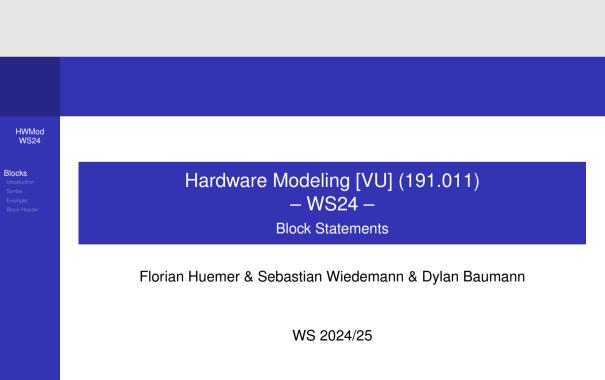In this lecture, we will explore block statements, a useful feature in VHDL that enhances code structure and organization.

HWMod
WS24

Blocks
Introduction
Syntax
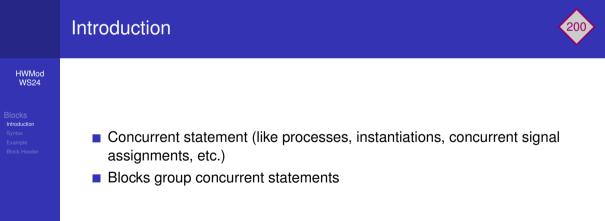Example
Block Header

Hardware Modeling [VU] (191.011)
– WS24 –
Block Statements

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2024/25

Block statements – often simply referred to as blocks – are concurrent statements. As a quick reminder, concurrent statements in VHDL are statements that can be used in the statement part of architectures. So far we have encountered processes, instances and concurrent signal assignments that fall into this category.
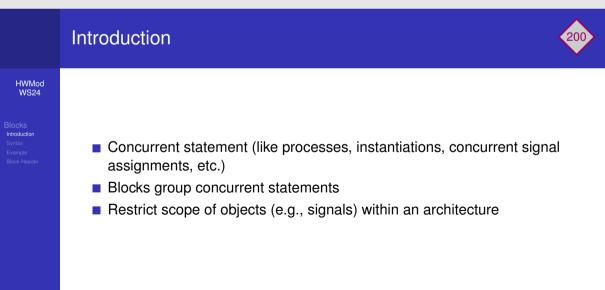
Introduction

200

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

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

Block statements essentially allow the grouping of a set of concurrent statements. Hence, they can be used to organize sections of code, making it more readable and easier to manage, especially in complex designs.

Introduction

200

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

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

1

They further allow to restrict the scope of certain VHDL objects, like signals, constants, types or subprograms to certain parts of an architecture. This can help, with reducing the likelihood of naming conflicts, and improves modularity. Confining an object's scope to a particular block, can reduce the potential for errors and simplify debugging and testing, especially in large and complex VHDL designs.

Introduction　200

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

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

You can think of blocks as "inline modules", as they combine certain aspects of entities, architectures and module instantiations in a single VHDL language construct. As such, you can for example use them to implement specialized sub-modules that you don't want to put into a separate fully-fledged module, because they are not intended to be used anywhere else in your design. This will become more clear when we look at some code examples.

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)

To again draw a quick comparison to the world of common software programming languages, you can think of a block as an inner class. Just as an inner class in software programming languages like Java or C# provides a way to encapsulate variables or methods, and restrict their visibility to a specific section of code, a VHDL block serves a similar purpose in hardware design. It allows for certain declarations (such as signals, constants, and types) to be limited in scope, visible only within the block itself.

Introduction ◆ 200

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

- 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

Before we look at some example code, let us first introduce the formal syntax specification of block statements. Blocks are introduced with a label followed by a colon and the keyword "block".

Block Statement - Syntax

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

■ Block syntax

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

Note that in contrast, to – for example – processes, the identifier representing the block label is not optional!

Block Statement - Syntax

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

- Block syntax
  ```
  BLOCK_LABEL : block [ ( guard_condition ) ] [ is ]
    block_header
    block_declarative_part
  begin
    block_statement_part
  end block;
  ```
- Label is **not** optional!

After the block keyword the optional guard condition follows. This is a language feature, we will neither cover nor use in this course. However, we still wanted to mention it for the sake of completeness. In case you want to learn about it, we linked the VHDL language reference.

Block Statement - Syntax

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

■ 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) ◈

The optional block header allows to specify an explicit interface to the block, and looks very similar to an entity declaration and an instantiation. An upcoming slide will be dedicated to this feature and explain it in detail.

Block Statement - Syntax

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

- 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) ◈
- Optional block header: explicit block interface specification

2

Finally, we have the block declarative part followed by the keyword "begin" and the statement part. These parts look exactly like the respective parts in architectures and can, thus, contain the same elements. This also means that a block can contain other sub-blocks in its statement part. Note that the scope of objects declared in a block's declarative part is restricted to the block itself. This means that they cannot be accessed from outside the block – for example from another process in the architecture that contains the block. However, the block – meaning both its declarative and statement part – can access all objects from the architecture that contains it. In nested block structures, the objects of all containing constructs – for example other blocks – can be accessed up until the architecture as outer-most level.

Block Statement - Syntax

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

■ 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) ◈
■ 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
  ■ can access objects from outer scope

Alright, now let's look at some code! Let's say we have some module, simply called "demo" that internally needs a single full adder. We could, of course, create a separate entity/architecture pair for the full adder and add it as an instance to our module. However, here we want to demonstrate how to employ a block to achieve the desired result.

Block Statement - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```
1 architecture arch of demo is
```

First, we declare some signals that represent the inputs and outputs of the full adder sub-circuit. Since, we have already used this example circuit several times throughout this course, you should be familiar with them.

Block Statement - Example

```vhdl
1 architecture arch of demo is
2   signal a, b, cin : std_ulogic;
3   signal cout, sum : std_ulogic;
4 begin
```

In the statement part of the architecture, we can then create a block that implements the full adder.

Block Statement - Example

```vhdl
1 architecture arch of demo is
2   signal a, b, cin : std_ulogic;
3   signal cout, sum : std_ulogic;
4 begin
5   full_adder : block
```

For that purpose we first declare the signals "x", "y" and "z" that we are going to need to implement the circuit.

Block Statement - Example

```vhdl
1 architecture arch of demo is
2  signal a, b, cin : std_ulogic;
3  signal cout, sum : std_ulogic;
4 begin
5  full_adder : block
6   signal x, y, z : std_ulogic;
```

In the statement part of the block we then add the five familiar concurrent signal assignments that describe the functionality of the full adder.

Block Statement - Example

```vhdl
1 architecture arch of demo is
2  signal a, b, cin : std_ulogic;
3  signal cout, sum : std_ulogic;
4 begin
5  full_adder : block
6   signal x, y, z : std_ulogic;
7  begin
8   x <= a xor b;
9   y <= a and b;
10  sum <= cin xor x;
11  z <= cin and x;
12  cout <= y or z;
13  end block;
```

Finally, we add a process called "some-logic" that can then do something with the inputs and outputs of the full adder.

Block Statement - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```vhdl
1 architecture arch of demo is
2  signal a, b, cin : std_ulogic;
3  signal cout, sum : std_ulogic;
4 begin
5  full_adder : block
6    signal x, y, z : std_ulogic;
7  begin
8    x <= a xor b;
9    y <= a and b;
10   sum <= cin xor x;
11   z <= cin and x;
12   cout <= y or z;
13  end block;
14
15  -- do something with a, b, etc.
16  some_logic: process(all)
17   [...]
18 end architecture;
```

3

Note that, because of the scoping rules this process cannot access the signals "x", "y" and "z", declared in the full adder block. However, the block itself can and does access the signals declared on the architecture level.

Block Statement - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```
1 architecture arch of demo is
2  signal a, b, cin : std_ulogic;
3  signal cout, sum : std_ulogic;
4 begin
5  full_adder : block
6   signal x, y, z : std_ulogic;
7  begin
8   x <= a xor b;
9   y <= a and b;
10  sum <= cin xor x;
11  z <= cin and x;
12  cout <= y or z;
13 end block;
14
15 -- do something with a, b, etc.
16 some_logic: process(all)
17  [...]
18 end architecture;
```

Note

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

3

■ Defines an explicit interface to a block

As promised on the syntax-slide, we still need to cover the block header construct. As already mentioned before, the block header allows to define an explicit interface for the block statement. Instead of simply accessing the relevant signals in the outer scope of the block, we can use the block header to explicitly wire a block into an architecture like we would do with a regular instance.
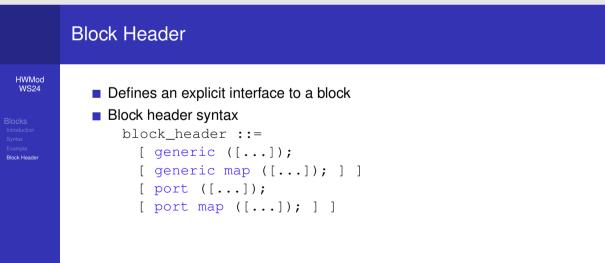
Block Header

■ Defines an explicit interface to a block

The syntax for the block header looks like a combination of an entity declaration and an instantiation. First a "generic" clause lists the generic parameters of the block. The syntax is exactly the same as for entities. Hence, we don't need to go into any further detail here. The generic clause can then be immediately followed up with a "generic-map" clause that assigns values to the declared generics. For the "generic-map" clause the same syntax as for instantiation is used. Likewise a "port" clause defines the physical interface signals, which can then be mapped to signals of the outer scope by a "port-map" clause.

Block Header

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

■ Defines an explicit interface to a block

■ Block header syntax

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

Notice that all of these four clauses are optional. However, obviously a generic or port map clause can only be specified if a respective port or generic clause is present. Furthermore, if the generic or port map clause is omitted, it must be ensured that the respective port or generic clause has default values. Otherwise, a compilation error will be raised.

Block Header

HWMod
WS24

Blocks
Introduction
Syntax
Example
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

Finally, please also note, that specifying a block header does not prevent the block from accessing objects from the outer scope in any way.

Block Header

HWMod
WS24

Blocks
Introduction
Syntax
Example
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 Statements
  └─Block Header
    └─**Block Header - Example**

To exemplify how block headers can be used in practice, let's rework the previous example to use a block header instead of simply accessing the signals in the architecture that contains the full adder block.

Block Header - Example

```
1 architecture arch2 of demo is
```

Hence, we again declare an architecture with some local signals, that we are then connecting to our full adder block.

Block Header - Example

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

└─Block Statements
  └─Block Header
    └─**Block Header - Example**

Then we can continue to declare the block and use a port clause to define its interface.

Block Header - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```
1 architecture arch2 of demo is
2   signal i1, i2, i3, o1, o2 : std_ulogic;
3 begin
4   full_adder : block
5     port (
6       a, b, cin : in std_ulogic;
7       sum, cout : out std_ulogic
8     );
```

The "port-map" clause is then used to connect these interface signals to the local signals declared in the architecture.

Block Header - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```
1 architecture arch2 of demo is
2  signal i1, i2, i3, o1, o2 : std_ulogic;
3 begin
4  full_adder : block
5   port (
6    a, b, cin : in std_ulogic;
7    sum, cout : out std_ulogic
8   );
9   port map (
10    a => i1, b => i2, cin => i3,
11    cout => o1, sum => o2
12   );
```

Finally, we declare our block local signals and add our familiar concurrent signal assignments.

Block Header - Example

```
1  architecture arch2 of demo is
2   signal i1, i2, i3, o1, o2 : std_ulogic;
3  begin
4   full_adder : block
5    port (
6     a, b, cin : in std_ulogic;
7     sum, cout : out std_ulogic
8    );
9    port map (
10    a => i1, b => i2, cin => i3,
11    cout => o1, sum => o2
12   );
13   signal x, y, z : std_ulogic;
14  begin
15   x <= a xor b; y <= a and b;
16   z <= cin and x;
17   cout <= y or z; sum <= cin xor x;
18  end block;
19  [...]
20 end architecture;
```

There are two things we can observe about this example: First the interface to the block is now completely clear from its definition. It is clear which signals are accessed in which way, and we no longer need to interfere with signals in the architecture's scope. This makes the block more self-contained, and it would make it, for example, much easier to move it into another architecture.

Block Header - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```
1 architecture arch2 of demo is
2  signal i1, i2, i3, o1, o2 : std_ulogic;
3 begin
4  full_adder : block
5   port (
6    a, b, cin : in std_ulogic;
7    sum, cout : out std_ulogic
8   );
9   port map (
10   a => i1, b => i2, cin => i3,
11   cout => o1, sum => o2
12  );
13  signal x, y, z : std_ulogic;
14  begin
15   x <= a xor b; y <= a and b;
16   z <= cin and x;
17   cout <= y or z; sum <= cin xor x;
18  end block;
19  [...]
20 end architecture;
```

Note

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

Moreover, should we at some point decide to move the block into its own entity, the code change would be trivial.

Block Header - Example

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

```vhdl
1  architecture arch2 of demo is
2   signal i1, i2, i3, o1, o2 : std_ulogic;
3  begin
4   full_adder : block
5    port (
6     a, b, cin : in std_ulogic;
7     sum, cout : out std_ulogic
8    );
9    port map (
10    a => i1, b => i2, cin => i3,
11    cout => o1, sum => o2
12   );
13   signal x, y, z : std_ulogic;
14  begin
15   x <= a xor b; y <= a and b;
16   z <= cin and x;
17   cout <= y or z; sum <= cin xor x;
18  end block;
19  [...]
20 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).

Thank you for listening! We recommend you to immediately take the self-check test in TUWEL, to see if you understood the material presented in this lecture.

HWMod
WS24

Blocks
Introduction
Syntax
Example
Block Header

Lecture Complete!