This is a cheat sheet for vhdl to help when in doubt about syntax or buidling blocks.

VHDL Cheat Sheet

By Ismail El Badawy

Table of Contents

- Code snippets

Libraries

0. IEEE

```
Library ieee;
```

1. Standard Logic.

```
use ieee.std_logic_1164.all;
```

2. Math for numerics

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

Entities

1. Declaring an entity:

Here we declare an entity with Three ports an input port called `FirstPort` an output port called `SecondPort` they are both `std_logic`.

A third port is used as bidirectional with 16 bits.

```
entity EntityName is
port (
    FirstPort : in std_logic,
    SecondPort : out std_logic,
    ThirdPort : inout std_logic_vector(15 downto 0)
);
end entity;
```

2. Generic Entity declaration

Here we declared the entity with two generic integers that can be used throughout the code of the entity.

```
entity EntityName is
-- generics fit here.
```

```
generic (n : integer := 16, m : integer := 32);
port(

);
end entity;
```

Architecture

1. Writing an architecture

```
Architecture architecturename of entityname is
begin

end architecture;
```

2. Using intermediary signals

```
Architecture architecturename of entityname is
-- All signals fit here
Signal wire : std_logic;
Signal bus : std_logic_vector(number downto 0);
begin

end architecture;
```

3. Using an old entity inside an architecture

```
Architecture architecturename of entityname is
-- Use the entity declaration but instead of the word entity write component.

component componentname is
-- if the component has a generic
generic(n : integer := 16);
port(
    -- port declaration here.
);
end component;

begin

end architecture;
```

You might want to instantiate it.

```
label : entityname generic map(16) port map(portlist);
-- Ommit generic map if the component is not generic.
```

Process

A process is executed sequentially rather than the usual concurrent execution.

Using process

```
-- Process lies inside an architecure block
Architecture name of entityname is
begin

process(-- sensitivity list
    clk, reset
)
begin
    -- inside the process block you can use conditions.
    if(reset = '0') then
        output <= '0';
    elsif (reset = '1') then
        -- Something
    else
        -- Something
    end if;
-- Process ends with this.
end process;

end architecture;
```

Assignment

1. Normal Assignment

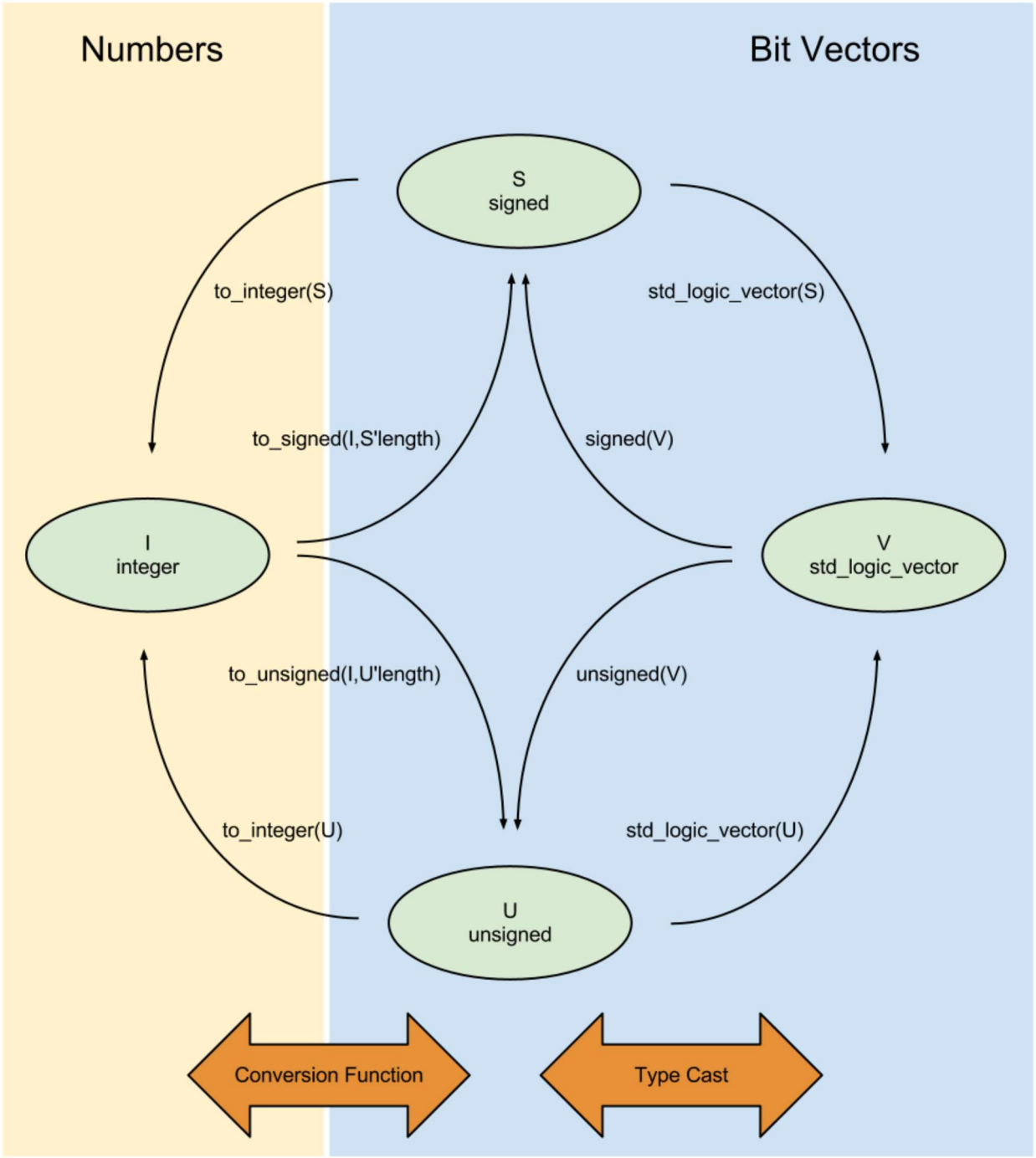
```
    B <= '0';
```

2. Using `when`

```
-- Actual code inside archtiecture
B <= "0000" when enable = '0'
        else "0001" when A = "00"
        else "0010" when A = "01"
```

```
        else "0100" when A = "10"
        else "1000" when A = "11";
```

Type Conversion



Operators

Mathematical

Operator	Function
**	Power
rem	Remainder
mod	Modulus
/	Divide
*	Multiply
-	Subtract
+	Add
abs	Absolute value

Logical Operators

Operator
and
or
not
nor
nand
xor
xnor

Relational Operators

Operator	Function
=	Equal?
/=	Not Equal?
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

Test Bench

A test bench is an entity with no ports.

Assert

```
Assert(condition) Report "Error message"
Severity error;
```

Wait For

```
WAIT FOR 10ns;
```

Example

```
entity testbench is
end testbench;

architecture testing of testbench is
component and2 is
    port(
        A, B : in std_logic;
        C : out std_logic
    );
end component;
Signal testa, testb, testc : std_logic;
begin
    process
    begin
        testa <= '0';
        testb <= '0';
        Wait for 10ns;
        assert(testc = '0') report "C must have been 0 when inserting 00"
        Severity error;
    wait;
    end process;
    uut : and2 port map(A => testa, B => testb, C => testc);
```

README.md

Register (1 bit)

```
-- Do not forget imports.s
entity reg is
port(
        clk, reset, input, enable: in std_logic;
        output : out std_logic);
end entity;

architecture my_arch of reg is
begin
process (clk, reset)
begin
        if (reset = '1') then
                output <= '0';
        elsif rising_edge(clk) and enable = '1' then
                output <= input;
        end if;
end process;
end architecture;
```

N-Bits Register

Using the 1 bit register :

```
entity nreg is
generic ( n : integer := 16);
port(
        clk, reset, enable : in std_logic;
        input : in std_logic_vector(n-1 downto 0);
        output : out std_logic_vector(n-1 downto 0));
end entity;

architecture my_arch of nreg is
component reg is
port(
        clk, reset, input, enable: in std_logic;
        output : out std_logic);
end component;
begin
        loop1 : for i in 0 to n-1 generate
                fx: reg port map (clk, reset, input(i), enable, output(i));
        end generate;
end architecture;
```

Tri-state buffer

```
entity tri_state_buff is
generic(n : integer := 16);
port(
        A : in std_logic_vector(n-1 downto 0);
        B : out std_logic_vector(n-1 downto 0);
        C : in std_logic);
end entity;

architecture dataflow of tri_state_buff is
begin
        B <= A when C = '1'
        else (others => 'Z');
end architecture;
```

1-bit Full Adder

```
entity FullAdder is
port(
        A, B: in std_logic;
        Cin: in std_logic;
        sum: out std_logic;
        Cout: out std_logic);
end entity;

Architecture dataflow of FullAdder is
begin
        sum <= A xor B xor Cin;
        Cout <= (A and B) or (Cin and (A xor B));
end Architecture;
```

N-bit Full Adder

Using the 1 bit full adder :

```
entity NAdder is
generic(n: integer := 8);
port(
        A, B: in std_logic_vector(n-1 downto 0);
        Cin: in std_logic;
        Cout : out std_logic;
        sum : out std_logic_vector(n-1 downto 0));
end entity;

Architecture my_adder_architecture of NAdder is
```

```
component FullAdder is
port(
        A, B: in std_logic;
        Cin: in std_logic;
        sum: out std_logic;
        Cout: out std_logic);
end component;
Signal temp: std_logic_vector(n-1 downto 0);
begin
        f0 : FullAdder port map(A(0), B(0), Cin, sum(0), temp(0));
        loop1: for i in 1 to n-1 generate
                carries: FullAdder port map(A(i),B(i), temp(i - 1), sum(i), temp(
        end generate;
        Cout <= temp(n-1);
end Architecture;
```

Releases

No releases published

Packages

No packages published