

Concurrent Statement

Natural Concept for Describing Hardware

- **Concurrent Signal Assignment**
- **Conditional Signal Assignment**
- **Selected Signal Assignment**
- **Block Statement**
- **Concurrent Assertion Statement**
- **Process Statement**

Concurrent Signal Assignment

Represent an Equivalent Process Statement

target <= expression [after time_expression] ;

- **Signals are associated with TIME**
- **With "after", the assignment is scheduled to a future simulation time**
- **Without "after", the assignment is scheduled at a DELTA TIME after the current simulation time**

Conditional Signal Assignment

More than One Expression

```
target <= { expression [ after time_exp ] when condition else }  
          expression [ after time_exp ] ;
```

- **Condition / expression except for the last expression**
- **One and only one of the expressions is used at a time**

Selected Signal Assignment

Only One Target

with choice_expression select

```
target <= { expression [ after time_exp ] when choices , }  
          expression [ after time_exp ] when choices ;
```

- "when others" is used when all the cases were not treated

Block Statement (1)

A Set of Concurrent Statements

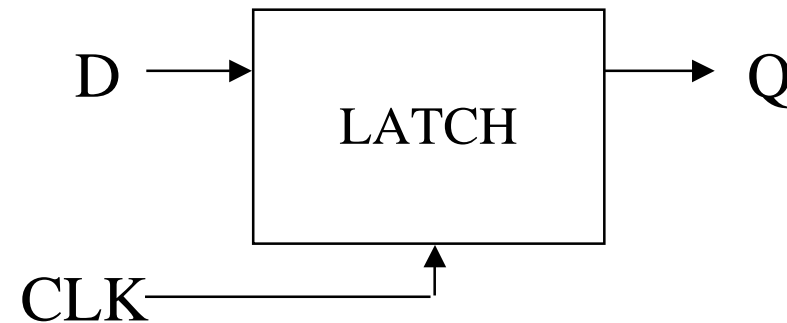
```
label : block
    { block_declarative_part }
begin
    { concurrent_statements }
end block [ label ] ;
```

- **Used to organize a set of concurrent statements hierarchically**

Block Statement (2)

In Synchronous Descriptions

```
latch : block ( CLK = '1' )  
begin  
    Q <= GUARDED D ;  
end block latch ;
```



Assertion Statement

Only One Target

assert condition

```
[ report error_message ]  
[ severity severity_level ] ;
```

- **If the condition is false, it reports a diagnostic message**
- **Useful for detecting condition violation during simulation**
- **Not used in synthesis**

Process Statement (1)

A Set of Sequential Statements

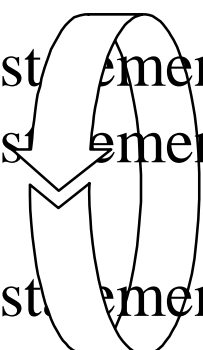
```
[ label : ] process [ ( sensitivity_list ) ]  
    { process_declarative_part }  
begin  
    { sequential_statements }  
end process [ label ] ;
```

- **All processes in a design executes CONCURRENTLY**
- **At a given time, ONLY ONE sequential statement executed within each process**
- **Communicates with the rest of a design through signals**

Process Statement (2)

A Pseudo Infinite Loop

```
process
begin
    sequential_statement_1 ;
    sequential_statement_2 ;
    -----
    sequential_statement_n ;
end process;
```



- **A Synchronization Mechanism is Needed**

Process Statement (3)

Synchronization Mecanism

```
wait  
  [ on signal_name { signal_name } ]  
  [ until boolean_expression ]  
  [ for time_expression ] ;
```

- **Objects being waited upon should be SIGNALS**

Process Statement (4)

The Sensitivity List

```
process [ ( sensitivity_list ) ]  
begin  
    { sequential_statements }  
end process ;
```

- **Equivalent to a "wait" statement as the last statement**
wait on sensitivity_list ;

Sequential Statement

Insight Into Statements within Processes

- **Variable Assignment**
- **Signal Assignment**
- **If**
- **Case**
- **Null**
- **Assertion**
- **Loop**
- **Next**
- **Exit**
- **Wait**
- **Procedure Calls**
- **Return**

Variable Assignment Statement

Immediate Assignment

`target_variable := expression ;`

- **Always executed in ZERO SIMULATION TIME**
- **Used as temporary storages**
- **Can not be seen by other concurrent statements**

Signal Assignment Statement (1)

Defines a **DRIVER** of the Signal

```
target_signal <= [ transport ] expression [ after time_expression ] ;
```

- **Within a process, ONLY ONE driver for each signal**
- **When assigned in multiple processes, it has MULTIPLE DRIVERS. A RESOLUTION FUNCTION should be defined**

Signal Assignment Statement (2)

Inertial Delay Model (Default)

```
target_signal <= expression [ after time_expression ] ;
```

- **Same TIMING aspects than concurrent signal assignment**
- **Useful in modeling devices that ignore spikes on the inputs**

Signal Assignment Statement (3)

Inertial Delay Model (Default)

```
signal S : BIT := '0' ;  
process  
    S <= '1' after 5 ns ;  
    S <= '0' after 10 ns ;  
end ;
```

Overrides the first
assignment

```
signal S : BIT := '0' ;  
process  
    S <= '0' after 10 ns ;  
    S <= '1' after 5 ns ;  
end ;
```

Overrides the first
assignment

Signal Assignment Statement (4)

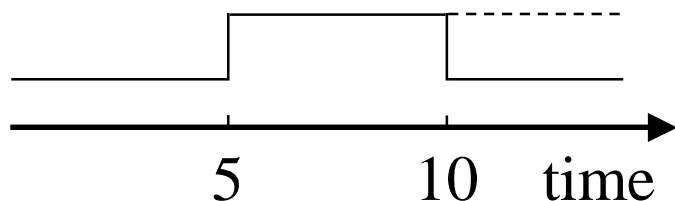
Transport Delay Model

`target_signal <= transport expression [after time_expression] ;`

```

signal S : BIT := '0' ;
process
  S <= '1' transport after 5 ns;
  S <= '0' transport after 10 ns;
end ;

```



```

signal S : BIT := '0' ;
process
  S <= '1' transport after 10 ns;
  S <= '0' transport after 5 ns;
end ;

```



CONCLUSION (1)

- VHDL is an OPEN language with many features
- With VHDL, any discrete system can be modeled

CONCLUSION (2)

Each user has its own needs depending on:

- His background
- His environment

We defined a SUBSET of VHDL