Aula XII – Modelagem de Circuitos Sequenciais em VHDL

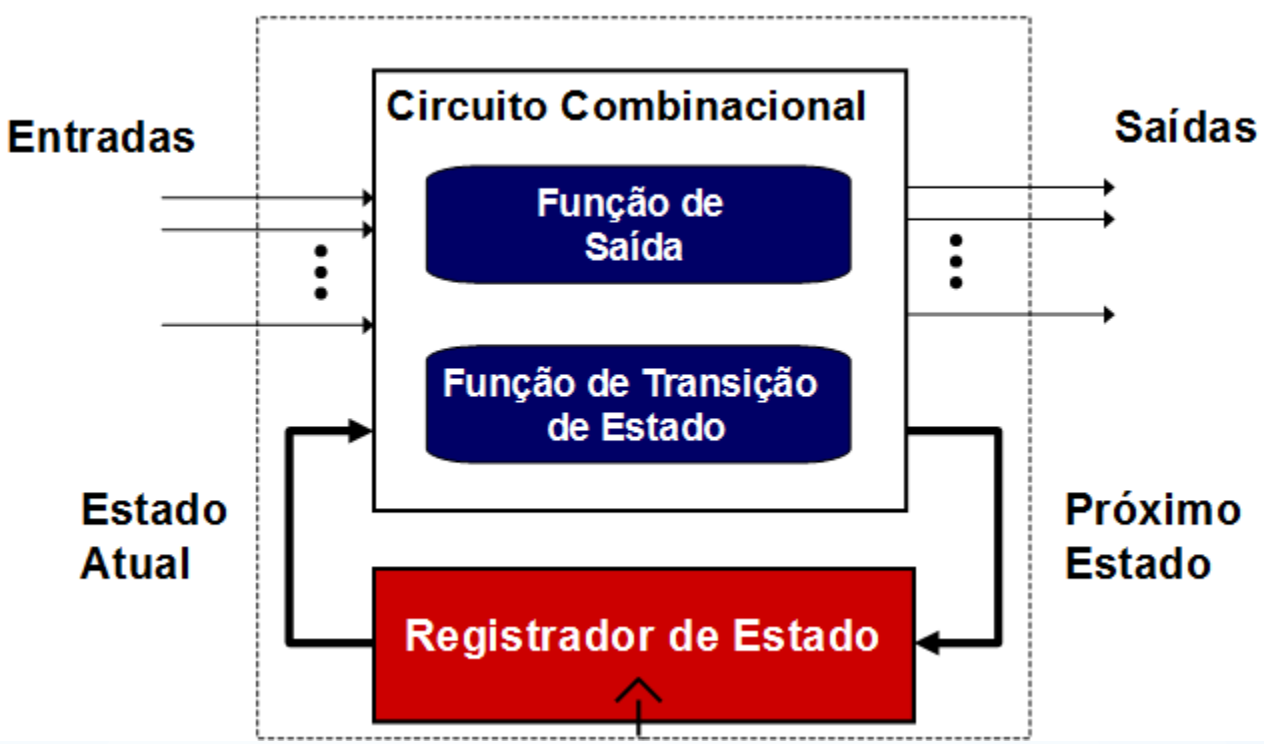
Tópicos da aula

- **Blocos construtivos básicos**
- **Inferência de flip-flops**
- **Registradores**
- **Contadores**
- **Máquinas de Estado**

Circuitos Sequenciais

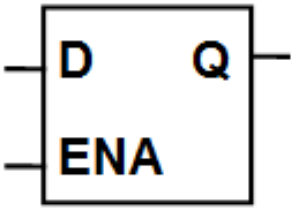
- **São aqueles cujas saídas dependem do estado atual e passado das entradas**
- **Exemplos:**
  - Registradores
  - Contadores
  - Máquinas de Estado

Arquitetura genérica de um circuito sequencial

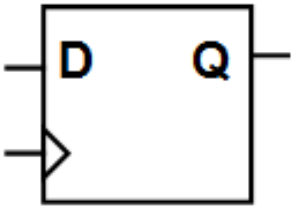


Blocos construtivos básicos

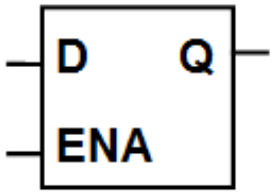
- Latches



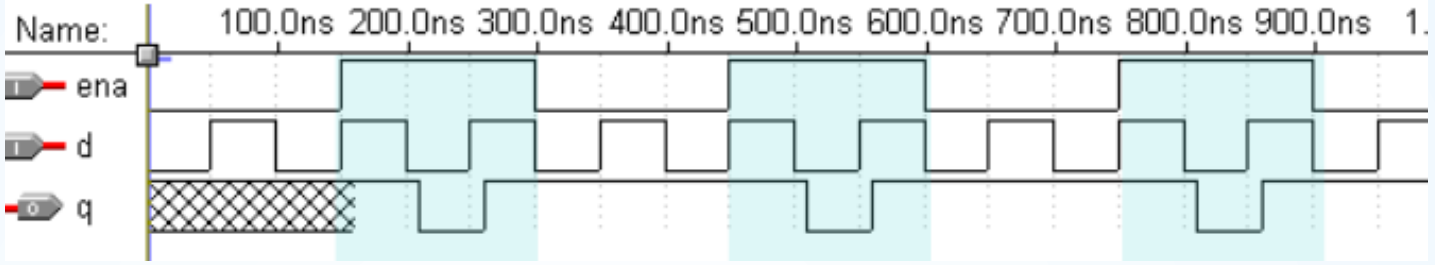
- Flip-flops



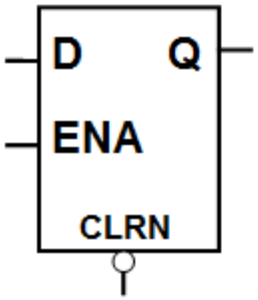
Latch transparente



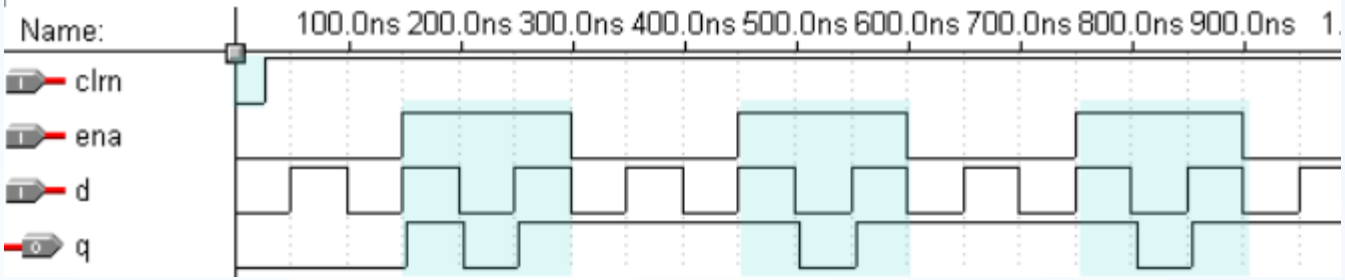

```
ENTITY latch_d IS
  PORT (d   : IN  BIT;  -- data input
        ena : IN  BIT;  -- enable
        q   : OUT BIT); -- data output
END latch_d;

---------------------------------------
ARCHITECTURE behavior OF latch_d IS
BEGIN
  PROCESS(d,ena)
  BEGIN
    IF (ena = '1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END behavior;
```



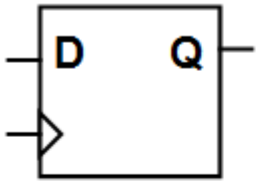
Latch com clear



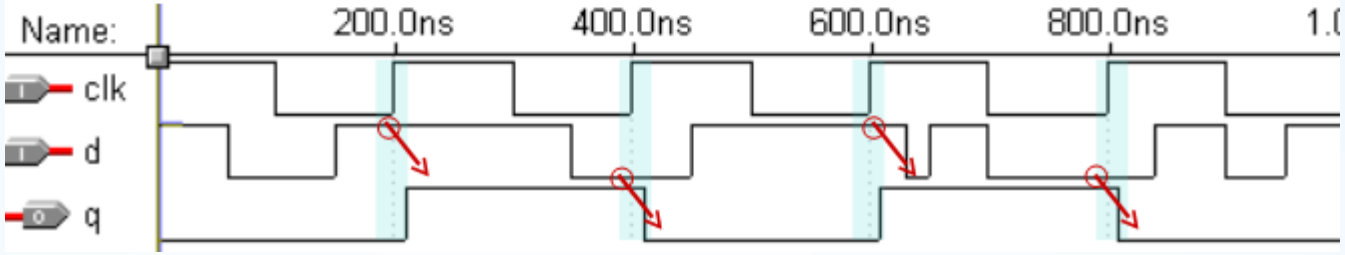

```
ENTITY latch_d_clear IS
  PORT (d    : IN  BIT;  -- data input
        ena  : IN  BIT;  -- enable
        clrn: IN  BIT;  -- clear
        q    : OUT BIT); -- data output
END latch_d_clear;

-----------------------------------------
ARCHITECTURE behavior OF latch_d_clear IS
BEGIN
  PROCESS(d,ena,clrn)
  BEGIN
    IF (clrn = '0') THEN
      Q <= '0';
    ELSIF (ena = '1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END behavior;
```



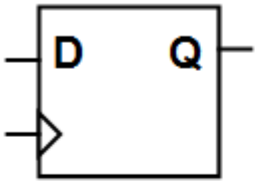
Flip-flop tipo D



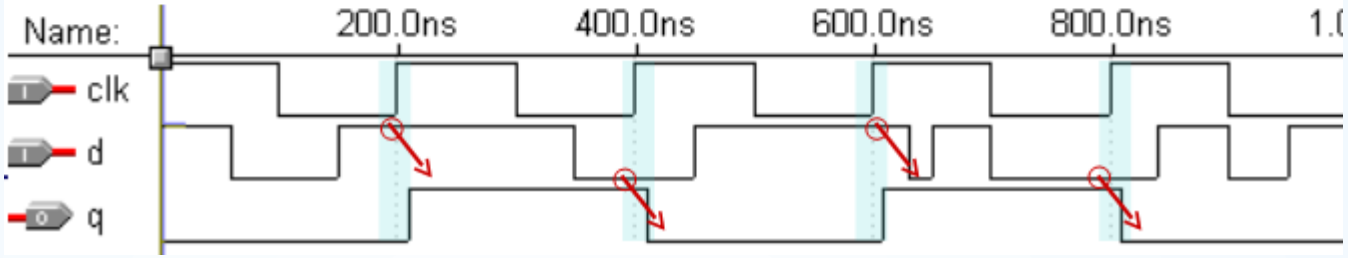

```
ENTITY ff_d IS
  PORT (d   : IN  BIT;  -- data input
        clk : IN  BIT;  -- clock
        q   : OUT BIT); -- data output
END ff_d;

---------------------------------------
ARCHITECTURE arch_1 OF ff_d IS
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL clk='1';
    Q <= D;
  END PROCESS;
END arch_1;
```



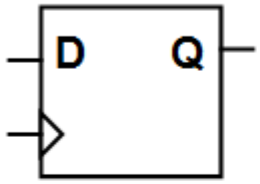
Flip-flop tipo D



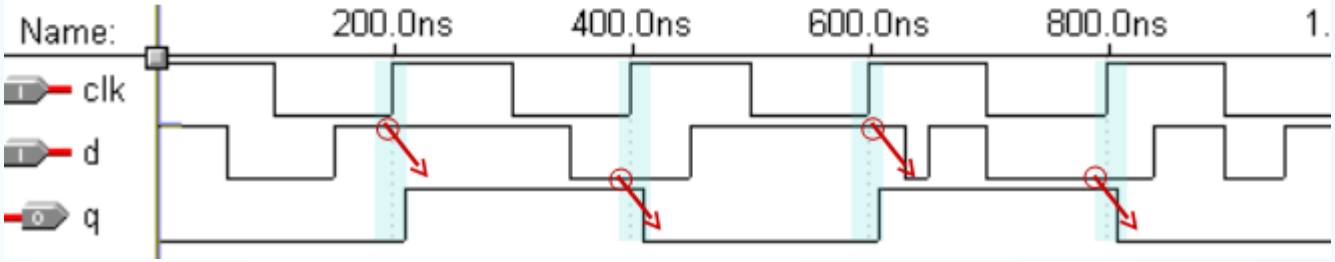

```
ENTITY ff_d IS
  PORT (d   : IN  BIT;  -- data input
        clk : IN  BIT;  -- clock
        q   : OUT BIT); -- data output
END ff_d;

---------------------------------------
ARCHITECTURE arch_2 OF ff_d IS
BEGIN
  PROCESS(clk)
  BEGIN
    IF (clk'EVENT AND clk='1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END arch_2;
```



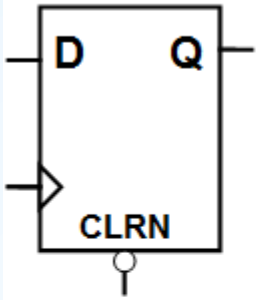
Flip-flop tipo D



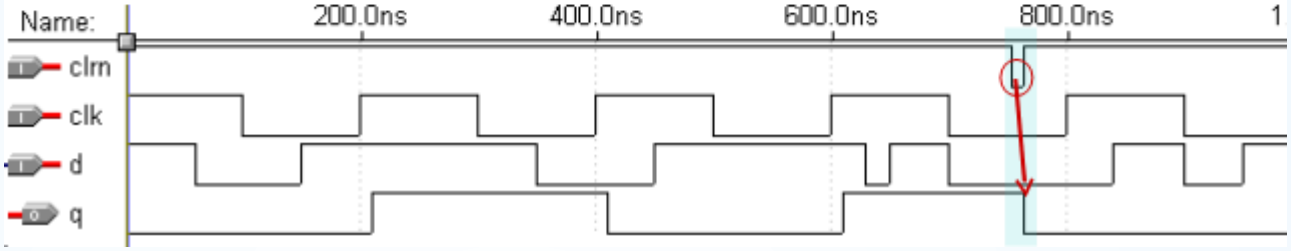

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY ff_d_std_logic IS
  PORT (d   : IN  STD_LOGIC;  -- data input
        clk : IN  STD_LOGIC;  -- clock
        q   : OUT STD_LOGIC); -- data output
END ff_d_std_logic;

-----------------------------------------
ARCHITECTURE arch_1 OF ff_d_std_logic IS
BEGIN
  PROCESS(clk)
  BEGIN
    IF rising_edge(clk) THEN
      Q <= D;
    END IF;
  END PROCESS;
END arch_1;
```



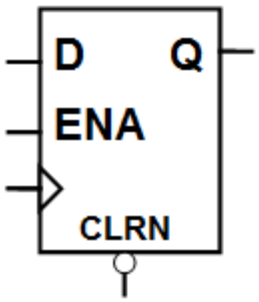
Flip-flop tipo D com clear



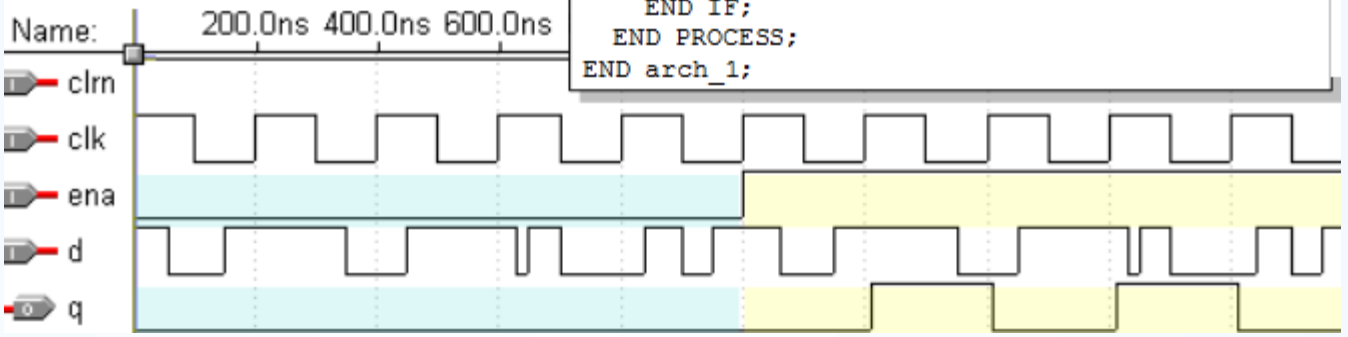

```
ENTITY ff_d_clear IS
  PORT (d    : IN  BIT;  -- data input
        clk  : IN  BIT;  -- clock
        clrn: IN  BIT;  -- clear
        q    : OUT BIT); -- data output
END ff_d_clear;
---------------------------------------
ARCHITECTURE arch_1 OF ff_d_clear IS
BEGIN
  PROCESS(clk,clrn)
  BEGIN
    IF (clrn='0') THEN
      Q <= '0';
    ELSIF (clk'EVENT AND clk='1') THEN
      Q <= D;
    END IF;
  END PROCESS;
END arch_1;
```



Flip-flop tipo D com clear e enable




```
ENTITY ff_d_clear_ena IS
  PORT (d   : IN  BIT;  -- data input
        clk : IN  BIT;  -- clock
        clrn: IN  BIT;  -- clear
        ena : IN  BIT;  -- enable
        q   : OUT BIT); -- data output
END ff_d_clear_ena;
---------------------------------------
ARCHITECTURE arch_1 OF ff_d_clear_ena IS
BEGIN
  PROCESS(clk,clrn)
  BEGIN
    IF (clrn='0') THEN
      Q <= '0';
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (ena='1') THEN
        Q <= D;
      END IF;
    END IF;
  END PROCESS;
END arch_1;
```



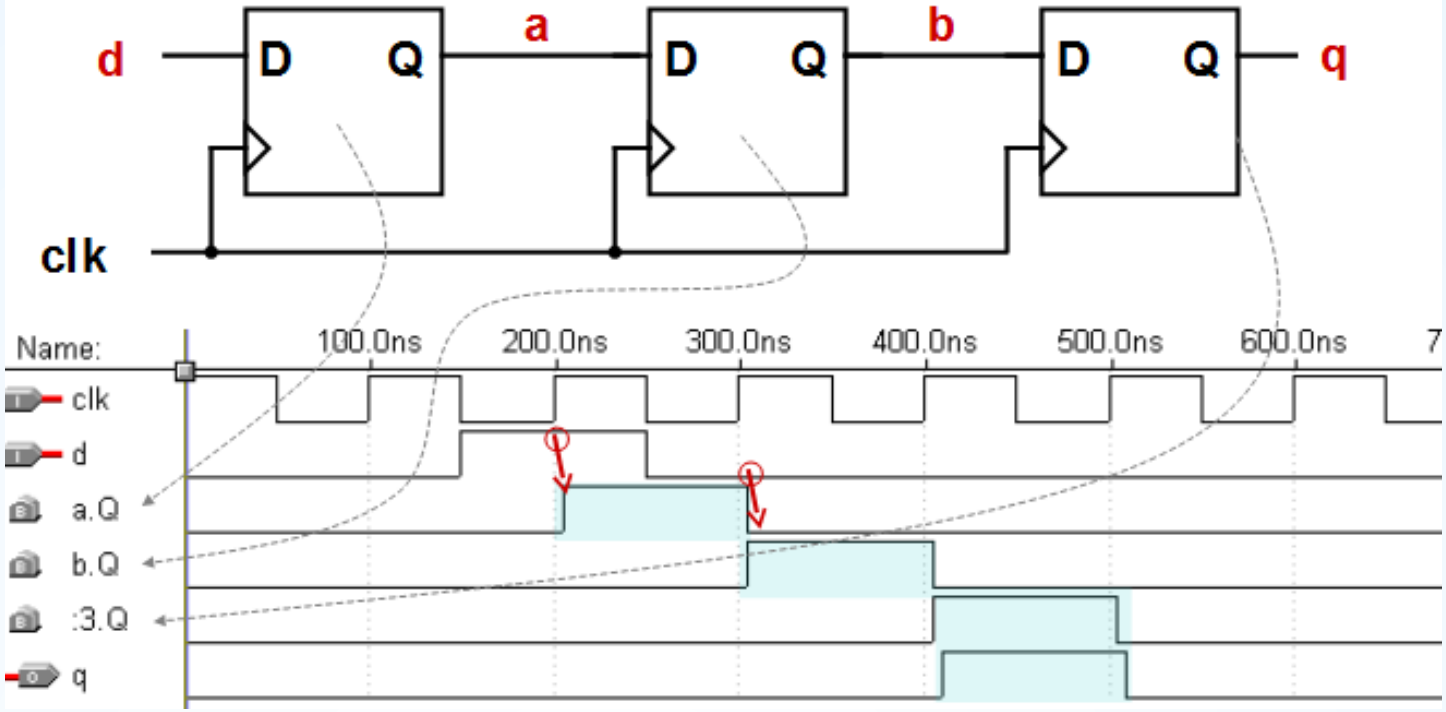
Inferência de flip-flops

- **Quando flip-flops serão inferidos para reg1.vhd?**

```
ENTITY reg1 IS
  PORT ( d   : IN  BIT;
         clk : IN  BIT;
         q   : OUT BIT);
END reg1;
-------------------------------------
ARCHITECTURE arch_1 OF reg1 IS
  SIGNAL a, b : BIT;
BEGIN
  PROCESS(clk)
  BEGIN
    IF (clk'EVENT AND clk='1') THEN
      a <= d;
      b <= a;
      q <= b;
    END IF;
  END PROCESS;
END arch_1;
```

Inferência de flip-flops

- Quando flip-flops serão inferidos reg1.vhd?



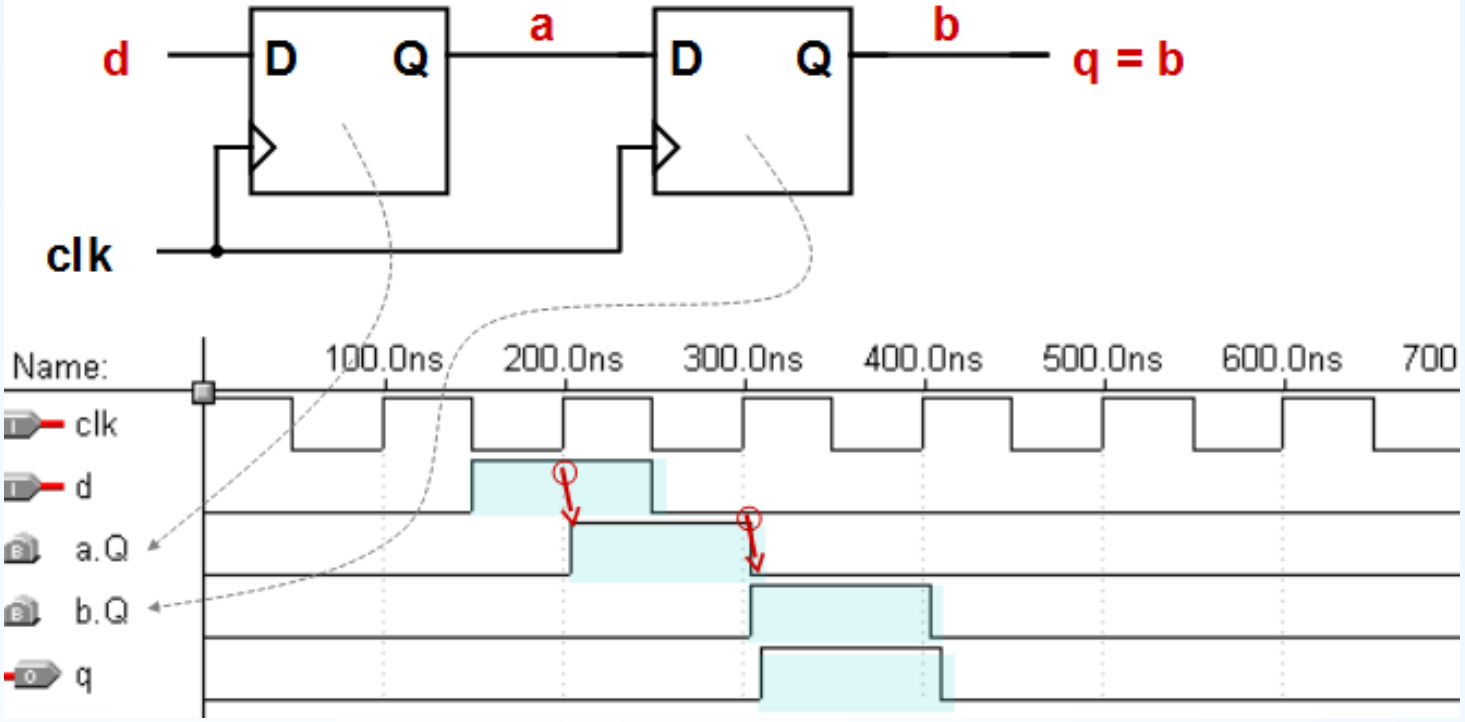
Inferência de flip-flops

- **Quando flip-flops serão inferidos para reg2.vhd?**

```
ENTITY reg2 IS
  PORT ( d   : IN  BIT;
         clk : IN  BIT;
         q   : OUT BIT);
END reg1;
-----------------------------------
ARCHITECTURE arch_1 OF reg2 IS
  SIGNAL a, b : BIT;
BEGIN
  PROCESS(clk)
  BEGIN
    IF (clk'EVENT AND clk='1') THEN
      a <= d;
      b <= a;
    END IF;
  END PROCESS;
  q <= b;
END arch_1;
```

Inferência de flip-flops

- Quando flip-flops serão inferidos reg2.vhd?



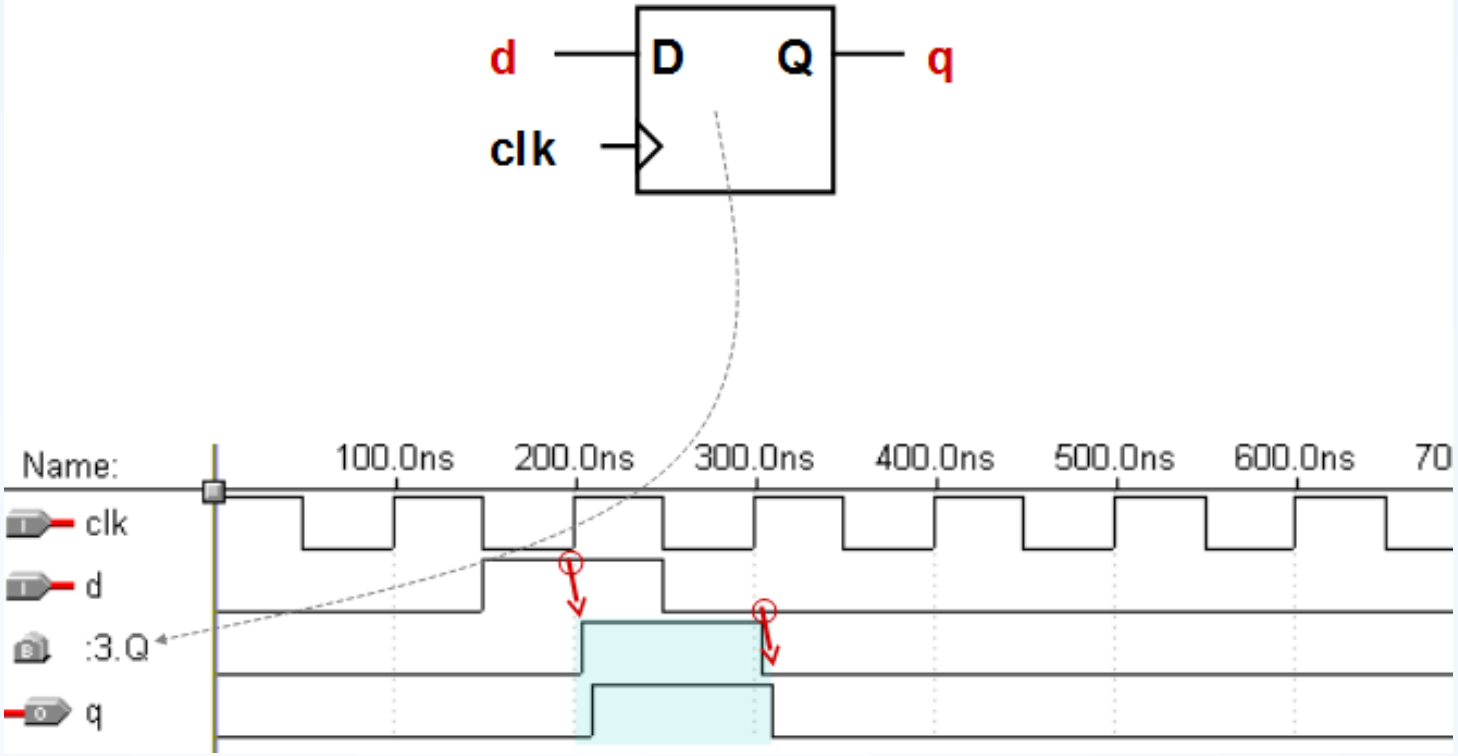
Inferência de flip-flops

- **Quando flip-flops serão inferidos para reg3.vhd?**

```
ENTITY reg3 IS
  PORT ( d   : IN  BIT;
         clk : IN  BIT;
         q   : OUT BIT);
END reg1;
-----------------------------------
ARCHITECTURE arch_1 OF reg3 IS
BEGIN
  PROCESS(clk)
    VARIABLE a, b : BIT;
  BEGIN
    IF (clk'EVENT AND clk='1') THEN
      a := d;
      b := a;
      q <= b;
    END IF;
  END PROCESS;
END arch_1;
```

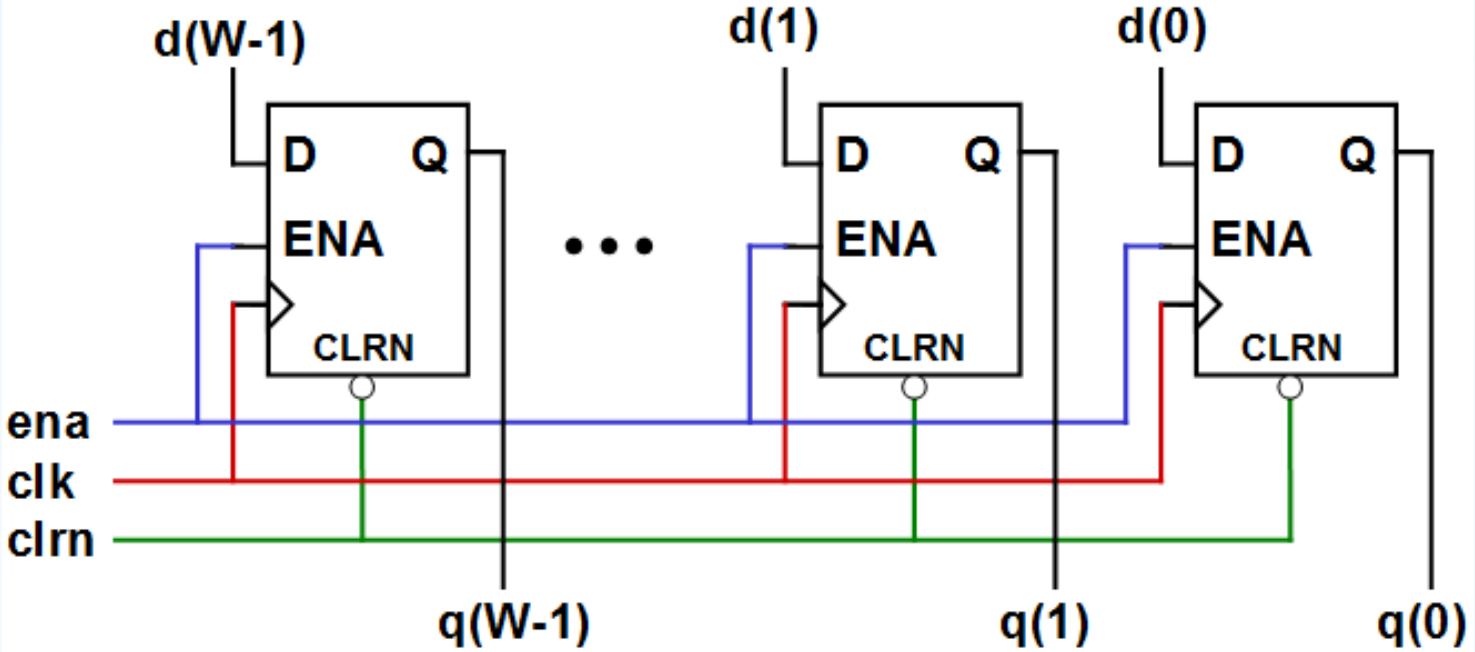
Inferência de flip-flops

- Quando flip-flops serão inferidos reg3.vhd?



Registradores

- Registrador Paralelo-Paralelo de W bits



Registradores

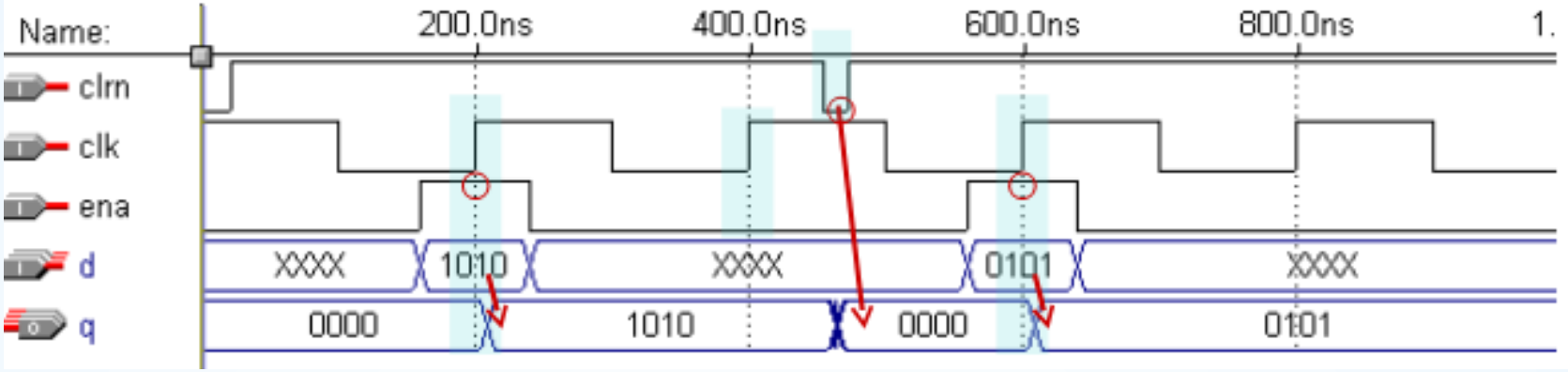
Registrador Paralelo-Paralelo de W bits

```
ENTITY reg_pp_Wbits IS
  GENERIC(W : NATURAL := 4);
  PORT (d   : IN  BIT_VECTOR(W-1 DOWNTO 0);-- data input
        clk : IN  BIT;  -- clock
        clrn: IN  BIT;  -- clear
        ena : IN  BIT;  -- enable
        q   : OUT BIT_VECTOR(W-1 DOWNTO 0));-- data output
END reg_pp_Wbits;

ARCHITECTURE arch_1 OF reg_pp_Wbits IS
BEGIN
  PROCESS(clk,clrn)
  BEGIN
    IF (clrn='0') THEN
      q <= (OTHERS => '0');
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (ena='1') THEN
        q <= d;
      END IF;
    END IF;
  END PROCESS;
END arch_1;
```

Igual ao flip-flop tipo D com clear e enable

Registradores

Registrador Paralelo-Paralelo de W bits

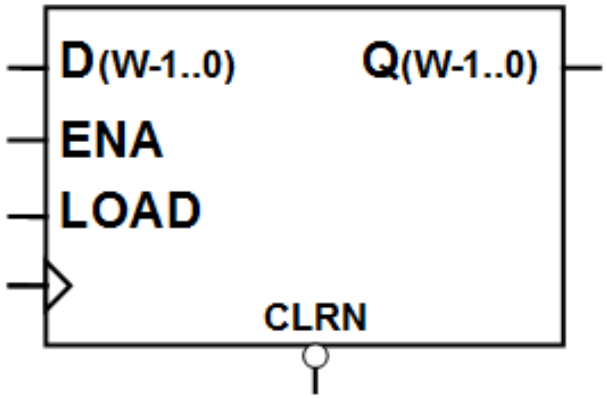


Contadores

- **Incrementador de W bits**
  - Aplicação típica: Program Counter (PC)

CLRN	Ena	Load	Qf
0	X	X	0
1	0	X	Qa
	1	1	D
		0	Qa+1

CLRN é assíncrono



Contadores

Incrementador de W bits – Entidade

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY counter_Wbits IS
  GENERIC(W : NATURAL := 4);
  PORT (d    : IN  STD_LOGIC_VECTOR(W-1 DOWNTO 0);-- data input
        clk  : IN  BIT;  -- clock
        clrn: IN  BIT;  -- clear
        ena  : IN  BIT;  -- enable
        load: IN  BIT;  -- load
        q    : BUFFER STD_LOGIC_VECTOR(W-1 DOWNTO 0));-- data
output
END counter_Wbits;
```

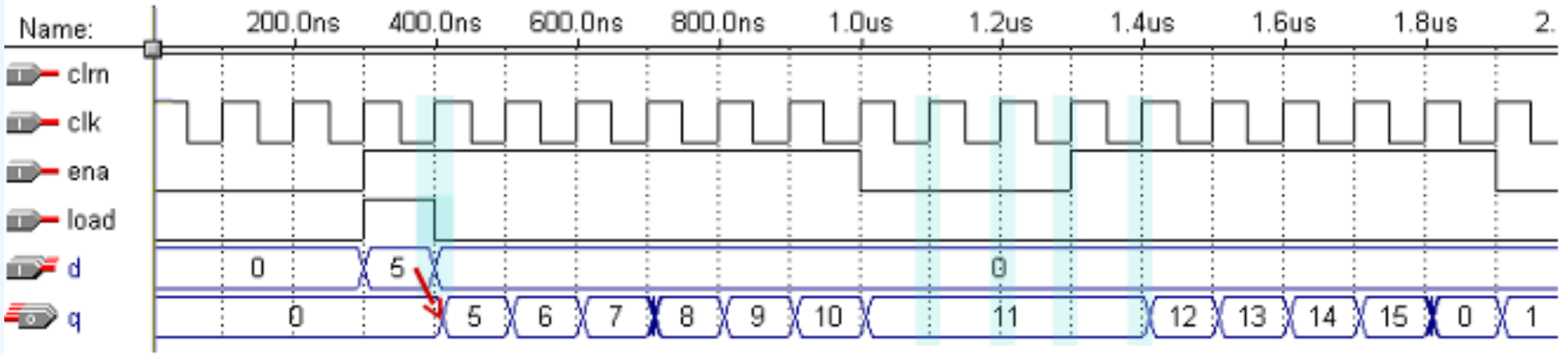
Contadores

Incrementador de W bits – Arquitetura

```
ARCHITECTURE arch_1 OF counter_Wbits IS
BEGIN
  PROCESS(clk,clrn)
  BEGIN
    IF (clrn='0') THEN
      q <= (OTHERS => '0');
    ELSIF (clk'EVENT AND clk='1') THEN
      IF (ena='1') THEN
        IF (load='1') THEN
          q <= d;
        ELSE
          q <= q+1;
        END IF;
      END IF;
    END IF;
  END PROCESS;
END arch_1;
```

Contadores

- Incrementador de W bits



FIM AULA XIII