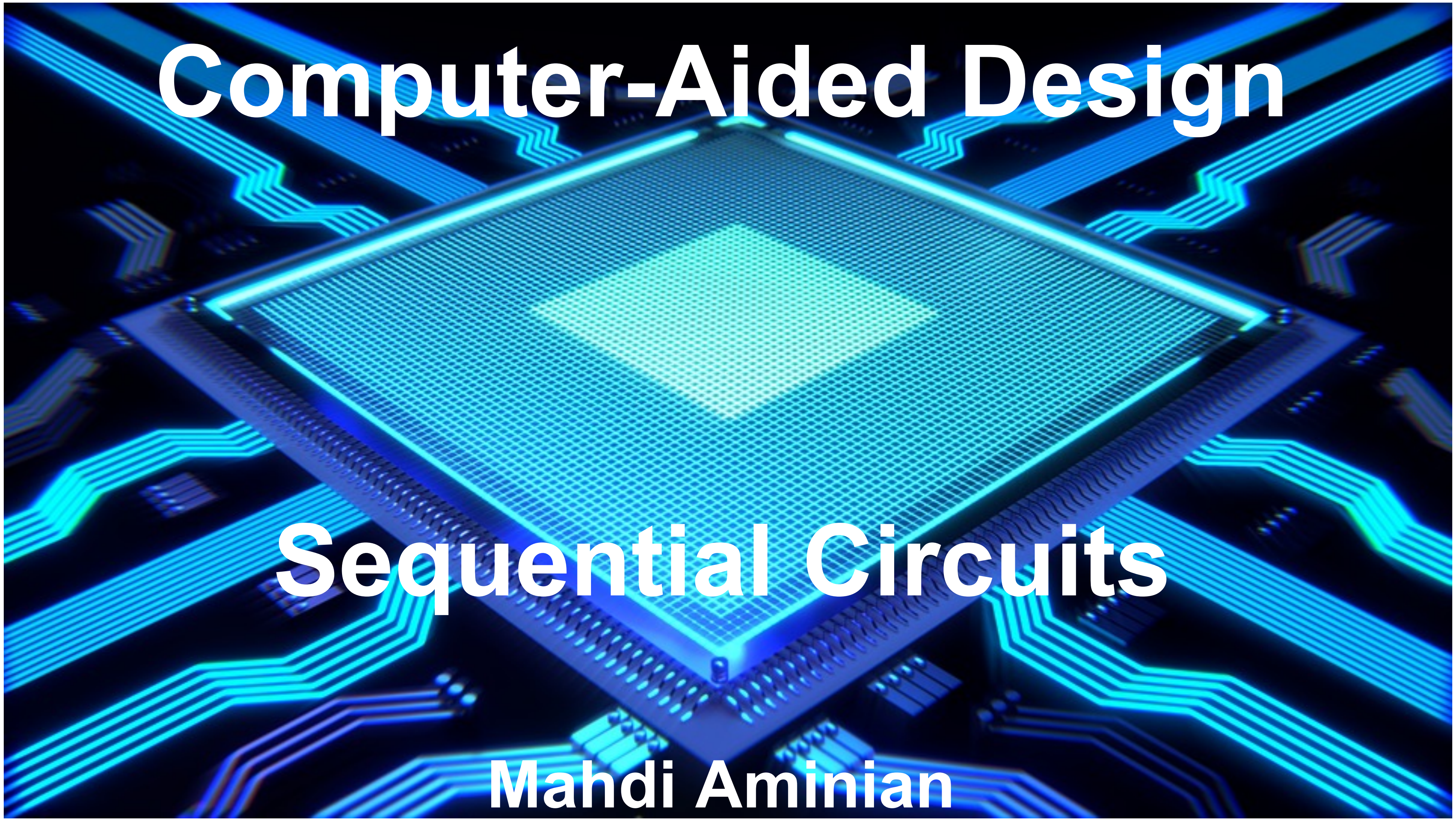


Computer-Aided Design

Sequential Circuits

Mahdi Aminian

The University Of Guilan
Rasht - Iran

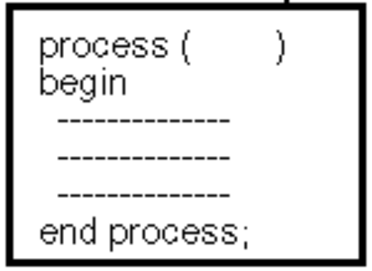
Some slides courtesy of:

- “Hardware Systems Modeling”, A. Vachoux, EPFL
- CAD slides from Dr. Saheb Zamani

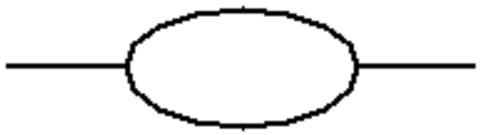
RTL Style

- در RTL مدار ترتیبی به دو بخش (ترکیبی و عناصر حافظه) تقسیم می شود.
- می توان برای هر بخش یک پروسس نوشت یا برای هر دو فقط یک پروسس نوشت.

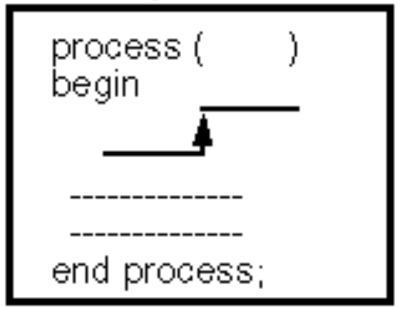
Combinational process



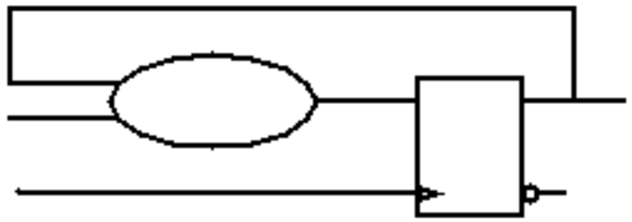
```
process (        )
begin
  --------------
  --------------
  --------------
end process;
```



Clocked process



```
process (        )
begin

  --------------
  --------------
end process;
```



Describing Sequential/Synchronous Behavior

- **Edge-sensitive (flip-flop)**

```
signal clk : std_logic;
```

```
process (clk)
begin
   if clk'event and clk = '1' then
   -- or: if rising_edge(clk) then
      clock-driven behavior
   end if;
end process;
```

```
process
begin
   wait until clk = '1';
   -- or: wait until rising_edge(clk);
   clock-driven behavior
end process;
```

- **Level-sensitive (latch)**

```
signal en : std_logic;
```

```
process (en, other read signals)
begin
   if en = '1' then
      transparent-latch behavior
   end if;
end process;
```

- Normally not preferred for synchronous designs
- May be mistakenly inferred by careless VHDL coding

- Signal clk is supposed to be a clock (periodically switching signal)
- Falling edge specification: clk = '0' or falling_edge(clk)
- Same clock signal and same edge in the same process

Flip-Flop vs. Latch Models

- **D flip-flop**
  - Edge-sensitive behavior

```
entity dff is
   port (
      signal clk, d : in  std_logic;
      signal q      : out std_logic);
end entity dff;

architecture rtl of dff is
begin
   process
   begin
      wait until rising_edge(clk);
      q <= d;
   end process;
end architecture rtl;
```

- **D latch**
  - Level-sensitive behavior

```
entity dlatch is
   port (
      signal en, d : in  std_logic;
      signal q     : out std_logic);
end entity dlatch;

architecture rtl of dlatch is
begin
   process (en, d)
   begin
      if en = '1' then
         q <= d;
      end if;
   end process;
end architecture rtl;
```

Description of Rising Clock Edge for Synthesis

برای عناصرحافظه: if یا wait until به صورت خاص

- **Standard for synthesis: IEEE 1076.6**

- *... if condition*
  - RISING_EDGE (*CLK*) (not always supported)
  - *CLK*'event *and* CLK='1'
  - *CLK*='1' *and CLK*'event
  - *not CLK*'stable *and CLK*='1'
  - *CLK*='1' *and* NOT *CLK*'stable

- *... wait until condition*
  - RISING_EDGE (CLK)
  - *CLK*'event *and CLK*='1'
  - *CLK*='1' *and CLK*'event
  - not *CLK*'stable *and CLK*='1'
  - *CLK*='1' *and* not *CLK*'stable
  - *CLK*='1'

Rising_edge (falling_edge)

```
process
begin
  wait until RISING_EDGE(CLK);
  Q <= D;
end process;
```

- **In Std_Logic_1164 package**

```
function RISING_EDGE (signal CLK : std_ulogic)
          return boolean is
begin
    if (  CLK`event and CLK =`1`
                        and CLK`last_value=`0`) then
       return true;
    else
       return false;
    end if;
end RISING_EDGE;
```

From Combinational to Sequential Logic

- **Combinational**

```
entity E1 is
   port (
      signal A, B : in  std_logic;
      signal Z    : out std_logic);
end entity E1;

architecture comb of E1 is
   signal C, D : std_logic;
begin
   process (A, B, C, D)
   begin
      C <= A xor B;
      D <= A or (C xor B);
      Z <= C nand D;
   end process;
end architecture comb;
```

- **Sequential (clocked)**

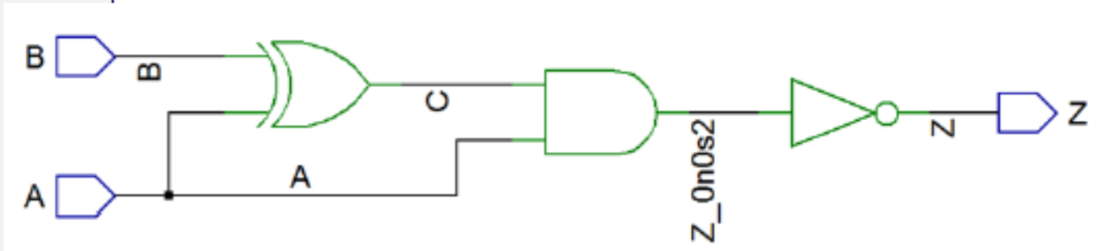
```
entity E2 is
   port (
      signal clk  : in  std_logic;
      signal A, B : in  std_logic;
      signal Z    : out std_logic);
end entity E2;

architecture clocked of E2 is
   signal C, D : std_logic;
begin
   process
   begin
      wait until rising_edge(clk);
      C <= A xor B;
      D <= A or (C xor B);
      Z <= C nand D;
   end process;
end architecture clocked;
```

From Combinational to Sequential Logic

- **Combinational**

```
entity E1 is
   port (
      signal A, B : in  std_logic;
      signal Z    : out std_logic);
end entity E1;

architecture comb of E1 is
   signal C, D : std_logic;
begin
   process (A, B, C, D)
   begin
      C <= A xor B;
      D <= A or (C xor B);
      Z <= C nand D;
   end process;
end architecture comb;
```

- **Design entity E1(comb)**

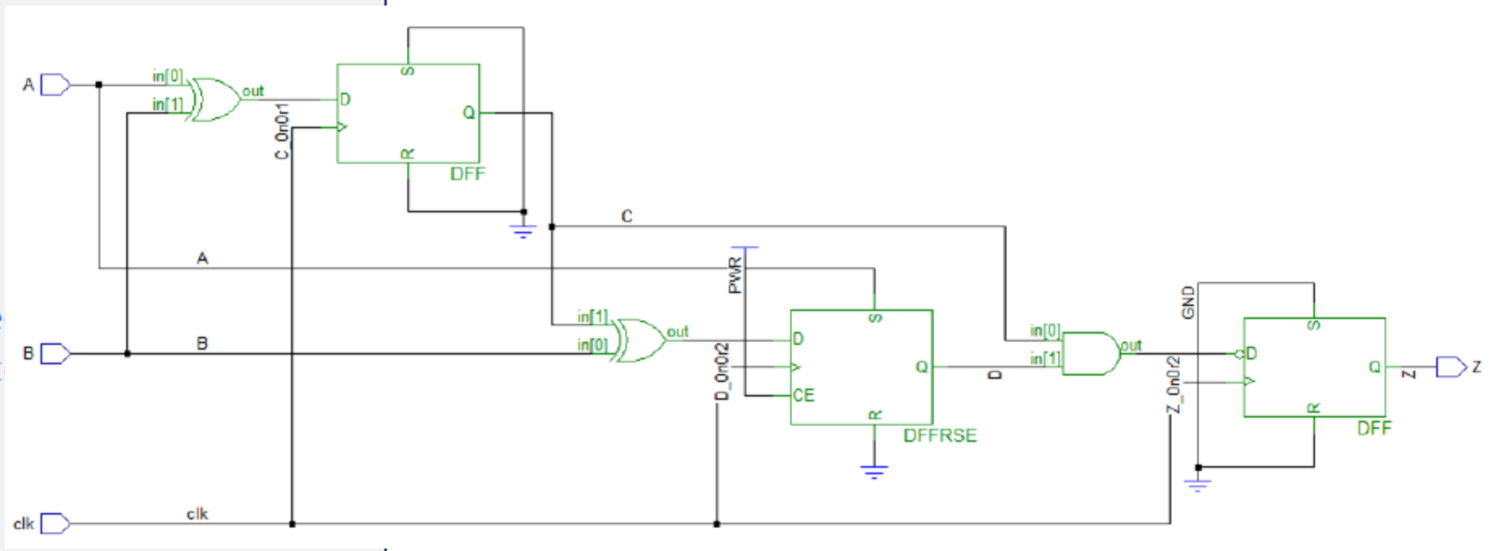


From Combinational to Sequential Logic

- **Sequential (clocked)**

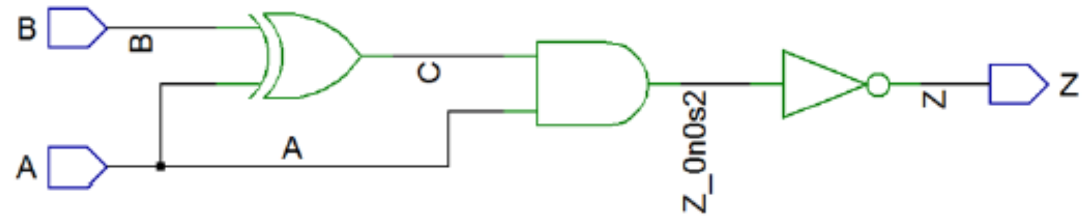
```
entity E2 is
   port (
      signal clk   :
      signal A, B :
      signal Z     :
end entity E2;

architecture clocke
   signal C, D : st
begin
   process
   begin
      wait until rising_edge(clk);
      C <= A xor B;
      D <= A or (C xor B);
      Z <= C nand D;
   end process;
end architecture clocked;
```

- **Design entity E2(clocked)**

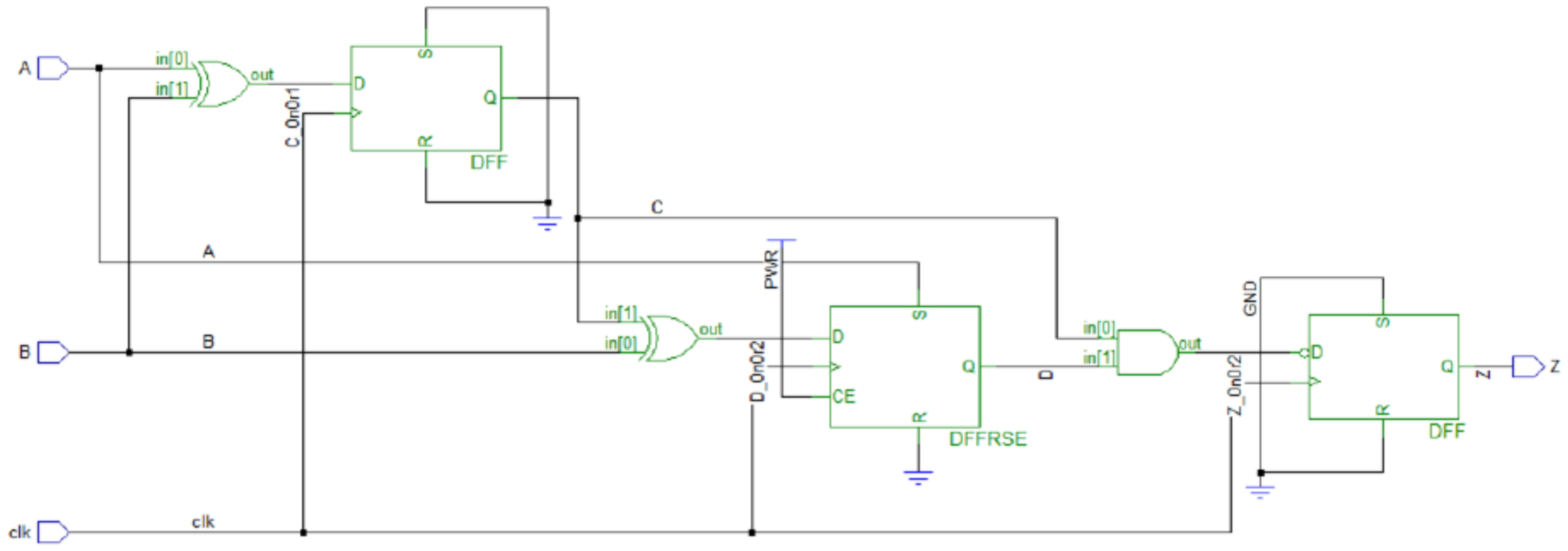


From Combinational to Sequential Logic

- **Design entity E1(comb)**



- **Design entity E2(clocked)**



Avoid Latches

```
entity E3 is
   port (
      signal en   : in  std_logic;
      signal A, B : in  std_logic;
      signal Z    : out std_logic);
end entity E3;
```

- Latches inferred

```
architecture latched of E3 is
   signal C, D : std_logic;
begin
   process (en, A, B, C, D)
   begin
      if en = '1' then
         C <= A xor B;
         D <= A or (C xor B);
         Z <= C nand D;
      end if;
   end process;
end architecture latched;
```

- No latch inferred

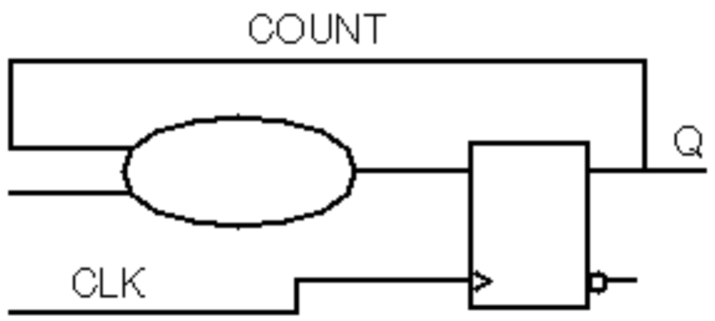
```
architecture comb of E3 is
   signal C, D : std_logic;
begin
   process (en, A, B, C, D)
   begin
      C <= '0';
      D <= '0';
      Z <= '0';
      if en = '1' then
         C <= A xor B;
         D <= A or (C xor B);
         Z <= C nand D;
      end if;
   end process;
end architecture comb;
```

Register Inference

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity COUNTER is
port ( CLK    : in std_ulogic;
         Q    : out integer  range 0 to 15 );
end COUNTER;

architecture A of COUNTER is
  signal COUNT  : integer  range 0 to 15 ;
begin
  process (CLK)
  begin
    if CLK`event and CLK = `1` then
      if (COUNT >= 9) then
        COUNT <= 0;
      else
        COUNT <= COUNT +1;
      end if;
    end if;
  end process;
   Q <= COUNT;
end A;
```

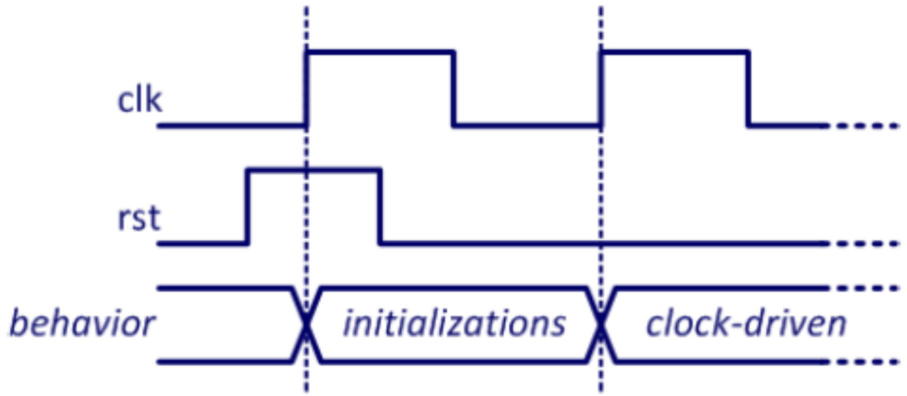
- **One-digit BCD counter**
- **For all signals which receive an assignment in clocked processes, memory is synthesized.**
  - COUNT: 4 FF
    - (constrained integer)
    - Q not used in clocked process.

Problem: doesn’t have power-on reset



Reset for Edge-Sensitive Behavior

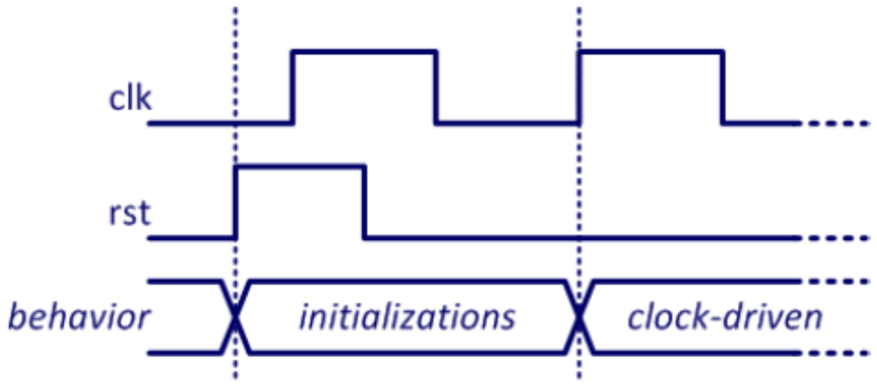
- **Synchronous reset**



```
signal clk, rst : std_logic;
```

```
process
begin
   wait until rising_edge(clk);
   if rst = '1' then
      initialization(s)
   else
      clock-driven behavior
   end if;
end process;
```

- **Asynchronous reset**



```
signal clk, rst : std_logic;
```

```
process (clk, rst)
begin
   if rst = '1' then
      initialization(s)
   elsif rising_edge(clk) then
      clock-driven behavior
   end if;
end process;
```

Asynchronous Set/Reset

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity ASYNC_FF is
port (    D, CLK, SET, RST : in std_ulogic;
          Q                : out std_ulogic);
end ASYNC_FF;

architecture A of ASYNC_FF is
begin
   process  (CLK, RST, SET)
   begin
     if (RST = `1`) then
       Q <= `0`;
     elsif SET ='1' then
       Q <= '1';
     elsif (CLK`event and CLK = `1`) then
       Q <= D;
     end if;
   end process;
end A;
```

- **if/elsif - structure**
- **The last elsif has an edge**
- **No else**

- برای **set/reset** سنکرون فقط **clk** در لیست حساسیت قرار می گیرد (می توان با **wait until** هم مدلسازی کرد).
- اما برای آسنکرون فقط با لیست حساسیت می توان مدلسازی کرد
- حتماً همهٔ ورودیهای آسنکرون در لیست حساسیت وارد شوند والا نتیجهٔ شبیه سازی با سنتز متفاوت می شود.

Example: D Flip-Flop with Reset

```
entity dff is
   port (
      signal clk, rst : in  std_logic;
      signal d        : in  std_logic;
      signal q        : out std_logic);
end entity dff;
```

- **Synchronous reset**

```
architecture sync_rst of dff is
begin
   process
   begin
      wait until rising_edge(clk);
      if rst = '1' then
         q <= '0';
      else
         q <= d;
      end if;
   end process;
end architecture sync_rst;
```

- **Asynchronous reset**

```
architecture async_rst of dff is
begin
   process (clk, rst)
   begin
      if rst = '1' then
         q <= '0';
      elsif rising_edge(clk) then
         q <= d;
      end if;
   end process;
end architecture async_rst;
```

- **These two templates must be strictly used for synthesis**
  - Initialization and clocked behaviors may be more complex

1-bit Register -> 4-bit ?

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity Register_1bit is
port ( CLK, Reset, Load   : in std_logic;
         DataIN    : in std_logic;
         Q    : out  std_logic );
end  Register_1bit;

architecture RTL of  Register_1bit  is
  signal Reg_Input, Q_Reg  : std_logic ;
begin
  process (CLK, Reset)
  begin
    if (Reset = ‘1’) then
                Q_Reg <= ‘0’;
    elsif CLK’event and CLK = `1` then
                Q_Reg <= Reg_Input;
    end if;
  end process;
```

```
  process (Load, Q_Reg, DataIN)
  begin
    if (Load = ‘1’) then
              Reg_Input <= DataIN;
    else
              Reg_Input <= Q_Reg;
    end if;
  end process;

  Q <= Q_Reg;

end RTL;
```

4-bit Shift-Register

- ??