Laboratório 04

Instruções:

- Quando for demonstrar seu trabalho, tome nota do número da placa utilizada. O número da placa será utilizado para atribuir a nota ao grupo.
- A última página deste documento contém um checklist com todos os arquivos que fazem parte da entrega.
- Os nomes dos arquivos devem ser seguidos, e isso faz parte da avaliação.
- A entrega deverá estar em único arquivo .ZIP, com o nome **T_Lab04_RA.zip**, **T** é a turma, e **RA** é o RA do componente do grupo que fará a entrega. Por exemplo, B_Lab04_123456.zip é a entrega do grupo do aluno com o RA 123456, na turma B.
- Não divida ou agrupe em pastas os arquivos dentro do .ZIP.
- A entrega deve ser feita pelo Google Forms (https://forms.gle/qBnoCDXBQec8tpvE6). Você deve estar autenticado com uma conta do Google - pode ser uma conta pessoal ou da DAC.
- Apenas um integrante do grupo precisa fazer a entrega.
- Preste especial atenção aos nomes das entidades e sinais (entradas e saídas) descritos nos laboratórios. Isso também faz parte da avaliação.
- Se mais do que um arquivo for recebido para a mesma entrega, o último recebido será considerado. Utilize o mesmo RA do aluno entregando.
- Faça o download do arquivo **lab04_material_v2020.2.zip**. Esse arquivo já contém as descrições de *entity* necessárias para implementar os circuitos. Utilize elas, e não as altere.

Parte I - Somador Ripple Carry

Um somador binário de *n* bits é um sistema combinacional que tem duas entradas (*x* e *y*) de *n* bits que representam os operandos da operação de adição e uma saída de *n* bits que representa o resultado. Sinais adicionais de entrada e saída, chamados de *carry-in* (cin) e *carry-out* (cout) são usados para facilitar a implementação de somadores maiores.

I.1. Projete na entidade *full_adder* <full_adder.vhd>, com VHDL estrutural, um somador completo de um bit (com os *carries*), utilizando somente portas lógicas. Elabore uma simulação para testar o funcionamento do seu circuito.

ENTREGA: Arquivo **full_adder.vhd** e um screenshot dos resultados da simulação em **full_adder.png**.

I.2. Usando o circuito do item *a* implemente um somador *ripple-carry adder* genérico de N bits (entidade *ripple_carry* <ripple_carry.vhd>). O circuito deve ter uma saída adicional para indicar overflow. Não utilize pacotes (packages).

ENTREGA: Arquivo **ripple_carry.vhd** e um screenshot de resultados da simulação em **ripple_carry.png** (o screenshot pode mostrar apenas um trecho da simulação)**.**

I.3. Instancie o componente *ripple-carry adder* para N=4 na entidade *ripple_carry_board* <ripple_carry_board.vhd> e faça as seguintes ligações de entrada e saída: SW(7..4) para entrada do operando x e SW(3..0) para entrada do operando y; HEX4 mostra o operando x em hexadecimal, HEX2 o operando y em hexadecimal, e HEX0 mostra o resultado em hexadecimal; LEDR(0) acende se houver *overflow*.
Dicas: Utilize seu circuito *bin2hex* (Laboratório 02) para controlar os visores de sete segmentos. Lembre-se que a representação em hexadecimal não mostrará o sinal dos números.

Parte II - ALU

Uma unidade aritmética lógica (*arithmetic – logic* unit) ALU é um módulo capaz de realizar um conjunto de funções aritméticas e lógicas. Para isto, uma ALU tem vetores de entrada/saída de dados, bem como entradas e saídas de controle. A Figura 1 mostra uma possível especificação de uma ALU de 4 bits.

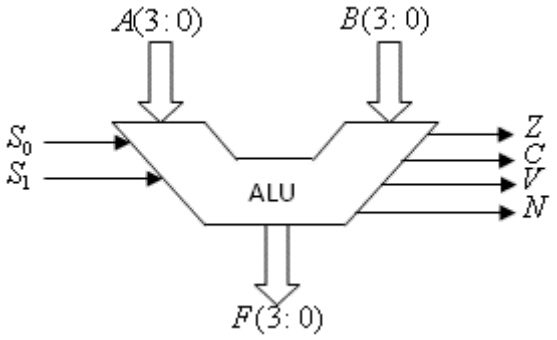

Figura 1: ALU de 4 bits

A ALU da Figura 1 tem entradas (A e B) de 4 bits, e s_0 e s_1 são sinais de controle para selecionar a operação (ver tabela 1 a seguir). A saída F de 4 bits contém o resultado da operação. Z é 1 se o resultado da operação for zero, e 0 caso contrário.

Os sinais C, V e N são 0 para operações lógicas e tem o seguinte significado para operações aritméticas. C é 1 se houver um *carry* em operações de soma, V é 1 se houver *overflow* e N é igual a 1 se o resultado for negativo. C, V e N são 0 nos restante dos casos.

S_0	S_1	Operação (F)
0	0	A+B
1	0	A-B
0	1	AND
1	1	OR

Tabela 1: Operações ALU

II.1. Modifique a implementação de uma das suas entidades de controle de visores de 7 segmentos (bin2dec ou bin2hex, do Laboratório 02) para tratar entradas de 4 bits em complemento de 2. Utilize o arquivo <two_comp_to_7seg.vhd>, em que a entrada bin(3..0) é o número a ser representado, a saída segs(6..0) são os segmentos de *g* a *a* (segs(0)=a,

segs(6)=g) do visor e a saída neg é 1 quando o número for negativo e 0 em caso contrário.

ENTREGA: Arquivo **two_comp_to_7seg.vhd**.

II.2. Usando o somador *ripple-carry adder* da parte I, implemente uma ALU de quatro bits (**alu.vhd**) com as operações descritas pela Tabela 1. A ALU deve implementar as operações de soma e subtração utilizando **o mesmo** *ripple-carry adder* (ou seja, instanciar ele apenas uma vez). Não utilize pacotes (packages).

ENTREGA: Arquivo **alu.vhd** e um screenshot de resultados de simulação em **alu.png**. Inclua na sua simulação alguns casos extremos (por exemplo, overflow).
Dica: utilize a entrada de carry-in do somador para conversão para complemento de 2.

II.3. Instancie o circuito da ALU para gravação na placa utilizando a entidade *alu_board* (arquivo **alu_board.vhd**). Utilize as entradas SW(9..8) como s_0 e s_1, SW(7..4) como operador A e SW(3..0) como operador B. Represente o operador A em HEX4, B em HEX2 e a saída F em HEX0, com o segmento g do visor logo à esquerda (HEX5, HEX3 e HEX1, respectivamente) representando o sinal negativo, quando necessário. Para operações lógicas mostre os valores em hexadecimal (utilizando seu circuito bin2hex do Laboratório 02), e para operações aritméticas mostre os valores em decimal com sinal. Utilize os leds LEDR(3..0) para representar Z,C,V,N, respectivamente.

- ENTREGA -

Entregue um único arquivo comprimido em formato **ZIP** de nome **T_Lab04_RA.zip**, onde **RA** é o RA do aluno entregando e **T** é a turma, contendo:

- Arquivos **full_adder.vhd** e **full_adder.png** do item **I.1**.
- Arquivos **ripple_carry.vhd** e **ripple_carry.png** do item **I.2**.
- Arquivo **two_comp_to_7seg.vhd** do item **II.1**.
- Arquivos **alu.vhd** e **alu.png** do item **II.2**.