

# INTRODUCTION TO FPGA PROGRAMMING

---

## LESSON 05: TYPES, ARRAYS AND ARITHMETIC FUNCTIONS

Dr. Davide Cieri<sup>1</sup>

<sup>1</sup>Max-Planck-Institut für Physik, Munich

September 2024

**MAX-PLANCK-INSTITUT**  
FÜR PHYSIK

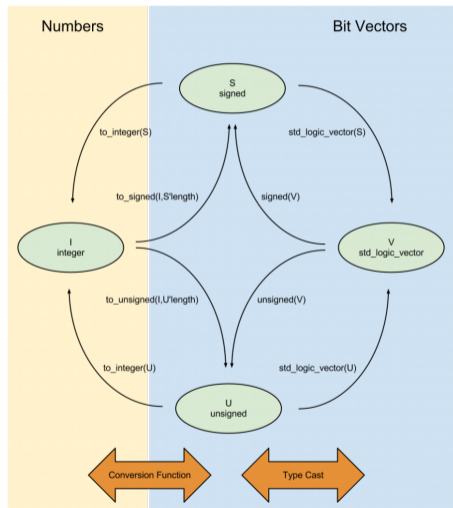


## RECAP: VHDL TYPES

- VHDL is a strongly typed language (like C)
  - All objects in VHDL must have a type (signals, variables, ports, etc...)
  - Objects can only have value of that typed
  - Objects must be declared before using them
  - Operations are allowed only between same-type objects
  - Advantage: Less mistakes. Disadvantage: Longer codes
- Types can be built-in, provided by third-party (IEEE), or user-defined.

# TYPE CONVERSION

- To assign or compare values of different types, you need type casting or conversion functions
- Type Casting if source and destination are closely related (relation is defined in the type definition)
- Conversion Function if the types are not related



## ATTRIBUTES OF TYPES AND SUBTYPES

- Types have intrinsic information (attributes) that can be accessed using the ' tick notation
  - T'high returns greatest value of type T
  - T'low is the lowest value of type T.
  - Full list [here](#).
- Subtypes are a restricted range of a scalar base type.

```
-- Syntax
```

```
subtype <subtype_name> is <base_type> range <range_constraint >;
```

```
-- Example
```

```
subtype t_count is integer range 0 to 10;
```

## ENUMERATED TYPES

- Enumerated types are constructed from a fixed list of values
- E.g. `std_ulogic` or `boolean`

```
-- Syntax
type type_name is (type_element, type_element, ...);
-- Example
type STD_ULOGIC is ( 'U',           -- Uninitialized
                    'X',           -- Forcing Unknown
                    '0',           -- Forcing 0
                    '1',           -- Forcing 1
                    'Z',           -- High Impedance
                    'W',           -- Weak Unknown
                    'L',           -- Weak 0
                    'H',           -- Weak 1
                    '_'            -- Don't care
                    );
```

- Available attributes: `Pos(s)`, `Val(x)`, `Succ(s)`, `Pred(s)`, `Leftof(s)`, `Rightof(s)`

## PHYSICAL TYPES

- Physical types are numeric types, built upon a base unit with successive definitions based on multiples of the base unit
- `time` is a built-in physical type

```
-- Syntax
type <type_name> is range <left_bound>
    to <right_bound>;
units
    <primary_unit_name>;
    <secondary_unit_name> = <multiplier> <
        primary_unit_name>;
    <third_unit_name> = <multiplier> <
        secondary_unit_name>;
    ...
end units;
```

```
-- Example
type Time is range;
units
    fs;           -- femtosecond
    ps = 1000 fs; -- picosecond
    ns = 1000 ps; -- nanosecond
    us = 1000 ns; -- microsecond
    ms = 1000 us; -- millisecond
    sec = 1000 ms; -- second
    min = 60 sec; -- minute
    hr = 60 min;  -- hour
end units;
```

## ARRAYS

- Collection of values of the same type

```
-- Syntax
type <type_name> is array <range> of <type>;
-- Example
type bus is array (7 downto 0) of std_logic_vector;
```

- Index can also be an enumerated type
- Two pre-defined arrays in VHDL

```
type string is array (positive range <>) of character;
type bit_vector is array (positive range <>) of bit;
```

- The notation `range <>` means that the array is unconstrained. Its size will be defined when declared.

## DEFAULT AND INITIAL VALUES

- The default value of a signal is the 'left value of its type definition
  - First value in an enumerated list, smallest value in a numeric range
  - E.g. Default for `std_logic` is 'U'
- Initial values may be assigned in the declaration for simulation (ignored in synthesis)

```
signal a : std_logic := '0';
```



## IEEE DATA TYPES

- `ieee.std_logic_1164.all` data types:

```
-- Unresolved (std_ulogic)
signal unres : std_ulogic := '1';
-- Resolved (std_logic)
signal res : std_logic := '1';
-- Vectors of std_logic (std_logic_vector)
signal vec : std_logic_vector(3 downto 0) := "1010";
```

- `ieee.numeric_std.all` data types:

```
-- Unsigned vectors
signal sig_uns : unsigned(3 downto 0) := "1010"; -- 10
-- Signed vectors
signal sig_sig : signed(3 downto 0) := "1010"; -- -6
```

- Signed are represented using the **two's complement notation**.

## ACCESSING VECTORS

- Vectors can be accessed by bit, part-selection or whole words

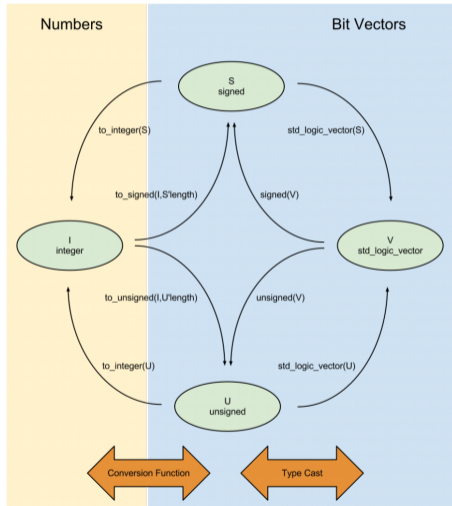
```
signal vec : std_logic_vector(7 downto 0) := (others => '0'); -- vec is 00000000
vec <= "10100000"; -- vec is now 10100000
vec(0) <= '1'; -- vec is now 10100001
vec(2 downto 1) <= "01"; -- vec is now 10100011
vec(7 downto 5) <= vec(2 downto 0); -- vec is now 01100011
```

## OVERFLOW / UNDERFLOW

- Unsigned case
  - 3-bit unsigned (0 to 7)
  - Overflow  $7 + 1 = 0$
  - Underflow  $0 - 1 = 7$
- Signed case
  - 3-bit signed (-4 to 3)
  - Overflow  $3 + 1 = -4$
  - Underflow  $-4 - 1 = 3$

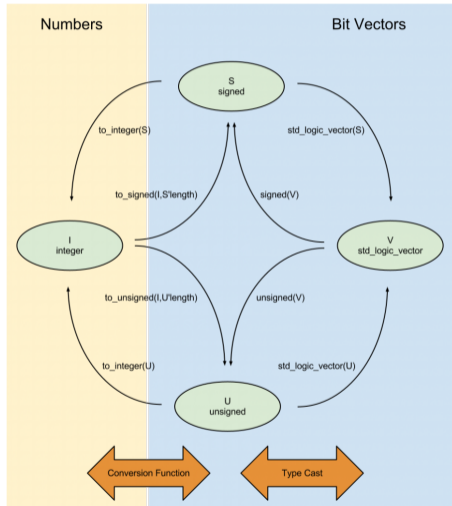
# NUMERIC TYPE CASTING AND CONVERSION FUNCTIONS

```
signal vec_std  : std_logic_vector(3 downto 0) := "
    1000";
signal vec_uns  : unsigned(3 downto 0);
signal vec_sig  : signed(3 downto 0);
signal vec_std2 : std_logic_vector(3 downto 0);
-- Casting to unsigned
vec_uns <= unsigned(vec_std);
-- Casting to signed
vec_sig <= signed(vec_uns);
-- Casting to std_logic_vector
vec_std2 <= std_logic_vector(vec_sig);
```



# NUMERIC TYPE CASTING AND CONVERSION FUNCTIONS

```
signal vec_uns : unsigned(3 downto 0) := "1011";
signal vec_sig : signed(3 downto 0) := "1010";
signal int1 : integer;
-- Converting to integer and back to unsigned
int1 <= to_integer(vec_uns);
vec_uns <= to_unsigned(int1, 4);
-- Converting to integer and back to signed
int1 <= to_integer(vec_sig);
vec_sig <= to_signed(int1, 4);
```



## NUMERAL AND BIT-STRING LITERAL VALUES

- Integer values can be expressed using any base from 2 to 16 (default 10)

```
signal a : integer;  
a <= 16#FF#;  
a <= 10#99#;  
a <= 2#1010_1010#; -- underscore used for clarity
```

- Vector values can be expressed using bit-string literals (Bases B,O,X)

```
signal b : std_logic_vector(7 downto 0);  
a <= X"FF"; -- hexadecimal base  
a <= B"1010_1010"; -- Binary base  
signal c : std_logic_vector(8 downto 0);  
c <= O"777"; -- Octal base. 3 digits require 9 bits
```

- The length of the bit-string must match the vector
- If they differ, use the resize function

```
signal addr : unsigned(10 downto 0);  
addr <= resize(unsigned(X"0F"), addr'length);
```

## VHDL RECORDS

- VHDL records are a useful way to group different signals in a single type.
- Similar to C structure

```
type t_rec1 is record                -- Declare a record with two fields
  f1 : std_logic;
  f2 : std_logic_vector(7 downto 0);
end record t_rec1;

constant zero_rec1 : t_rec1 := (    -- Declare a constant: declare the value of each field
  f1 => '0',                          -- Recommended: use name binding
  f2 => (others => '0')
);

constant one_rec1 : t_rec1 := (      -- Not recommended: positional binding for record constants
  '1',
  (others => '1')
);
```

## ARRAY OF VECTORS

- We have seen that you can define array of any type
- We can also define arrays of IEEE types

```
type t_array_std is array(3 downto 0) of std_logic_vector(7 downto 0);  
type t_array_uns is array(3 downto 0) of unsigned(7 downto 0);
```

- You can access the array using the indexes

```
signal array_std is t_array_std;  
array_std(0);    -- returns the first std_logic_vector in the array  
array_std(1)(0); -- returns the first bit of the second std_logic_vector in the array
```



## UNCONSTRAINED ARRAYS

- Arrays can be constrained or unconstrained
  - Constrained: the array boundaries are defined in the typed
  - Unconstrained: boundaries are defined in the object declaration

```
-- Constrained array example
```

```
type constrained_array is array(3 downto 0) of std_logic_vector(7 downto 0);  
signal constrained_signal : constrained_array;
```

```
-- Unconstrained array example
```

```
type unconstrained_array is array(integer range <>) of std_logic_vector(7 downto 0);  
signal unconstrained_signal : unconstrained_array(3 downto 0);
```

## MULTIDIMENSIONAL ARRAYS AND ARRAY OF ARRAYS

- Multidimensional arrays allows you to access only one array location at the time

-- Syntax

```
type <array_type_name> : array (<range 0>, <range 1>, ... , <range N>) of <type>;
```

-- Example

```
type t_2D_array : array(3 downto 0, 3 downto 0) of std_logic_vector(3 downto 0);  
signal mem2d : t_2d_array;  
mem2d(1,3); -- Access std_logic_vector at location (1,3)
```

- Array of arrays allows you to access both sub-array and array locations

-- Syntax

```
type <array_type_name> : array (<range>) of <another_array_type>;
```

-- Example

```
type t_array0 : array(3 downto 0) of std_logic_vector(3 downto 0)  
type t_array1 : array(3 downto 0) of t_array0;  
signal mem : t_array1;  
mem(1); -- Access t_array1 at location (1)  
mem(1)(3); -- Access std_logic_vector at location (1,3)
```

## ARITHMETIC OPERATIONS IN VHDL

Operator	Definition	Example
+	Addition	<code>c &lt;= a + b;</code>
-	Subtraction	<code>c &lt;= a - b;</code>
*	Multiplication	<code>c &lt;= a * b;</code>
/	Division	<code>c &lt;= a / b;</code>
**	Power	<code>a ** b</code>
mod	Modulus of <code>a</code> divided by <code>b</code>	<code>c &lt;= a mod b;</code>
rem	Remainder of <code>a</code> divided by <code>b</code>	<code>c &lt;= a rem b;</code>
abs	Absolute value of a signed signal	<code>abs(a)</code>
-	Negated value of a signed signal	<code>-a</code>

- Operators supports a mixture of `signed+integer`, `unsigned+integer`
- No direct support to mix signed and unsigned

## CONCATENATING AND RESIZING SIGNALS

Operator	Definition	Example
&	Concatenate two signals	a & b
resize()	Resize the vector to a new size	resize(a, width)

```
-- Concatenating example
signal a : std_logic_vector(3 downto 0) := "1010";
signal b : std_logic_vector(3 downto 0) := "0010";
signal c : std_logic_vector(7 downto 0);
c <= a & b; -- 10100010
-- Resizing example
c <= resize(a, 8); -- "00001010"
```

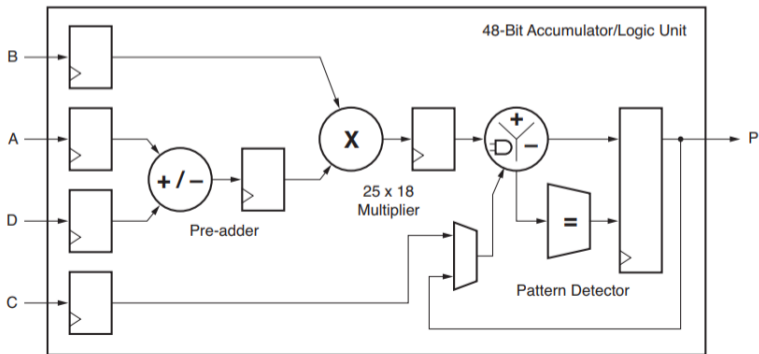
## SHIFT AND ROTATE VECTORS

Function	Description	Example
<code>shift_left</code>	Shifts bits to the left, filling with zeros.	<code>shift_left(a, 2);</code>
<code>shift_right</code>	Shifts bits to the right, filling with zeros.	<code>shift_right(a, 2);</code>
<code>rotate_left</code>	Rotates bits to the left.	<code>rotate_left(a, 2);</code>
<code>rotate_right</code>	Rotates bits to the right.	<code>rotate_right(a, 2);</code>

```
-- Example
signal uns_vec : unsigned (7 downto 0) := "10000010";
signal sig_vec : unsigned (7 downto 0) := "10000010";
shift_left(uns_vec, 1);  -- "00000100"
shift_left(sig_vec, 1); -- "00000100" (sign is lost)
shift_right(uns_vec, 1); -- "01000001"
shift_right(sig_vec, 1); -- "10000001" (sign is kept)
```

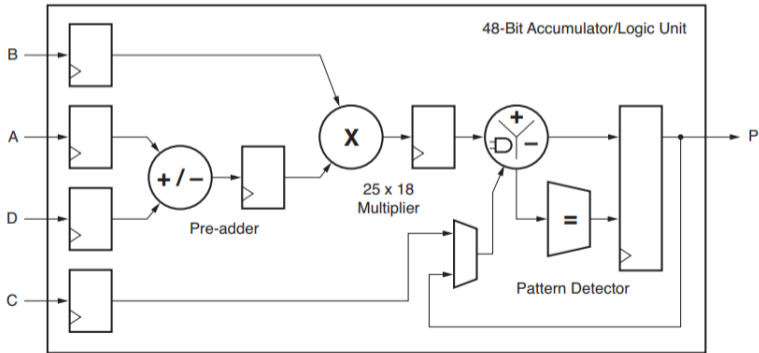
## DIGITAL SIGNAL PROCESSING (DSP) ON FPGA

- Majority of nowadays FPGAs have dedicated, full-custom hardware for arithmetic processing, called Digital Signal Processing or DSP slices
- Artix-7 has 90 DSP48E1 slices available
- Full Documentation [here](#)



## DSP BASIC FUNCTIONS

- Multiply, Accumulate, Pre-Adder, Post Adder/Subtract/Logic Unit, Pattern Detector
- Vivado automatically instantiate DSP slices to implement arithmetic functions
  - Users can force usage of DSP/Registers using `attributes` (more in another lesson)



UG479\_c1\_21\_032111

## LAB 09: DESIGN AN ARITHMETIC LOGIC UNIT



The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

©2006, Pearson Education, Inc, Upper Saddle River, NJ. All rights reserved

- [allaboutfpga.com](http://allaboutfpga.com)

- [nandland.com](http://nandland.com)

- [docs.amd.com](http://docs.amd.com)

- <https://www.symmetryelectronics.com/>

- <https://www.edn.com/>

- <https://www.bitweenie.com/>