ‘0’, a saída z_out recebe ‘0’.

2) Entradas: a_in = “0111**1**110”, bit_sel = “011” ; Saídas: r_out = “0111**1**110”, z_out = ‘1’

   Na saída r_out, o bit 3 (bit_sel = “011”) foi ajustado para ‘1’.
   Como na entrada a_in o bit 3 é igual a ‘1’, a saída z_out recebe ‘1’.