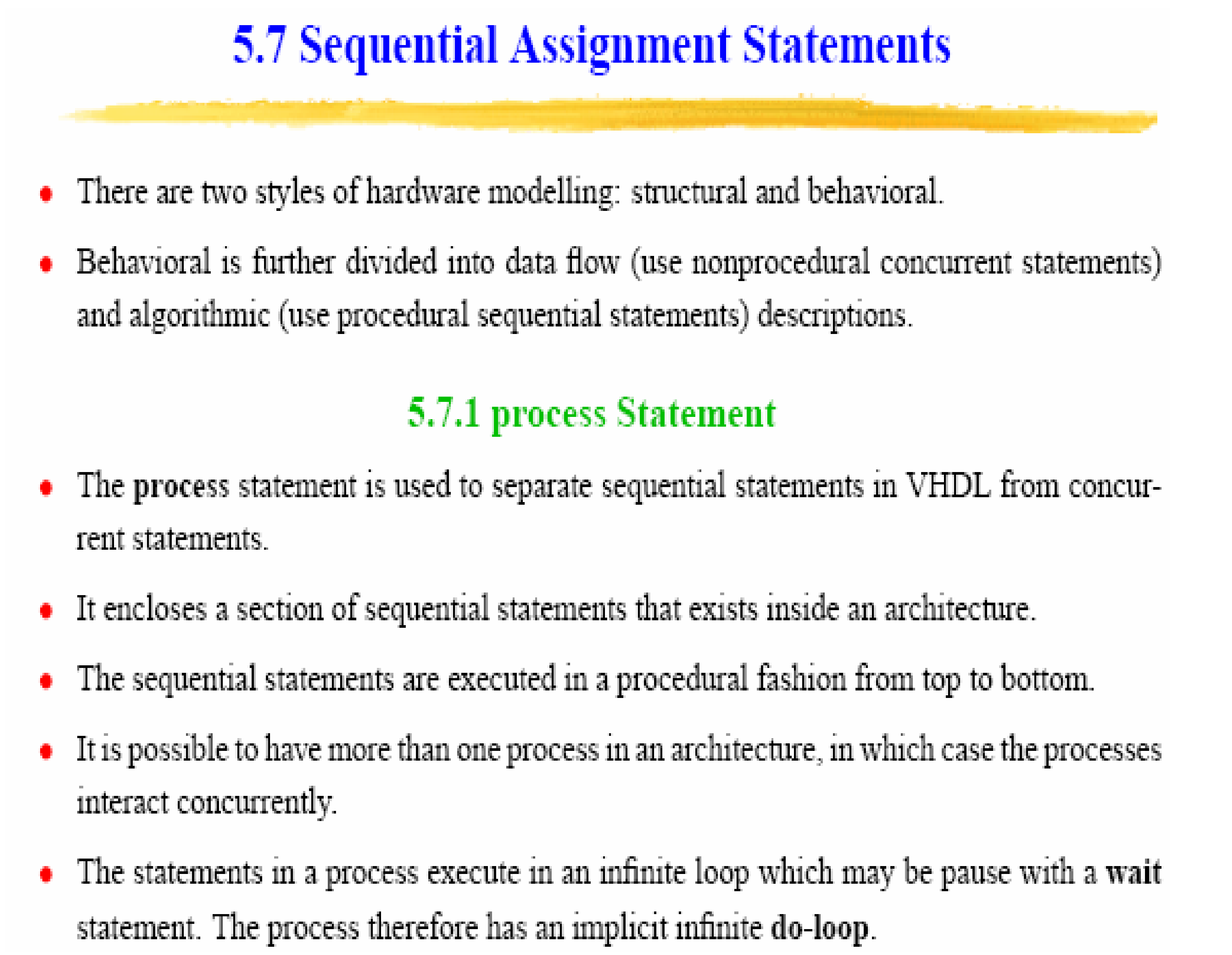


5.7 Sequential Assignment Statements

- There are two styles of hardware modelling: structural and behavioral.
- Behavioral is further divided into data flow (use nonprocedural concurrent statements) and algorithmic (use procedural sequential statements) descriptions.

5.7.1 process Statement

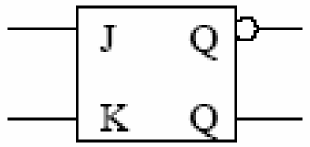
- The **process** statement is used to separate sequential statements in VHDL from concurrent statements.
- It encloses a section of sequential statements that exists inside an architecture.
- The sequential statements are executed in a procedural fashion from top to bottom.
- It is possible to have more than one process in an architecture, in which case the processes interact concurrently.
- The statements in a process execute in an infinite loop which may be pause with a **wait** statement. The process therefore has an implicit infinite **do-loop**.

- The syntax of a process is:

```
[PROCESS_LABEL:]
process [(SIGNAL_NAME {, SIGNAL_NAME})]   -- Sensitivity list
   -- Declarative part (only available within the process)
   [variable declarations]
begin
   -- Statement part
   [Simple signal assignment statements]
   [Variable assignment statements]
   [if statements]
   [case statements]
   [loop statements]
   [wait statement]
end process [PROCESS_LABEL];
```

- The two architectures on the following slide perform the same functionality.
- However, HB1 contains a conditional concurrent statement, while in HB2 this statement is defined in a process statement. The process waits for a change in J, K, Q, or Q_BAR.

A JK-Latch Example



```
entity JK_LATCH is
   port (J, K : in BIT;
         Q : inout := '0';
         Q_BAR : inout := '1');
end JK_LATCH;

architecture BH1 of JK_LATCH is
begin
   Q <= Q when (J='0' and K='0') else
       '0' when (J='0' and K='1') else
       '1' when (J='1' and K='0') else
       Q_BAR;
   Q_BAR <= not Q;
end BH1;
```

```
architecture BH2 of JK_LATCH is
begin
   process
   begin
      if (J='0' and K='0') then
         Q <= Q;
      elsif (J='0' and K='1') then
         Q <= '0';
      elsif (J='1' and K='0') then
         Q <= '1';
      else
         Q <= Q_BAR;
      end if;

      wait on J, K, Q, Q_BAR;
   end process;
   Q_BAR <= not Q;
end BH2;
```

if, case and loop Statements

- These are sequential statements used for purposes of selection and repetition in the programming code.
- For the **if** statement, the condition expressions are evaluated in order until one is met. For the **case** statement, all alternatives have equal priority.
- Syntax of **if**:

```
if expression then
   statement;
   { statement; }
elsif expression then
   statement;
   { statement; }
else
   statement;
   { statement; }
end if;
```

- Syntax of **case**:

```
case expression is
   when constant_value =>
      statement;
      { statement; }
   when constant_value =>
      statement;
      { statement; }
   when others =>
      statement;
      { statement; }
end case;
```

- There are three forms of the **loop** statement in VHDL. Each have the following syntax:
- The **for-loop** executes a set of sequential statements a certain number of times:

```
[LOOP_LABEL:]
for variable_name in range loop
   statement;
   { statement; }
end loop [LOOP_LABEL];
```

- The **while-loop** repeatedly executes a set of statements while the condition is satisfied:

```
[LOOP_LABEL:]
while boolean_expression loop
   statement;
   { statement; }
end loop [LOOP_LABEL];
```

- The infinite-loop is similar to the above, but simply uses the **loop** statement alone. i.e.:

```
[LOOP_LABEL:]
loop
   statement;......
```

The Use of Variables in a process

- **variable** data objects are not related to a wire in a circuit.
- Variables can be use in ways that **signal** objects can't.
- Values are assigned to variables using the := operator, rather than the <= operator used for signals.
- For example, let us consider a design which counts the number of 1's in a 3-bit input:

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

entity NUMBITS is
    port(X : in STD_LOGIC_VECTOR(1 to 3);
        Count : buffer INTEGER range 0 to 3);
end NUMBITS;
```

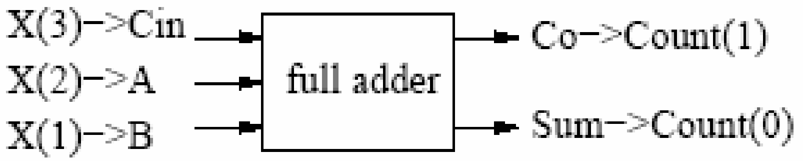
```
architecture BEHAVIOUR of NUMBITS is
begin
    process (X)     -- count the number of bits in X with the value 1
    begin
        Count <= 0;       -- the 0 with no quotes is a decimal number
        for I in 1 to 3 loop
            if X(I) = '1' then
                Count <= Count+1;
            end if;
        end loop;
    end process;
end BEHAVIOUR;
```

- Here, the signal Count is defined as **buffer** because it is used on the right hand side of the equation.
- This does not represent a sensible circuit with the "Count <= Count+1" type of statement since Count will be an unstable signal. The code should be modified as follows.

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

entity NUMBITS is
   port(X : in STD_LOGIC_VECTOR(1 to 3); Count : out INTEGER range 0 to 3);
end NUMBITS;

architecture BEHAVIOUR of NUMBITS is
begin
   process (X)    --count the number of bits in X with the value 1
      variable Tmp : INTEGER;
   begin
      Tmp := 0;      -- The assignment of variable is :=
      for I in 1 to 3 loop
         if X(I) = '1' then
            Tmp := Tmp + 1;
         end if;
      end loop;
      Count <= Tmp;
   end process;
end BEHAVIOUR;
```

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

entity NUMBITS is
   port(X : in STD_LOGIC_VECTOR(1 to 3); Count : out INTEGER range 0 to 3);
end NUMBITS;

architecture BEHAVIOUR of NUMBITS is
begin
   process (X)    --count the number of bits in X with the value 1
      variable Tmp : INTEGER;
   begin
      Tmp := 0;      -- The assignment of variable is :=
      for I in 1 to 3 loop
         if X(I) = '1' then
            Tmp := Tmp + 1;
         end if;
      end loop;
      Count <= Tmp;
   end process;
end BEHAVIOUR;
```

- We may implement this design with a hardware circuit. For example, a full adder:



5.7.4 The wait Statement and Sensitivity List

- Execution of sequential statements in a process can be paused with a **wait** statement.
  - **wait on** `X;` – Wait for the signal `X` to change value.
  - **wait until** `(X = '1');` – Wait until the signal `X` changes to the value 1.
  - **wait for** `4 ms;` – Wait for a period of time (4 ms) before continuing.
- Alternatively, a sensitivity list follows the **process** statement. Both of the following function the same by waiting for a change in `ALARM_A` or `ALARM_B`:

```
process (ALARM_A, ALARM_B)
begin
    ...
end
```

```
process
begin
    ...
    wait on ALARM_A, ALARM_B;
end
```

- When using a sensitivity list, we may not use any **wait** statements in the process.

5.8 Register and Simple Sequential Circuit Descriptions

5.8.1 D Latch and D Flip-Flop

- Latches trigger on a clock level, while flip-flops trigger on a clock edge.

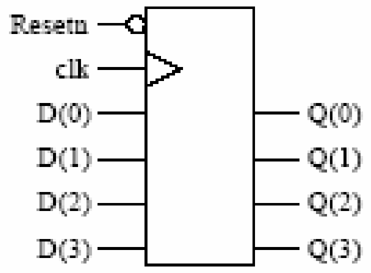
```
library ieee;                              -- D latch definition.
use ieee.std_logic_1164.all;
entity latch is
   port(D, clk : in STD_LOGIC; Q : out STD_LOGIC);
end latch;

architecture Behaviour of latch is
begin
   process (D, clk)
   begin
      if clk = '1' then
         Q <= D;
      end if
   end process;
end Behaviour;
```

- D flip-flop definition:

```
library ieee;                    -- D flip-flop definition.
use ieee.std_logic_1164.all;

entity flipflop is
   port(D, clk : in  STD_LOGIC;
        Q      : out STD_LOGIC);
end flipflop;

architecture Behaviour of flipflop is
begin
   process (clk)
   begin
      if clk'EVENT and clk = '1' then
         Q <= D;
      end if
   end process;
end Behaviour;
```

5.8.3 An Example 4-bit Register

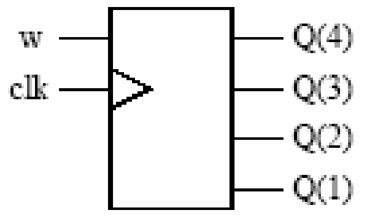
```
library ieee;                    -- A four-bit register.
use ieee.std_logic_1164.all;

entity reg4 is
   port(D : in STD_LOGIC_VECTOR(3 downto 0);
        Resetn, clk : in STD_LOGIC;
        Q : out STD_LOGIC_VECTOR(3 downto 0));
end reg4;

architecture Behaviour of reg4 is
begin
   process (Resetn, clk)
   begin
      if Resetn = '0' then
         Q <= "0000";            -- or Q <= (others => '0');
      elsif clk'EVENT and clk = '1' then
         Q <= D;
      end if;
   end process;
end Behaviour;
```



- Question: How to modify the above code so it represents an n-bit register?

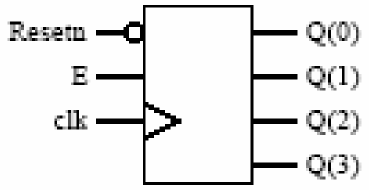
5.8.4 An Example Shift Register

```
library ieee;                          -- A four-bit shift register.
use ieee.std_logic_1164.all;

entity shift4 is
   port(w, clk : in  STD_LOGIC;
        Q      : out STD_LOGIC_VECTOR(1 to 4);
end shift4;

architecture Behaviour of shift4 is
   signal Sreg : STD_LOGIC_VECTOR(1 to 4);
begin
   process (clk)
   begin
      if clk'EVENT and clk='1' then
         Sreg(1) <= Sreg(2);
         Sreg(2) <= Sreg(3);
         Sreg(3) <= Sreg(4);
         Sreg(4) <= w;
      end if;
   end process;
   Q <= Sreg;
end Behaviour;
```



5.8.5 An Example 4-bit Counter

```
library ieee;                    -- A four-bit counter.
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count4 is
   port(Resetn, E, clk : in  STD_LOGIC;
        Q : out STD_LOGIC_VECTOR (3 downto 0));
end count4;

architecture Behaviour of count4 is
   signal Count : STD_LOGIC_VECTOR (3 downto 0);
begin
   process (clk, Resetn)
   begin
      if Resetn = '0' then
         Count <= "0000";
      elsif (clk'EVENT and clk = '1') then
         if E = '1' then
            Count <= Count + 1;
         end if;
      end if;
   end process;
   Q <= Count;
end Behaviour;
```


Resetn
E
clk
Q(0)
Q(1)
Q(2)
Q(3)