Hardware Modeling [VU] (191.011)
– WS25 –

Generate Statements

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 16:10 (f8a58e9)

Introduction

- Containers for concurrent statements (like blocks)
- Conditional and iterative/repetitive code evaluation/generation
- Similar to blocks
  - However, no block header allowed!
  - Can also be nested
- Three variants
  - if, case (activate code based on condition)
  - for (replicate code based on a range)
- Condition or range must be static at compile-time

if generate - Syntax

- “Conditional blocks”
- Comparable to if-else directives (e.g., `#if`, `#ifdef`) of the C preprocessor
- If-generate syntax

```
GENERATE_LABEL :
  if [ IF_LABEL : ] condition generate
    generate_statement_body
  { elsif [ ELSIF_LABEL : ] condition generate
    generate_statement_body }
  [ else [ ELSE_LABEL : ] generate
    generate_statement_body ]
  end generate;
```

- Conditions
  - **static** Boolean expressions (involving e.g., generics, constants)
  - At most one statement body is active

if generate - Syntax (cont'd)

- Generate statement body

```
generate_statement_body ::=
    [ block_declarative_part
  begin ]
    block_statement_part
  [ end; ]
```

- Same declarative/statement part as for blocks and architectures
- Same scoping rules as for blocks

if generate - Example


```
1  architecture arch3 of demo is
2   signal a, b, cin, cout, sum : std_ulogic;
3  begin
4   fa_gen : if ADDER_TYPE = "exact" generate
5    fa_inst : entity work.fa
6    port map(
7     a => a, b => b, cin => cin,
8     cout => cout, sum => sum
9    );
10  elsif ADDER_TYPE = "approximate" generate
11   signal temp, temp2 : std_ulogic;
12  begin
13   temp <= a or b; temp2 <= a and b;
14   sum <= temp xor cin; cout <= temp2 or cin;
15  else generate
16   assert false report "invalid option" severity failure;
17  end generate;
18  [...]
19 end architecture;
```

case generate

- Activates **one** of multiple alternatives
- All alternatives must be covered (recall `others`)!
- Case-generate syntax

```
GENERATE_LABEL :
  case expression generate
    case_generate_alternative
    { case_generate_alternative }
  end generate;
```

- One or more alternatives

```
case_generate_alternative ::=
  when [ ALTERNATIVE_LABEL : ] choices =>
    generate_statement_body
```

for generate - Syntax

- Replicate concurrent statements in a loop
- For-generate syntax

```
GENERATE_LABEL :
  for IDENTIFIER in discrete_range generate
    generate_statement_body
  end generate;
```

- The **discrete range** can be
  - a simple integer range expression (e.g., `0 to 7`)
  - a range obtained via the `'range` attribute
  - a discrete (i.e., enum or integer) type (e.g., `std_ulogic`, `natural`)
  - a discrete type with a range constraint (e.g., `natural range 0 to 1`)
- Datatype of loop variable depends on discrete range

for generate - Discrete Range Examples


```
1  entity for_gen_demo is
2  end entity;
3
4  architecture arch of for_gen_demo is
5   type myint_t is range 0 to 42;
6   constant X : string := "demo";
7  begin
8   forgen : for i in [...] generate
9    process
10   begin
11    report to_string(i);
12    wait;
13   end process;
14  end generate;
15 end architecture;
```

- `3 downto 0`
  ⇒ 3, 2, 1, 0
- `bit`
  ⇒ `'0','1'`
- `X'range`
  ⇒ 1, 2, 3, 4
- `std_ulogic range '0' to 'Z'`
  ⇒ `'0','1','Z'`
- `myint_t`
  ⇒ 0, 1, ..., 42

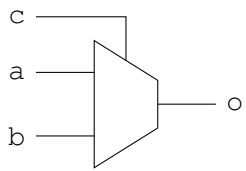
for generate - Loop Semantics


```
1 forgen:
2 for i in std_ulogic range '0' to 'Z'
3 generate
4  [declarations]
5 begin
6  [statements]
7 end generate;
```

⇒

```
1  \forgen('0')\ : block
2   constant i : std_ulogic := '0';
3   [declarations]
4  begin
5   [statements]
6  end block;
7
8  \forgen('1')\ : block
9   constant i : std_ulogic := '1';
10  [declarations]
11 begin
12  [statements]
13 end block;
14
15 \forgen('0')\ : block
16  constant i : std_ulogic := 'Z';
17  [declarations]
18 begin
19  [statements]
20 end block;
```

for generate - Example




```vhdl
entity mux21 is
 port (
  a, b : in std_ulogic;
  c : in std_ulogic;
  o : out std_ulogic
 );
end entity;

architecture arch of mux21 is
begin
 o <= a when c = '0' else b;
end architecture;
```

```vhdl
entity vec_mux21 is
 generic (
  N : positive
 );
 port (
  a, b : in std_ulogic_vector(N-1 downto 0);
  c : in std_ulogic;
  o : out std_ulogic_vector(N-1 downto 0)
 );
end entity;

architecture arch of vec_mux21 is
begin
 mux_gen : for i in 0 to N-1 generate
 begin
  mux21_inst : entity work.mux21
  port map (
   c => c, a => a(i), b => b(i), o => o(i)
  );
 end generate;
end architecture;
```

Lecture Complete!