Hardware Modeling [VU] (191.011) – WS25 –

Block Statements

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 16:03 (f8a58e9)

Introduction

- Concurrent statement (like processes, instantiations, concurrent signal assignments, etc.)

Introduction

- Concurrent statement (like processes, instantiations, concurrent signal assignments, etc.)
- Blocks group concurrent statements

Introduction

- Concurrent statement (like processes, instantiations, concurrent signal assignments, etc.)
- Blocks group concurrent statements
- Restrict scope of objects (e.g., signals) within an architecture

Introduction

- Concurrent statement (like processes, instantiations, concurrent signal assignments, etc.)
- Blocks group concurrent statements
- Restrict scope of objects (e.g., signals) within an architecture
- Can be viewed as “inline module” or “module light” (combined module declaration and instantiation)

Introduction

- Concurrent statement (like processes, instantiations, concurrent signal assignments, etc.)
- Blocks group concurrent statements
- Restrict scope of objects (e.g., signals) within an architecture
- Can be viewed as “inline module” or “module light” (combined module declaration and instantiation)
- Can be loosely compared to inner (nested) classes in e.g., Java

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!
- Optional guard condition (not covered in this course) ◇617

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!
- Optional guard condition (not covered in this course) ◇617
- Optional block header: explicit block interface specification

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!
- Optional guard condition (not covered in this course) ◇617
- Optional block header: explicit block interface specification
- Declarative/statement part
  - can contain the same objects as in the respective parts of architectures → blocks can be nested

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!
- Optional guard condition (not covered in this course) ◇617
- Optional block header: explicit block interface specification
- Declarative/statement part
  - can contain the same objects as in the respective parts of architectures → blocks can be nested
  - cannot be accessed from outside the block

Block Statement - Syntax

- Block syntax

```
BLOCK_LABEL : block[ ( guard_condition ) ] [ is ]
  block_header
  block_declarative_part
begin
  block_statement_part
end block;
```

- Label is **not** optional!
- Optional guard condition (not covered in this course) 617
- Optional block header: explicit block interface specification
- Declarative/statement part
  - can contain the same objects as in the respective parts of architectures → blocks can be nested
  - cannot be accessed from outside the block
  - can access objects from outer scope

Block Statement - Example

```
architecture arch of demo is
```

Block Statement - Example

```
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
```

Block Statement - Example

```
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
 full_adder : block
```

Block Statement - Example

```vhdl
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
 full_adder : block
  signal x, y, z : std_ulogic;
```

Block Statement - Example

```
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
 full_adder : block
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b;
  y <= a and b;
  sum <= cin xor x;
  z <= cin and x;
  cout <= y or z;
 end block;
```

Block Statement - Example

```
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
 full_adder : block
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b;
  y <= a and b;
  sum <= cin xor x;
  z <= cin and x;
  cout <= y or z;
 end block;

 -- do something with a, b, etc.
 some_logic: process(all)
  [...]
end architecture;
```

Block Statement - Example

```vhdl
architecture arch of demo is
 signal a, b, cin : std_ulogic;
 signal cout, sum : std_ulogic;
begin
 full_adder : block
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b;
  y <= a and b;
  sum <= cin xor x;
  z <= cin and x;
  cout <= y or z;
 end block;

 -- do something with a, b, etc.
 some_logic: process(all)
  [...]
end architecture;
```

Note

The process `some_logic` **cannot** access the signals `x`, `y` and `z`.

Block Header

- Defines an explicit interface to a block

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
  [ generic ([...]); [ generic map ([...]); ] ]
  [ port ([...]); [ port map ([...]); ] ]
```

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
   [ generic ([...]); [ generic map ([...]); ] ]
   [ port ([...]); [ port map ([...]); ] ]
```

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
  [ generic ([...]); [ generic map ([...]); ] ]
  [ port ([...]); [ port map ([...]); ] ]
```

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
  [ generic ([...]); [ generic map ([...]); ] ]
  [ port ([...]); [ port map ([...]); ] ]
```

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
  [ generic ([...]); [ generic map ([...]); ] ]
  [ port ([...]); [ port map ([...]); ] ]
```

- Everything is optional
  - Port/generic map clause only valid if a port/generic clause is present
  - If port/generic map clauses are omitted the respective port/generic clause must have default values

Block Header

- Defines an explicit interface to a block
- Block header syntax

```
block_header ::=
  [ generic ([...]); [ generic map ([...]); ] ]
  [ port ([...]); [ port map ([...]); ] ]
```

- Everything is optional
  - Port/generic map clause only valid if a port/generic clause is present
  - If port/generic map clauses are omitted the respective port/generic clause must have default values
- Block header does not prevent the block from accessing objects from the outer scope

Block Header - Example

```
architecture arch2 of demo is
```

Block Header - Example

```
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
```

Block Header - Example

```
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
 full_adder : block
  port (
   a, b, cin : in std_ulogic;
   sum, cout : out std_ulogic
  );
```

Block Header - Example

```vhdl
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
 full_adder : block
  port (
   a, b, cin : in std_ulogic;
   sum, cout : out std_ulogic
  );
  port map (
   a => i1, b => i2, cin => i3,
   cout => o1, sum => o2
  );
```

Block Header - Example

```vhdl
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
 full_adder : block
  port (
   a, b, cin : in std_ulogic;
   sum, cout : out std_ulogic
  );
  port map (
   a => i1, b => i2, cin => i3,
   cout => o1, sum => o2
  );
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b; y <= a and b;
  z <= cin and x;
  cout <= y or z; sum <= cin xor x;
 end block;
 [...]
end architecture;
```

Block Header - Example

```
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
 full_adder : block
  port (
   a, b, cin : in std_ulogic;
   sum, cout : out std_ulogic
  );
  port map (
   a => i1, b => i2, cin => i3,
   cout => o1, sum => o2
  );
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b; y <= a and b;
  z <= cin and x;
  cout <= y or z; sum <= cin xor x;
 end block;
 [...]
end architecture;
```

Note

Block is now self-contained. We no longer need to directly access signals from the architecture.

Block Header - Example

```
architecture arch2 of demo is
 signal i1, i2, i3, o1, o2 : std_ulogic;
begin
 full_adder : block
  port (
   a, b, cin : in std_ulogic;
   sum, cout : out std_ulogic
  );
  port map (
   a => i1, b => i2, cin => i3,
   cout => o1, sum => o2
  );
  signal x, y, z : std_ulogic;
 begin
  x <= a xor b; y <= a and b;
  z <= cin and x;
  cout <= y or z; sum <= cin xor x;
 end block;
 [...]
end architecture;
```

Note

Block is now self-contained. We no longer need to directly access signals from the architecture.

Note

Trivial to move the block into a separate module (entity / architecture).

Lecture Complete!