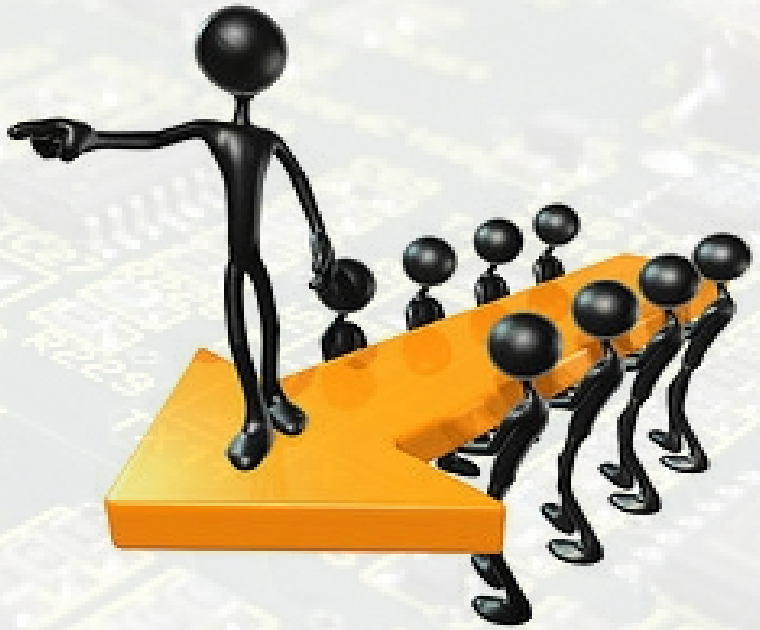
Sistemas Digitais

Aulas práticas de laboratório

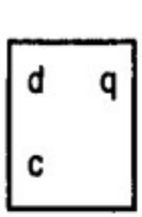
Circuitos sequenciais no VHDL

Prof. Dr. Eng. Rafael Luiz Cancian

Introdução

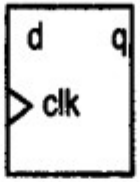


Elementos Básicos de Memória



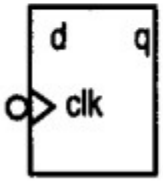
c	q*
0	q
1	d

(a) D latch



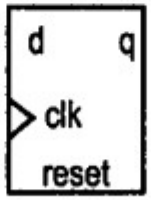
clk	q*
0	q
1	q
↓	d

(b) positive-edge-triggered D FF



clk	q*
0	q
1	q
↓	d

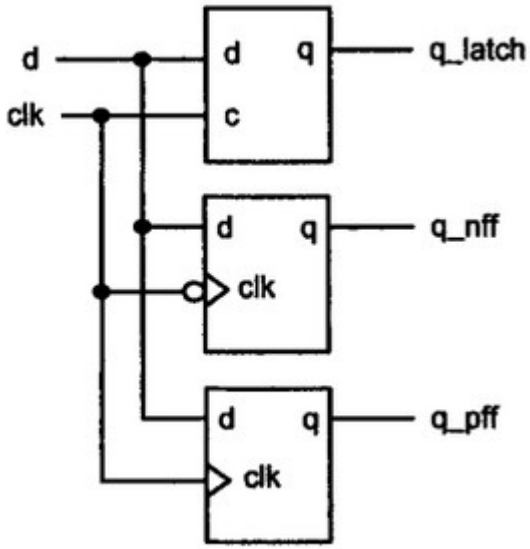
(c) negative-edge-triggered D FF

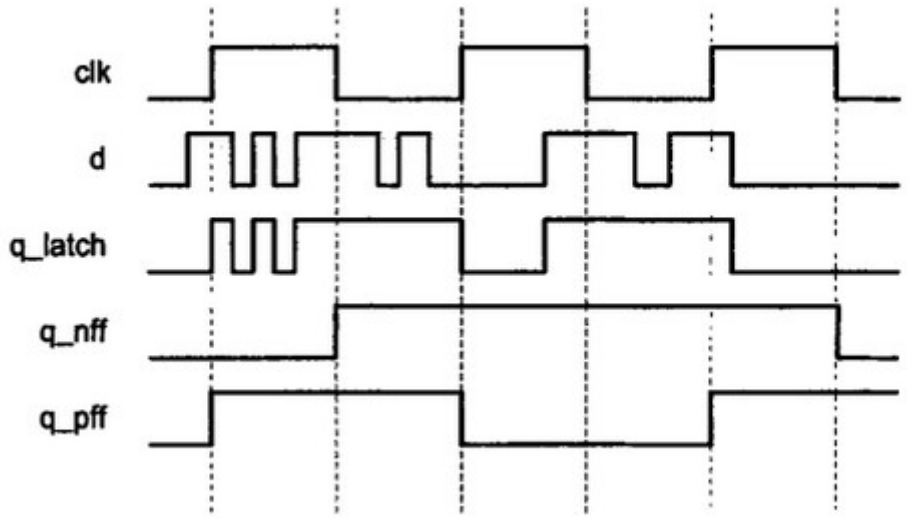


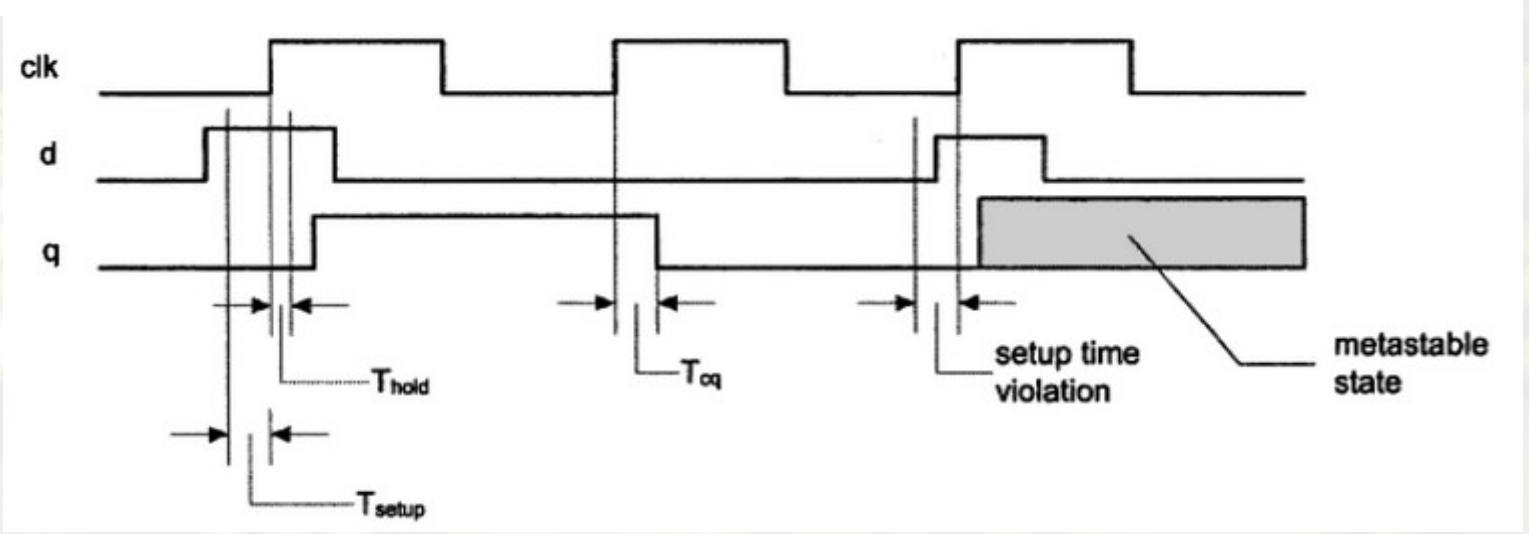
reset	clk	q*
1	-	0
0	0	q
0	1	q
0	↓	d

(d) D FF with asynchronous reset

Elementos Básicos de Memória







Circuitos Síncronos e Assíncronos

- Há três tipos de circuitos sequenciais:
  - Circuitos globalmente síncronos (ou apenas síncronos): Usam FF como elementos de memória e todos os FF são controlados (sincronizados) por um único sinal global de clock. São o tipo mais importante de circuitos sequenciais para projetos complexos.
  - Circuitos assíncronos localmente síncronos: Um circuito composto por vários subsistemas. Cada subsistema é internamente síncrono e usa seu próprio clock, mas os subsistemas (e o circuito de forma global) são assíncronos entre si.
  - Circuitos globalmente assíncronos: Não usam qualquer sinal de clock para coordenador operações nos elementos de memória.

Conteúdo

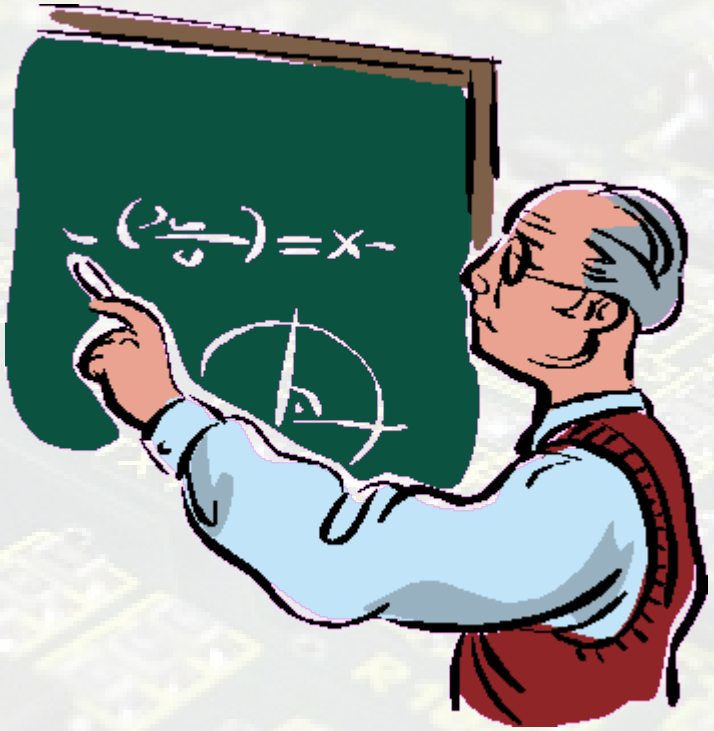
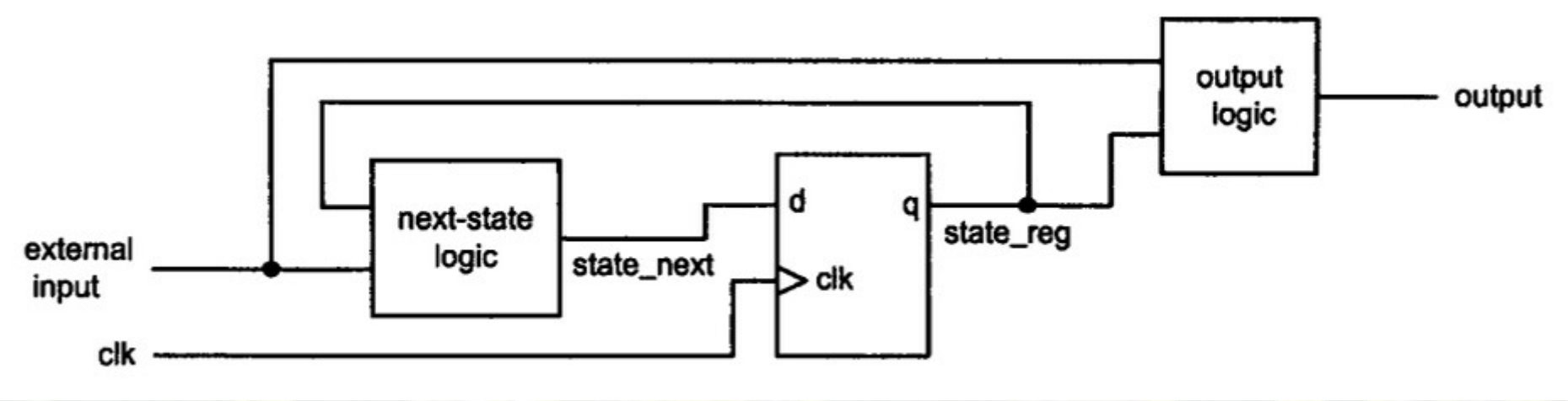


Diagrama Conceitual de TODOS os Circuitos Sequenciais

- TODOS os circuitos sequenciais seguem exatamente o mesmo diagrama conceitual.



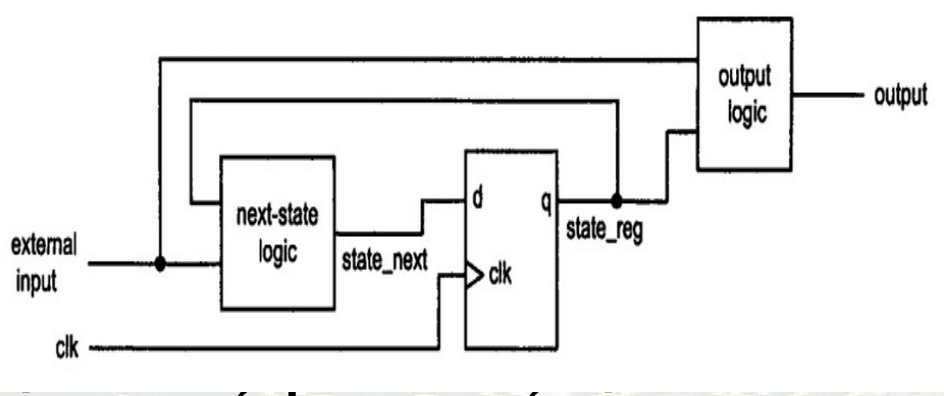
- Pode haver um "*Output Logic*" para cada saída do circuito.
- Quando uma saída depende apenas de "state_reg" ela é dita "Saída de Moore", senão é uma "Saída de Mealy".

Descrição de Circuitos Sequenciais

- Os componentes "*Next-State Logic*" e "*Output Logic*" são combinacionais e devem ser descritos separadamente.
- Por serem combinacionais, esses elementos devem ser descritos usando apenas comandos concorrentes do VHDL.
  - A única exceção é quando a lógica (comportamento) do componente é complexa de mais para ser descrita num comando concorrente com poucas linhas de código.
- O registrador (elemento de memória) é a única parte realmente "sequencial" do circuito, e é descrito usando um *process*.

Descrição de Circuitos Sequenciais

- Toda “Next-State Logic” tem como entradas as entradas do circuito e o estado atual do circuito (*state_reg*) e tem como única saída o próximo estado (*state_next*).



- Os únicos sinais internos de qualquer circuito sequencial são o estado atual e o próximo estado (*state_reg e state_next*).
- Cada saída tem sua própria “*Output Logic*”, que pode depender apenas do estado atual ou também de alguma entrada do circuito (saída assíncrona).

Descrição de Circuitos Sequenciais

- Base para qualquer circuito sequencial (um registrados de *"width"* bits):

```
ibrary ieee;
use ieee.std_logic_1164.all;

entity registerN is
        generic(  width: positive;
                  resetValue: integer := 0 );
        port(     -- control
                  clock, reset, load: in std_logic;
                  -- data
                  input: in std_logic_vector(width-1 downto 0);
                  output: out std_logic_vector(width-1 downto 0));
end entity;
```

sempre parametrizável,visando reuso. Não usar valores default (ao menos nos primeiros primeiros parâmetros), evitando uso como top-level e forçando generic map na instanciação.

separar sinais de controle e de dados

Descrição de Circuitos Sequenciais

```
use ieee.numeric_std.all;

architecture behav0 of registerN is
subtype state is std_logic_vector(width-1 downto 0);
signal currentState, nextState: state;
begin
    -- next-state logic
    nextState <= input when load='1' else currentState;
    -- memory element
    process(clock, reset) is
    begin
        if reset='1' then
            currentState <= std_logic_vector(to_signed(resetValue, currentState'length));
        elsif rising_edge(clock) then
            currentState <= nextState;
        end if;
    end process;
    -- output logic
    output <= currentState;
end architecture;
```

a declaração deve ter apenas a especificação do tipo do "estado" e apenas dois sinais internos: o estado atual e o próximo estado. Só.

a lógica de próximo estado é combinacional e sempre atribui algo para o próximo estado, com base nas entradas e no estado atual

o registrador (elemento de memória) nunca muda (a não ser eventualmente a atribuição no estado de reset) e não deve ter mais nenhuma lógica (vai para a NS logic)

uma lógica de saída que atribui algo para uma saída, com base apenas no estado atual (moore) e possivelmente entradas (mealy)

Exemplos

- Contador de Módulo

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;

entity moduleCounter is
	generic( module: positive;
		 resetValue: integer := 0 );
	port(	 -- control
		 clock, reset, load: in std_logic;
		 enb: in std_logic;
		 -- data
		 input: in std_logic_vector(integer(ceil(log2(real(module))))-1 downto 0);
		 output: out std_logic_vector(integer(ceil(log2(real(module))))-1 downto 0)	);
end entity;
```

Exemplos

```
use ieee.numeric_std.all;

architecture behav0 of moduleCounter is
subtype state is unsigned(integer(ceil(log2(real(module))))-1 downto 0);
signal nextState, currentState: state;
begin
        -- next-state logic
        nextState <=    unsigned(input) when load='1' else
                        currentState when enb='0' else
                        to_unsigned(0, currentState'length) when currentState=module-1 else
                        currentState+1;
        -- memory element
        process(clock, reset) is
        begin
                if reset='1' then
                        currentState <= (to_unsigned(resetValue, currentState'length));
                elsif rising_edge(clock) then
                        currentState <= nextState;
                end if;
        end process;
        -- output logic
        output <= std_logic_vector(currentState);
end architecture;
```

num contador, o tipo escolhido para o estado foi “unsigned”

lógica ainda suficientemente simples para que seja usado um único comando concorrente.

o registrador (memória) não foi alterado.

Exemplos

- Uma FSM (Finite State Machine) é apenas um circuito sequencial como qualquer outro
  - Porém, costuma ter a lógica de próximo estado (e possivelmente também a lógica de saída) mais complexa, de modo que esses elementos podem vir a ser descritos usando comandos sequenciais.

SA: Sensor portão Aberto
SF: Sensor portão Fechado
SO: Sensor Obstáculo no portão
CR: Controle Remoto acionado
MT: Motor (00 | 01: desligado;
01: fechando;
10: abrindo)

```
library ieee;
use ieee.std_logic_1164.all;

entity PortaoGaragem is
	port(-- controle
		clock, reset: in std_logic;
		– dados
		SA, SF, SO, CR: in std_logic;
		MT: out std_logic_vector(1 downto 0) );
end;
```

Exemplos

```
...
-- lógica de proximo estado
process(estadoAtual,SA, SF, SO, CR) is
begin
        proximoEstado <= Fechando;
        case estadoAtual is
                when Fechado =>
                        if CR='1' then proximoEstado <= Abrindo;
                        eles proximoEstado <= Fechado;
                        end if;
                when Abrindo =>
                        if SA='1' then proximoEstado <= Aberto;
                        eles proximoEstado <= Abrindo;
                        end if;
                when Aberto =>
                        if CR='1' then proximoEstado <= Fechando;
                        else proximoEstado <= Aberto;
                        end if;
                when Fechando =>
                        if SO='1' then proximoEstado <= Abrindo;
                        elseif SF='1' then proximoEstado <= Fechado;
                        else  proximoEstado <= Fechando;
                        end if;
        end case;
end process;
```

a lógica desse circuito combinacional é complexa demais para um comando concorrente de algumas linhas, então nesse caso (e apenas nesse caso) permite-se que o comportamento seja descrito usando process e comandos sequenciais.

apenas o sinal de "próximo estado" deve ter algo atribuído. Para evitar atribuições "incompletas" e a geração de elementos de memória nesse circuito combinacional, é uma boa prática inicializar o sinal.

Exemplos

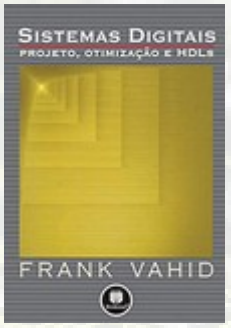
```
-- elemento de memória
process(clock, reset) is
begin
	if reset='1' then
		estadoAtual <= Fechando;
	elsif rising_edge(clock) then
		estadoAtual <= proximoEstado;
	end if;
end process;

-- lógica de saída
with EstadoAtual select
	MT <=	"00" when Aberto,
		"11" when Fechando,
		"01" when Fechado,
		"10" when Abrindo;
```

novamente, o registrador permanece inalterado, a não ser a atribuição do estado de reset, que depende do tipo do estado. Neste caso, é o estado inicial da FSM.

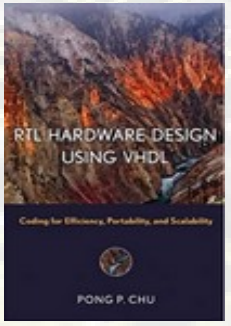
a lógica de saída é um circuito combinacional cujo comportamento é simples o suficiente para ser descrito com um único comando concorrente de algumas linhas. Portanto, essa deve ser a escolha feita.

Referências Bibliográficas

- Vahid, Frank. Sistemas Digitais: projeto, otimização e HDLs. Porto Alegre: Bookman, 2008. ISBN 978-85-7780-190-9



- Chu, Pomg P. RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability. Wiley-Interscience, 2006.



- Pedroni, Volnei. Circuit Design with VHDL. The MIT Press, 3th edition, 2020.