# Hardware Modeling [VU] (191.011)
# – WS25 –
## Numeric Standard Package

Florian Huemer & Sebastian Wiedemann & Dylan Baumann

WS 2025/26

Modified: 2025-12-16, 16:04 (f8a58e9)

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations
  - Only 32 / 64 bits guaranteed by standard (2008 / 2019)

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations
  - Only 32 / 64 bits guaranteed by standard (2008 / 2019)
  - Restricted to Boolean 0 and 1 vs. 9-valued logic

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations
  - Only 32 / 64 bits guaranteed by standard (2008 / 2019)
  - Restricted to Boolean 0 and 1 vs. 9-valued logic
  - Behavior on over-/underflow undefined $\Rightarrow$ wrap-around **or** runtime error 

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations
  - Only 32 / 64 bits guaranteed by standard (2008 / 2019)
  - Restricted to Boolean 0 and 1 vs. 9-valued logic
  - Behavior on over-/underflow undefined ⇒ wrap-around **or** runtime error

⇒ IEEE `numeric_std` package

# Integer Arithmetic in Hardware



- When possible use ranged `integer` for synthesizable code
- However: `integer` type has some limitations
  - Only 32 / 64 bits guaranteed by standard (2008 / 2019)
  - Restricted to Boolean 0 and 1 vs. 9-valued logic
  - Behavior on over-/underflow undefined ⇒ wrap-around **or** runtime error

⇒ IEEE `numeric_std` package

- Requires import from the `ieee` library

```
library ieee;
use ieee.numeric_std.all;
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: error: overflow detected
```

# The IEEE numeric_std Package





- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: error: overflow detected
```

```
1 process is
2   variable a : signed(31 downto 0)
3     := to_signed(integer'high, 32);
4 begin
5   report to_string(to_integer(a));
6   a := a + 1;
7   report to_string(to_integer(a));
8   wait;
9 end process;
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: error: overflow detected
```

```
1 process is
2   variable a : signed(31 downto 0)
3     := to_signed(integer'high, 32);
4 begin
5   report to_string(to_integer(a));
6   a := a + 1;
7   report to_string(to_integer(a));
8   wait;
9 end process;
```

# The IEEE numeric_std Package





- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: error: overflow detected
```

```
1 process is
2   variable a : signed(31 downto 0)
3     := to_signed(integer'high, 32);
4 begin
5   report to_string(to_integer(a));
6   a := a + 1;
7   report to_string(to_integer(a));
8   wait;
9 end process;
```

```
[...]: 2147483647
```

# The IEEE numeric_std Package



- `unsigned` and `signed` with respective operator definitions
- Wrap-around on over-/underflow behavior

```
1 process is
2   variable a : integer
3     := integer'high;
4 begin
5   report to_string(a);
6   a := a + 1;
7   report to_string(a);
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: error: overflow detected
```

```
1 process is
2   variable a : signed(31 downto 0)
3     := to_signed(integer'high, 32);
4 begin
5   report to_string(to_integer(a));
6   a := a + 1;
7   report to_string(to_integer(a));
8   wait;
9 end process;
```

```
[...]: 2147483647
[...]: -2147483648
```

# The unsigned and signed Types



- Resolved array types of `std_ulogic`

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers

- Examples

```
signal a : unsigned(3 downto 0) := "1111";  -- 15
```

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers

- Examples

```
signal a : unsigned(3 downto 0) := "1111";  -- 15
signal b :   signed(3 downto 0) := "1111";  -- -1
```

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers
  - Bit string literal initialization possible

- Examples

```
signal a : unsigned(3 downto 0) := "1111";  -- 15
signal b :   signed(3 downto 0) := "1111";  -- -1
```

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers
  - Bit string literal initialization possible
  - Elements are nine-valued ⇒ useful for debugging

- Examples

```
signal a : unsigned(3 downto 0) := "1111";  -- 15
signal b :   signed(3 downto 0) := "1111";  -- -1
```

# The unsigned and signed Types



- Resolved array types of `std_ulogic`
  - Can be interpreted as `std_logic_vector` for arithmetic
  - Represent unsigned / two's complement binary integers
  - Bit string literal initialization possible
  - Elements are nine-valued ⇒ useful for debugging
- Wrap around on over- / underflow ⇒ behavior of "real" hardware
- Examples

```
signal a : unsigned(3 downto 0) := "1111";  -- 15
signal b :   signed(3 downto 0) := "1111";  -- -1
```

# Arithmetic Operators



- Most common operators are defined and implemented

# Arithmetic Operators



- Most common operators are defined and implemented
  - Arithmetic: `+`, `-`, `*`, `/`, `rem`, `mod`, `abs`

# Arithmetic Operators



- Most common operators are defined and implemented
  - Arithmetic: `+`, `-`, `*` , `/`, `rem`, `mod`, `abs`
  - Relational: `>`, `<`, `<=`, `>=`, `=`, `/=`
  - Logical: Same as for `std_ulogic_vector`
  - Shift / rotate: `sll`, `srl`, `rol`, `ror`, `sla`, `sra`

# Arithmetic Operators



- Most common operators are defined and implemented
  - Arithmetic: `+`, `-`, `*` , `/`, `rem`, `mod`, `abs`
  - Relational: `>`, `<`, `<=`, `>=`, `=`, `/=`
  - Logical: Same as for `std_ulogic_vector`
  - Shift / rotate: `sll`, `srl`, `rol`, `ror`, `sla`, `sra`
  - `resize` function

# Arithmetic Operators



- Most common operators are defined and implemented
  - Arithmetic: `+`, `-`, `*`, `/`, `rem`, `mod`, `abs`
  - Relational: `>`, `<`, `<=`, `>=`, `=`, `/=`
  - Logical: Same as for `std_ulogic_vector`
  - Shift / rotate: `sll`, `srl`, `rol`, `ror`, `sla`, `sra`
  - `resize` function
- Useful overloads for integer / natural operand

# Arithmetic Operators (cont'd)



```
1  process is
2
3
4  begin
5
6
7
8
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5
6
7
8
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));
6
7
8
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6
7
8
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6    report to_string(resize(s, 5));  -- 11010
7
8
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6    report to_string(resize(s, 5));  -- 11010
7
8    report to_string(resize(u, 3));
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6    report to_string(resize(s, 5));  -- 11010
7
8    report to_string(resize(u, 3));  -- 010
9
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6    report to_string(resize(s, 5));  -- 11010
7
8    report to_string(resize(u, 3));  -- 010
9    report to_string(resize(s, 3));  -- 110
10
11
12
13
14
15
16   wait;
17 end process;
```

# Arithmetic Operators (cont'd)



```
1  process is
2    variable u : unsigned(3 downto 0) := "1010";
3    variable s :   signed(3 downto 0) := "1010";
4  begin
5    report to_string(resize(u, 5));  -- 01010
6    report to_string(resize(s, 5));  -- 11010
7
8    report to_string(resize(u, 3));  -- 010
9    report to_string(resize(s, 3));  -- 110
10
11   u := u + 1;
12   s := s - 2;
13   if u = 0 or s < -1 then
14     report "HWMod is awesome";
15   end if;
16   wait;
17 end process;
```

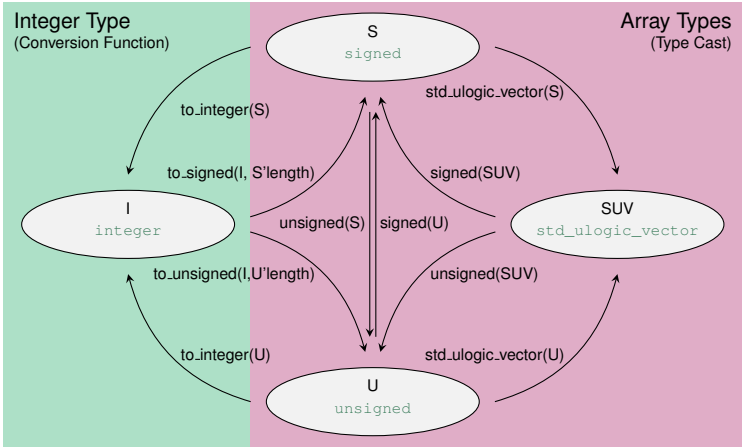
# Type Conversion

Conversion function from / to `integer`, type casts between array types

# Type Conversion



Conversion function from / to `integer`, type casts between array types



# Lecture Complete!