Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia de Computação e Automação
DCA0212.2 - Circuitos Digitais
Docente: Tiago Barros

Laboratório 04 - Registradores, contadores e bancos de registradores

Objetivos

1. Experimentar a descrição em VHDL de registradores, contadores e bancos de registradores;
2. Reforçar os conceitos de latches, flip-flops, registradores e contadores assíncronos e síncronos;
3. Pôr em prática conceitos aprendidos na disciplina Circuitos Digitais - Teoria.

Introdução

Na aula de hoje serão apresentados os blocos registradores. Os registradores são construídos a partir de latches ou flip-flops. Nesta aula, abordaremos os latches SR e D e os flip-flops SR e D. Os registradores abordados na aula serão projetados com o uso de latches D e flip-flops D. Além disso, serão apresentados os blocos contadores. Os contadores são divididos em contadores síncronos e assíncronos. Os contadores síncronos são aqueles em que a mudança do seu estado ocorre em cada pulso de clock, enquanto os contadores assíncronos não possuem uma entrada de clock. Nessa aula, trataremos apenas dos contadores síncronos, sendo apresentados os contadores crescentes e decrescentes. Por fim, veremos a construção de bancos de registradores, os quais são componentes de memória utilizados em blocos operacionais de circuitos digitais.

Circuitos Sequenciais x Circuitos Combinacionais

A saída de um circuito combinacional é uma função que depende apenas das entradas atuais do circuito. Um circuito combinacional não tem memória – não podemos armazenar bits em um circuito combinacional e depois ler esses bits. Os circuitos combinacionais em si são muito limitados em sua utilidade. Por outro lado, usualmente os projetistas usam os circuitos combinacionais como parte de circuitos maiores, chamados circuitos sequenciais – circuitos que realmente têm memória. Um circuito sequencial é um circuito cujas saídas dependem não somente das entradas atuais, mas também do seu estado atual, que é o conjunto de todos os bits armazenados no circuito. O estado do circuito, por sua vez, depende da sequência passada de valores que apareceram nas entradas do circuito.

O VHDL é uma linguagem não-sequencial, portanto, faz-se necessário uma estrutura que o torne sequencial, para auxiliar na implementação de circuitos sequenciais. A estrutura que permite isto é o PROCESS, cuja sintaxe pode ser vista no Quadro 1.

```
PROCESS (A, B); --entre parenteses estao os sinais da lista de sensibilidade
BEGIN
   --comandos
END PROCESS;
```

Quadro 1 – Sintaxe da estrutura PROCESS da linguagem VHDL.

Conceito importante: O trecho entre BEGIN e END é executado sequencialmente (a ordem em que os códigos estão apresentados importa). Por sua vez, o PROCESS é executado concorrentemente com as demais declarações. O PROCESS é invocado quando há uma mudança em algum sinal de sua lista de sensibilidade.

Uma vez definida a estrutura que executará um código sequencial em VHDL, precisamos definir quais comandos executados dentro desta estrutura que serão utilizados na implementação de um circuito sequencial. Uma forma de fazer isso é com o uso de outra estrutura do VHDL: a estrutura IF-ELSE-ELSIF, a qual é executada de forma sequencial, podendo somente ser utilizada em regiões de código sequencial. Funciona de forma semelhante à estrutura padrão de IF de linguagens de alto nível, obedecendo a sintaxe ilustrada no Quadro 2.

```
IF (condicao1) THEN
   --Comandos caso essa condição seja verdadeira
ELSIF (condicao2) THEN
   --Comandos caso essa condição seja verdadeira
ELSE
   --Comandos caso nenhuma condição anterior seja verdadeira

END IF;
```

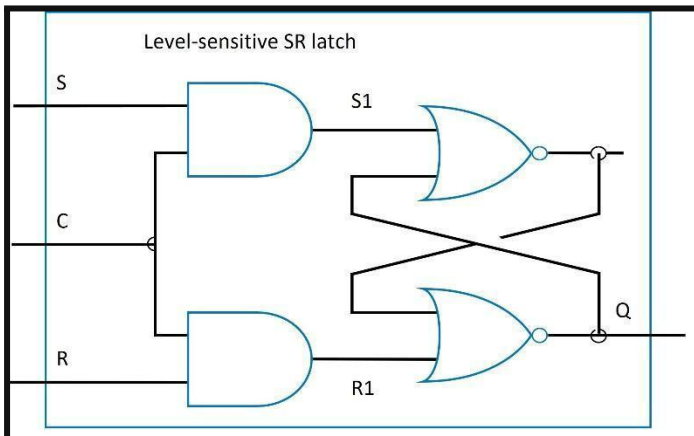
Quadro 2 – Sintaxe da estrutura IF-ELSE-ELSIF da linguagem VHDL.

Latches

Um latch é um dispositivo de armazenamento temporário que tem dois estados estáveis. Os latches são similares aos flip-flops por serem dispositivos biestáveis, ou seja, podem permanecer em um de dois estados estáveis usando uma configuração de realimentação, na qual as saídas são ligadas nas entradas opostas. A principal diferença entre os latches e os flip-flops é o método usado para a mudança de estado.

Latch SR (sensível ao nível) - O circuito consiste em um par de portas NOR acopladas, conforme ilustrado na Figura 1. Sempre que a entrada C for igual a ‘0’ (nível baixo), o circuito mantém o estado atual, independente dos valores das entradas S e R. Quando a entrada C for igual a ‘1’ (nível alto), o circuito sofrerá uma alteração de seu valor interno: se a entrada S for igual a ‘1’, a saída Q será igual a ‘1’; se a entrada R for igual a ‘1’, a saída Q será igual a ‘0’. Portanto, dizemos que, para um nível alto de C, a saída Q é setada para um nível alto se S for igual a ‘1’; por sua vez, se a entrada R for igual a ‘1’, a saída Q é

ressetada para um nível baixo. Este comportamento está ilustrado na tabela verdade da Figura 1. No Quadro 3, podemos observar a descrição VHDL do latch SR, a qual emprega as estruturas PROCESS e IF-ELSE-IF. Por fim, a Figura 2 ilustra as formas de onda obtidas através da simulação da descrição VHDL do latch SR.



C	S	R	Q
0	0	0	Q
0	0	1	Q
0	1	0	Q
0	1	1	Q
1	0	0	Q
1	0	1	0
1	1	0	1
1	1	1	X

Figura 1: Circuito e tabela verdade do latch SR.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
ENTITY latchSR IS
PORT(S, R, c : IN BIT;
                Q : OUT BIT);
END;
ARCHITECTURE behav OF latchSR IS
BEGIN
PROCESS (S, R, c)
BEGIN
   IF (c = '1' AND S = '1') THEN
      Q<='1';
   ELSIF (c = '1' AND R = '1') THEN
      Q<='0';
   END IF;
END PROCESS ;
END;
```

Quadro 3: Descrição de um latch SR sensível ao nível.

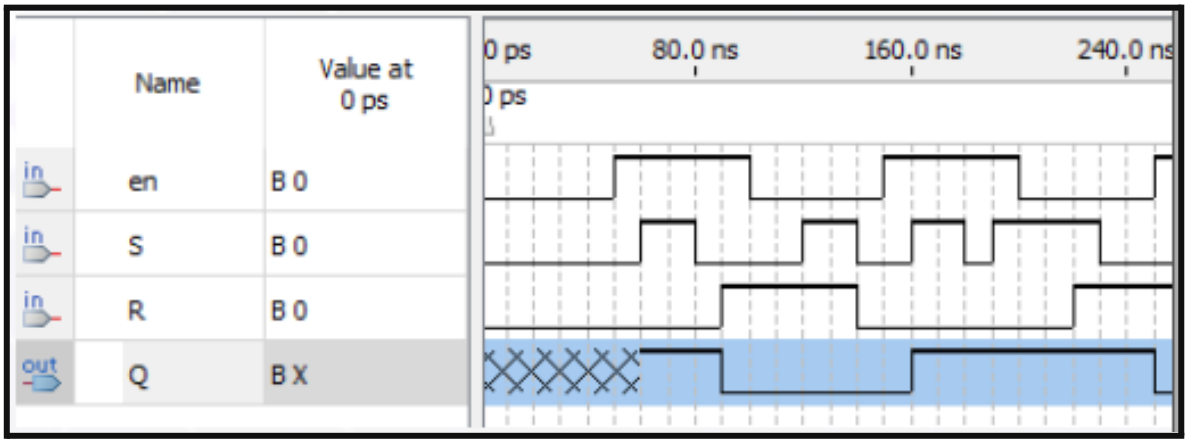
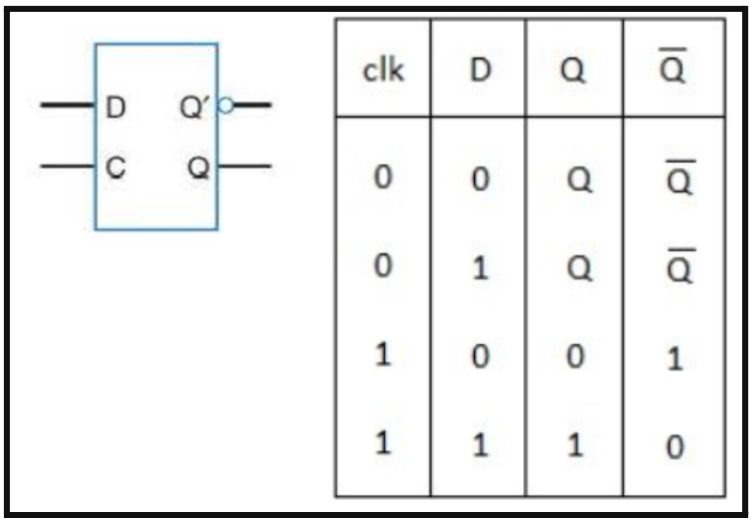


Figura 2: Formas de onda obtidas pela simulação da descrição VHDL do latch SR.

Latch D (sensível ao nível) - Uma das principais dificuldades que projetistas de circuitos digitais encontram, ao utilizar latches SR, é a necessidade de garantir que as entradas S e R nunca sejam iguais a '1' simultaneamente, uma vez que a entrada de ativição (C) seja igual a '1'. O latch D sensível ao nível, ilustrado na Figura 3, é um dispositivo proposto para lidar com este problema. Se a entrada C do latch D for igual a '0', a saída Q do latch D não será modificada, ou seja, seu estado interno é armazenado. Por outro lado, se a entrada C do latch D for igual a '1', a saída Q do latch será igual ao valor da entrada D: o latch é setado se D='1' e ressetado se D='0'.



clk	D	Q	$\overline{Q}$
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$
1	0	0	1
1	1	1	0

Figura 3: Representação em bloco e tabela verdade do latch D.

Flip-flops

Um problema dos latches é que se o valor da entrada D mudar enquanto o sinal C está ativado, ele ainda é levado em consideração. Para evitarmos este problema, utilizamos

os dispositivos flip-flops, os quais só registram o bit quando há mudanças de nível em sua entrada C (também chamada de *clock,* ou CLK).

Flip-flop SR - O flip-flop SR funciona de forma semelhante ao latch SR, porém, ao invés de ser sensível ao nível da entrada C (ou CLK), ele é sensível a mudança do nível da entrada C de um valor baixo (C='0') para um valor alto (C='1') ou vice-versa. Quando o flip-flop SR é sensível a uma mudança de nível baixo para alto, dizemos que ele é sensível à borda de subida de C. Neste caso, quando temos uma borda de subida em C, a saída Q será igual a '1' se a entrada S for igual a '1' e a saída Q será igual a '0' se a entrada R for igual a '1'. Na linguagem VHDL, modelamos a mudança de nível utilizando o comando EVENT. Portanto, uma borda de subida de uma varíavel clk é modelada como: clk ' EVENT AND clk = '1'. O Quadro 1 apresenta a descrição VHDL de um flip-flop SR e a Figura 4 ilustra as formas de onda obtidas pela simulação da descrição VHDL do flip-flop SR.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;
ENTITY flipflopSR IS
PORT(S, R, clk : IN BIT;
                 Q : OUT BIT);
END;
ARCHITECTURE behav OF flipflopSR IS
BEGIN
PROCESS (S, R, clk)
BEGIN
   IF (clk ' EVENT AND clk = '1' AND S = '1') THEN
      Q<='1';
   ELSIF (clk ' EVENT AND clk = '1' AND R = '1') THEN
      Q<='0';
   END IF;
END PROCESS ;
END;
```

Quadro 4: Descrição de um flip-flop SR.

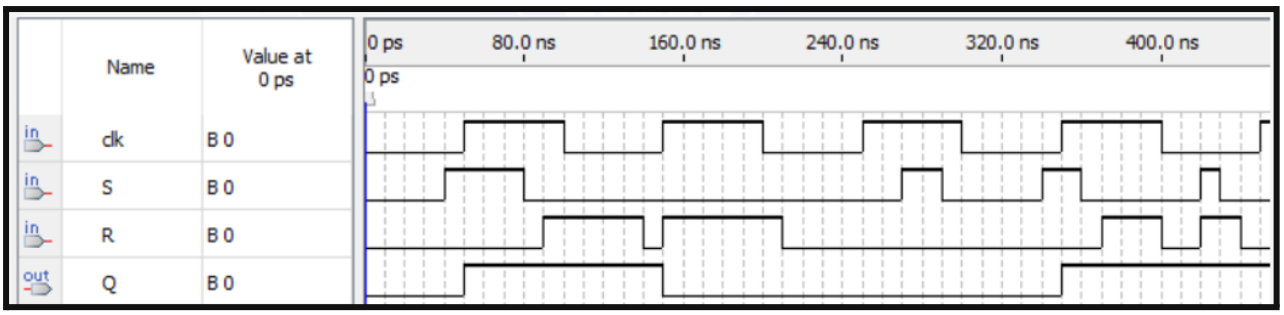
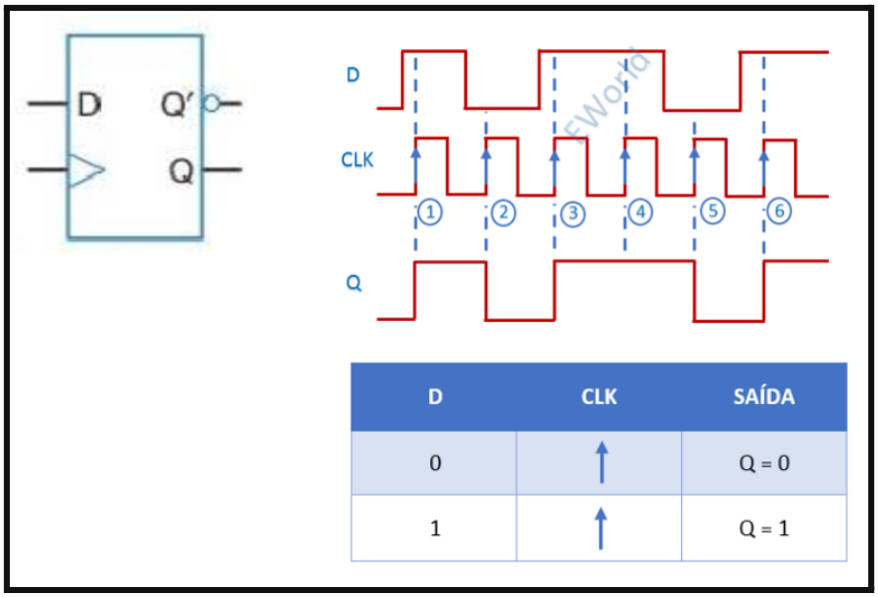


Figura 4: Formas de onda obtidas pela simulação da descrição VHDL do flip-flop SR.

Flip-flop D - O flip-flop D é um dispositivo sensível à mudança de nível de sua entrada C (ou CLK). No caso em que o flip-flop D é sensível a uma borda de subida (mudança de CLK='0' para CLK='1'), a sua saída Q será igual a entrada D sempre que houver uma borda de subida em CLK. Em todos os outros casos, a saída Q não sofrerá alterações, ela apenas armazenará seu estado anterior. A Figura 5 ilustra a representação em blocos do flip-flop D; suas formas de onda, contendo as entradas D e CLK e a saída Q; e sua tabela verdade.



D	CLK	SAÍDA
0	↑	Q = 0
1	↑	Q = 1

Figura 5: Representação em bloco, formas de onda e tabela verdade do flip-flop D.

Registradores

Registradores são componentes sequenciais usados para armazenar múltiplos bits. Um registrador básico é construído empregando uma sequência de latches ou flip-flops. Por possuírem um estado em que a saída é indesejável (estado em que SR = 11), latches e flip-flops SR não são usados para criar registradores. Em geral, são usados latches ou flip-flops D. Na Figura 6, podemos observar um registrador construído com flip-flops D, utilizado para armazenar 4 bits. Sempre que houver uma borda de subida na entrada CLK, todos os 4 flip-flops irão carregar, simultaneamente, suas entradas I0, I1, I2 e I3.

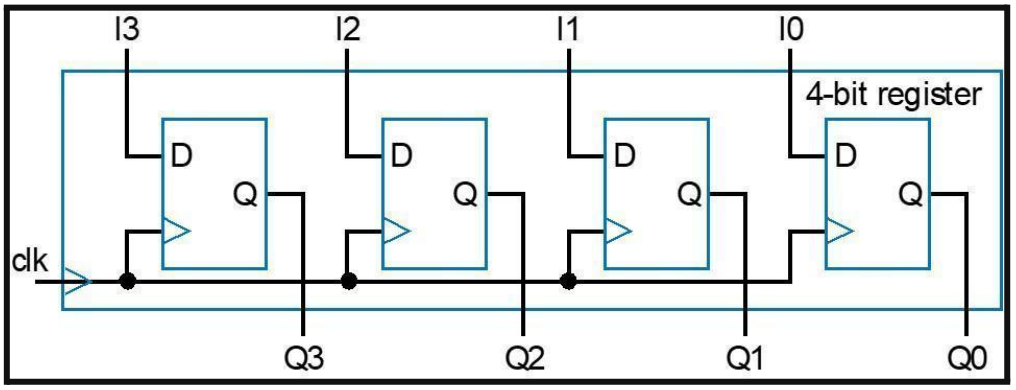


Figura 6: Registrador de 4 bits contruído a partir de flip-flops D.

Contadores

Um contador de *N* bits é um componente que pode incrementar ou decrementar o próprio valor a cada ciclo de relógio, quando uma entrada de habilitação é 1.

Incrementar significa adicionar 1 a sua contagem e decrementar significa subtrair 1 a sua contagem. Um contador que pode incrementar é conhecido como ***contador crescente***, um contador que pode decrementar é conhecido como ***contador decrescente*** e um contador que pode incrementar e decrementar é conhecido como ***contador crescente/decrescente***.

Um contador crescente de quatro bits produz a seguinte sequência: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 0000, 0001, etc. Note que quando o contador atinge o seu valor mais alto (1111), ele retorna para 0.

De forma similar, um contador decrescente dá uma volta completa quando chega a 0 e, em seguida, indo para o valor mais elevado. Na Figura 7 é apresentado o diagrama de blocos de um contador crescente de 4 bits. Quando cnt=1, o contador incrementa seu próprio valor a cada ciclo de relógio. Quando cnt=0, o contador mantém o seu valor atual.

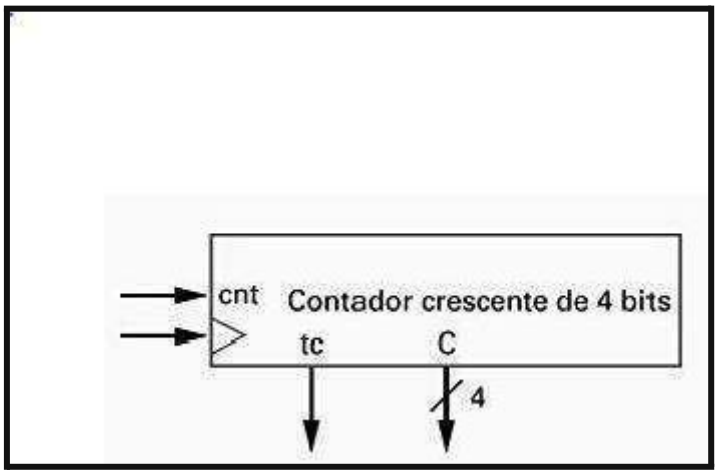


Figura 7: Diagrama de blocos de um contador crescente de 4 bits.

Contador crescente

Um contador crescente pode ser implementado utilizado registradores. A Figura 8 apresenta um exemplo de implementação, o qual utiliza um bloco incrementador para somar 1 ao valor atual do registrador. Quando cnt=0, o registrador mantém o seu valor atual e, quando cnt=1, o valor corrente no registrador é somado de 1.

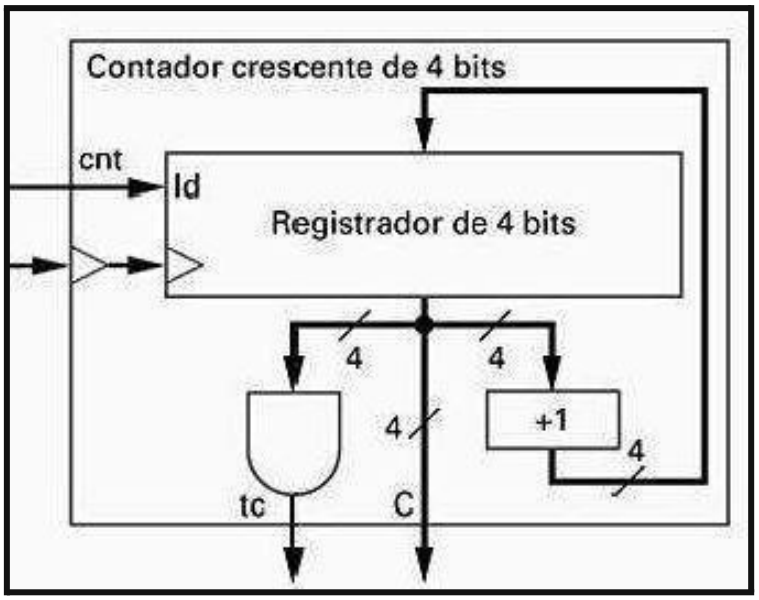


Figura 8: Contador síncrono crescente.

A tabela-verdade de um contador crescente é apresentada na Figura 9. Note que as saídas s0, s1, s2 e s3 assumem os valores dos bits de entrada. Quando o contador chega em sua contagem final, as suas saídas retornam para 0.

Entradas				Saídas				
a3	a2	a1	a0	c0	s3	s2	s1	s0
0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	0	1	1	0	0	1	0	0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	1	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	1	0
1	0	1	0	0	1	0	1	1
1	0	1	1	0	1	1	0	0
1	1	0	0	0	1	1	0	1
1	1	0	1	0	1	1	1	0
1	1	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0

Figura 9: Tabela-Verdade de um contador crescente (note que a coluna *c0* da tabela diz respeito ao *carry-out* do contador, que pode ou não estar presente no circuito).

Contador decrescente

Um contador decrescente pode ser projetado de forma semelhante a um contador crescente, substituindo o incrementador por um decrementador. Na Figura 10 é apresentada a estrutura de um decrementador de 4 bits.

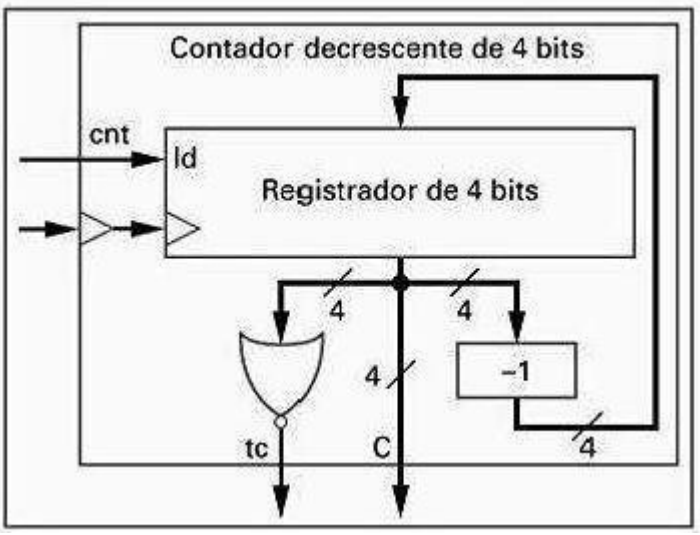


Figura 10: Estrutura de um contador decrescente de 4 bits.

Contador com carga (*load*) paralela:

Frequentemente, os contadores vêm com a capacidade de inicialização do valor de contagem. Isto é conseguido carregando-se o registrador do contador com dados em paralelo. A Figura 11 mostra o projeto de um contador crescente de quatro bits com carga paralela. Quando a entrada de controle *ld* é 1, o multiplexador 2 x 1 deixa passar a entrada L com os dados a serem carregados; quando *ld* é 0, o multiplexador deixa passar o valor incrementado. Além disso, fazemos uma operação OR com os sinais *ld* e *cnt* do contador para gerar o sinal de carga do registrador. Quando *cnt* é 1, o valor incrementado é carregado. Quando *ld* é 1, os dados da carga paralela são carregados. Mesmo que *cnt* seja 0, *ld* = 1 fará o registrador ser carregado. De modo semelhante, um contador decrescente ou um crescente/decrescente poderia ser ampliado para permitir carga paralela.

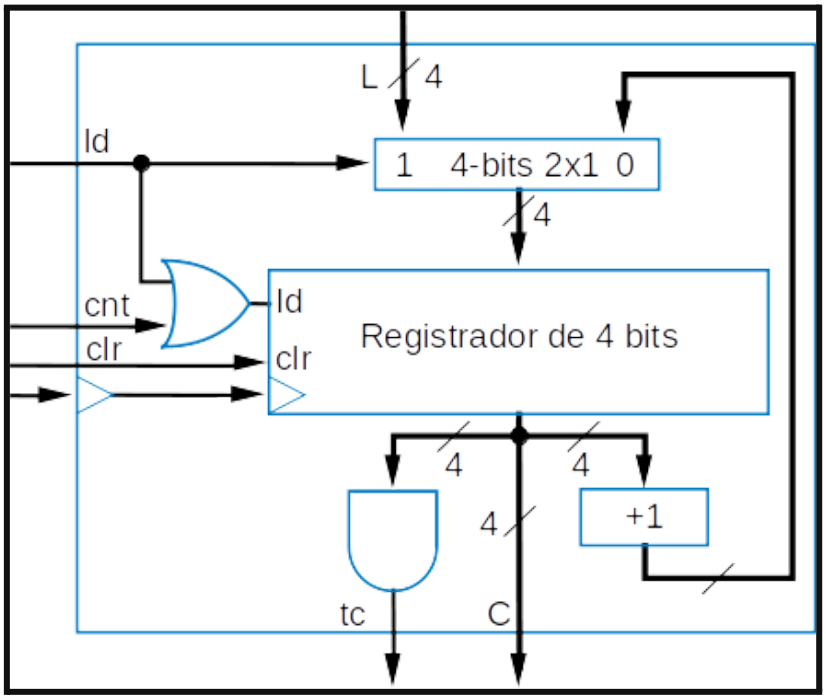


Figura 11: Estrutura de um contador crescente de 4 bits com carga paralela.

O Quadro 5 apresenta uma descrição comportamental de um contador com carga em paralelo, descrita em VHDL. Note que esta descrição faz uso da estrutura IF-ELSE-ELSIF da linguagem VHDL, a qual se trata de uma estrutura de execução sequencial e, portanto, deve ser descrita dentro da estrutura PROCESS da linguagem VHDL.

```
ENTITY contador IS
PORT(
  clk:    IN BIT; --entrada de clock
  ld:     IN BIT; --carrega os dados
  reset: IN BIT;
  data:  IN INTEGER RANGE 15 DOWNTO 0; -- entrada de dados
  q:      OUT INTEGER RANGE 15 DOWNTO 0); --saída de dados
END contador;
```

```
ARCHITECTURE comportamento OF contador IS
  BEGIN PROCESS(clk,reset)
    VARIABLE qv: INTEGER RANGE 15 DOWNTO 0; --variável para a saida

    BEGIN
      IF(reset = '1') THEN
        qv := 0;
      ELSIF(clk ' event and clk = '1') THEN
        IF(ld = '1') THEN
          qv := data;
        ELSE
```

```
        IF(qv >= 9) THEN
          qv := 0;
        ELSE
          qv := qv + 1;
        END IF;
      END IF;
    END IF;
    q <= qv;
  END PROCESS;
END;
```

Quadro 5: Descrição comportamental de um contador de 4 bits com carga paralela.

Bancos de Registradores:

Um banco de registradores (ou registrador de arquivos) ***M x N*** é um componente de memória de blocos operacionais (*datapath*) de um circuito digital que propicia um acesso eficiente a um conjunto de *M* registradores, de forma que cada registrador possui largura de *N* bits. A Figura 12 ilustra o símbolo para diagrama de blocos de um banco de registradores.

Figura 12: Símbolo para diagrama de blocos de um banco de registradores.

Para se escrever um valor no banco de registradores, coloca-se o dado a ser escrito em *W_data*. Para se indicar em qual dos registradores do banco de registradores o dado será escrito, coloca-se o endereço do registrador na entrada *W_addr*. Para habilitar uma escrita, em sincronismo com o *clock*, coloca-se *1* em *W_en*. O conjunto de entradas *W_data*, *W_addr* e *W_en* é conhecido como portas de escrita do banco de registradores.

No processo de leitura, especifica-se o endereço do registrador a ser lido na entrada *R_addr* e habilita-se a leitura fazendo *R_en = 1*. Esses valores farão com que o banco de registradores coloque na saída *R_data* o conteúdo do registrador que foi endereçado. O conjunto de *R_addr*, *R_en* e *R_data* é conhecido como portas de leitura de um banco de

registradores. As portas de leitura e escrita são independentes entre si, podendo-se escrever em um registrado e ler de outro (ou do mesmo) registrador simultaneamente. A Figura 13 ilustra o projeto interno de um banco de registradores 4x32.

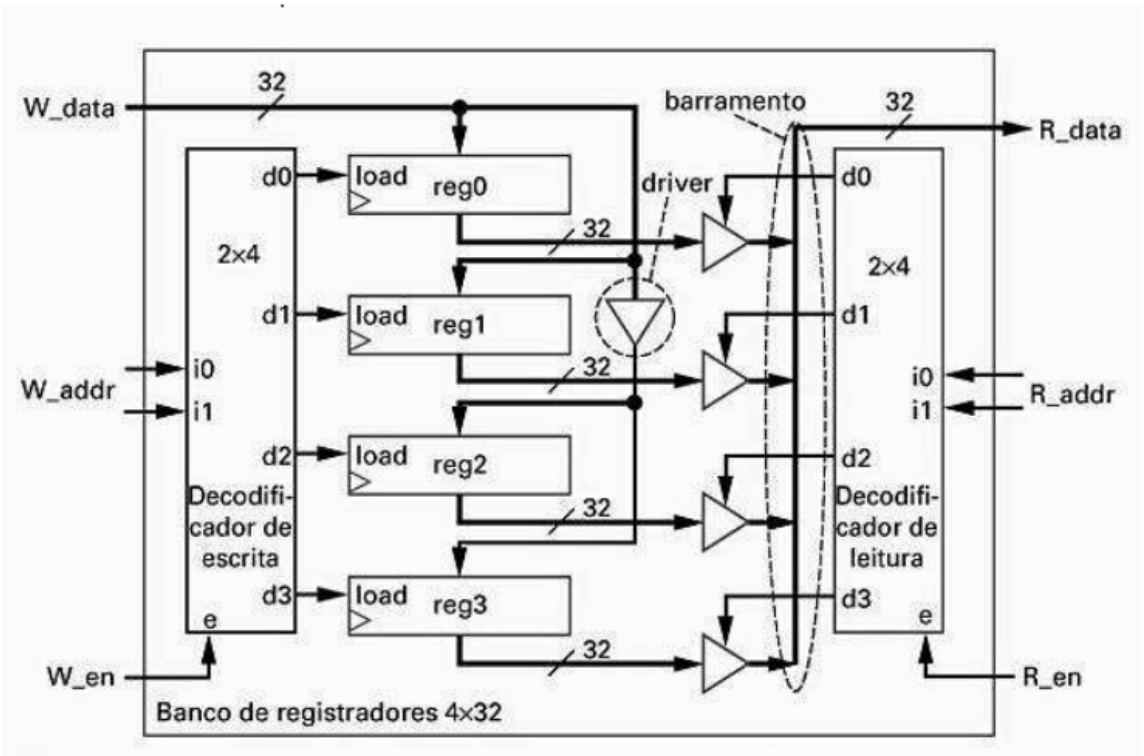


Figura 13: Projeto interno de um possível banco de registradores 4x32.

No projeto interno do banco de registradores da Figura 13, tem-se decodificadores de endereços de escrita e leitura; registradores formados por *flip-flops*, drivers e drivers de três estados, sendo estes dois últimos ilustrados na Figura 14. O driver é um componente cuja saída é igual à entrada, porém amplificada, no intuito de evitar o problema de *fanout*, o qual ocorre quando uma corrente se divide por muitos nós e barramentos, tendo sua amplitude atenuada. O driver de três estados, por sua vez, possui comportamento definido por sua entrada de controle, *c*. Quando *c = 1*, sua saída é igual à sua entrada; porém, quando *c = 0*, a saída do driver não é *0* nem *1*, mas está no estado de alta impedância, o qual é escrito como *'Z'*.

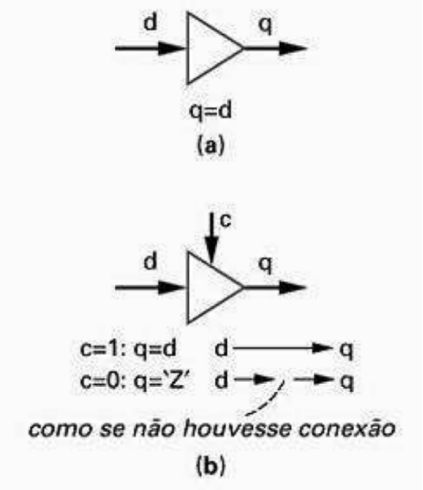


Figura 14: Driver (a) e driver de três estados (b).

Atividades:

Entregar um relatório descrevendo a execução das tarefas 1, 2 e 3. O relatório deve conter todos os códigos VHDL, formas de onda das simulações e explicações dos resultados verificados nas formas de onda.

1) **Registradores:**
   - **a)** Utilizando a linguagem VHDL, crie um latch D para 1 Bit.
   - **b)** Utilizando a linguagem VHDL, desenvolva um flip-flop D para 1 bit.
   - **c)** Utilizando a linguagem VHDL, desenvolva um registrador de quatro bits, baseado em latches D. Utilize componentes que contenham latches de 1 bit.
   - **d)** Utilizando a linguagem VHDL, desenvolva um registrador de quatro bits, baseado em flip-flops D. Utilize componentes que contenham flip-flops de 1 bit.

ATENÇÃO: Utilize como base a Figura 6 para a criação dos registradores. NÃO UTILIZEM BIT_VECTOR.

2) **Contadores:**
   - **a)** Implemente um contador crescente de 4 bits que conte até sua última contagem e retorne ao seu valor inicial. Adicione o sinal indicativo de término de contagem: tc=1 quando o contador atingir 1111 e tc=0 caso contrário.
   - **b)** Implemente um contador decrescente de 4 bits. Adicione o sinal indicativo de término de contagem: tc=1 quando o contador atingir 0000 e tc=0 caso contrário.

3) **Bancos de registradores:**
   - **a)** Implemente um banco de registradores 4x8, baseado na Figura 13. Siga as seguintes instruções:
     - **i)** Desconsidere o fênomeno de *fanout* (ou seja, não é necessário implementar o driver simples, de uma entrada e uma saída).
     - **ii)** Utilize *flip-flops* para formar os registradores.
     - **iii)** Implemente os drivers de três estados usando descrição comportamental.