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

1º semestre de 2025

1. Objetivo

Descrever em linguagem VHDL, comentar o código, simular o funcionamento e descrever os casos de teste e os resultados da simulação de uma unidade lógica e aritmética (ALU na sigla em inglês), conforme especificado a seguir.

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) em um arquivo no formato pdf, via Canvas. Além do relatório, deverá ser entregue um arquivo compactado (.zip ou .rar), com todos os arquivos do projeto gerados no ambiente Quartus. 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

2. ALU

Faz operações lógicas e aritméticas em palavras de 8 bits. Realiza 16 funções diferentes, entre operações lógicas, aritméticas, de rotação e de deslocamento. O circuito deverá ser totalmente combinacional e deverá ser descrito usando exclusivamente **código concorrente**, ou seja, não tem *latches* nem *flip-flops*. Todas as entradas e saídas deverão usar o tipo STD_LOGIC ou STD_LOGIC_VECTOR.

2.1. 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 em algumas operações aritméticas e de rotação)
op_sel[3..0]	Entrada de seleção da operação a ser realizada.

2.2. Saídas

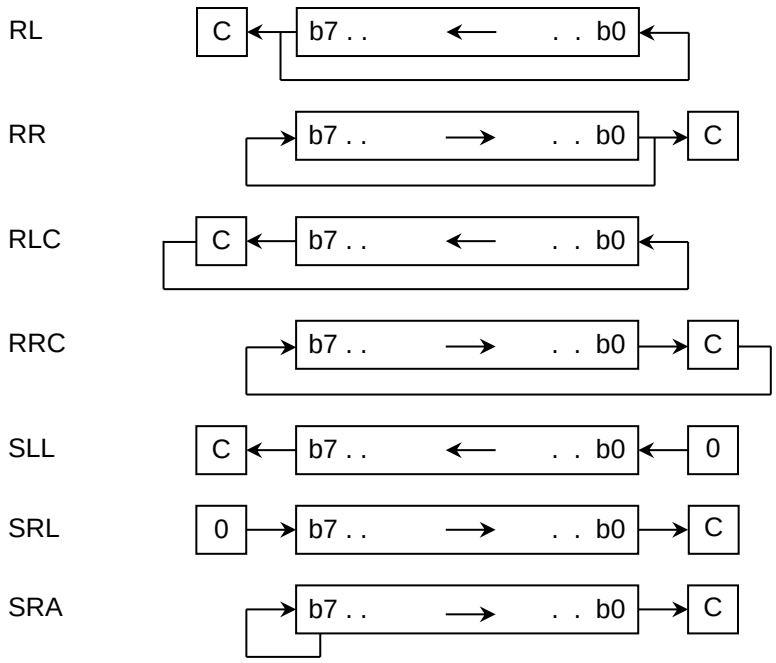
r_out[7..0]	Saída do resultado.
c_out	Saída de *carry/borrow*. Nas operações aritméticas de soma, este sinal é o *carry out* (vai um) no bit mais significativo. Nas operações de subtração, este sinal é o *borrow out* (empréstimo). Este sinal também é usado nas operações de rotação.
z_out	Saída de zero. Sinaliza (z_out = ‘1’) quando o resultado da operação é zero.
v_out	Saída de *overflow*. Sinaliza (v_out = ‘1’) quando há um *overflow* nas operações de soma e subtração. Um overflow ocorre quando: soma de dois números positivos com resultado negativo, soma de dois números negativos com resultado positivo, um número positivo menos um negativo com resultado negativo ou um número negativo menos um positivo com resultado positivo.

2.3. Operações

op_sel[3..0]	Mnemônico	Operação
0000	ADD	Soma sem *carry in*: r_out = a_in + b_in c_out = '1' se houver *carry* no bit mais significativo z_out = '1' se o resultado for igual a zero v_out = '1' se ocorreu *overflow*
0001	ADDC	Soma com *carry in*: r_out = a_in + b_in + c_in c_out = '1' se houver *carry* no bit mais significativo z_out = '1' se o resultado for igual a zero v_out = '1' se ocorreu *overflow*
0010	SUB	Subtração sem *carry in*: r_out = a_in - b_in c_out = '1' se houver *borrow* no bit mais significativo z_out = '1' se o resultado for igual a zero v_out = '1' se ocorreu *overflow*
0011	SUBC	Subtração com *carry in*: r_out = a_in - b_in – c_in c_out = '1' se houver *borrow* no bit mais significativo z_out = '1' se o resultado for igual a zero v_out = '1' se ocorreu *overflow*
0100	AND	AND lógico bit a bit: r_out = a_in AND b_in c_out = '0' z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
0101	OR	OR lógico bit a bit: r_out = a_in OR b_in c_out = '0' z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
0110	XOR	XOR lógico bit a bit: r_out = a_in XOR b_in c_out = '0' z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
0111	NOT	Complemento (inverte todos os bits) r_out = NOT a_in c_in = '0' z_in = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1000	RL	Rotação para esquerda: r_out = a_in[6..0], a_in[7] c_out = a_in[7] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1001	RR	Rotação para direita: r_out = a_in[0], a_in[7..1] c_out = a_in[0] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1010	RLC	Rotação para esquerda através do *carry*: r = a[6..0], c_in c_out = a_in[7] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)

1011	RRC	Rotação para direita através do *carry*: r_out = c_in, a_in[7..1] c_out = a_in[0] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1100	SLL	Deslocamento lógico para esquerda: r_out = a_in[6..0], '0' c_out = a_in[7] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1101	SRL	Deslocamento lógico para direita: r_out = '0', a_in[7..1] c_out = a_in[0] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1110	SRA	Deslocamento aritmético para direita: r_out = a_in[7], a_in[7..1], c_out = a_in[0] z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)
1111	PASS_B	*By-pass B* r_out = b_in c_out = '0' z_out = '1' se o resultado for igual a zero v_out = não interessa (*don't care*)

Operações de deslocamento e rotação do operando, conforme o desenho abaixo:



Exemplos para a operação ADD:

1) Entradas: a_in = 37h, b_in = 10h; Saídas: r_out = 47h, c_out = ‘0’, z_out = ‘0’
2) Entradas: a_in = 10h, b_in = F7h; Saídas: r_out = 07h, c_out = ‘1’, z_out = ‘0’
3) Entradas: a_in = 70h, b_in = 90h; Saídas: r_out = 00h, c_out = ‘1’, z_out = ‘1’

Exemplos para a operação SUB:

1) Entradas: a_in = 02h, b_in = 01h; Saídas: r_out = 01h, c_out = ‘0’, z_out = ‘0’
2) Entradas: a_in = 02h, b_in = 02h; Saídas: r_out = 00h, c_out = ‘0’, z_out = ‘1’
3) Entradas: a_in = 02h, b_in = 03h; Saídas: r_out = FFh, c_out = ‘1’, z_out = ‘0’