Chapter 3

Behavioral VHDL

Behavioral VHDL

- Process
- Sensitivity List
- Wait Statements
- If Statements
- Case Statements
- Loop Statements

Three Styles of VHDL

Behavioral

Boolean

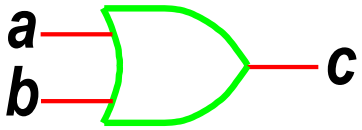
$c = a \vee b$

VHDL

```
c <= a OR b;
```

Structural

RTL level

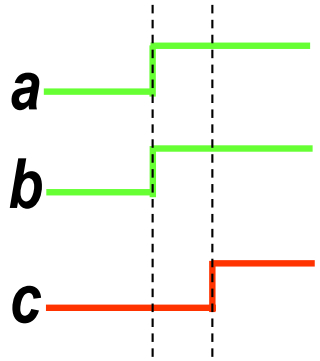


VHDL

```
COMPONENT and_gate
    port(
        i1,i2: in std_logic;
        o: out std_logic);
END COMPONENT;
        . . .
G0: and_gate
    port map(
            i1 => a, i2 => b,
            o1 => c);
```

Timing

Schematic



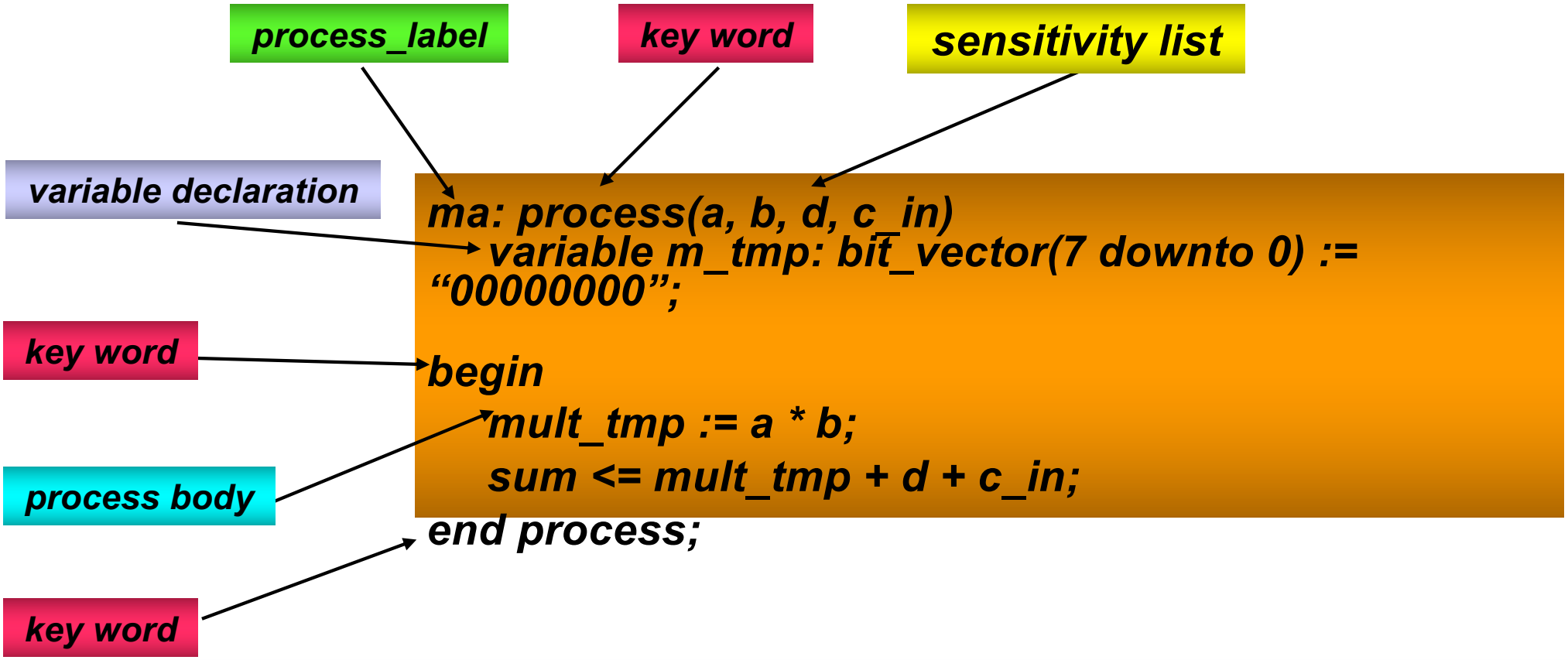
VHDL

```
c <= a OR b after 10 ns;
```

Modeling Styles

- Possibly many different architecture bodies of one entity corresponding to alternative implementations performing the same function (design functionality)
- Styles of architecture body
  - Behavioral
  - Structural
  - Mixed structural/behavioral

Process – Syntax 2



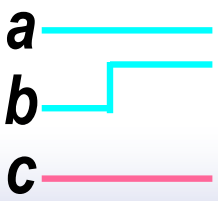
Example: Sensitivity List

b not in Sensitivity list

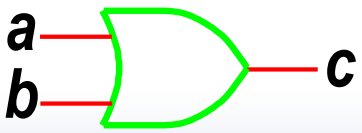
```
or_gate: process (a)
begin
    c <= a or b;
end process;
```

Sensitivity list

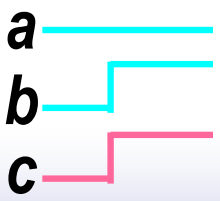
Pre-synthesis simulations



Synthesized netlist



Post-synthesis waveform



Wait Statements

- Suspends execution of process
- Three basic kinds of *wait* statement

```
wait on sensitivity-list;
```

  - *on* signals

```
wait until sensitivity-list;
```

  - *until* conditions

```
wait for sensitivity-list;
```

  - *for* signals

- Combinations of above conditions in single wait statement also possible

```
wait on sensitivity-list until boolean-expression for time-expression;
```

Wait On

```
wait on A, B, C;
```

- Process suspends and wait for event to occur on signals A, B, or C
  - Upon event on A, B, or C process resumes execution from next statement onwards
  - When wait is the last instruction, process resumes from first statement

Wait Until

```
wait until A=B;
```

- Process suspends until specified condition becomes true
  - Upon encountering even on signals A or B "=" condition evaluated
    - If fulfilled process resumes operation from next statement onwards, otherwise it remains suspended

Wait For

```
wait for 10 ns;
```

- Execution of wait statement at time T causes suspense of process for 10 ns
  - Next process statement executed at time T+10ns

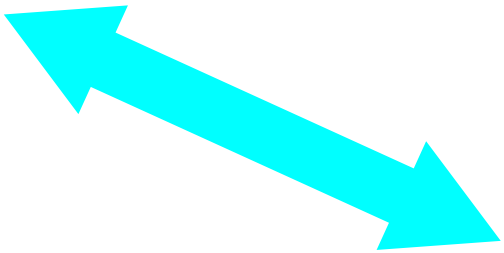
Wait On Sensitivity-List For Time-Expression

```
wait on CLOCK for 20 ns;
```

- Process suspended until event on CLOCK for a timeout of 20 ns

Example: Wait Statements

```
half_add: process is
begin
    sum  <= a XOR b after T_pd;
    carry <= a AND b after T_pd;
    wait on a, b;
end process;
```



```
half_add: process(a,b) is
begin
    sum  <= a XOR b after T_pd;
    carry <= a AND b after T_pd;
    -- not wait on a,b; needed
end process;
```

Clock Generation

- One of ways of clock generation for SIMULATION purposes only (testbenches) using wait for statement

```
entity clk is
    port(clk: out std_logic);
end clk;

architecture cyk of clk is
begin
   process
      begin
            clk <= '0';
            wait for 10 ns;
            clk <= '1';
            wait for 10 ns;
      end process;
end cyk;
```

No inputs to process -> no sensitivity list needed

Process: Allowed Statements

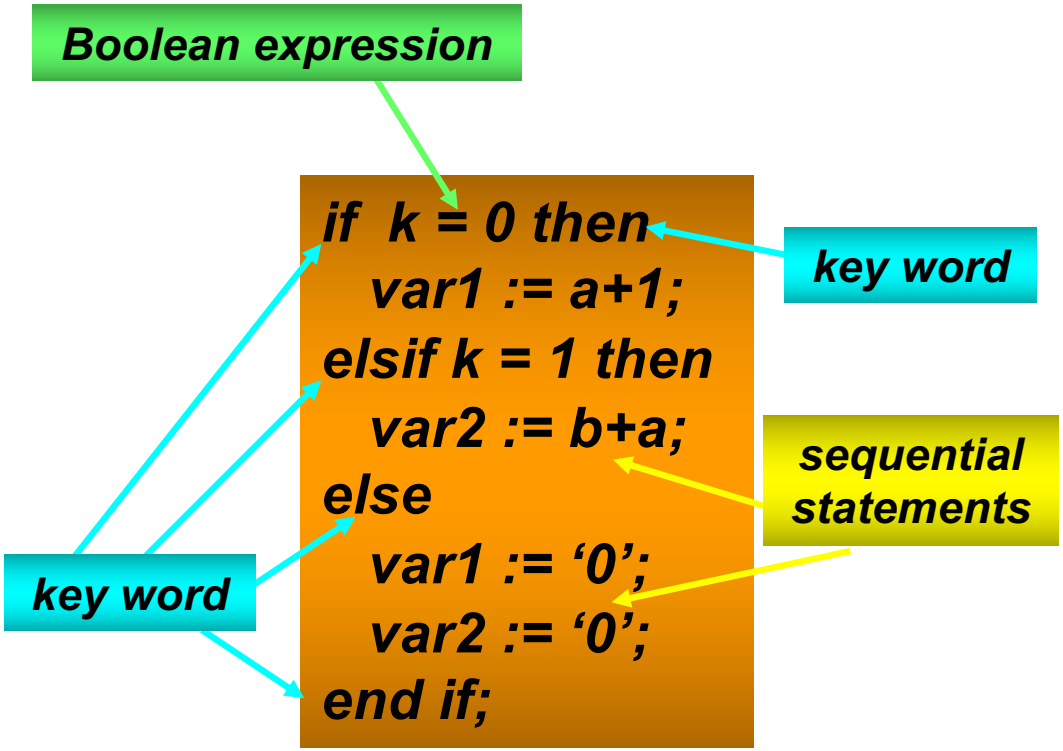
- Only following sequential statements allowed inside process:
  - **if** statement
  - **case** statement
  - **wait** statement
  - **loop** statement (for, while)
- Additionally allowed signal and variable assignments

If Statement

- Used to select sequence of statements for execution based on fulfilling some condition
  - Condition – any expression evaluating to Boolean value

```
if boolean-expression then
    sequential-statements
{elsif boolean-expression then  -- elsif clause; if can have 0 or
    sequential-statements}          -- more elsif clauses
[else
    sequential-statements]
end if;
```

If/else - Syntax



Modeling of rising clock edge

```
if ((clk'event) and (clk = '1')) then
   q1 <= d1;
   q2 <= d2;
end if;
```

Case Statement

- Concurrent statement

```
Case expression is
    when choices => sequential-statements   -- branch 1
    when choices => sequential-statements   -- branch 2
    -- any number of branches can be specified
    [when others => sequential-statements]   -- last branch
end case;
```

- Only one branch with matching condition selected for execution

If vs. Case

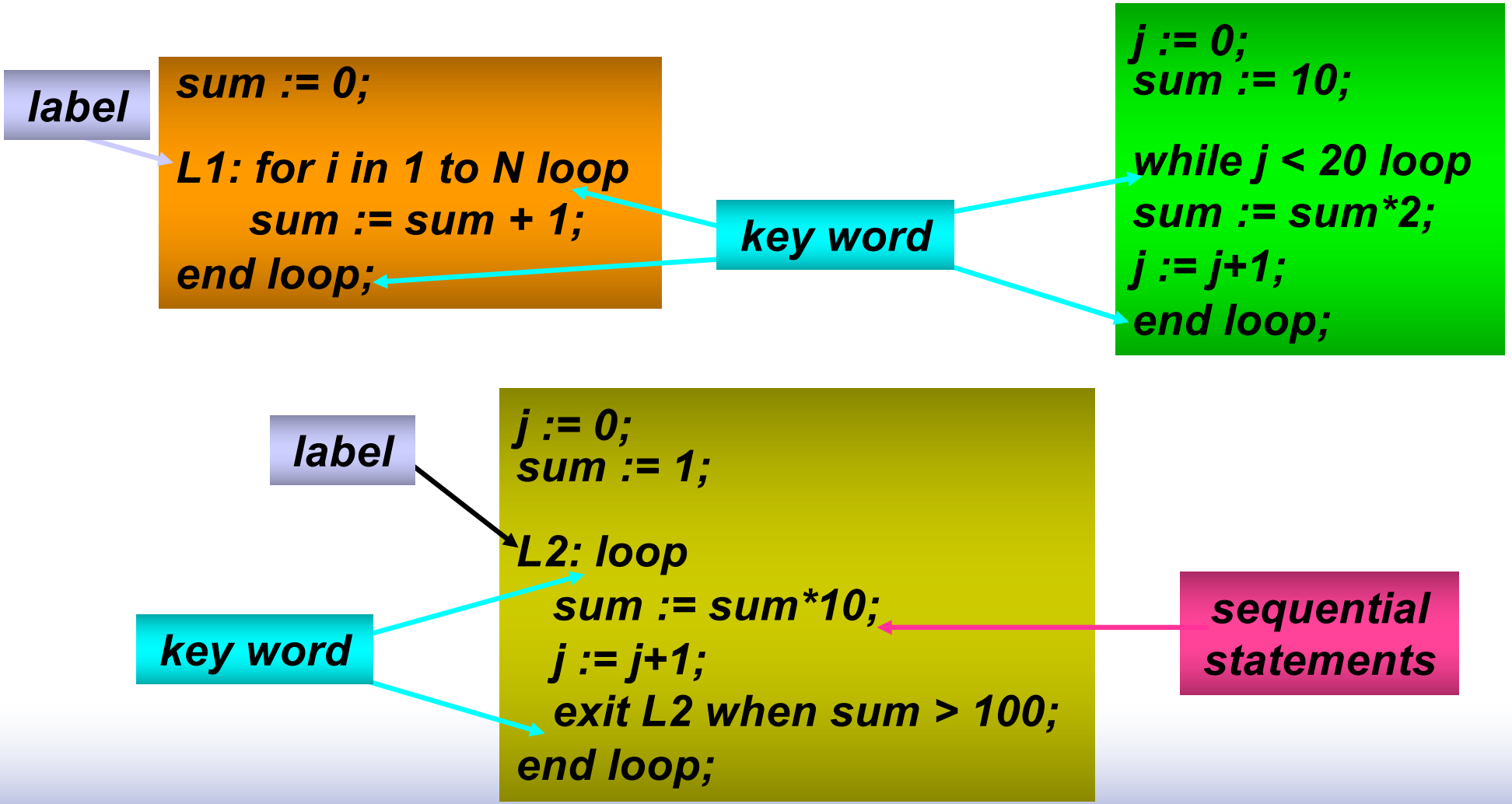
- **If/else**'s are synthesized as cascaded MUXes
  - If area is critical for design use if/else structure
    - Result – small but slow design
  - If time is critical for design use case statements
    - Result – fast but bigger designs

Loop Statements

- Used to iterate through a set of sequential statements
- Syntax

```
[loop label: ] iteration-scheme loop
    sequential-statements
end loop [loop label];
```

Loop - Three Forms



Loop For

- Syntax

```
for identifier in range
```

- Example: calculation of y^N, $N > 2$

```
power_N := y;
for i in 2 to N loop
    power_N := power_N * y;
end loop;
```

  - Body of loop executed N-1 times

Loop While

- Syntax

```
while boolean-expression
```

- Example:

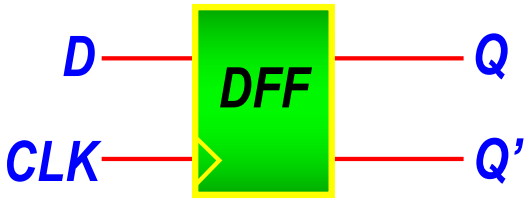
```
power_N := y;
while i < N loop
    power_N := power_N * y;
end loop;
```

Multiple Signal Assignments

- Signals can be assigned values several times within process
  - All of intermediate assignments invalid
  - Value assigned as last in single process execution cycle counts
    - ***Signal assignment does not take place until process execution finishes***
- ***If value of data must be changed several times in one process use variable instead of signal to represent it***

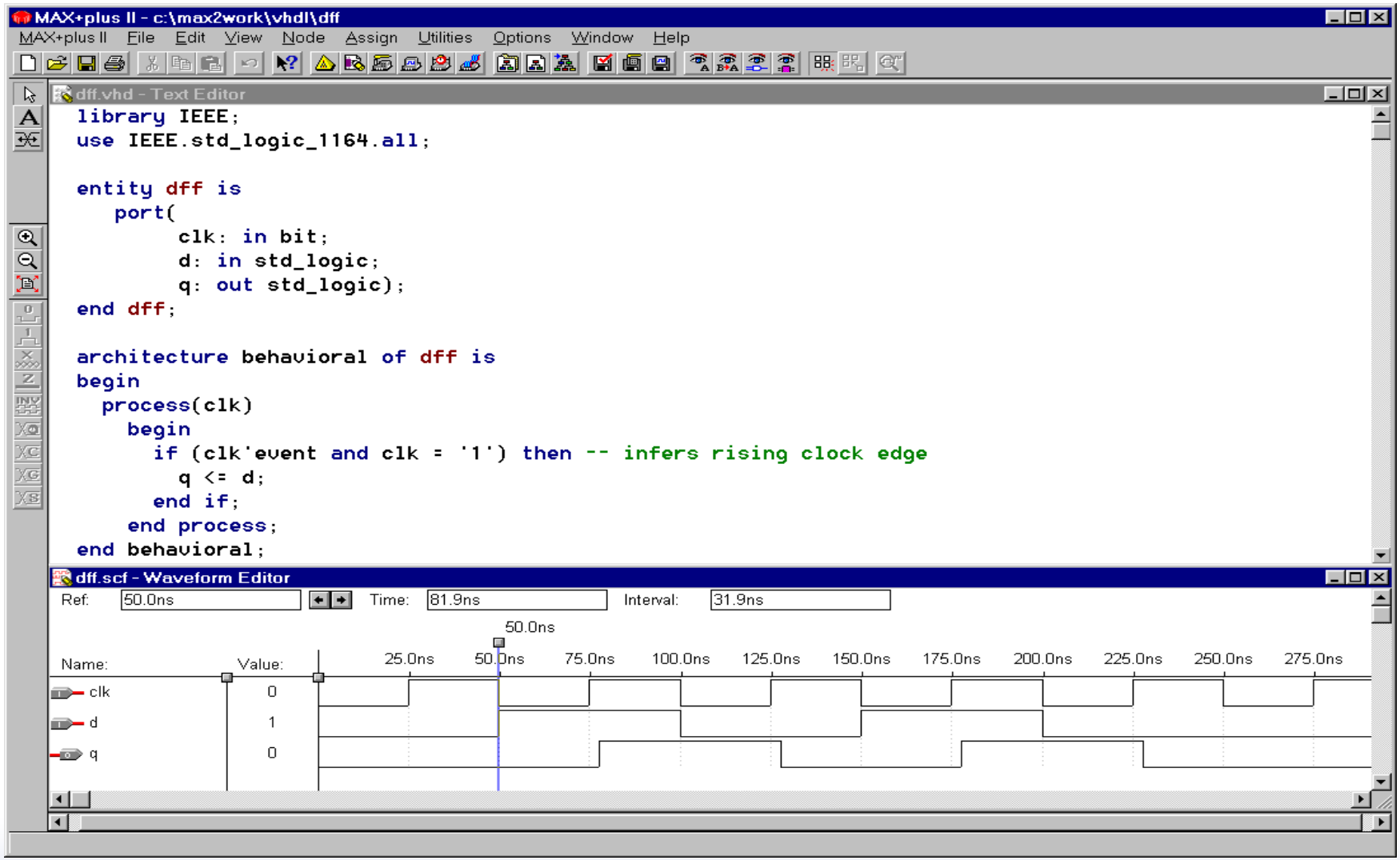
Example: D Flip-Flop

- D flip-flop (DFF) basic storage element
  - Inputs: D and clk
  - Outputs Q, Q'

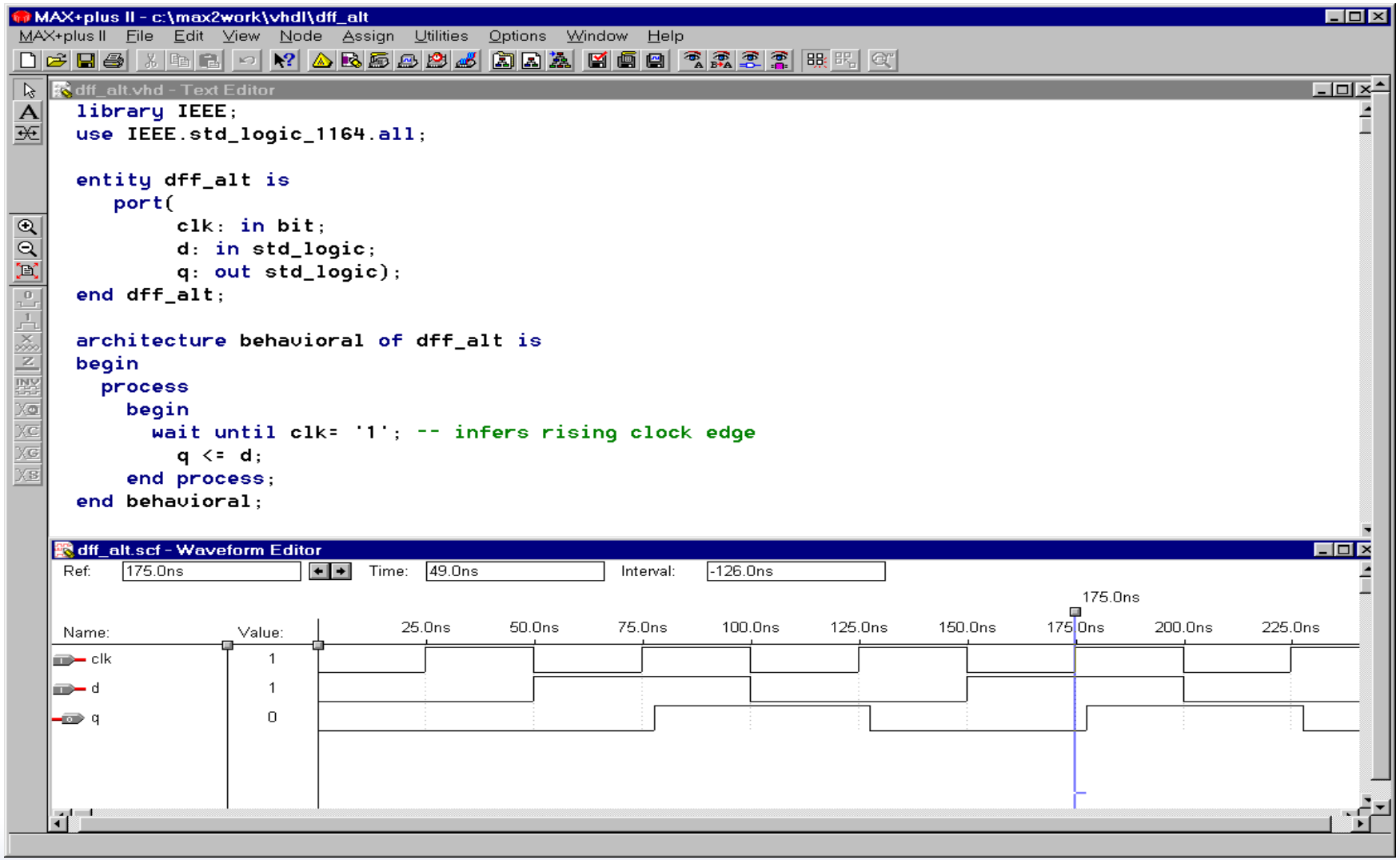


- Stored value updates with input data at line D only with rising clock edge (alternative designs using falling clock edge)
  - For all other clock values DFF retains its previous value assignment

Example: DFF – VHDL Code



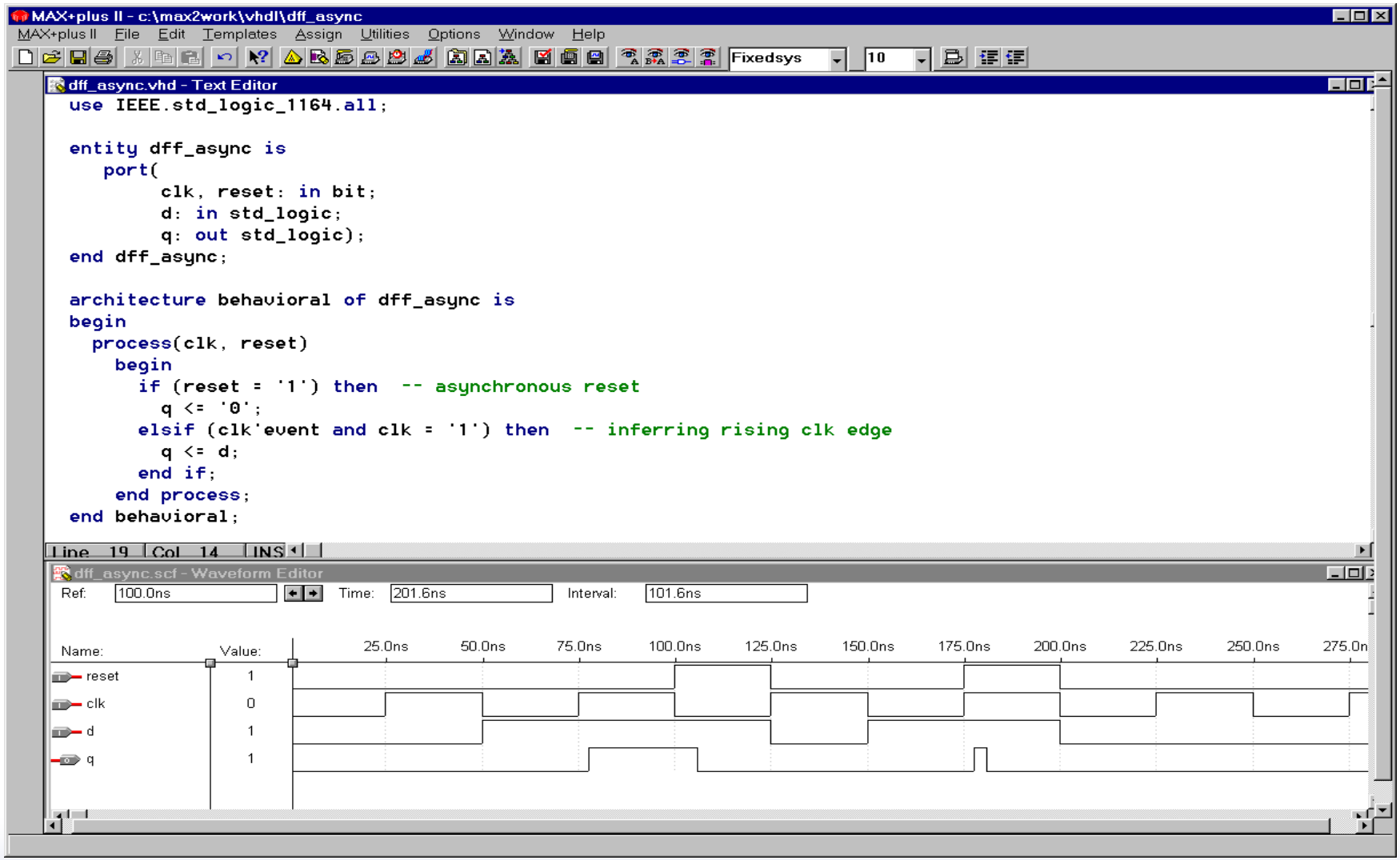
Example: DFF – Alternative Clocking Scheme



Asynchronous Reset

- Used to initialize (bring to known state) flip-flops at the beginning of design operation
  - Sometimes applied also during design operation
- Immediate effect upon encountering reset signal
  - Reset takes place immediately, regardless of the actual value of clock

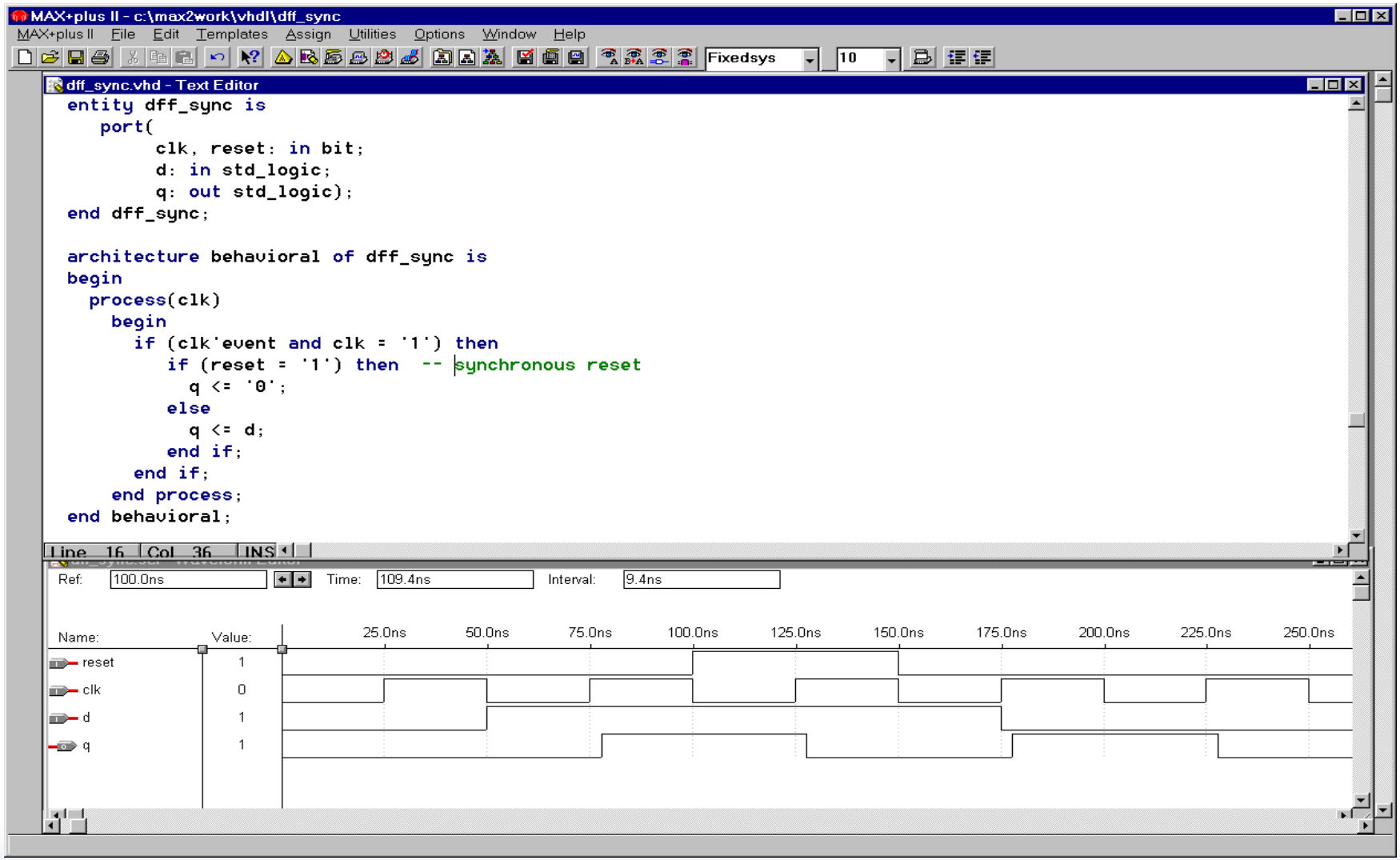
Example: Asynchronous Reset



Synchronous Reset

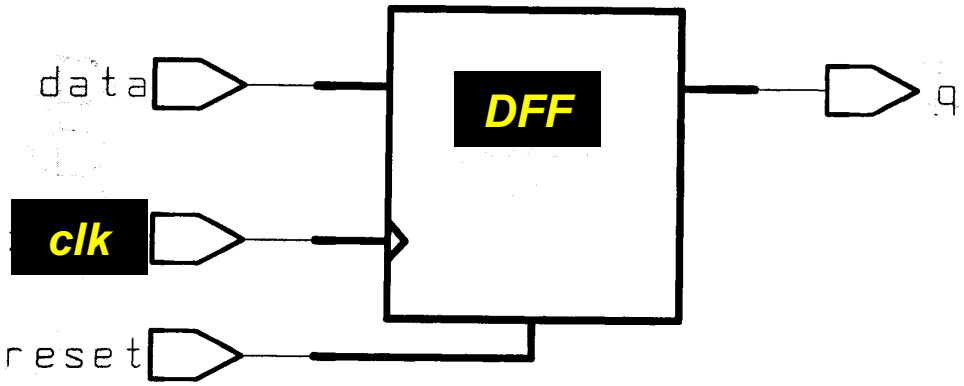
- Similarities with asynchronous case:
  - Used to reset value stored in DFF
- Differences form asynchronous case:
  - DFF reseted ONLY when
    - Reset signal high
    - Rising clock edge
      - Falling clock edge for alternative clocking scheme
- Process sensitive ***only*** on clock signal

Example: Synchronous Reset



Synthesized Circuits

DFF with asynchronous reset



DFF with synchronous reset