Sistemas Reconfiguráveis – Eng. de Computação

Especificações para o segundo trabalho

1º semestre de 2025

1. Objetivo

Nesse trabalho serão feitos quatro projetos independentes: **reg_bank**, especificado no item 2, **Wreg**, especificado no ítem 3, **stack,** item 4 e **prog_cnt**, item 5. Cada projeto consiste em descrever em linguagem VHDL, comentar o código, simular o funcionamento, descrever os casos de teste e comentar os resultados da simulação, de acordo com as especificações apresentadas. Poderá ser usado código concorrente ou sequencial, e todas as entradas e saídas deverão ser do tipo **std_logic** ou **std_logic_vector**.

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. 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.

2. Reg_bank

Esse conjunto de registradores implementa um banco de registradores com quatro registradores de 8 bits cada um, chamados R0 a R3. O registrador R3 armazena o *status* do processador. Os bits 0, 1 e 2 do registrador R3 armazenam, respectivamente, o resultado das saídas de **c**, **z** e **v** da ALU. Os demais bits de R7 não têm função especial de *status*.

Para a escrita no banco de registradores (R0 a R3), existe uma entrada de dados para escrita (8 bits), uma entrada de seleção para escrita (seleciona qual registrador receberá os dados) e uma entrada de habilitação para a escrita (só escreve se estiver habilitado). A escrita é feita síncrona com o sinal de *clock*. A leitura é feita através de duas saídas simultâneas, cada uma com a sua entrada de seleção (seleciona qual registrador vai para a saída). A leitura é feita de maneira assíncrona (independente do sinal de *clock*) e não tem habilitação.

As flags **c**, **z** e **v** (R3(0), R3(1) e R3(2), respectivamente) têm entradas dedicadas de dado e habilitação para escrita. Em todos, a escrita é feita síncrona com o sinal de *clock*. Cada um desses três bits tem uma saída dedicada, independente de seleção ou habilitação.

2.1. Entradas

clk_in	Entrada de *clock* para escrita em todos os registradores. A escrita acontece na borda de subida do *clock*, desde que habilitada.
nrst	Entrada de *reset* assíncrono. Quando ativada (nível lógico baixo), todos os registradores deverão ser zerados. Esta entrada tem preferência sobre todas as outras.
regn_di[7..0]	Entrada de dados para escrita nos registradores R0 a R3.
regn_wr_sel[1..0]	Entrada para selecionar qual registrador (de R0 a R3) vai ser escrito quanto a operação de escrita estiver habilitada por reg_wr_ena.
regn_wr_ena	Entrada de habilitação para escrita nos registradores R0 a R3, ativo em nível alto.
regn_rd_sel[1..0]	Entrada para selecionar qual registrador (de R0 a R3) vai ser lido na saída.
c_flag_in	Entrada de dado para escrita no registrador R3(0) (C).
z_flag_in	Entrada de dado para escrita no registrador R3(1) (Z).
v_flag_in	Entrada de dado para escrita no registrador R3(2) (V).
c_flag_wr_ena	Entrada de habilitação para escrita no registrador R3(0) (C), ativa em nível alto.

z_flag_wr_ena Entrada de habilitação para escrita no registrador R3(1) (Z), ativa em nível alto.

v_flag_wr_ena Entrada de habilitação para escrita no registrador R3(2) (V), ativa em nível alto.

Obs.: As entradas c_flag_wr_ena, z_flag_wr_ena, v_flag_wr_ena têm prioridade sobre a escrita em R3 através de reg_wr_ena e reg_wr_sel[1..0].

2.2. Saídas

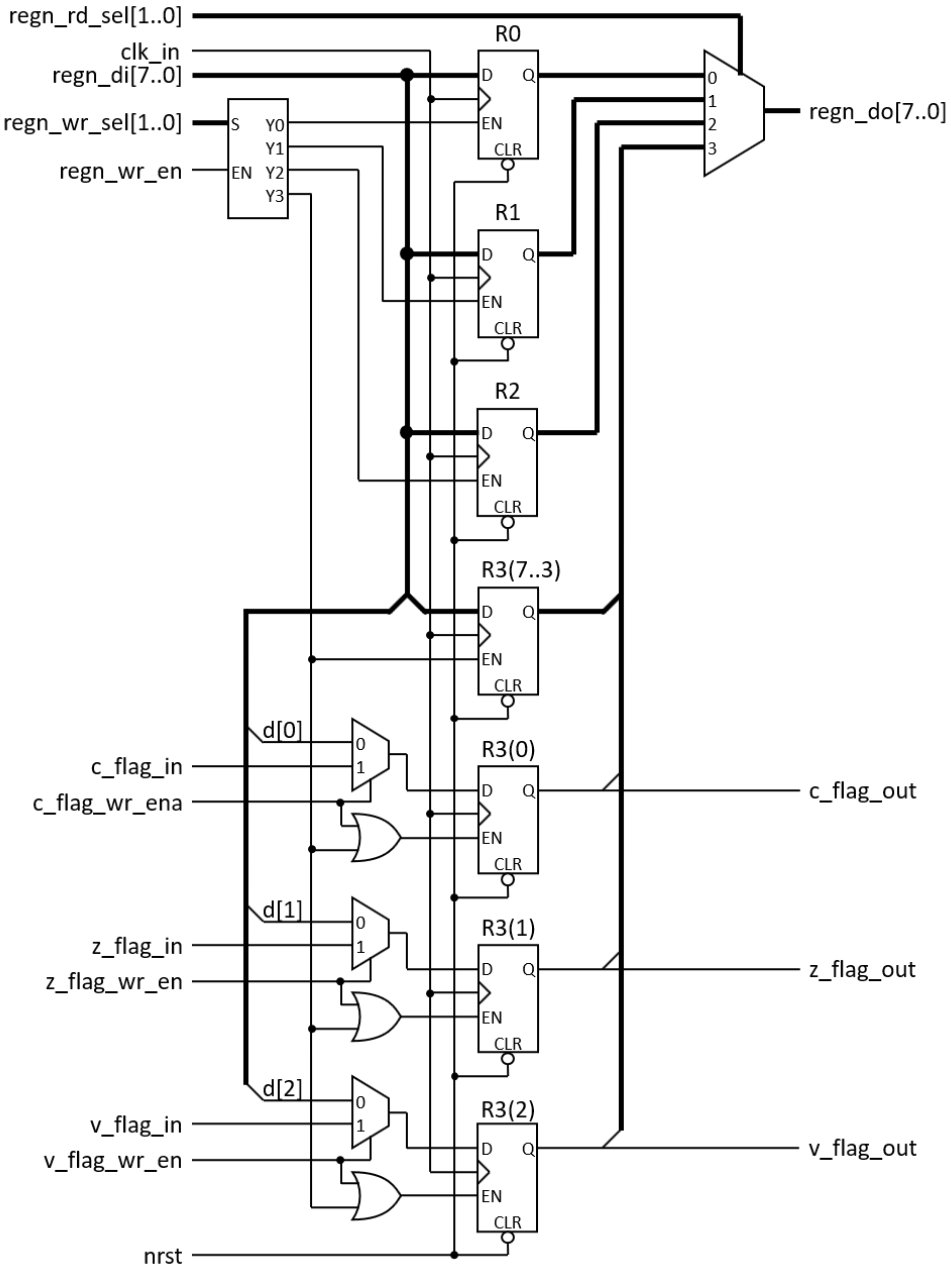
regn_do[7..0] Saída de dados de um dos registradores R0 a R3, selecionados pela entrada reg_rd_sel[1..0]. A saída é em lógica combinacional, não dependendo do *clock* nem de habilitação.

c_flag_out Saída do registrador R3(0) (C), não dependendo do *clock* nem de habilitação.

z_flag_out Saída do registrador R3(1) (Z), não dependendo do *clock* nem de habilitação.

v_flag_out Saída do registrador R3(2) (V), não dependendo do *clock* nem de habilitação.

2.3. Diagrama



3. Wreg

O bloco **Wreg** tem um registrador de 8 bits.

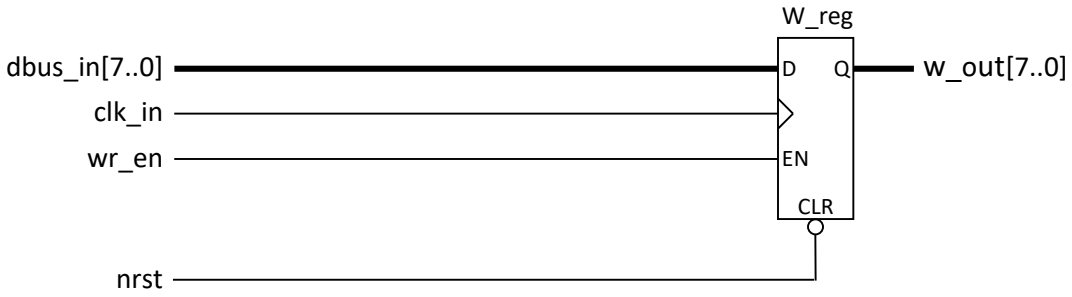
3.1. Entradas

nrst	Entrada de *reset* assíncrono. Quando ativada (nível lógico baixo), todos os bits do registrador deverão ser zerados. Esta entrada tem preferência sobre todas as outras.
clk_in	Entrada de *clock* para escrita no registrador. A escrita acontece na borda de subida do *clock*, desde que habilitada.
d_in[7..0]	Entrada de dados para escrita no registrador, com habilitação através de wr_en.
wr_en	Entrada de habilitação para escrita no registrador. Quando ativada (nível lógico alto), o registrador será escrito, síncrono com clk_in, com o valor de d_in. Caso contrário, nenhuma ação será efetuada.

3.2. Saída

w_out[7..0]	Saída de dados. Essa saída está sempre ativa, não dependendo de habilitação.

3.3. Diagrama



4. Stack

O bloco **stack** tem um conjunto de 8 registradores de 11 bits.

4.1. Entradas

nrst	Entrada de *reset* assíncrono. Quando ativada (nível lógico baixo), todos os bits dos registradores deverão ser zerados. Esta entrada tem preferência sobre todas as outras.
clk_in	Entrada de *clock* para escrita nos registradores. A escrita acontece na borda de subida do *clock*, desde que habilitada.
stack_in[10..0]	Entrada de dados para a pilha.
stack_push	Entrada de habilitação para colocar valores na pilha. Quando ativada (nível lógico alto), na borda de subida do sinal de *clock*, o valor presente na entrada stack_in deve ser escrito no primeiro registrador da pilha. Ao mesmo tempo, o segundo registrador recebe o conteúdo do primeiro, e assim por diante, até o oitavo registrador, que recebe o conteúdo do sétimo registrador.
stack_pop	Entrada de habilitação para retirar valores da pilha. Quando ativada (nível lógico alto), na borda de subida do sinal de *clock*, o conteúdo do segundo registrador deve ser transferido para o primeiro, o do terceiro registrador para a segundo, e assim por diante, até o sétimo

registrador, que recebe o conteúdo do oitavo. Esse, por sua vez, recebe o valor zero (todos os bits iguais a ‘0’).

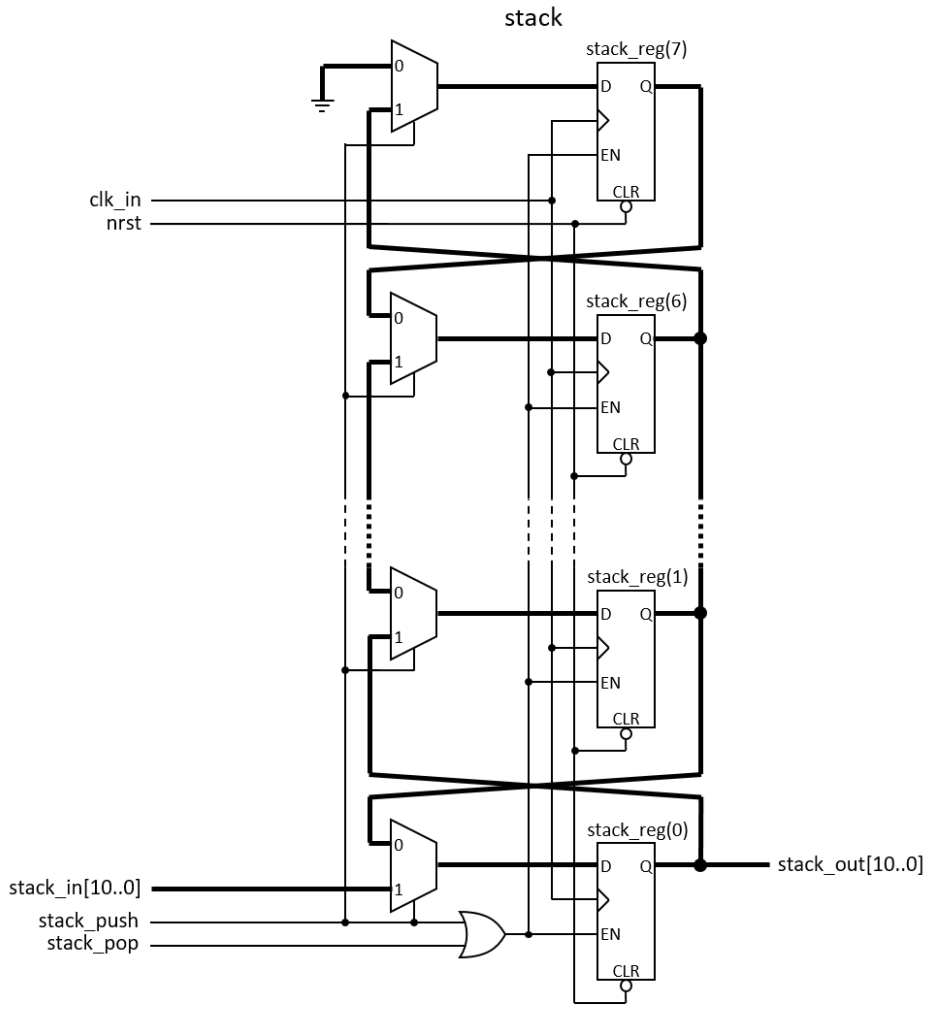
Obs. 1: Normalmente, ao operações de *push* e *pop* não ocorrem simultaneamente. No entanto, para fins de codificação, caso as entradas stack_push e stack_pop sejam ativadas ao mesmo tempo, a operação *push* deve ter preferência.

Obs. 2: Os valores armazenados na pilha através da operação *push* podem ser recuperados através da operação *pop*, na ordem inversa com que foram escritos (primeiro a entrar = último a sair). No entanto, mais de 8 operações *push* em sequência farão com que o conteúdo mais antigo da pilha seja perdido (chamado de *stack overflow*).

4.2. Saídas

stack_out[10..0] Saída correspondente à primeira posição (topo) da pilha. Essa saída está sempre ativa, não dependendo de habilitação

4.3. Diagrama



5. Prog_cnt

Contador síncrono módulo 2048 (de 0 a 2047), com *reset* assíncrono e *load* síncrono.

Esse contador tem duas saídas, sendo que uma corresponde a saída dos flip-flops do contador e outra corresponde à entrada dos flip-flops, ou seja, é o valor a ser carregado nos flip-flops na próxima transição do *clock*.

Existe uma entrada para controle do funcionamento do contador, que seleciona entre o carregamento do valor armazenado na pilha (*stack*), o carregamento de um novo valor para o contador ou o incremento do contador. Quando nenhuma dessas funções estiver ativada, o contador permanece como está.

Para melhor clareza do projeto, recomenda-se que seja codificado em dois *process*, um com a parte combinacional e outro com a parte sequencial.

5.1. Entradas

nrst — Entrada de *reset* assíncrono. Quando ativada (nível lógico baixo), o contador deverá ser zerado. Esta entrada tem preferência sobre todas as outras.

clk_in — Entrada de *clock* para incremento ou carga do contador. O incremento ou a carga acontece na borda de subida do *clock*, desde que habilitado.

pc_ctrl[1..0] — Entrada de controle do funcionamento do contador, de acordo a tabela abaixo. Todos os carregamentos são síncronos com clk_in.

Valor	Função
00	Permanece como está
01	Carrega um novo valor para o contador (entrada new_pc_in)
10	Carrega o valor presente no topo da pilha (entrada from_stack)
11	Incrementa o contador (+1)

new_pc_in[10..0] — Entrada para carga no contador, quando selecionado pela entrada pc_ctrl = “010”.

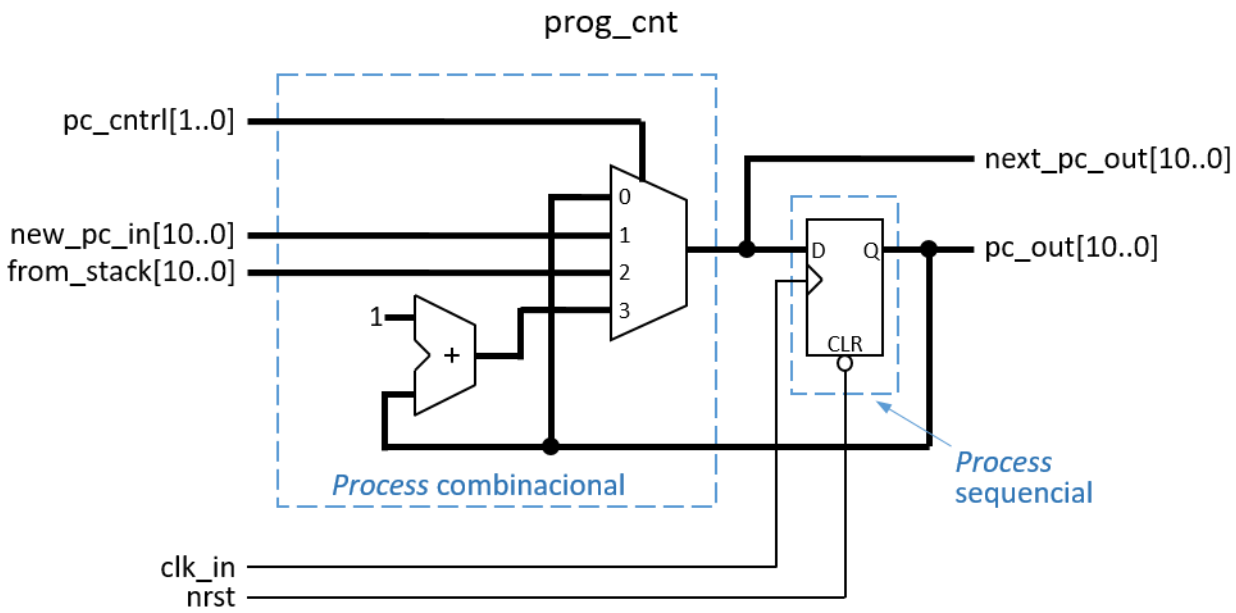
from_stack[10..0] — Entrada para carga no contador, quando selecionado pela entrada pc_ctrl = “001”.

5.2. Saídas

next_pc_out[10..0] — Saída do próximo valor do contador (combinacional)

pc_out[10..0] — Saída do contador

5.3. Diagrama



5.4. Ligação prog_cnt com stack

Ligação a ser feita no quarto trabalho: