

# *Chapter 3*

## *Behavioral VHDL*

# ***Behavioral VHDL***

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

# Three Styles of VHDL

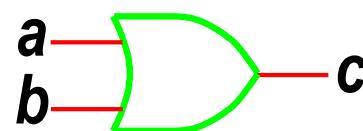
## Behavioral

**Boolean**  
 $c = a \vee b$

**VHDL**  
 $c \leq a \text{ OR } b;$

## Structural

**RTL level**

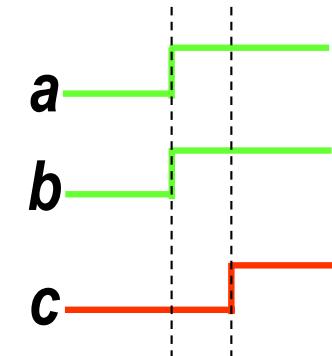


**VHDL**

```
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**



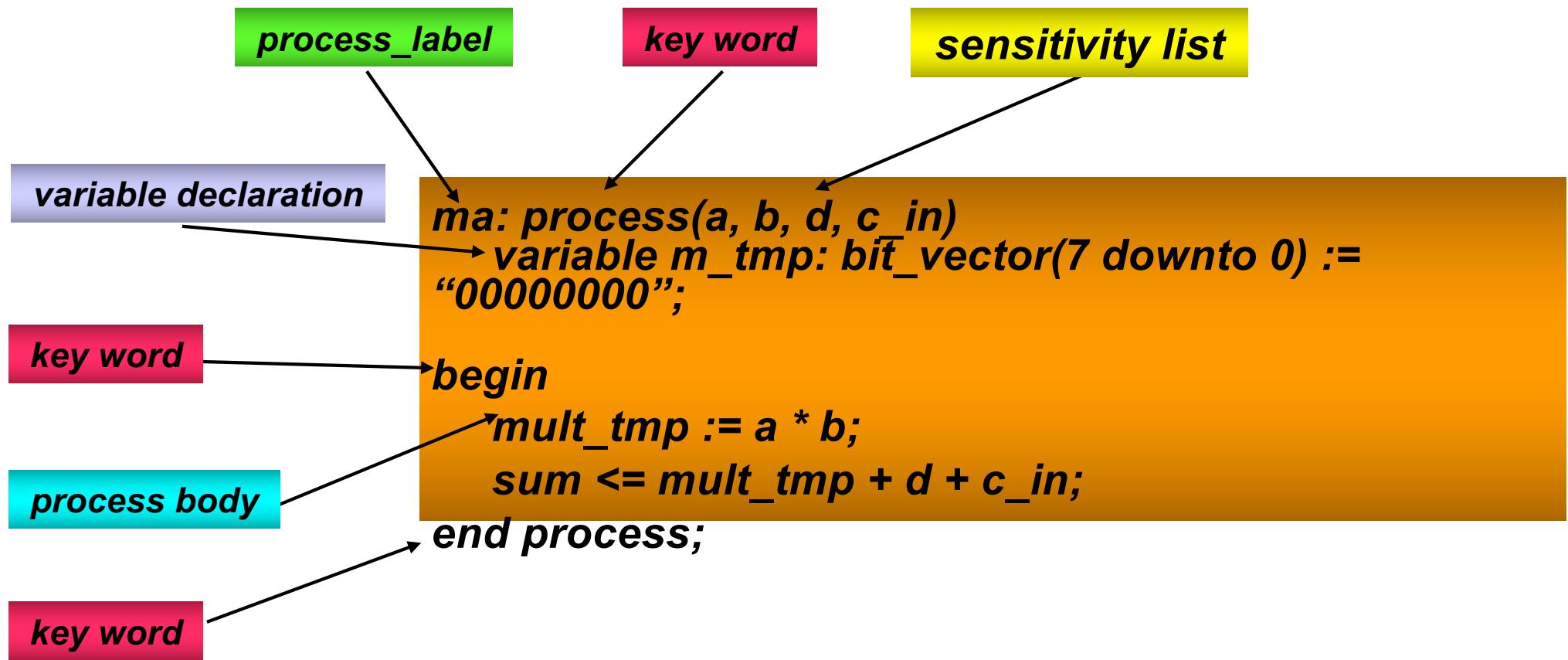
**VHDL**

```
 $c \leq a \text{ OR } b \text{ after } 10 \text{ ns};$ 
```

# ***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

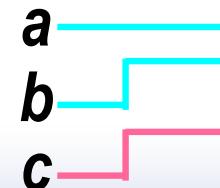
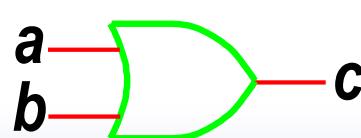
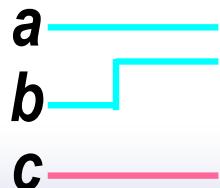
```
or_gate: process (a)
begin
  c <= a or b;
end process;
```

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

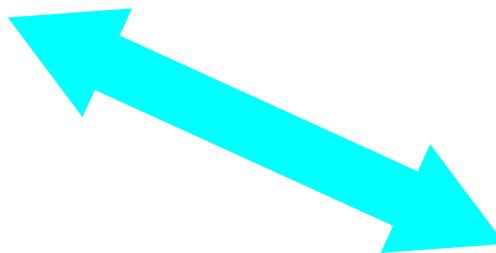
# **Wait On Sensitivity-List For Time-Expression**

*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

**No inputs to process ->  
no sensitivity list needed**

```
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;
```

# **Process: Allowed Statements**

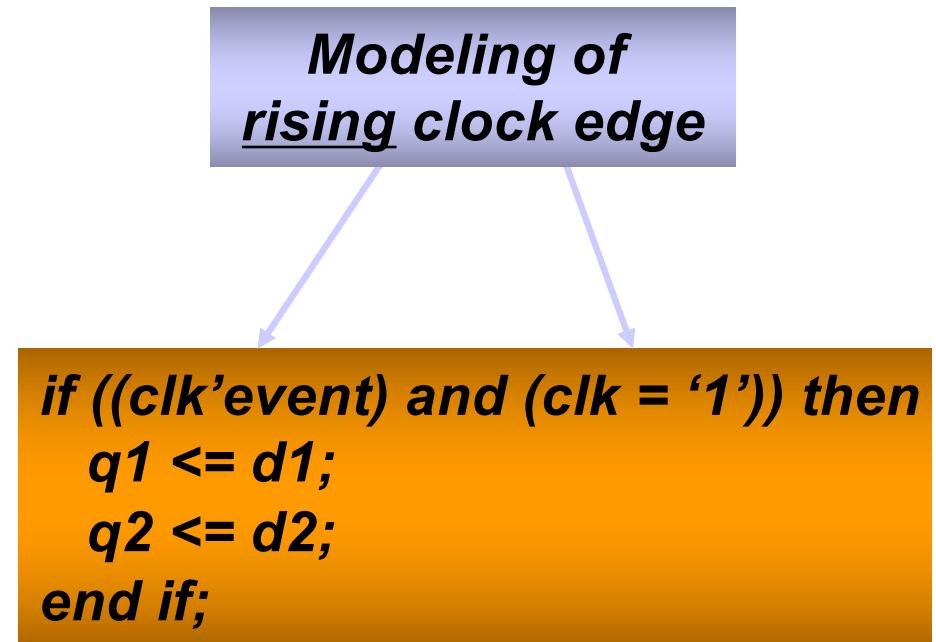
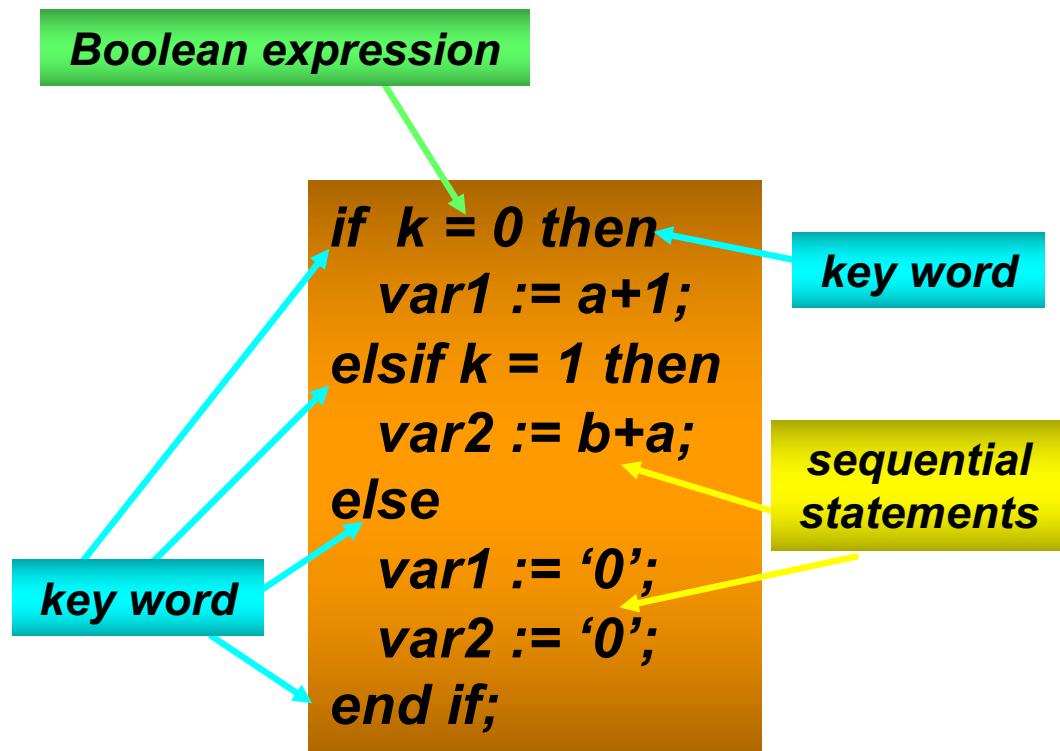
- Only following sequential statements allowed inside process:
  - **if** statement
  - **case** statement
  - **wait** statement
  - **loop** statement (for, while)
- Additionally allowed signal and variable assignments

# 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



# 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

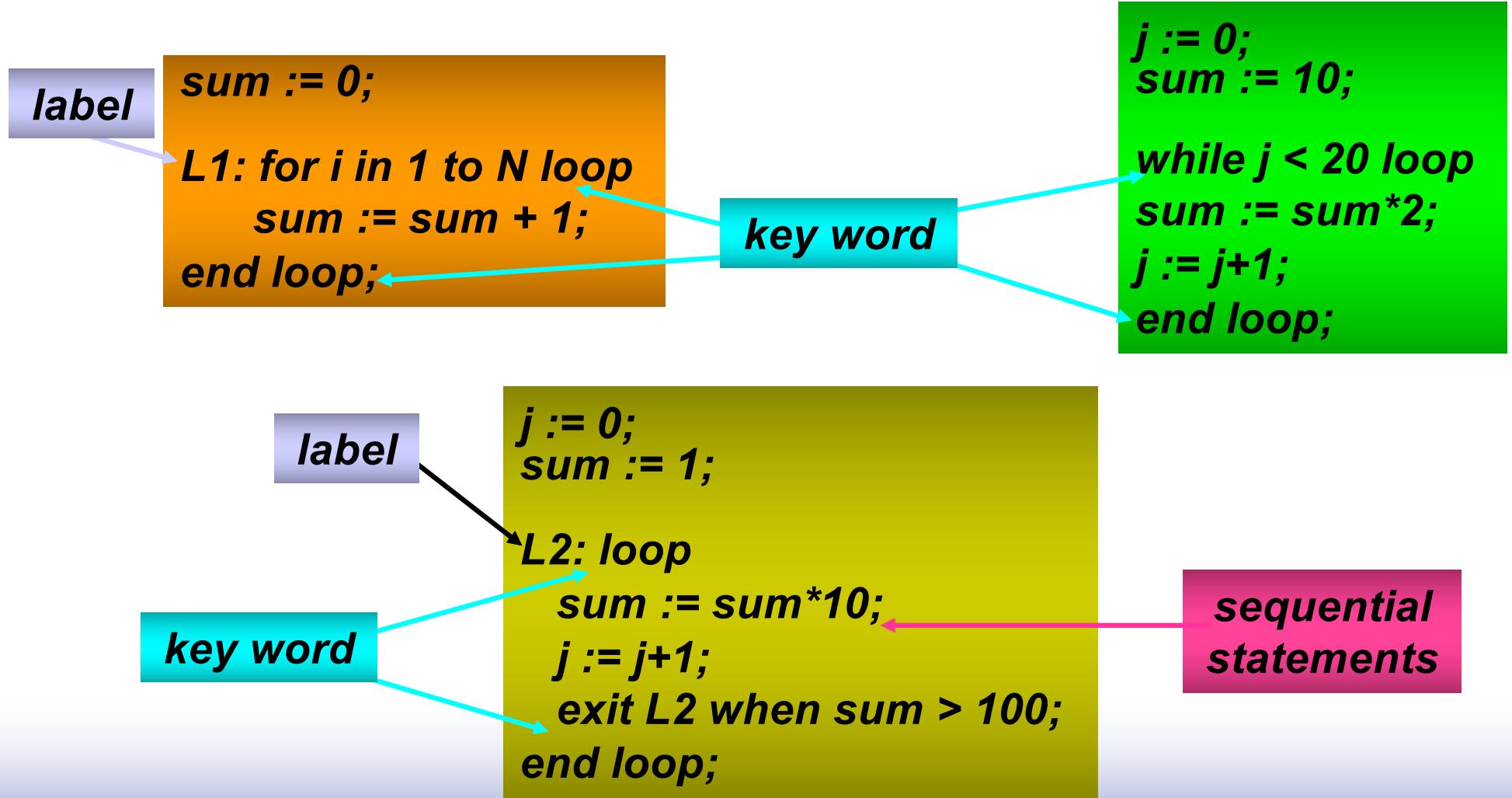
- **If/else's** are synthesized as cascaded MUXes
  - If area is critical for design use if/else structure
    - Result – small but slow design
  - If time is critical for design use case statements
    - Result – fast but bigger designs

# **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;
```

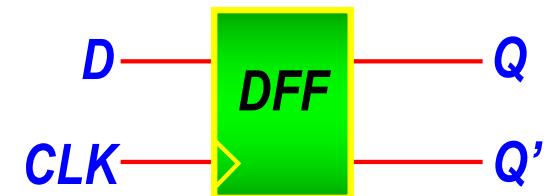
# **Multiple Signal Assignments**

- 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

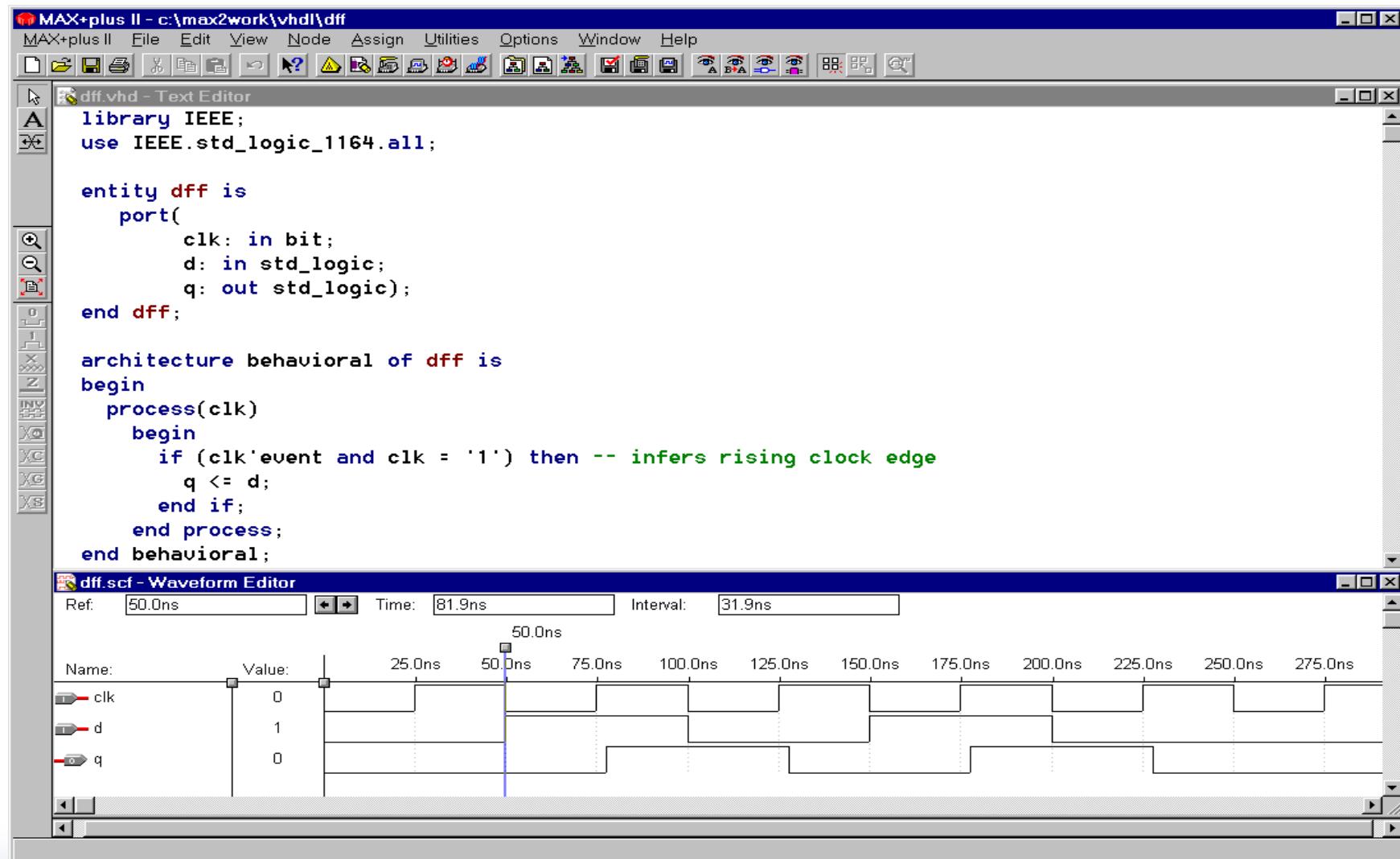
- Inputs: D and clk
  - Outputs Q, Q'



- 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

The screenshot shows the MAX+plus II software interface. The top window is a Text Editor displaying VHDL code for a DFF alternative clocking scheme. The bottom window is a Waveform Editor showing the timing of the clock (clk), data (d), and output (q) signals.

**VHDL Code (Text Editor):**

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dff_alt is
  port(
    clk: in bit;
    d: in std_logic;
    q: out std_logic);
end dff_alt;

architecture behavioral of dff_alt is
begin
  process
  begin
    wait until clk= '1'; -- infers rising clock edge
    q <= d;
  end process;
end behavioral;
```

**Waveform Editor (Waveform Editor):**

Ref: 175.0ns Time: 49.0ns Interval: -126.0ns

Name:	Value:
clk	1
d	1
q	0

Timing Data (approximate values):

Time (ns)	clk (Value)	d (Value)	q (Value)
0.0	0	1	0
25.0	1	1	1
50.0	0	1	1
75.0	1	0	0
100.0	0	0	0
125.0	1	0	0
150.0	0	0	0
175.0	1	0	0
200.0	0	0	0
225.0	1	0	0

# **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

MAX+plus II - c:\max2work\vhdl\dff\_async

dff\_async.vhd - Text Editor

```
use IEEE.std_logic_1164.all;

entity dff_async is
  port(
    clk, reset: in bit;
    d: in std_logic;
    q: out std_logic);
end dff_async;

architecture behavioral of dff_async is
begin
  process(clk, reset)
  begin
    if (reset = '1') then -- asynchronous reset
      q <= '0';
    elsif (clk'event and clk = '1') then -- inferring rising clk edge
      q <= d;
    end if;
  end process;
end behavioral;
```

Line 19 | Col 14 | INS |

dff\_async.scf - Waveform Editor

Name:	Value:	25.0ns	50.0ns	75.0ns	100.0ns	125.0ns	150.0ns	175.0ns	200.0ns	225.0ns	250.0ns	275.0ns
reset	1											
clk	0	1	0	1	0	1	0	1	0	1	0	1
d	1											
q	1											

The screenshot shows the MAX+plus II software interface. The top window is a Text Editor displaying VHDL code for a DFF with an asynchronous reset. The bottom window is a Waveform Editor showing the timing of four signals: reset, clk, d, and q. The reset signal is high from 0 to 50 ns. The clk signal has events at 25 ns, 50 ns, 75 ns, 100 ns, 125 ns, 150 ns, 175 ns, 200 ns, 225 ns, and 250 ns. The d signal is high from 50 ns to 100 ns. The q signal remains high during the reset period and then follows the d signal's value.

# **Synchronous Reset**

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

# Example: Synchronous Reset

The screenshot shows the MAX+plus II software interface. The top menu bar includes File, Edit, Templates, Assign, Utilities, Options, Window, and Help. The toolbar below has various icons for file operations and design tools. The main window title is "MAX+plus II - c:\max2work\vhdl\dff\_sync". The text editor tab is open, displaying the VHDL code for a synchronous reset DFF:

```
entity dff_sync is
  port(
    clk, reset: in bit;
    d: in std_logic;
    q: out std_logic);
end dff_sync;

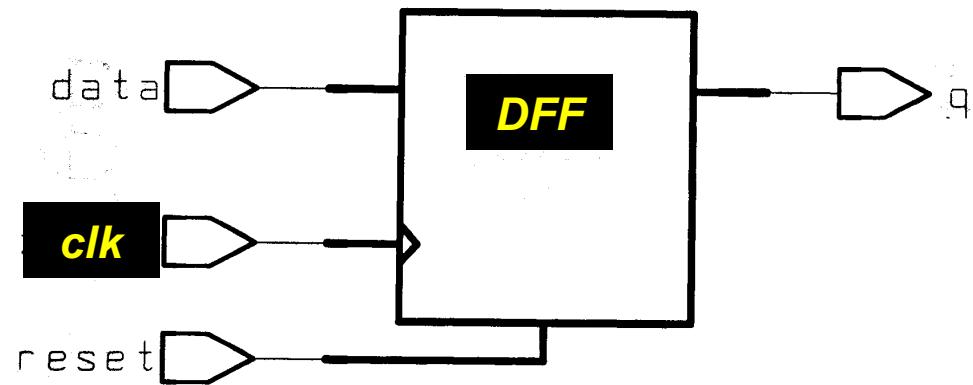
architecture behavioral of dff_sync is
begin
  process(clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then -- synchronous reset
        q <= '0';
      else
        q <= d;
      end if;
    end if;
  end process;
end behavioral;
```

The waveform viewer at the bottom shows the timing of the signals. The time scale is from 0.0ns to 250.0ns. The signals are:

Name	Value	25.0ns	50.0ns	75.0ns	100.0ns	125.0ns	150.0ns	175.0ns	200.0ns	225.0ns	250.0ns
reset	1	1	1	1	1	1	1	1	1	1	1
clk	0	0	1	1	0	0	1	1	0	0	1
d	1	1	1	1	1	1	1	1	1	1	1
q	1	1	1	0	1	1	0	1	1	0	1

# Synthesized Circuits

*DFF with  
asynchronous reset*



*DFF with  
synchronous reset*

