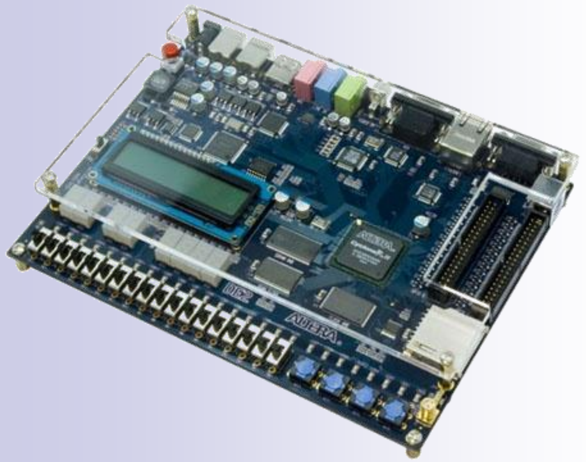
Logic Systems and Processors
cz:Logické systémy a procesory

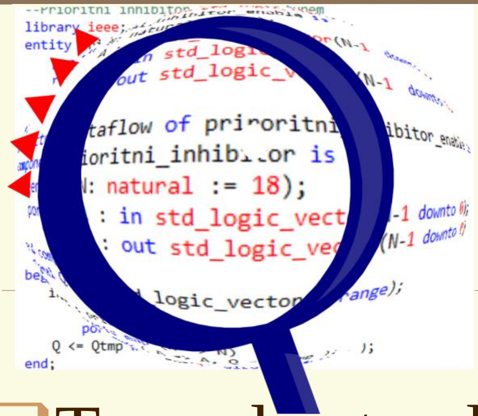
Lecturer: Richard Šusta

richard@susta.cz, susta@fel.cvut.cz,
+420 2 2435 **7359**



Version: 1.0

CTU-FEE in Prague, CR – subject BE5B35LSP



Why should we learn VHDL ?

- To understand modern electronics, we need to know a little about circuits and learn their properties through experiments.
- However, the clarity of graphical block diagrams decreased with the increasing number of their elements, which already caused the **circuit design crisis** around 1980.
- To solve it, complex schemas were gradually replaced by text-based description languages, so we will explain one of them, but only its basics.

VHDL

VHSIC HDL

Very High-Speed Integrated Circuit HDL

HDL=Hardware Description Language

for formal description

and design of electronic circuits

VHDL history

- 1981: Designed by the U.S. DoD (Department of Defense) to solve the circuit design crisis. VHDL is based on the Ada language, also developed by the U.S. DoD in 1970.
  Note: ADA is named after Ada Lovelace, the daughter of Lord Byron. She was a mathematician and author of the first computer algorithm.
  - 1983-85: VHDL developed by Intermetrics, IBM and Texas Instruments
  - 1986: **Rights transferred to IEEE**, *pronounced "Eye-triple-E,"* (Institute of Electrical and Electronics Engineers)
  - The U.S. DoD requires a VHDL description of each supplied ASIC.
- 1987: IEEE standard VHDL 1987
- VHDL-1993, VHDL-2008, VHDL-2019: Revised standards with new features
- **but VHDL-1993 is still the most frequent version**

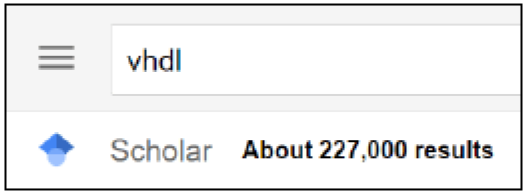
<u>We must use **VHDL 1993** with only some **VHDL 2008** elements</u>:

Quartus Lite supports VHDL 1987, VHDL 1993
and some basics elements of VHDL 2008

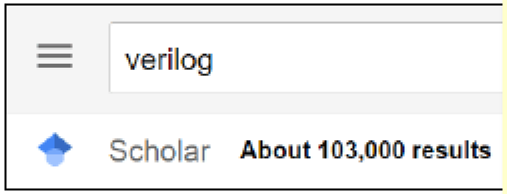
GHDL supports almost full VHDL 2008

Verilog versus VHDL

Academic usages


vhdl

Scholar About 227,000 results


verilog

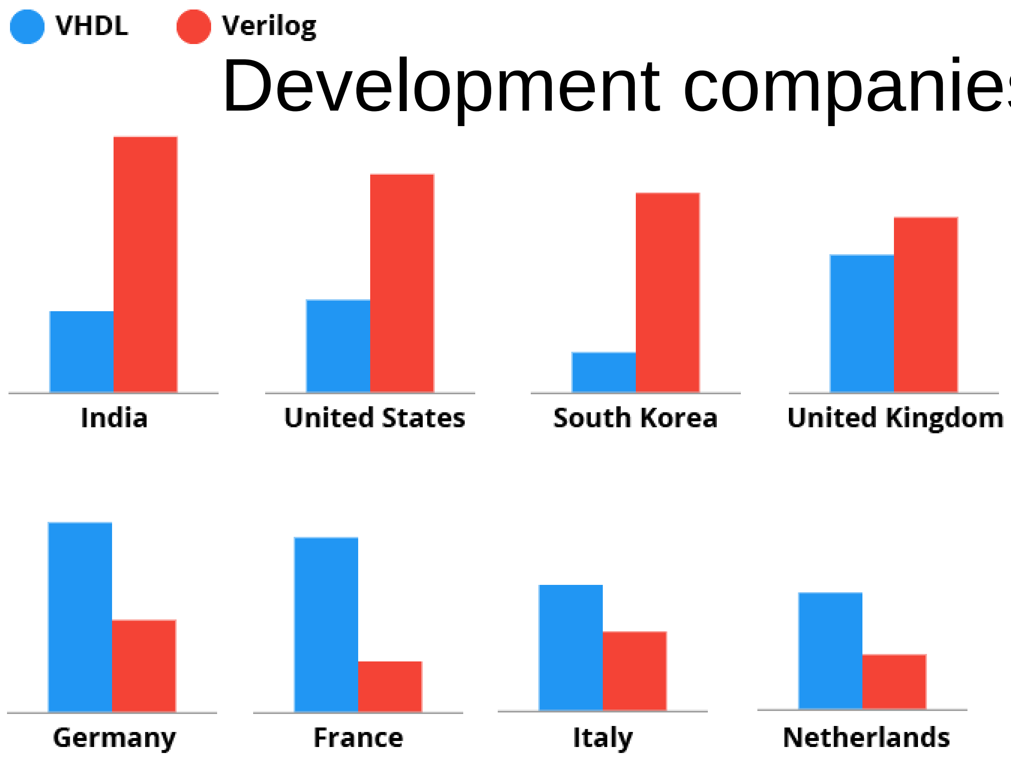
Scholar About 103,000 results

We dare say that the choice of the HDL language is less important than the proper coding style.

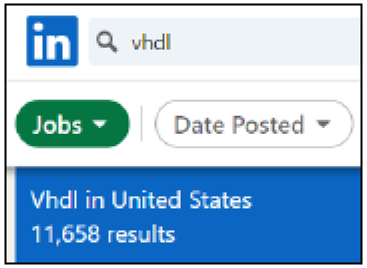
With knowledge of one HDL, you quickly master another.

And strict VHDL will better guide beginners to correct hardware descriptions.

Development companies

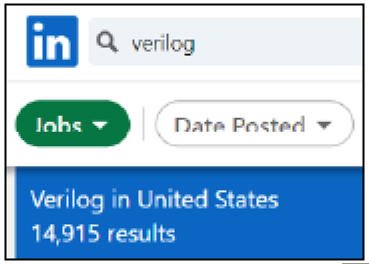


Jobs in the USA


vhdl

Jobs Date Posted

Vhdl in United States
11,658 results


verilog

Jobs Date Posted

Verilog in United States
14,915 results

Image source VhdlWhiz.com (Dated 6 December 2022)

Description of Combinational Logic via "Dataflow Design Style"

Textbooks:

Circuit Design With VHDL dataflow and structural

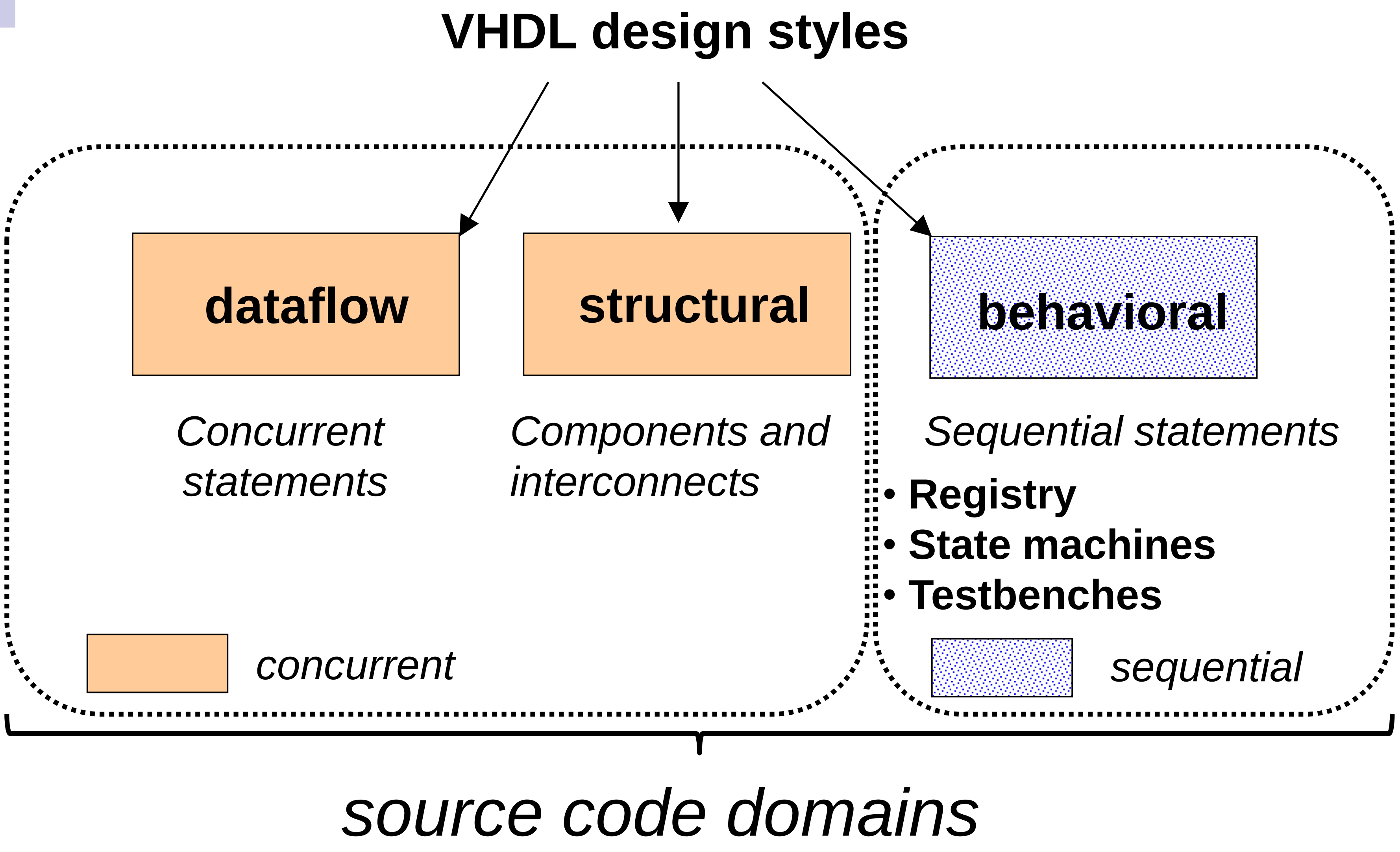
(96 pages, 20 solved examples and 9 practice tasks)

Books in DCENET library:

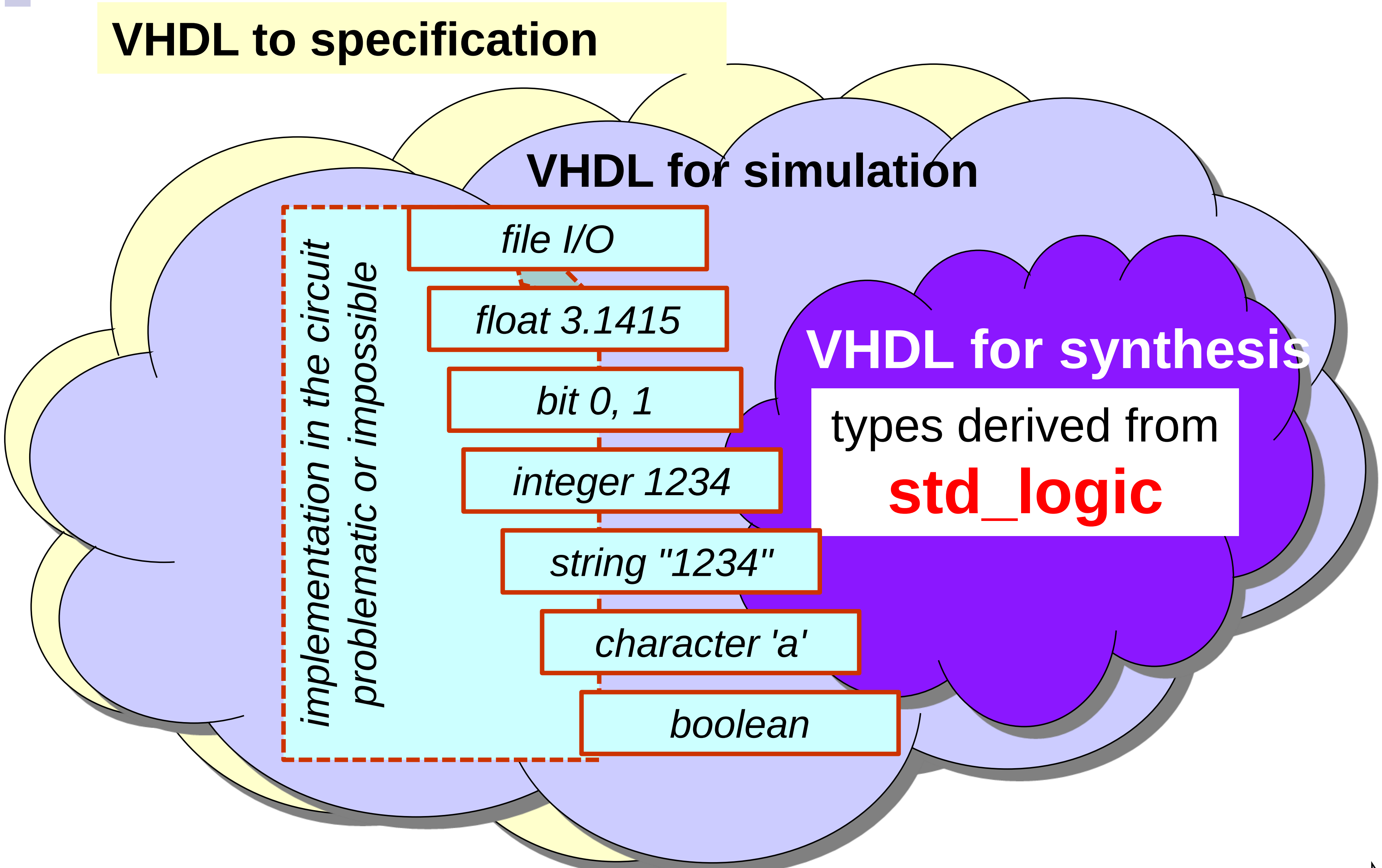
The Designers Guide to VHDL, 910 pages

The Low-Carb VHDL Tutorial, 87 pages

VHDL Code Domains



VHDL data types



The last lecture: 9 values of MVL-9 used in VHDL

Stiff source,
i.e., with **a low output impedance,**
in the case of voltage logic

Forcing 1	**'1'**
Forcing Unknown	'X'
Forcing 0	**'0'**

Non-stiff (weak) source
with **higher output impedance**
in the case of voltage logic

Weak 1	*'H'*
Weak Unknown	*'W'*
Weak 0	*'L'*

Wired Or/And

FPGAs usually cannot implement

High impedance (disconnected)

High Impedance	**'Z'**

Design note
- any value can be here

Don't Care	**'-'**

Simulation reports

Uninitialized	**'U'**

Resolution Table

```
type std_ulogic is ('U','X','0','1','Z','W','L','H', '-' );
```

```
subtype std_logic is resolved to std_ulogic;
```

connection	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'-'
'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'	'U'
'X'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'
'0'	'U'	'X'	'0'	'X'	'0'	'0'	'0'	'0'	'X'
'1'	'U'	'X'	'X'	'1'	'1'	'1'	'1'	'1'	'X'
'Z'	'U'	'X'	'0'	'1'	'Z'	'W'	'L'	'H'	'X'
'W'	'U'	'X'	'0'	'1'	'W'	'W'	'W'	'W'	'X'
'L'	'U'	'X'	'0'	'1'	'L'	'W'	'L'	'W'	'X'
'H'	'U'	'X'	'0'	'1'	'H'	'W'	'W'	'H'	'X'
'-'	'U'	'X'	'X'	'X'	'X'	'X'	'X'	'X'	'X'

Type vs Subtype

- In strongly typed languages, such as VHDL, the matching of the internal type structure is not tested; the type names are only important.
- **Type** definitions create isolated data types not convertible to types derived from a type otherwise named.
- However, there is an option to define a **subtype** that narrows the original type. This will put it in the group of an existing type and it can be converted to its members.

More: Circuit Design With VHDL dataflow and structural

Chapter 3.3 on pages 24 to 26.

1-bit Types

- all are enumerated types ("enumerated types")

- Bit - '0' and '1' *- not suitable for circuit synthesis*
- Boolean - TRUE and FALSE in *conditions only*
- std_logic is of type **Multi-Valued Logic**

  '1', '0', 'X', 'Z', 'U', '-', 'L', 'H', 'W'

All are type in strict VHDL; they are mutually

unconvertable by type casts!

Basic libraries

library ieee;

use ieee.std_logic_1164.all;

- *Access to all names declared in the STD_LOGIC_1164 package of the IEEE standard logical data type library and some basic functions.*

use ieee.numeric_std.all;

- *IEEE recommended library, which contains definitions of types for **signed**, **unsigned**, and **integer** numbers along with arithmetic operations.*

You can insert standard libraries into all VHDL codes.
They don't slow Quartus - they are precompiled.

Basic Rules for Source Code

- VHDL is not case sensitive, but we should keep capitalization. If we define symbol Data, then referring to it as "data" of "DATA" is wrong style.
- VHDL does not understand diacritics and non ASCII characters, not even in comments.
- Identifier must start with a letter, last character cannot be an underscore, two connected underscores are not allowed.
- All statements end with a semi-colon.
- Comments precede with (--) The rest of line is treated as a comment. Carriage return terminates a comment.
- VHDL 2008 allows /* */ block comments. In VHDL 93, we can comment only lines.

Reminder

A, B: in std_logic_vector (7 downto 0);

Z: out std_logic_vector (1 to 16);

A, B:

7	6	5	4	3	2	1	0

downto is selected for numeric vectors

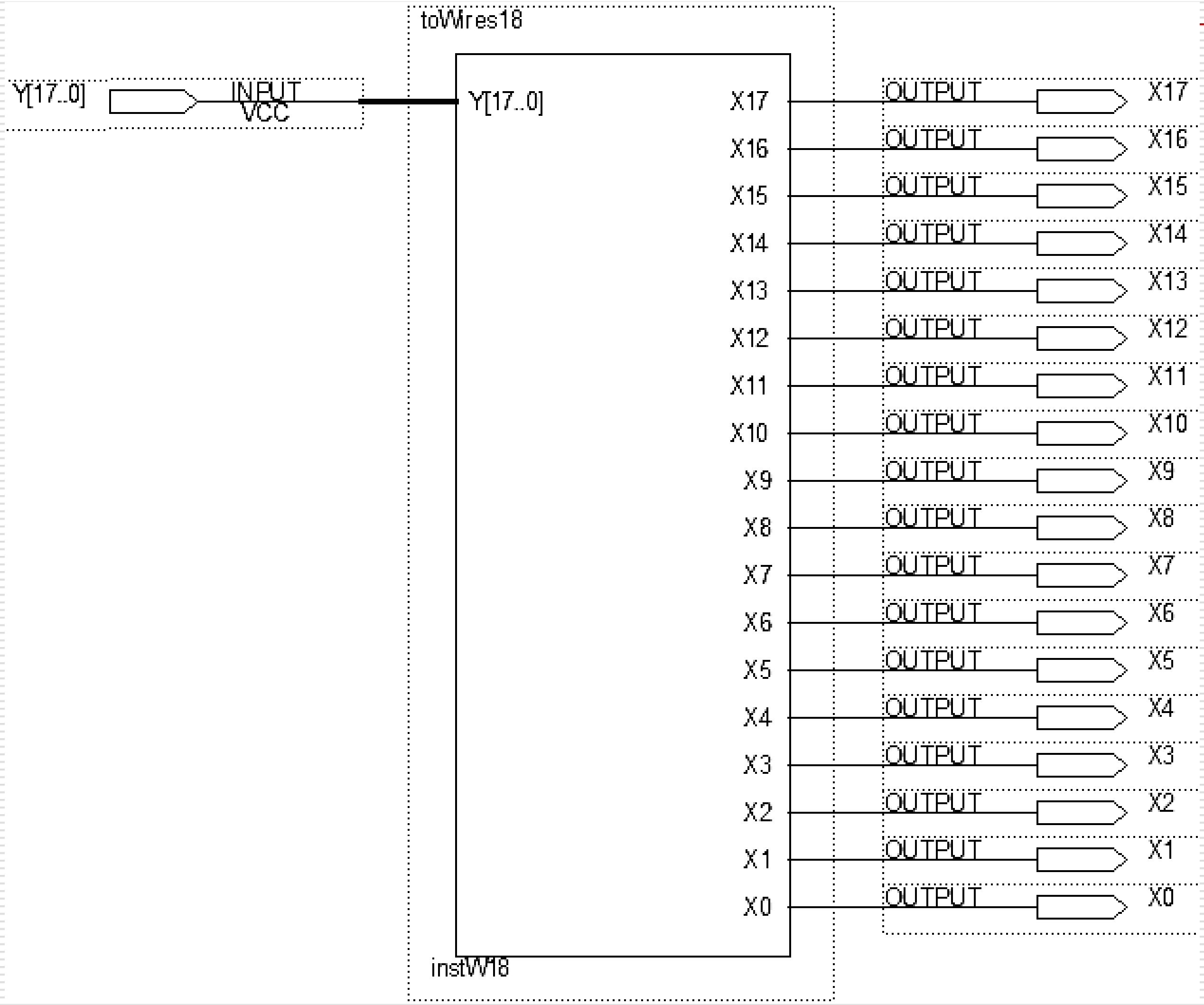
Z:

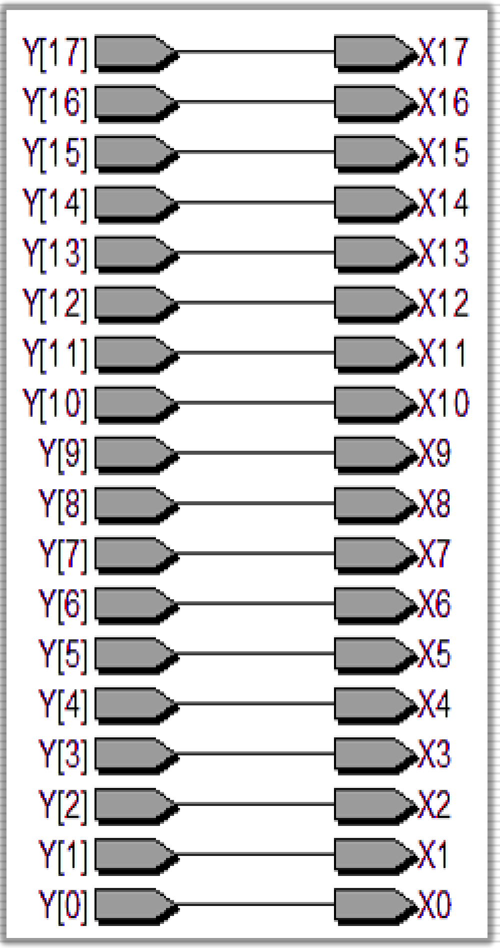
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Numbers and Buses

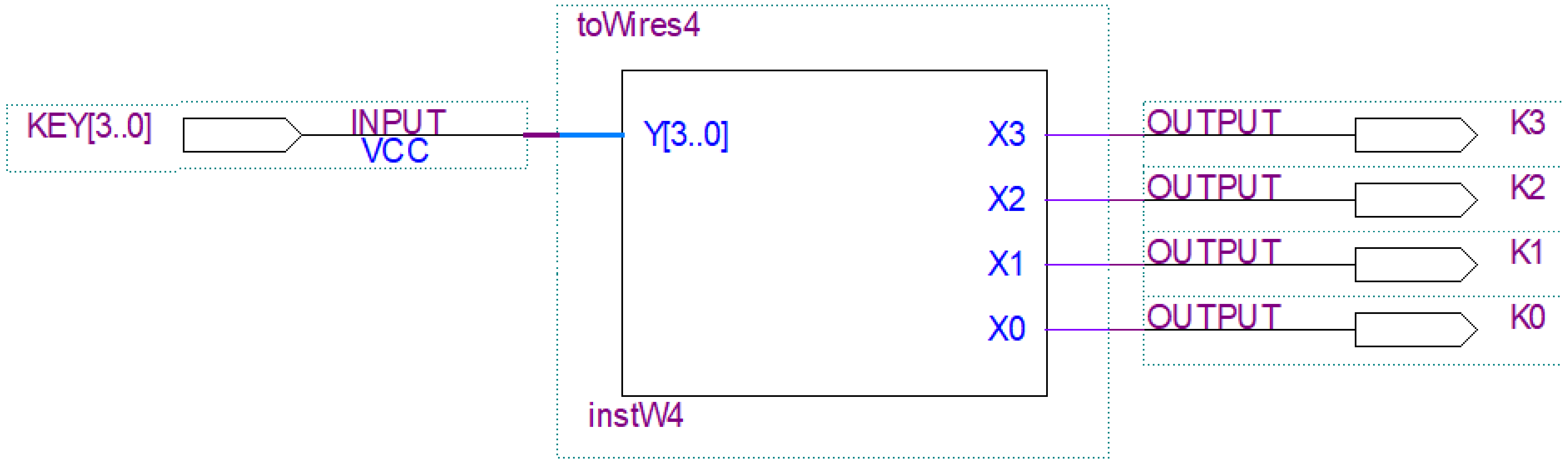
- **Std_logic** is one lonely wire.
- **Std_logic_vector**
  is a numbered collection of wires. A vector of wires is created as an array (bus) in terms of digital hardware.
- Numbers signed and unsigned
  are also vectors — but with associated information about their numeric values.
  We will discuss them in the next lecture.

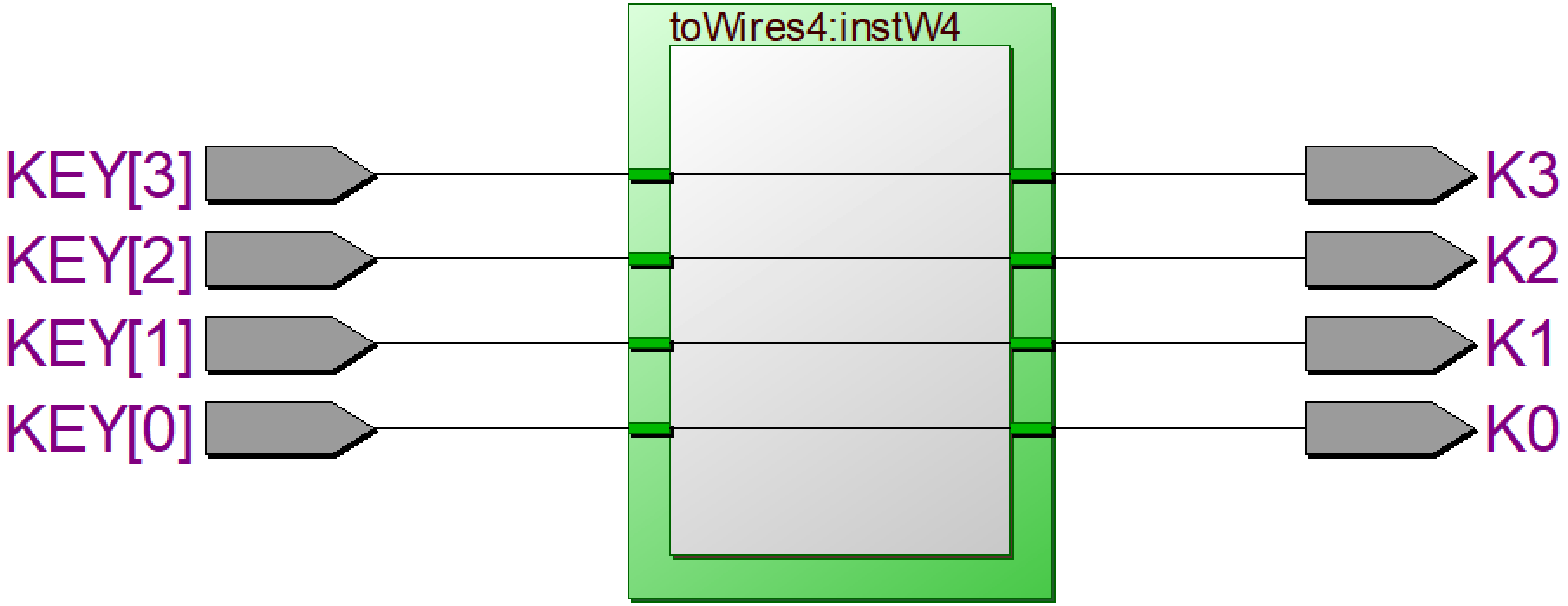
toWires18


toWires18
Y[17..0]
INPUT
VCC
Y[17..0]
X17
X16
X15
X14
X13
X12
X11
X10
X9
X8
X7
X6
X5
X4
X3
X2
X1
X0
OUTPUT
instW18

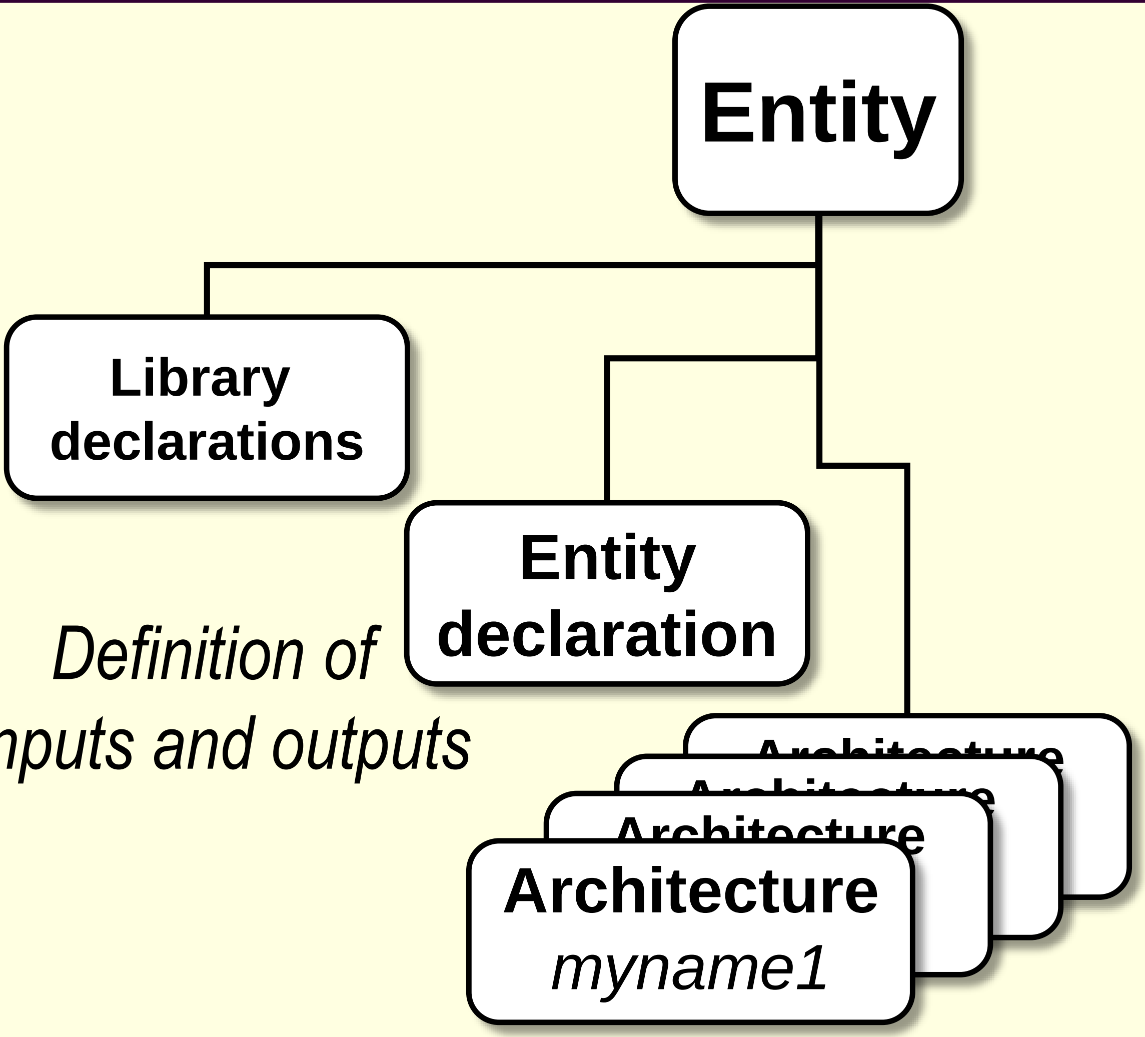

Y[17] X17
Y[16] X16
Y[15] X15
Y[14] X14
Y[13] X13
Y[12] X12
Y[11] X11
Y[10] X10
Y[9] X9
Y[8] X8
Y[7] X7
Y[6] X6
Y[5] X5
Y[4] X4
Y[3] X3
Y[2] X2
Y[1] X1
Y[0] X0

Reduced toWires18 to toWires4





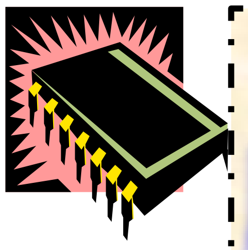
VHDL file structure



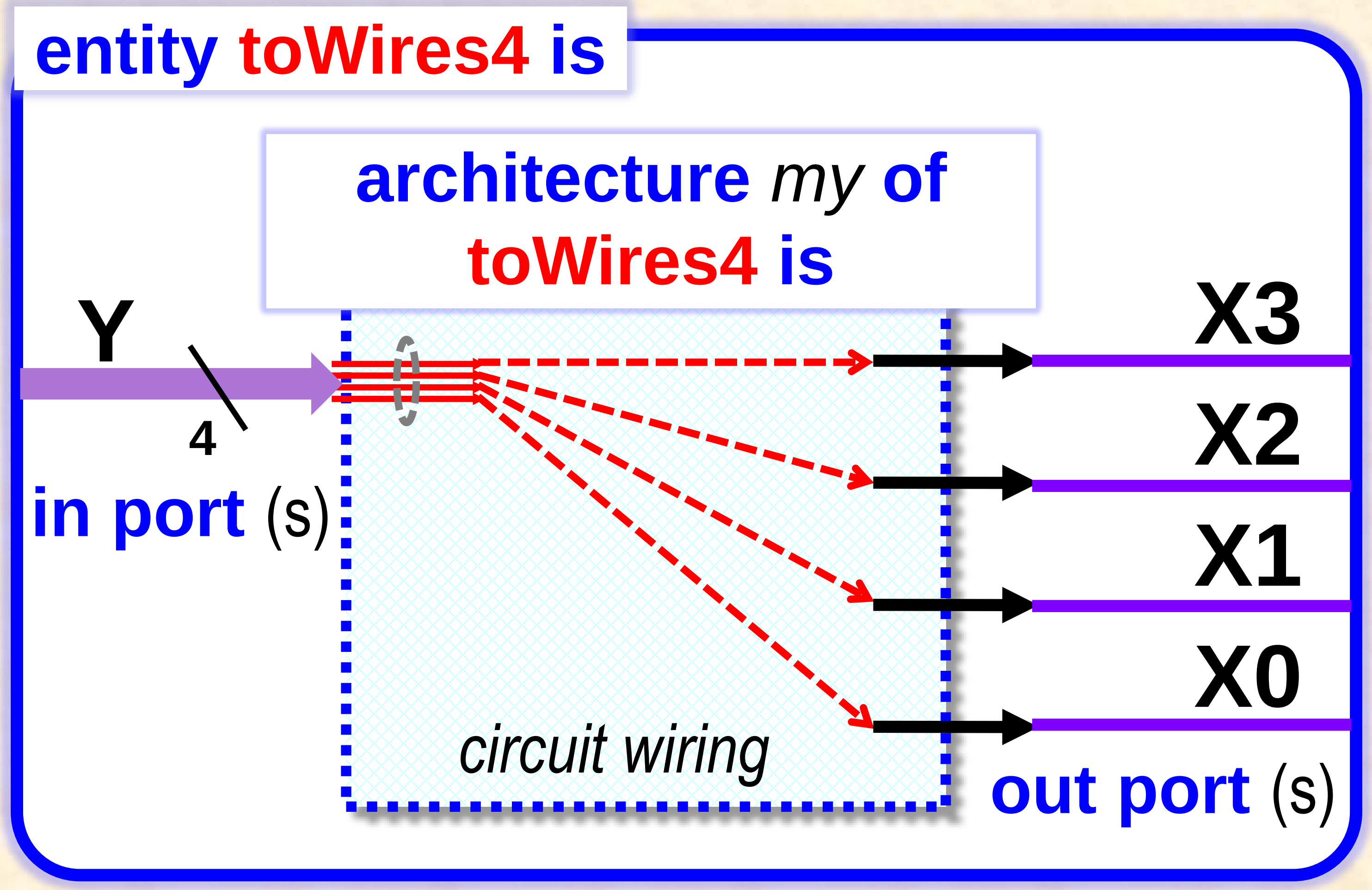
Definition of inputs and outputs

At least one architecture ≡ circuit description

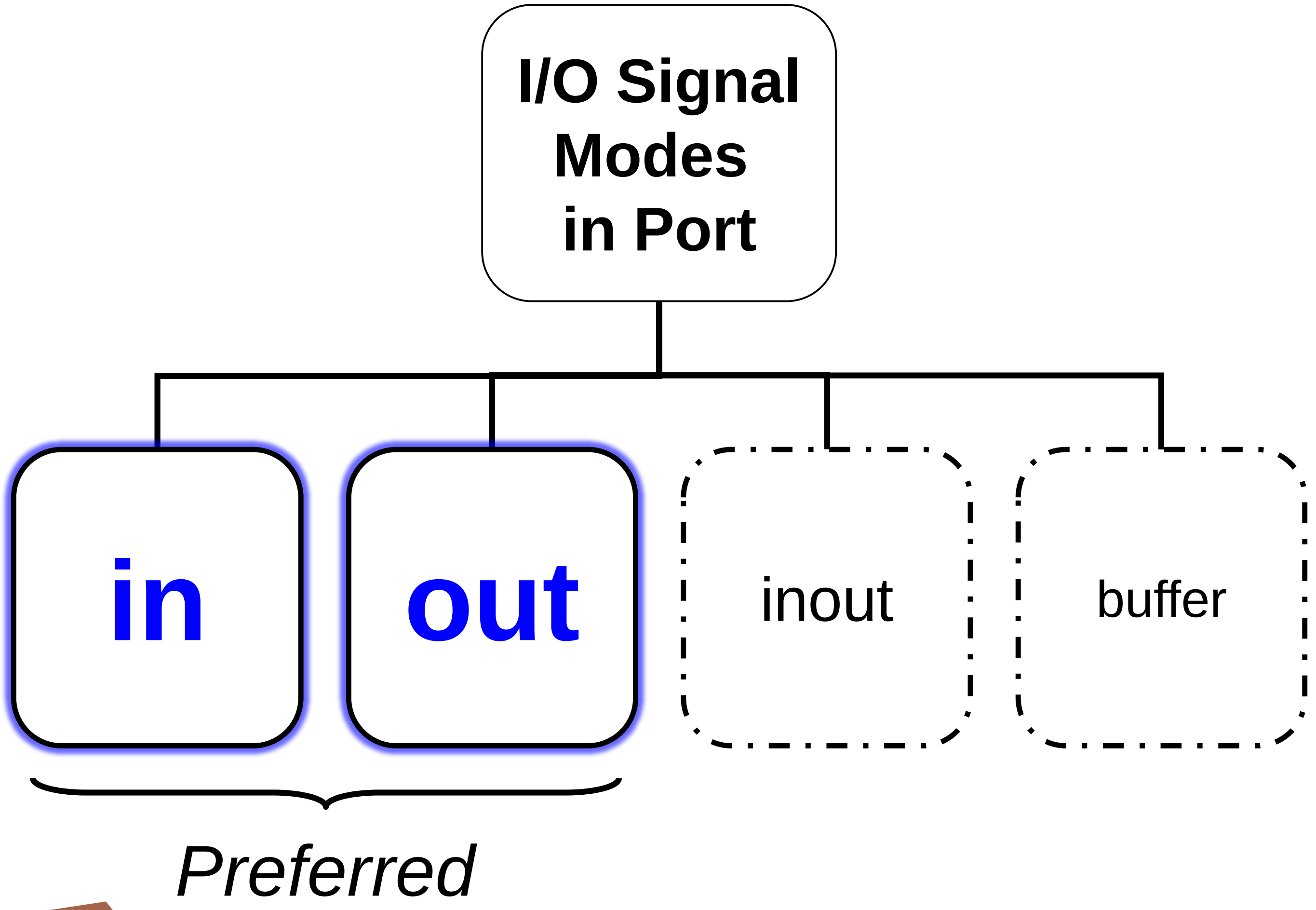
Naming Trinity



File: toWires4.vhd

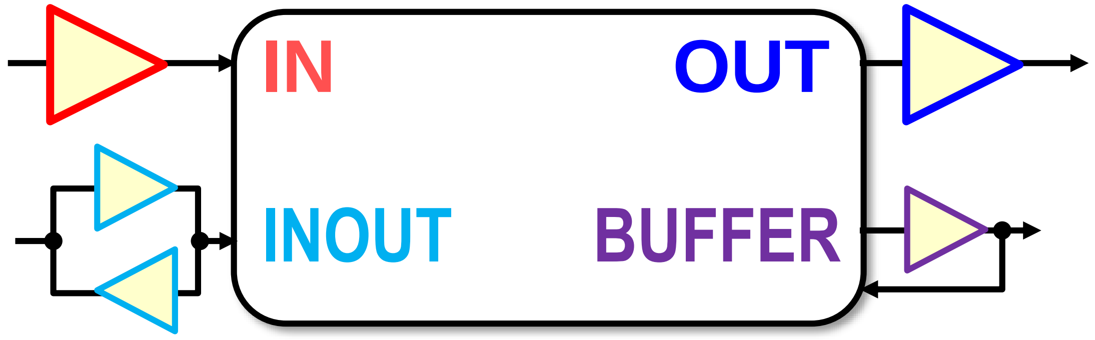


IO modes possible in the port declaration



IN, OUT, INOUT, BUFFER modes

- IN: read-only data input in
- OUT: write-only output of outgoing data.
  In VHDL93 its value cannot be read.
- BUFFER: output as well as OUT,
  but its value is readable in VHDL1993.
- INOUT: bidirectional bus wire

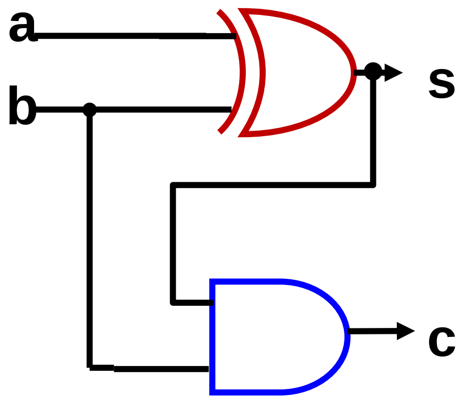


BUFFER and INOUT modes require more complex implementations. IN and OUT are preferred.

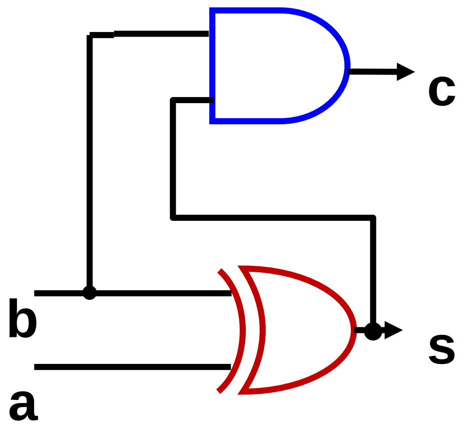
Concurrent assignment <= ≡ wire, connection

In what order will the operations be performed?

```
s <= a xor b;
c <= a and s;
```



```
c <= a and s;
s <= a xor b;
```



- In the circuit at the same time;
- Although they are executed sequentially in the simulation, a random order is chosen in each iteration to emulate the circuit behavior.

Main Statements of toWires4.vhd

Inputs and outputs

```
Y : in std_logic_vector(3 downto 0);
X3, X2, X1, X0 : out std_logic;
```

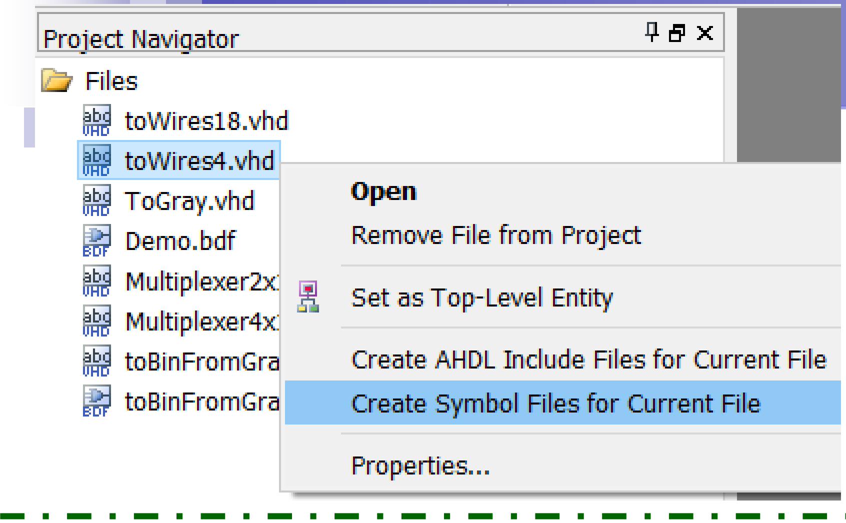
Custom wiring

```
X0<=Y(0); X1<=Y(1);
X2<=Y(2); X3<=Y(3);
```

toWires4.vhd

```
/* toWires4 converts bus to 4 wires.
   It is a simplified form of toWires18 DCE/library */
library ieee; use ieee.std_logic_1164.all;

entity toWires4 is
port( Y : in std_logic_vector(3 downto 0);
      X3,X2,X1,X0 : out std_logic);
end entity;

architecture rtl of toWires4 is
begin -- architecture
   X0<=Y(0); X1<=Y(1); X2<=Y(2); X3<=Y(3);
end architecture;
```

Creating a Symbol



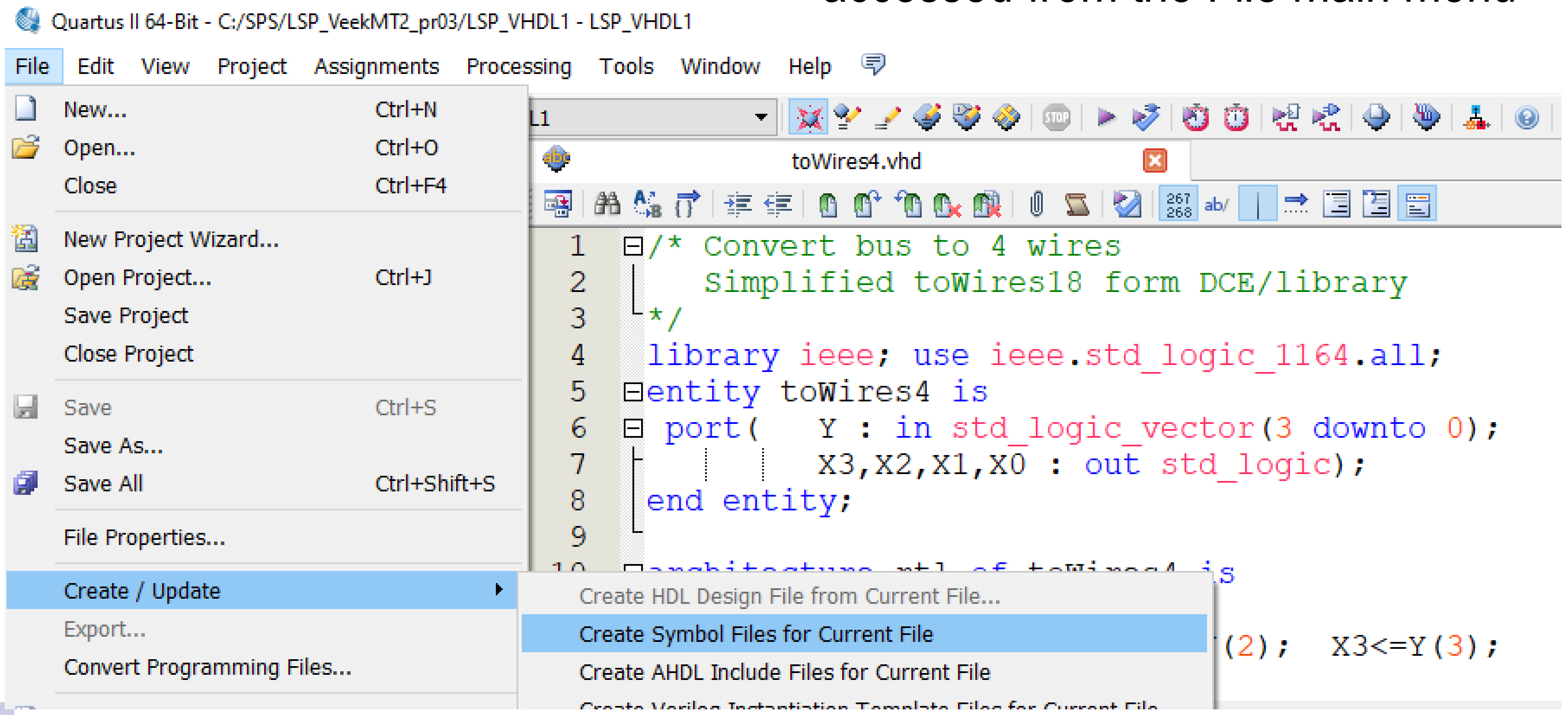
Project Navigator

Files
- toWires18.vhd
- toWires4.vhd
- ToGray.vhd
- Demo.bdf
- Multiplexer2x
- Multiplexer4x
- toBinFromGra
- toBinFromGra

Open
Remove File from Project
Set as Top-Level Entity
Create AHDL Include Files for Current File
Create Symbol Files for Current File
Properties...

❖ Be an evocation of the right mouse on a file in the [Files] tab

❖ If the file is actively open, it can also be accessed from the File main menu



Quartus II 64-Bit - C:/SPS/LSP_VeekMT2_pr03/LSP_VHDL1 - LSP_VHDL1

File Edit View Project Assignments Processing Tools Window Help

New...	Ctrl+N
Open...	Ctrl+O
Close	Ctrl+F4
New Project Wizard...	
Open Project...	Ctrl+J
Save Project	
Close Project	
Save	Ctrl+S
Save As...	
Save All	Ctrl+Shift+S
File Properties...	
Create / Update	
Export...	
Convert Programming Files...	

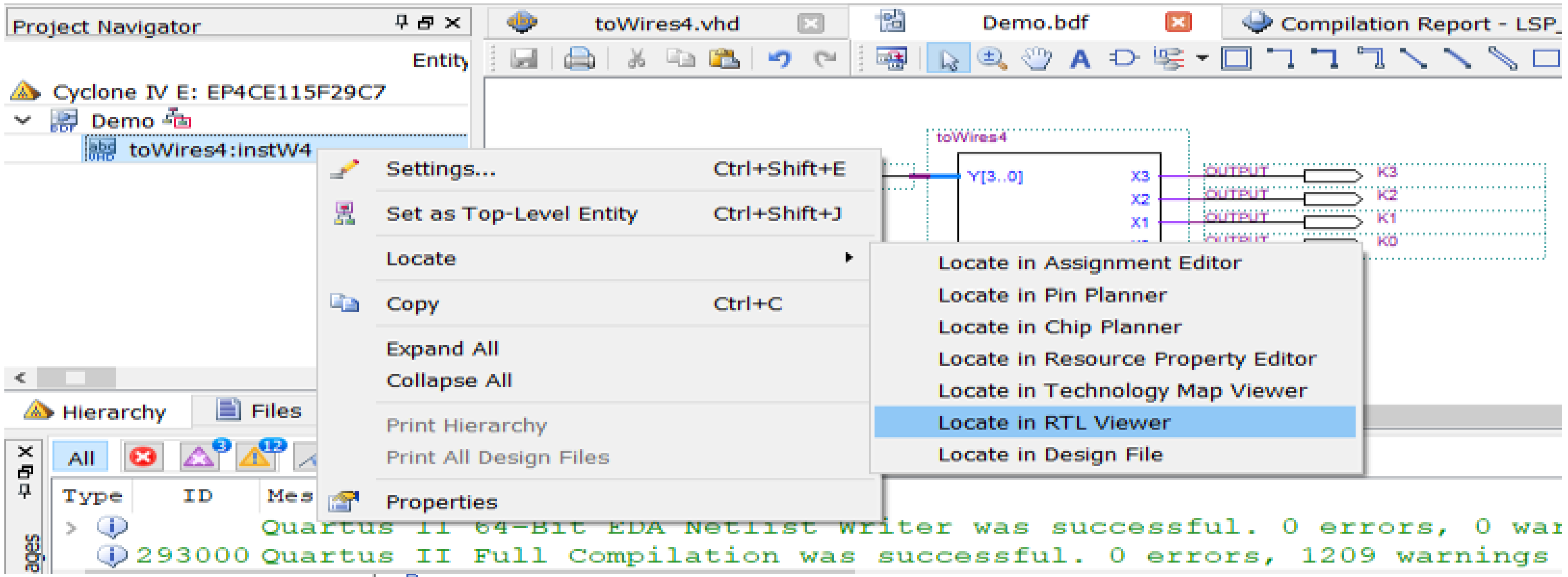
Create HDL Design File from Current File...
Create Symbol Files for Current File
Create AHDL Include Files for Current File

toWires4.vhd

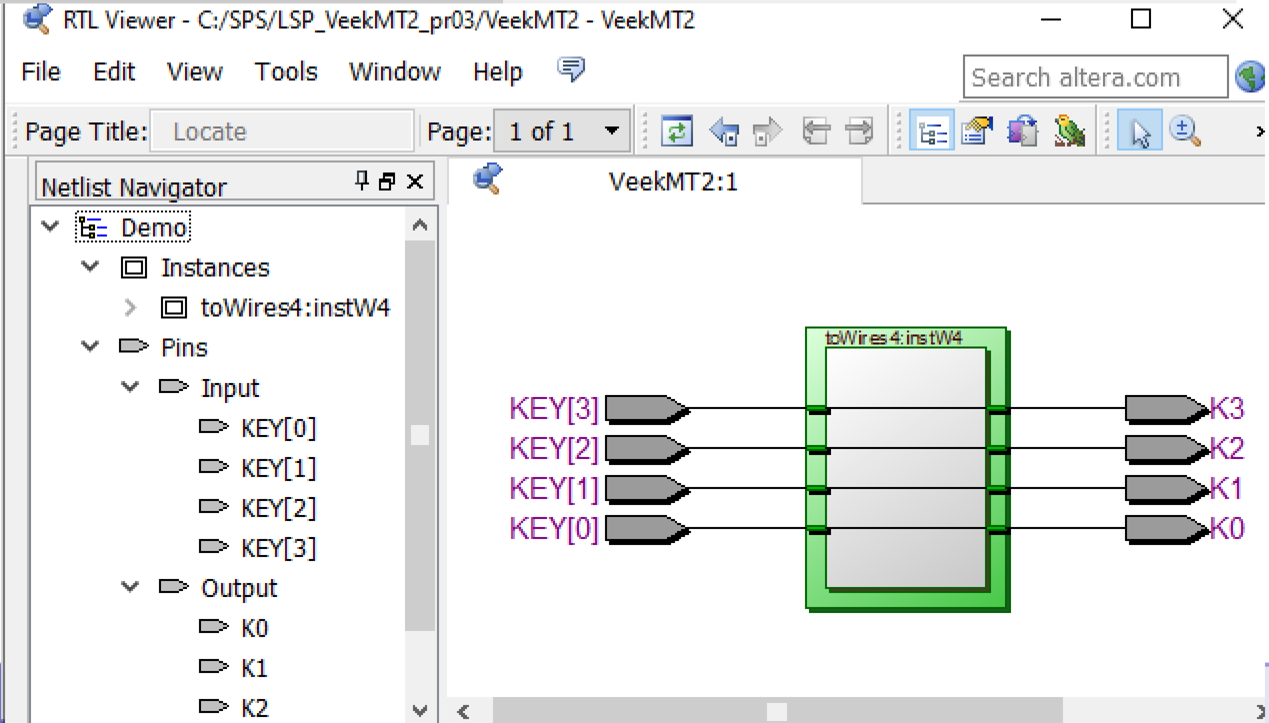
```
/* Convert bus to 4 wires
   Simplified toWires18 form DCE/library
*/
library ieee; use ieee.std_logic_1164.all;
entity toWires4 is
 port(    Y : in std_logic_vector(3 downto 0);
          X3,X2,X1,X0 : out std_logic);
end entity;

architecture rtl of toWires4 is
```

```
(2);   X3<=Y(3);
```

(Register-Transfer Level) RTL Viewer: toWires4



```
Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 war
293000 Quartus II Full Compilation was successful. 0 errors, 1209 warnings
```



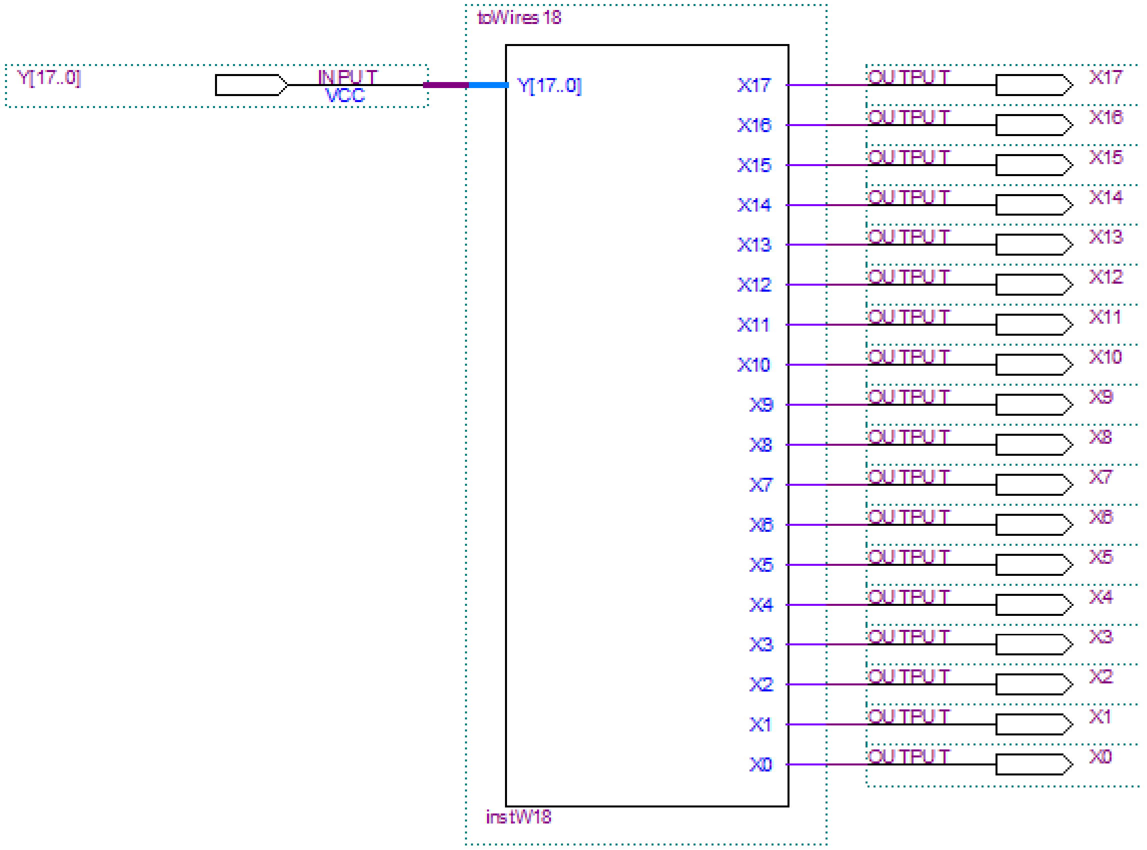
toWires18.vhd

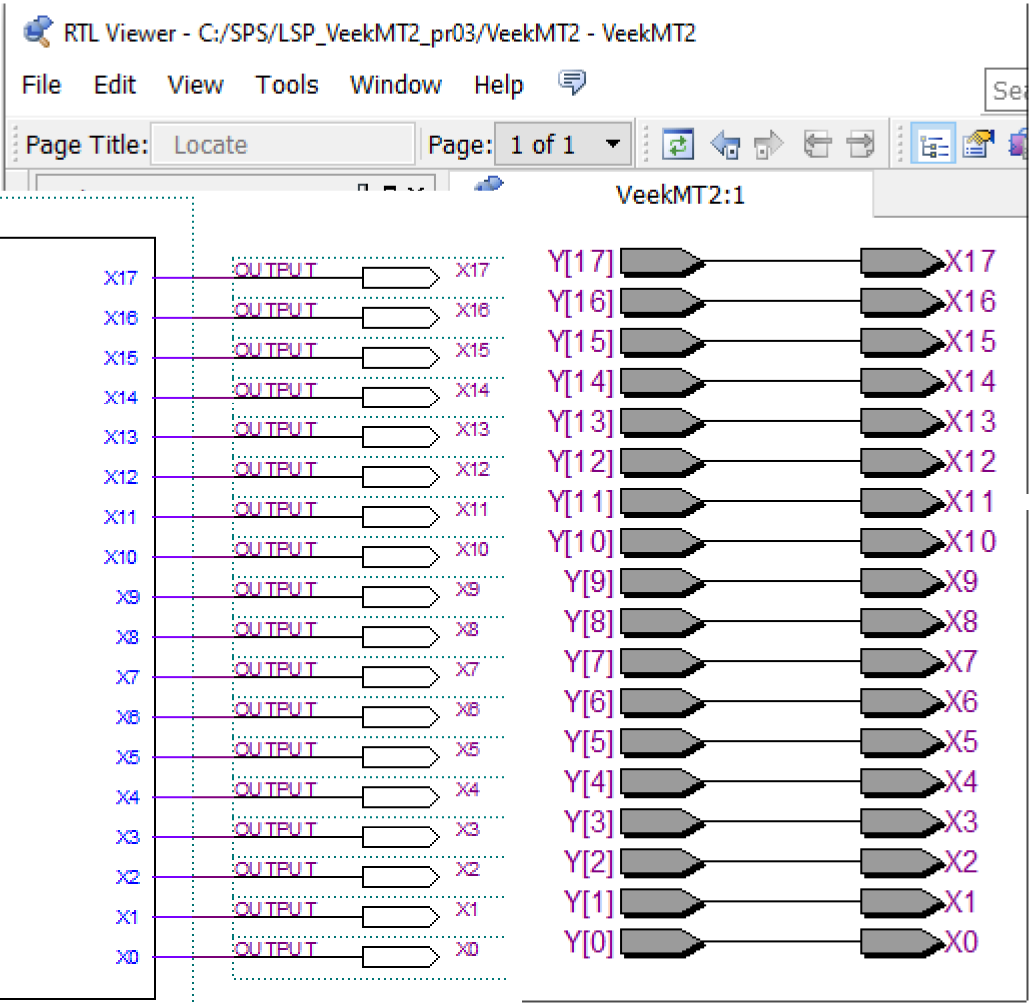
```
library ieee; use ieee.std_logic_1164.all;
entity toWires18 is
port ( Y : in std_logic_vector(17 downto 0);
   X17,X16,X15,X14,X13,X12,X11,X10,X9 :out std_logic;
   X8,X7,X6,X5,X4,X3,X2,X1,X0 :out std_logic);
end entity;
architecture rtl of toWires18 is
begin -- architecture
  X0<=Y(0); X1<=Y(1); X2<=Y(2); X3<=Y(3); X4<=Y(4);
  X5<=Y(5); X6<=Y(6); X7<=Y(7); X8<=Y(8); X9<=Y(9);
  X10<=Y(10);X11<=Y(11); X12<=Y(12);X13<=Y(13);
  X14<=Y(14);X15<=Y(15);X16<=Y(16);X17<=Y(17);
end architecture;
```

Important parts of toWires18.vhd

```
port ( Y : in std_logic_vector(17 downto 0);
   X17,X16,X15,X14,X13,X12,X11,X10,X9 :out std_logic;
   X8,X7,X6,X5,X4,X3,X2,X1,X0 :out std_logic);


  X0<=Y(0); X1<=Y(1); X2<=Y(2); X3<=Y(3); X4<=Y(4);
  X5<=Y(5); X6<=Y(6); X7<=Y(7); X8<=Y(8); X9<=Y(9);
  X10<=Y(10);X11<=Y(11); X12<=Y(12);X13<=Y(13);
  X14<=Y(14);X15<=Y(15);X16<=Y(16);X17<=Y(17);
```

Schema + RTLViewer: toWires18





TASKS DIRECTLY SOLVABLE BY MULTIPLEXER

[Pic: https://en.wikipedia.org/wiki/Stepping_switch]

Multiplexers :-)

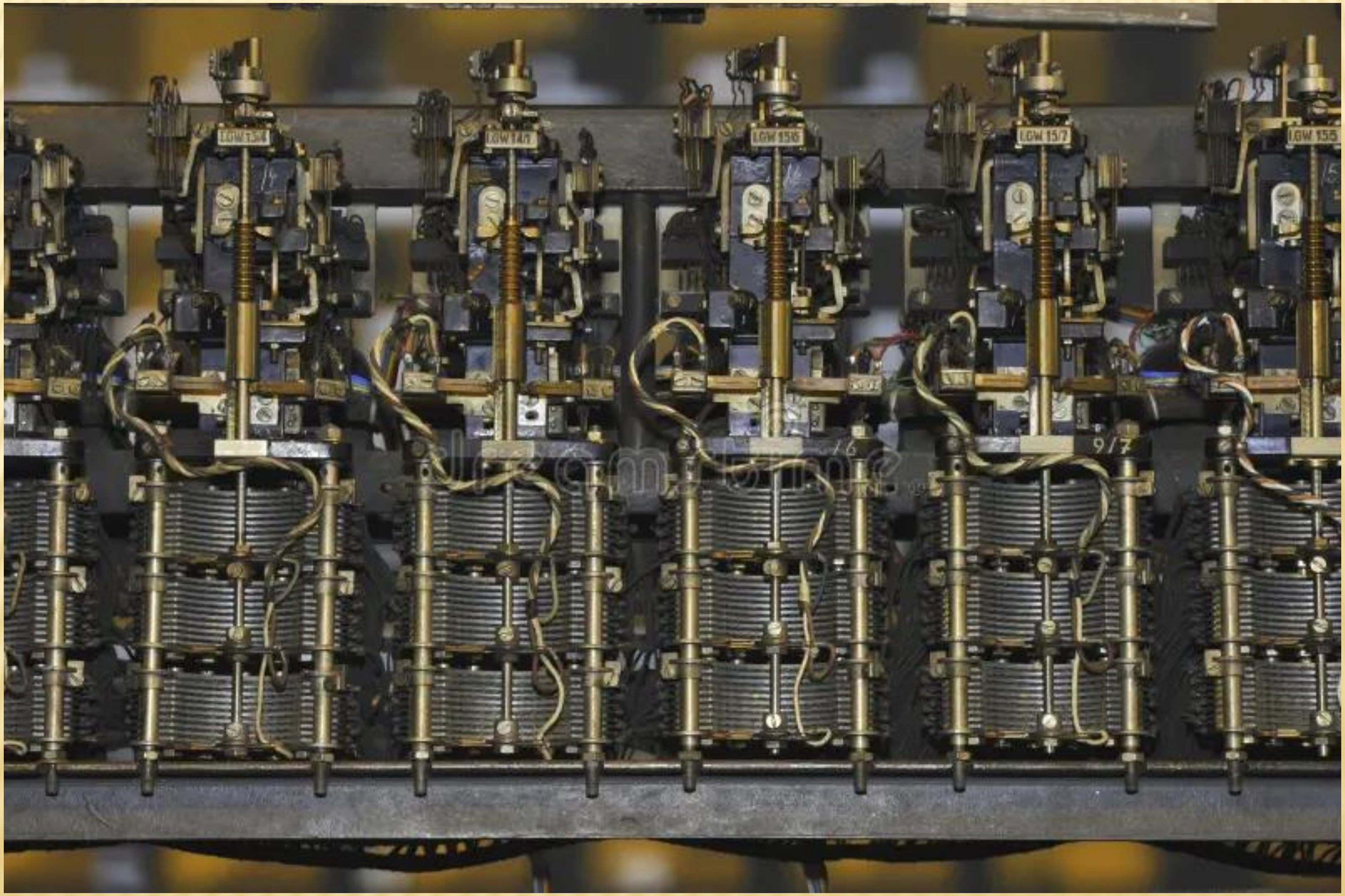


Image: www.dreamstime.com

Almost all "Concurrent" commands (connections) dataflow description *

<=

with-select-when

when-else

Used For

<=	with-select-when	when-else
Concurrent assignments	Selective assignments	Conditional assignments

implemented by

<=	with-select-when	when-else
wires	multiplexer selection 1 of $N=2^m$	cascades 2-input multiplexers

* *The list does not include **for-generate** to repeated insertions of codes and **process** - sequential code domain.*

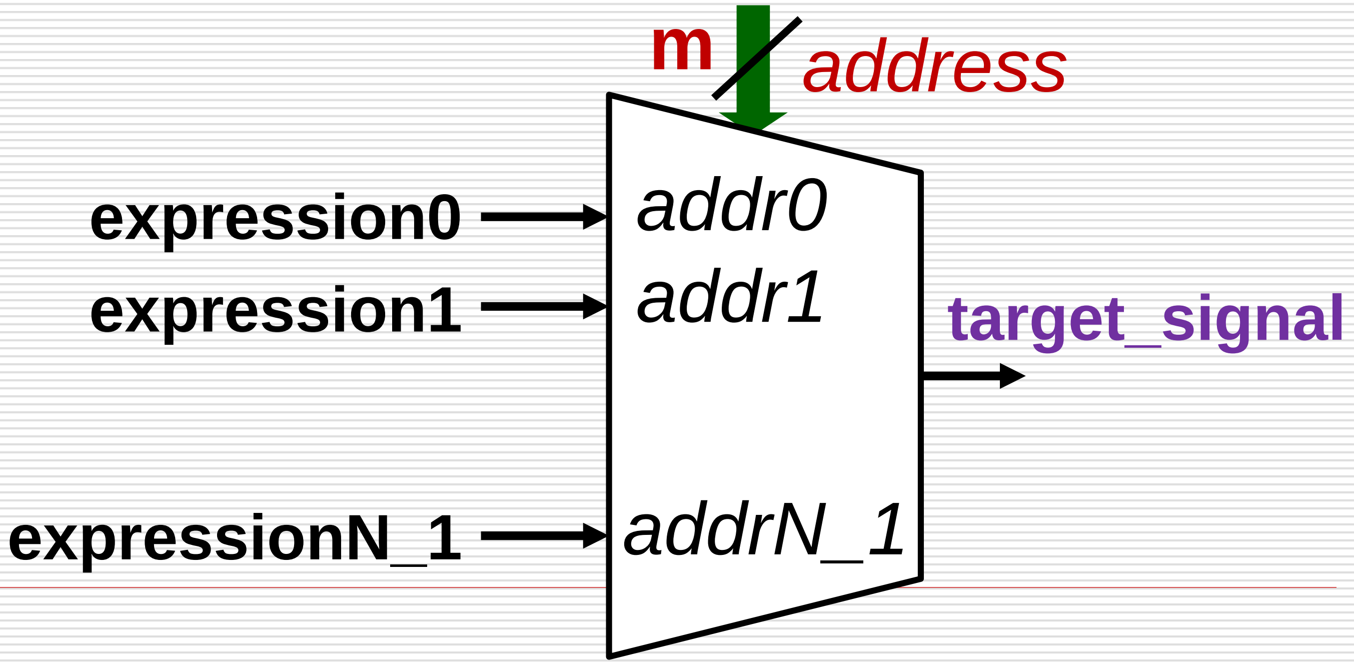
With -Select-When

Selective Signal Assignment

m-bit address selects one of the N=2^m -1 inputs

```
with address select
        target_signal <= expression0 when addr0,
                         expression1 when addr1,
                         . . .
                        expressionN_1 when addrN_1;
```

All inputs of any multiplexer must always obtain assigned values!



Select Statement

- *Conceptually, we can consider the selected signal assignment as a switch that routes the input to the output specified by an address.*
- *All choices must be included, unless the others clause is used as the last choice. Ranges and selections can be used as the choice, but it is not allowed for choices to overlap.*
- More: Circuit Design With VHDL dataflow and structural
  Chapter 3, page 22.

Switch vs with..select

```
bool target_signal, x, y; int address;
 //...some statements
 switch ( address&0x3 )
   {
    case 0: target_signal = true; break; // expression0
    case 1: target_signal = (x^y); break; // expression1
    case 2: target_signal = !y; break; // expression2
    default: target_signal = false; break; // expressionN_1
   }
```

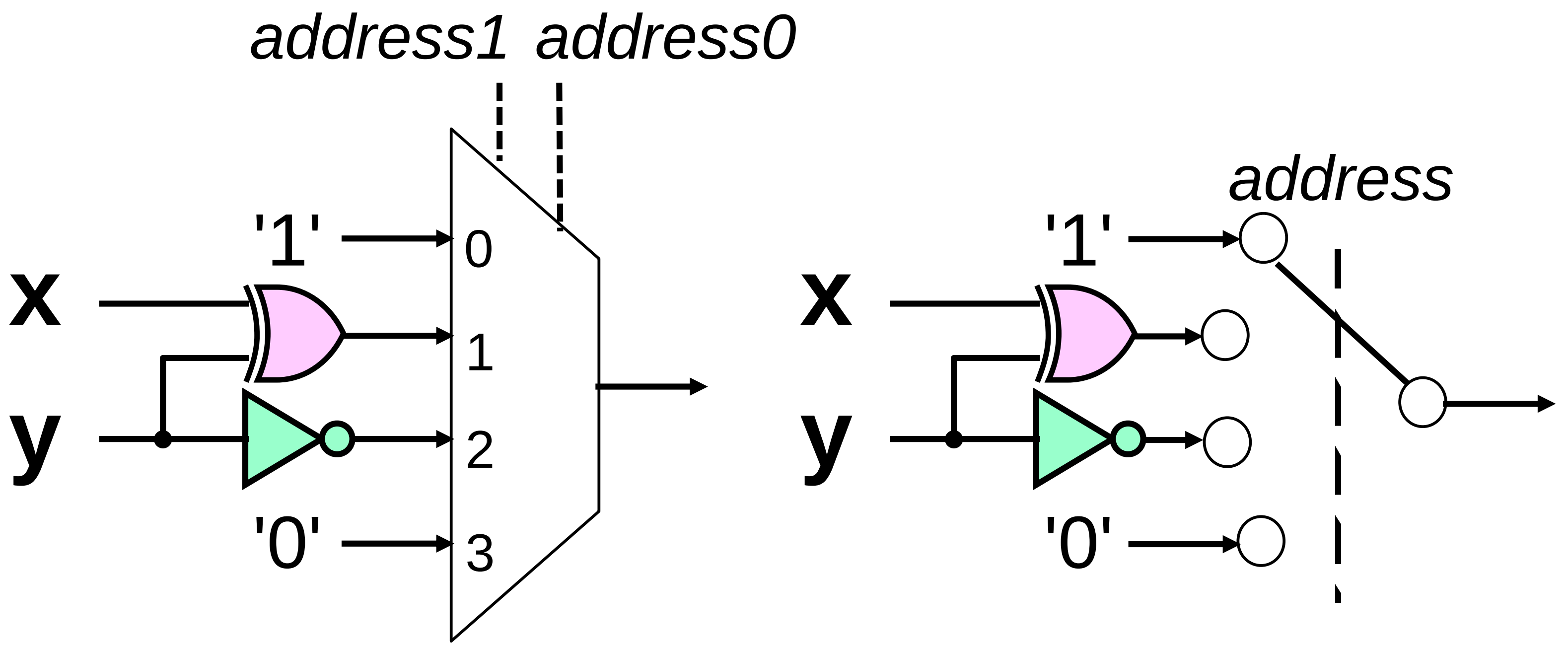
Switch vs with..select

```
bool target_signal, x, y; int address;
 //...some statements
 switch (address&0x3)
      {case 0: target_signal = true; break; // expression0
       case 1: target_signal = (x^y); break; // expression1
       case 2: target_signal = !y; break; // expression2
       default: target_signal = false; break; // expressionN_1
      }
```

```
signal target_signal, x, y : std_logic;
signal address : std_logic_vector(1 downto 0);
begin – architecture----------------------------------------------
-----
with address select
      target_signal <= '1' when "00",
                       x or y when "01",
                       not y when "10",
                      '0' when others; -- default
```

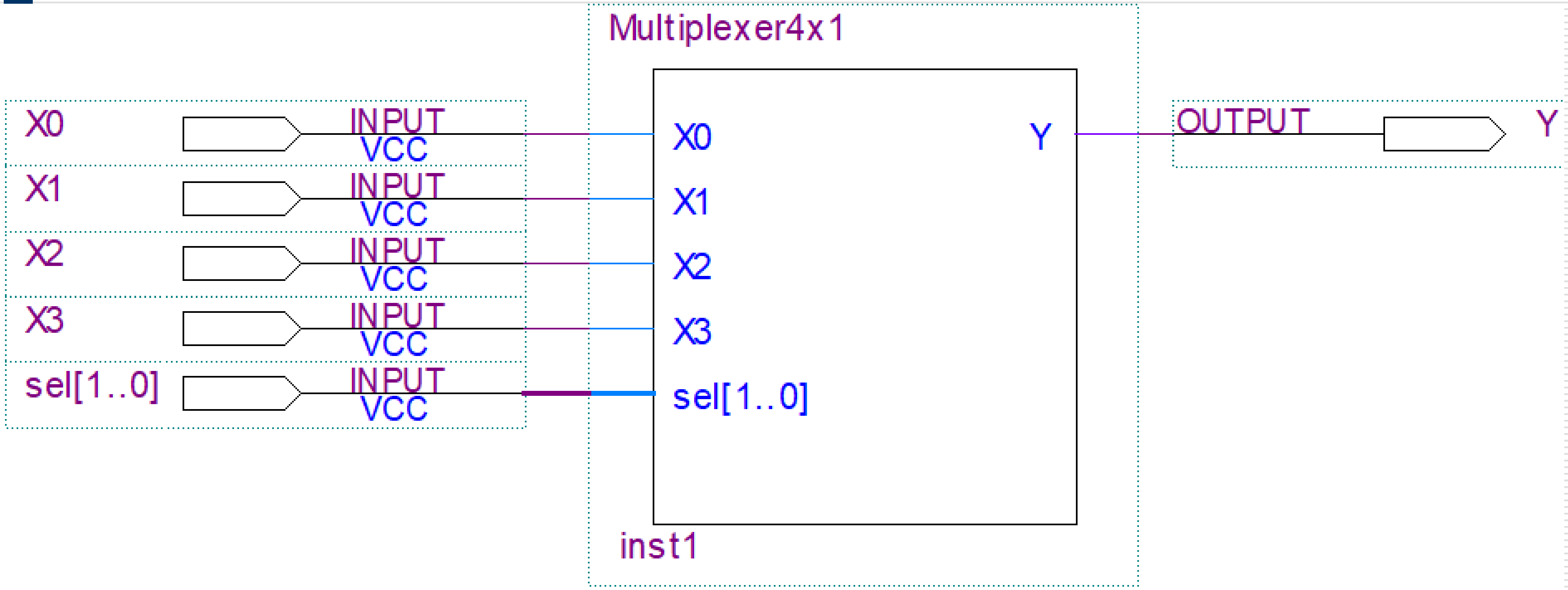
***others** - everything not mentioned so far, in MVL-9 the address signal can have 81 values*

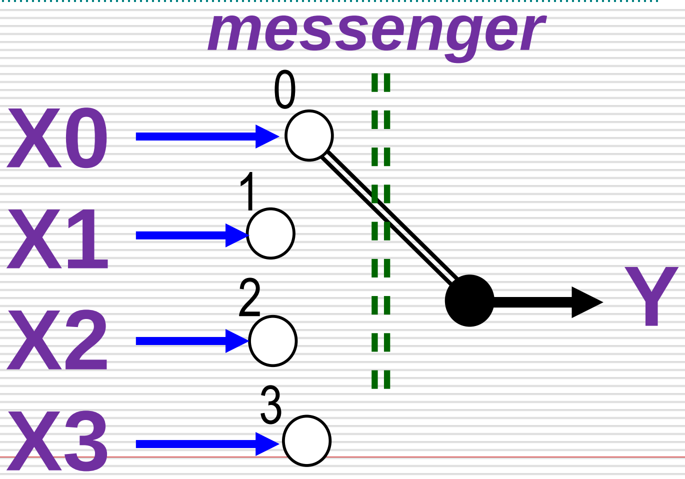
with...select



```
with address select
        target_signal <= '1' when "00",
                          x or y when "01",
                          not x when "10",
                          '0' when others; --
default
```

DCE/library Muxliplexer4x1





Main parts Multiplexer4x1.vhd

Inputs and outputs

```
   X0,X1,X2,X3 : in std_logic;
  sel : in std_logic_vector(1 downto 0);
   Y : out std_logic;
```

Multiplexer command

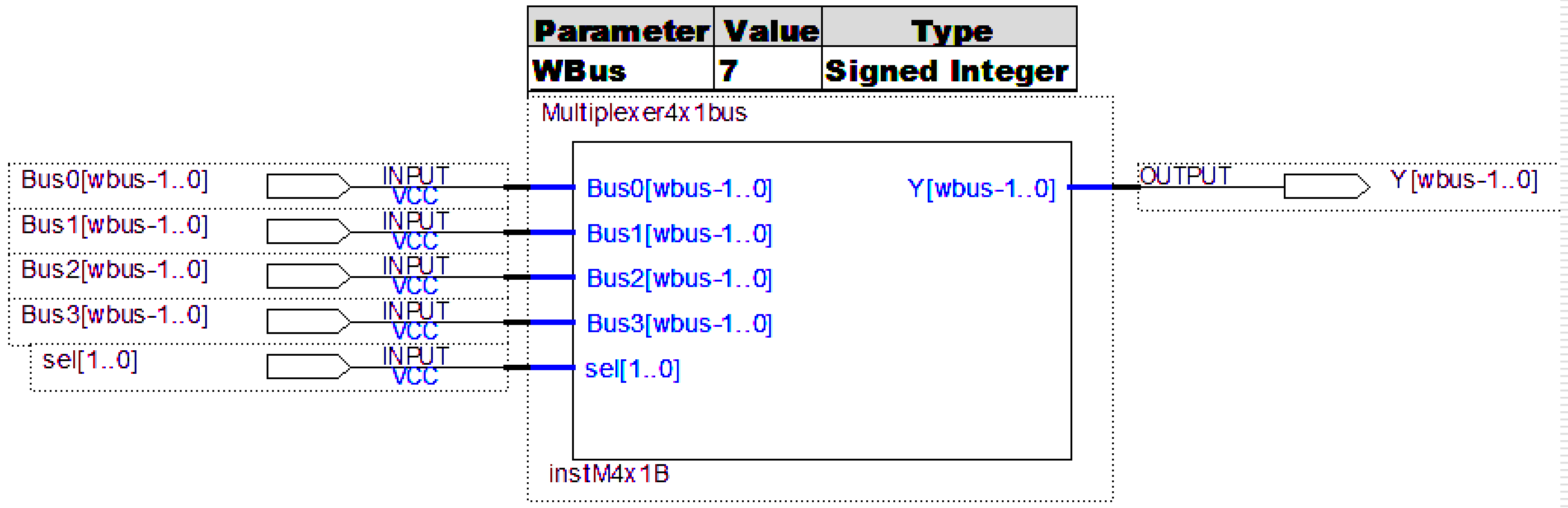
```
with sel select
   y <= X0 when "00", X1 when "01",
        X2 when "10",
        X3 when others;
```

The VHDL keyword others specifies all members not yet listed.

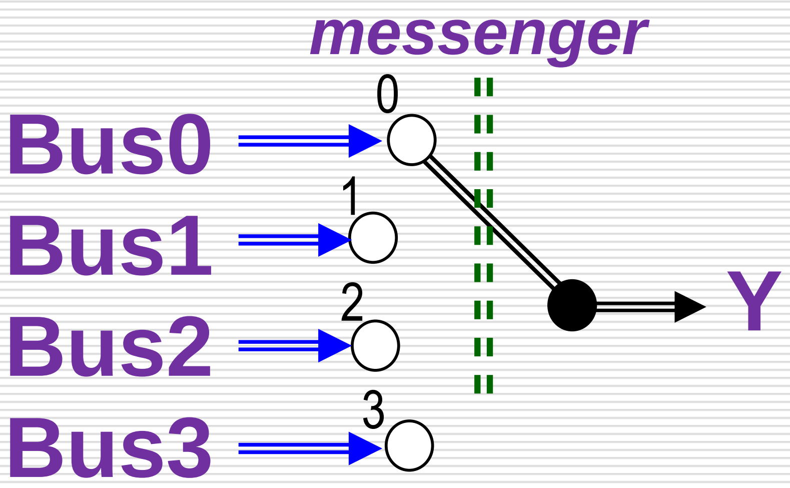
Full code: multiplexer4x1.vhd

```
library ieee;use ieee.std_logic_1164.all;
entity Multiplexer4x1 is
port -- inputs and outputs
( X0, X1, X2, X3 : in std_logic;
  sel : in std_logic_vector(1 downto 0);
   Y : out std_logic);
end entity;
architecture behavorial of Multiplexer4x1 is
begin
with sel select
     y <= X0 when "00", X1 when "01", X2 when "10",
     X3 when others;
end architecture;
```

DCE/library Muxliplexer4x1bus



Parameter	Value	Type
WBus	7	Signed Integer



Muxltiplexer4x1 vs Multiplexer4x1bus

<u>The signal names only changed in the entity:</u>

```
with sel select
y <= Bus0 when "00", Bus1 when "01",
   Bus2 when "10", Bus3 when others;


with sel select
 y <= X0 when "00", X1 when "01",
        X2 when "10", X3 when others;
```

Multiplexer4x1 vs Multiplexer4x1bus

```
entity Multiplexer4x1bus is
generic ( WBus : natural := 16 ); -- The width of input buses
port( Bus0, Bus1, Bus2, Bus3 : in std_logic_vector((WBUS-1) downto 0);
                               sel : in std_logic_vector(1 downto 0);
                                Y : out std_logic_vector((WBUS-1) downto 0) );
begin --The consistency check of our generic parameter
assert WBus>0 report "Expected the width of buses WBus>0"
        severity ERROR;
end entity;
=======================================
entity Multiplexer4x1 is
port ( X0, X1, X2, X3 : in std_logic;
                  sel : in std_logic_vector(1 downto 0);
                    Y : out std_logic );
end entity;
```

Multiplexer4x1bus.vhd

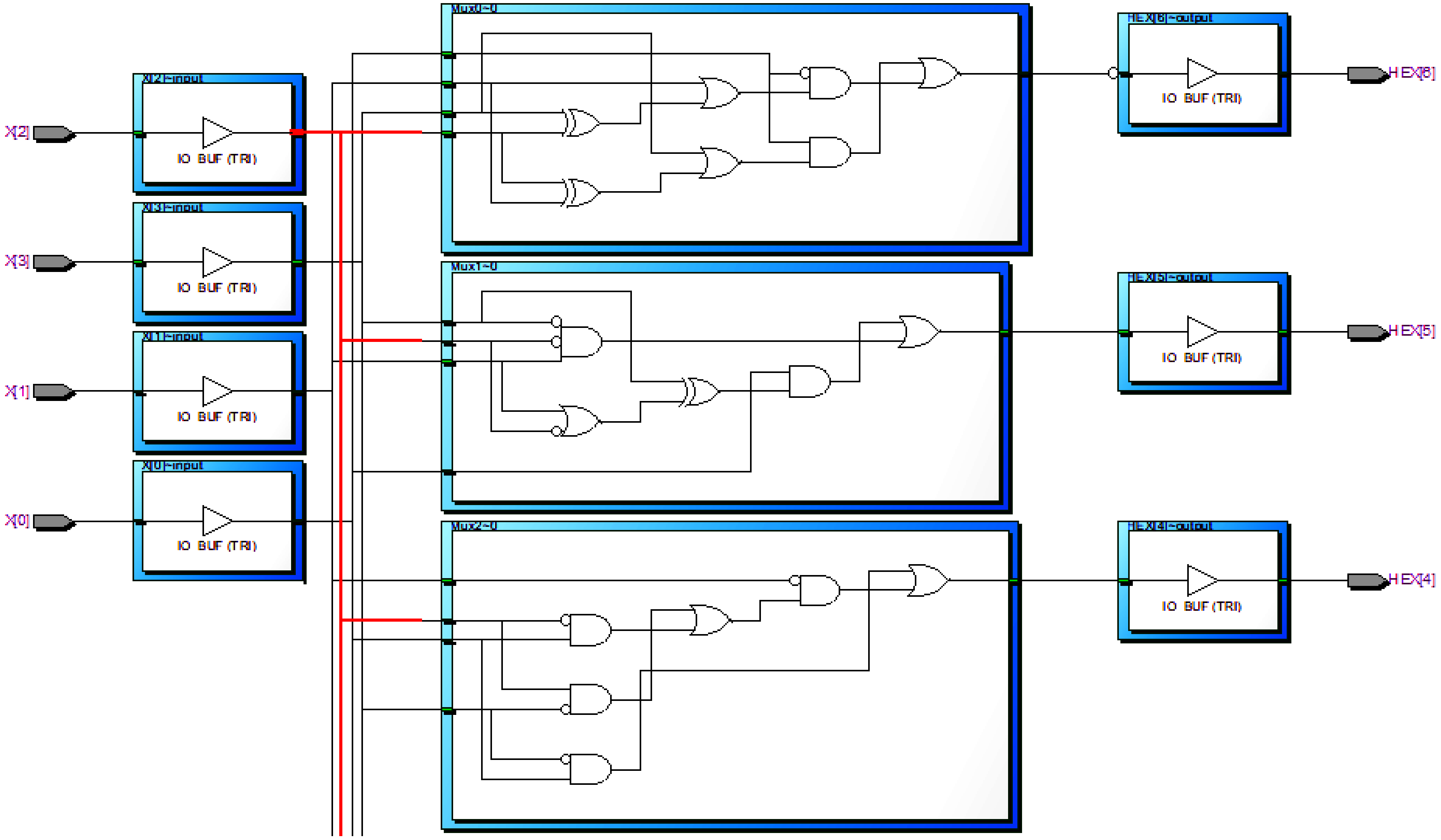
```
library ieee;use ieee.std_logic_1164.all;
entity Multiplexer4x1bus is
generic ( WBus : natural := 16 ); -- The width of input buses
port( Bus0, Bus1, Bus2, Bus3 : in std_logic_vector((WBUS-1) downto 0);
                                 sel : in std_logic_vector(1 downto 0);
                                  Y : out std_logic_vector((WBUS-1) downto 0));
begin
  assert WBus>0 report "Expected the width of buses WBus>0"
  severity ERROR;
end entity;
architecture behavorial of Multiplexer4x1bus is
begin
with sel select
  y <= Bus0 when "00", Bus1 when "01", Bus2 when "10",
         Bus3 when others;
end architecture;
```

to7SegHex written by multiplexer

```
library ieee; use ieee.std_logic_1164.all;
entity to7SegHex IS
      port( X :in std_logic_vector(3 downto 0); -- A(3)-MSB A(0)-LSB hex input
      HEX: out std_logic_vector(6 downto 0) );-- HEX[6..0] output to 7 segment display
end entity;
architecture behavioural of to7SegHex IS
begin
   -- On VEEK-MT2 board, 7segment LEDs are on when its input = '0'
   with X select
    hex <= "1000000" when "0000", "1111001" when "0001", -- 0, 1
           "0100100" when "0010", "0110000" when "0011", -- 2, 3
           "0011001" when "0100", "0010010" when "0101", -- 4, 5
           "0000010" when "0110", "1111000" when "0111", -- 6, 7
           "0000000" when "1000", "0010000" when "1001", -- 8, 9
           "0001000" when "1010", "0000011" when "1011", -- A, b
           "1000110" when "1100", "0100001" when "1101", -- C, d
           "0000110" when "1110", "0001110" when others; -- E, F "1111"
end architecture;
```

Technology Map Viewer

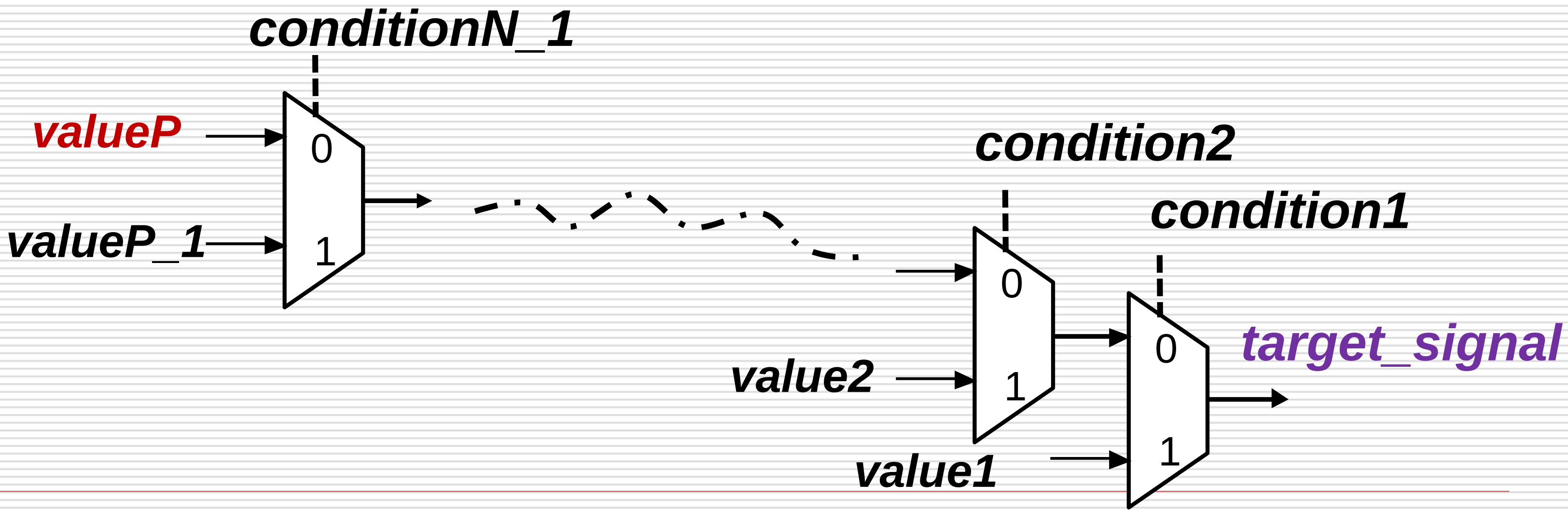
will show that Quartus has converted the multiplexor into logic equations



Implementation of *Conditional Concurrent Assignment*

When - Else

```
target_signal <=  value1 when condition1 else
                  value2 when condition2 else
                  . . .
                  valueP_1 when conditionN_1 else
                  ValueP;
```

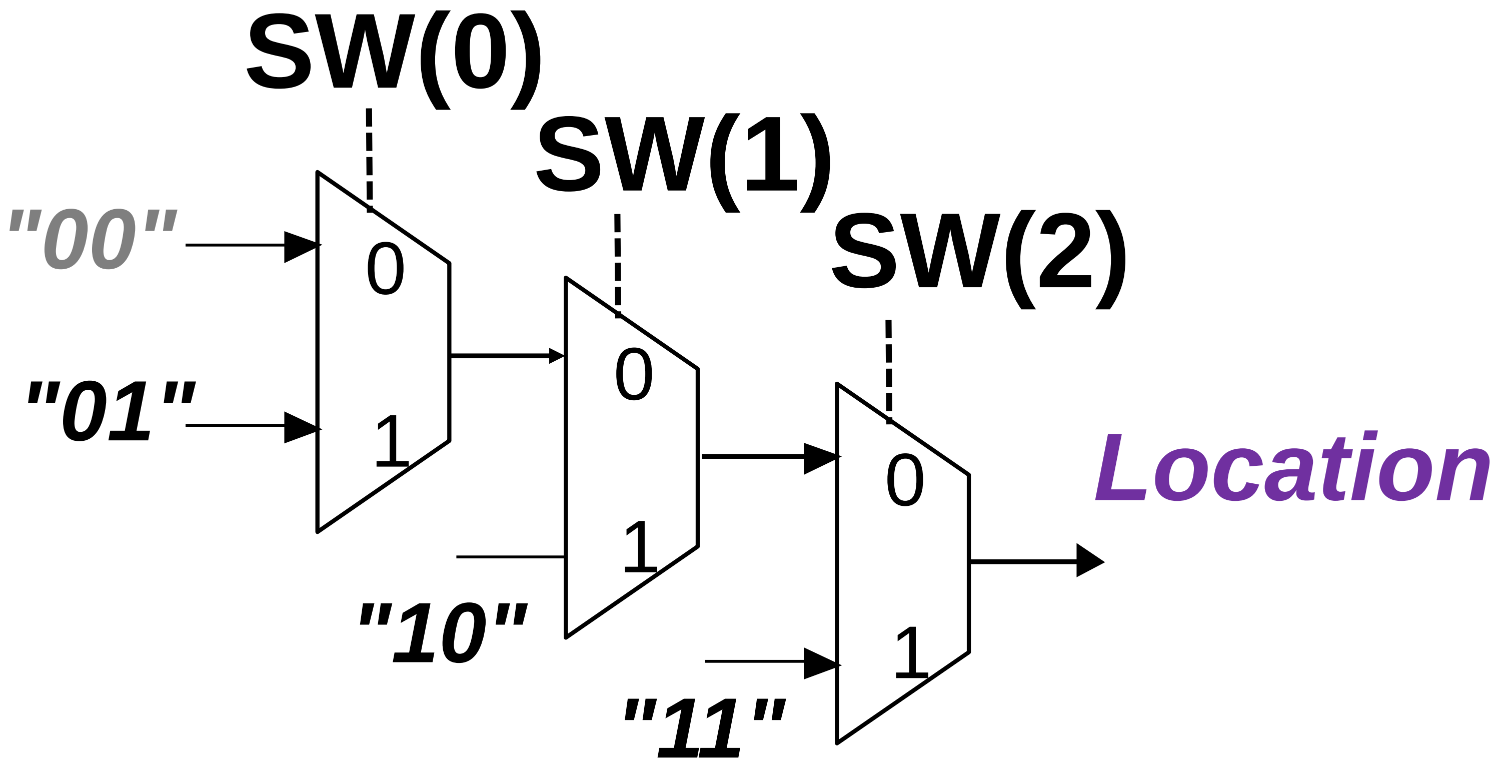


When-else

- **When-else** statement creates a cascade of Boolean expressions (multiplexers) arranged in descending order, from the highest priority to the lowest - about the output signal definitely the first expression for which the condition is fulfilled.
- The structure of the **when-else** statement implies a priority structure; Boolean expressions are evaluated in the specified order and the earlier one has a higher priority. The first true condition, it sends its value to the output, all the lower ones are already ignored.

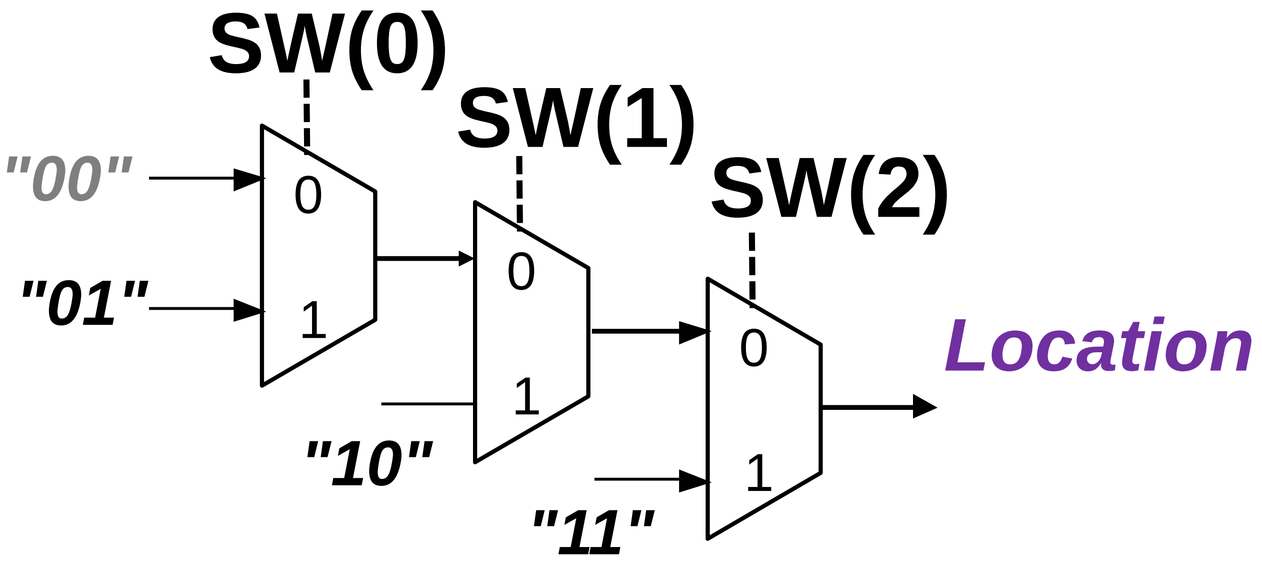
Example: position of the highest switch

SW(0):1, SW(1):2, SW(2):3



Switch position

```
byte Location; bool sw2, sw1, sw0; -- switch
//...some statements
if (sw2) Location = 3;
else {
        if (sw1) Location = 2;
         else     Location = (sw0 ? 1 : 0);
        }
```



Switch position

```
byte Location; bool sw2, sw1, sw0; -- switch
//...some statements
if (sw2) Location = 3;
else { if (sw1) Location = 2;
        else { Location = (sw0 ? 1 : 0); }
       }
```

```
signal Location : std_logic_vector(1 downto 0);
-----
begin -- architecture-----------------------------------
   Location <= "11" when SW(2) else
                "10" when SW(1) else
                "01" when SW(0) else "00";
end architecture;
```

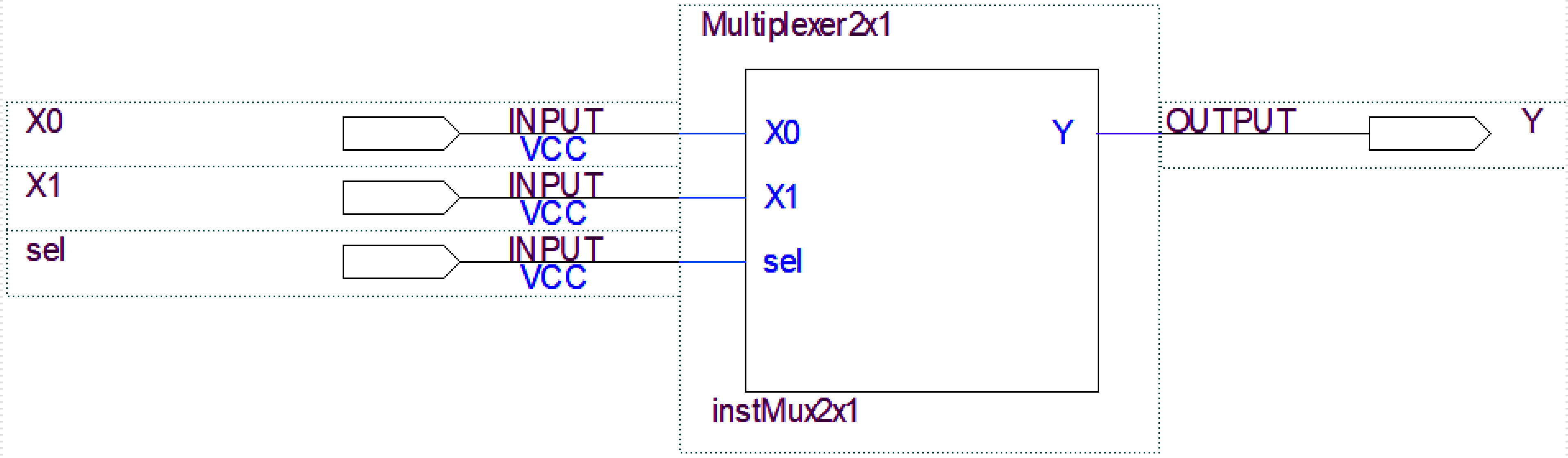
From the beginning of electronics till nowadays, many electronic circuits are based on multiplexers, so we should know possible constructions with them.

[Pic: https://en.wikipedia.org/wiki/Stepping_switch]

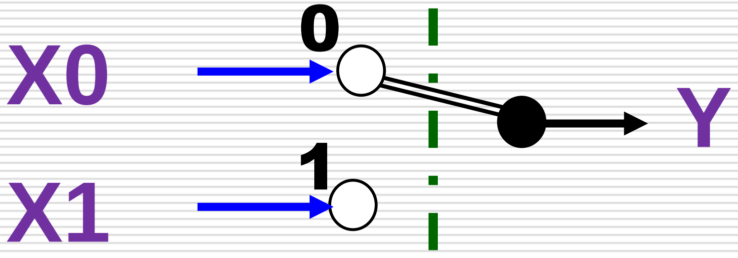
Example

Multiplexer 2 to 1

DCE/library Multiplexer2x1



sel=selection



Multiplexor 2x1 is a conditional assignment

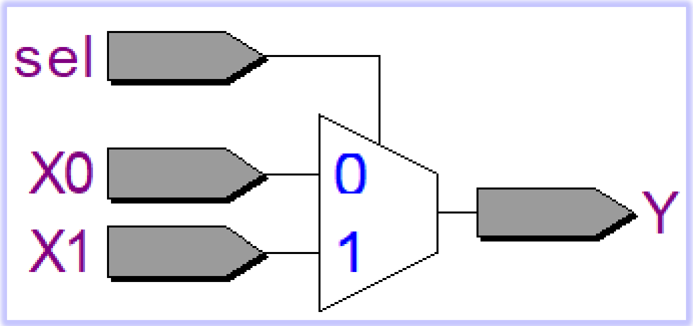
```
bool x, y, z;
//...
target_signal = z ? x && y : x || y;
```

Two main ways of describing it:

```
signal target_signal1, target_signal2, x, y, z : std_logic;
begin -- architecture --
-- 1. one 2x1 multiplexer from the cascade
target_signal1 <= x and y when z else x or y;

-- 2nd multiplexor 2x1
with z select
       target_signal2 <= x and y when z,
                         x or y when others;
end architecture;
```

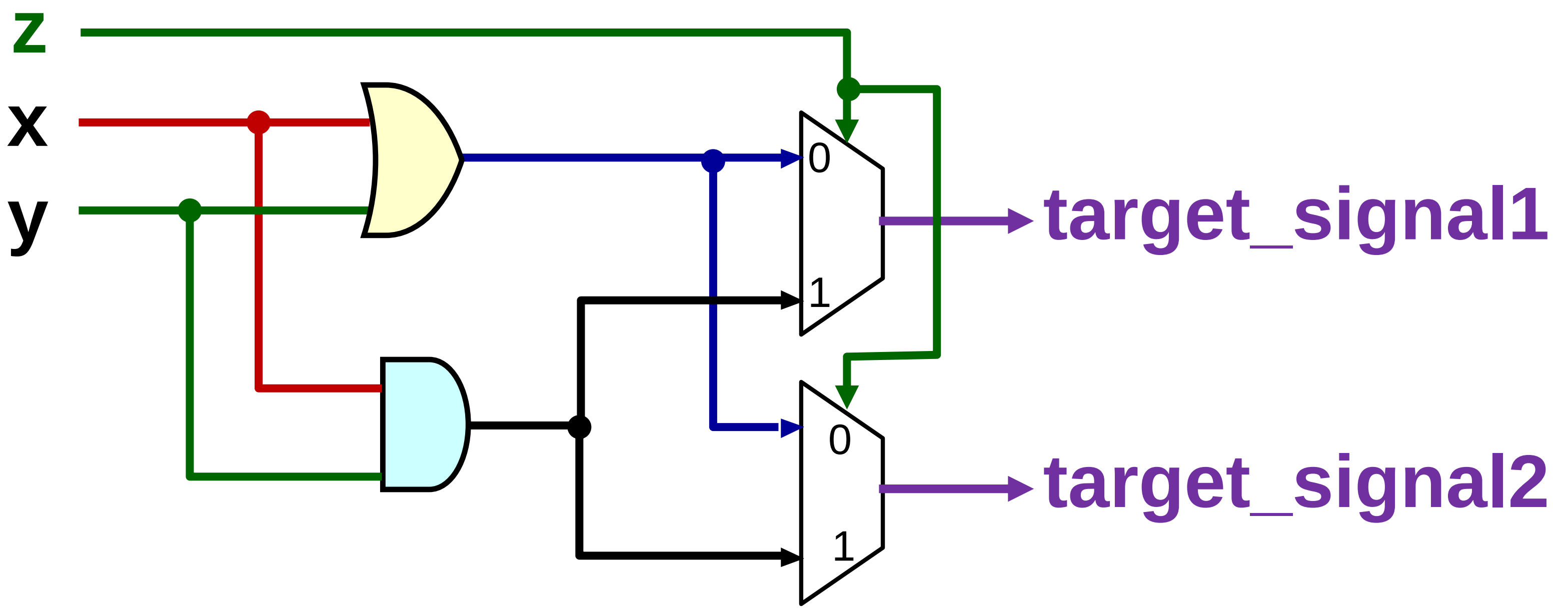
Multiplexor 2x1

```
library ieee;use ieee.std_logic_1164.all;
entity Multiplexer2x1 is
port ( X0, X1 : in std_logic;
            sel : in std_logic;
              Y : out std_logic);
end entity;
architecture behavorial of Multiplexer2x1 is
begin
    Y <= X1 when sel else X0;
  -- Y <= X1 when sel='1' else X0; -- VHDL93
end architecture;
```



Both options lead to the same wiring

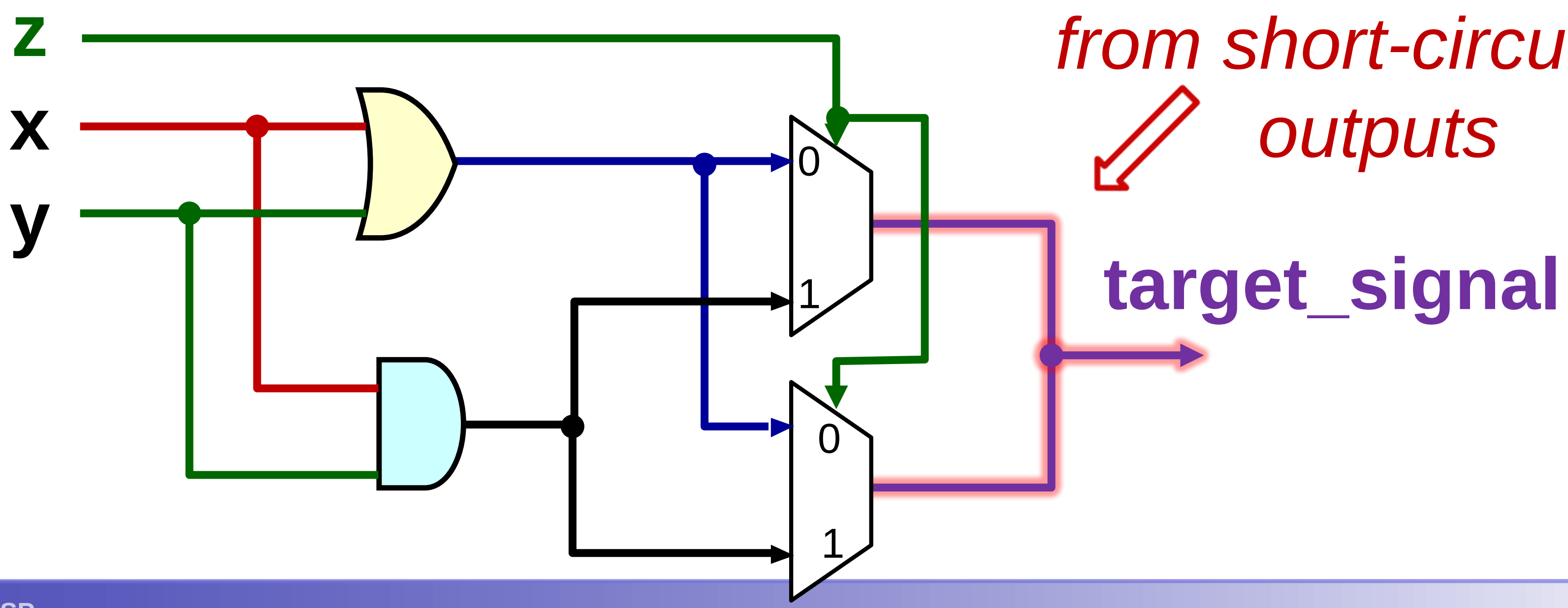
```
signal target_signal1, target_signal2, x, y, z : std_logic;
begin -- architecture --
    target_signal1 <= x and y when z else x or y;

    with z select
      target_signal2 <= x and y when z,
                        x or y when others;
```



The signal is not a variable, but a conductor!

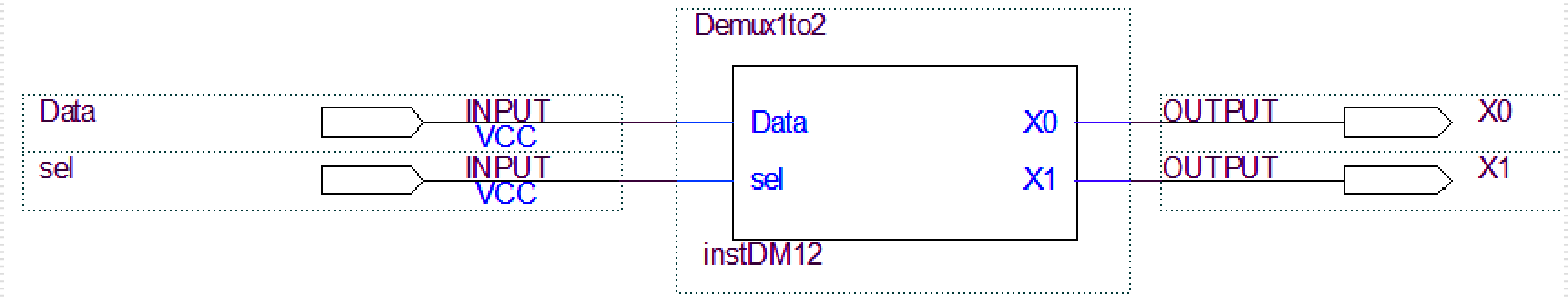
```
signal target_signal, x, y, z : std_logic;
begin -- architecture --
  target_signal <= x and y when z else x or y;

  with z select -- redefine first value --
   target_signal <= x and y when z,
                               x or y when others;
end architecture;
```

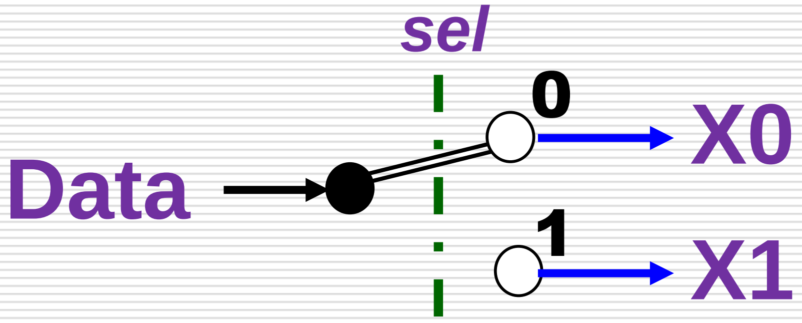
Fan-out from short-circuiting outputs

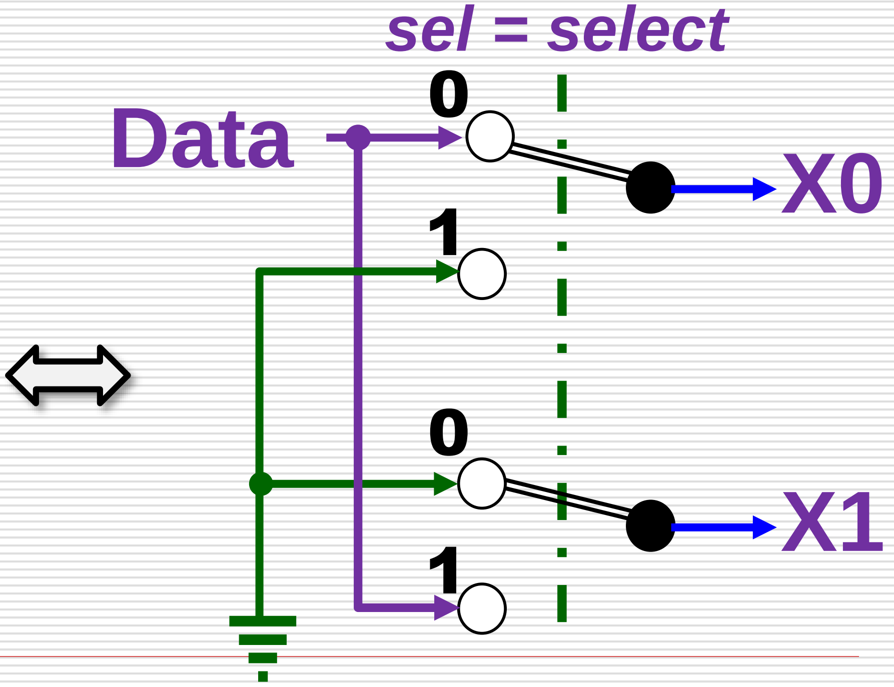


DEMUX FROM MUX

DCE/library Demux1to2 by Mux



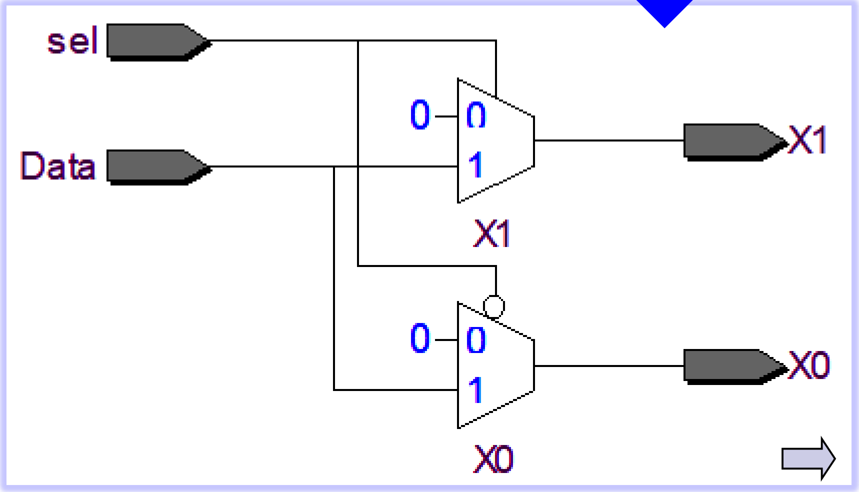




DCE/library Demux1to2

The Demux code is one description from many other possibilities.

```
library ieee;use ieee.std_logic_1164.all;
entity Demux1to2 is
port ( Data : in std_logic; -- data input
              sel : in std_logic; -- selection of output
          X0, X1 : out std_logic);
end entity;
architecture behavorial of Demux1to2 is
begin
   X0 <=  '0' when sel else data;
   X1 <= data when sel else '0';
end architecture;
```



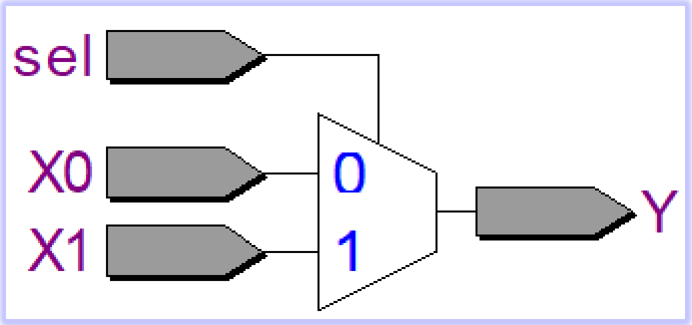
MUX OF BUSES

Multiplexer2x1bus

```
library ieee; use ieee.std_logic_1164.all;
entity Multiplexer2x1bus is
generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
port (Bus0, Bus1 : in std_logic_vector((WBUS-1) downto 0);
               sel : in std_logic;
               Y : out std_logic_vector((WBUS-1) downto 0));
  begin
    assert WBus>0 report "Excepted the width of buses WBus>0"
    severity ERROR;
end entity;
architecture behavorial of Multiplexer2x1bus is
begin
         y <= Bus1 when sel else Bus0;
end architecture;
```

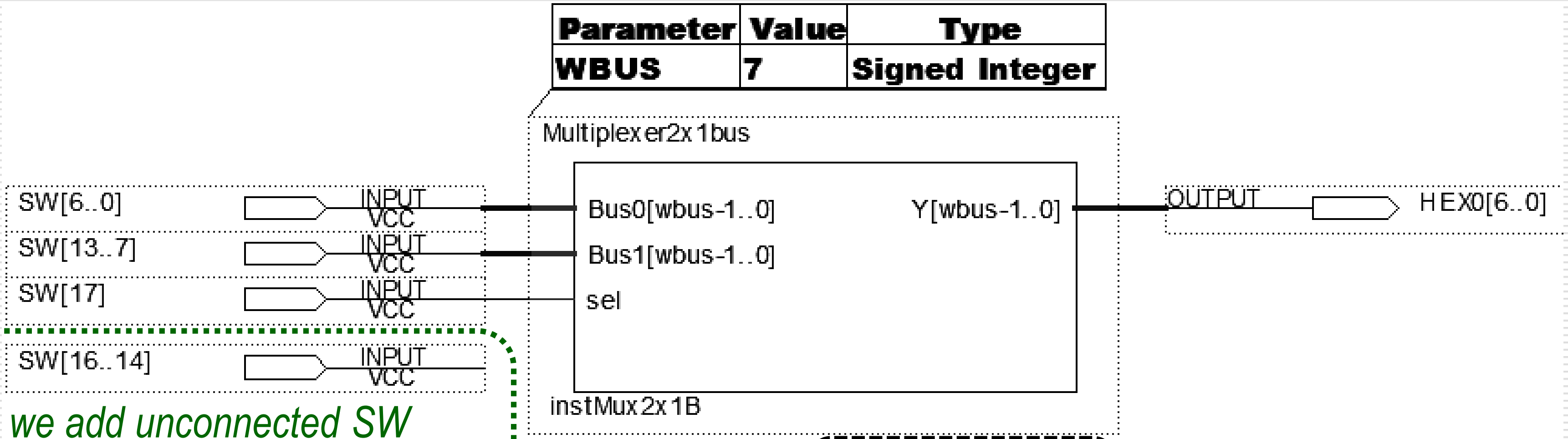
Multiplexor 2x1 vs Buses

```
library ieee;use ieee.std_logic_1164.all;
entity Multiplexer2x1 is  port ( X0, X1 : in std_logic;  sel : in std_logic;
                                 Y : out std_logic);
end entity;
architecture behavorial of Multiplexer2x1 is
begin Y <= X1 when sel else X0;
end architecture;
```



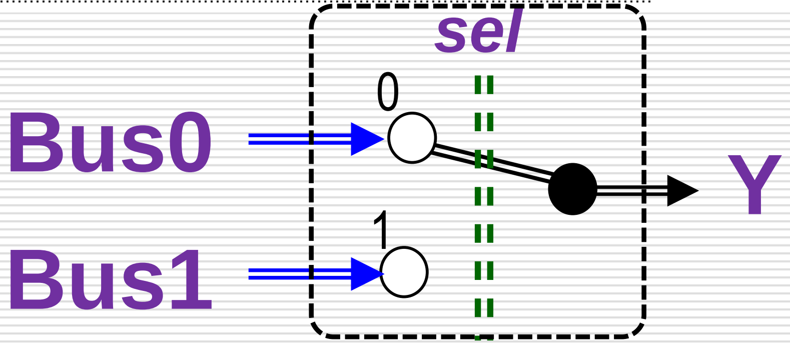
```
library ieee; use ieee.std_logic_1164.all;
entity Multiplexer2x1bus is
generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
port (Bus0, Bus1 : in std_logic_vector((WBUS-1) downto 0);  sel : in std_logic;
                  Y : out std_logic_vector((WBUS-1) downto 0));
begin assert WBus>0 report "Excepted the width of buses WBus>0"  severity ERROR;
end entity;
architecture behavorial of Multiplexer2x1bus is
begin   y <= Bus1 when sel else Bus0;
end architecture;
```

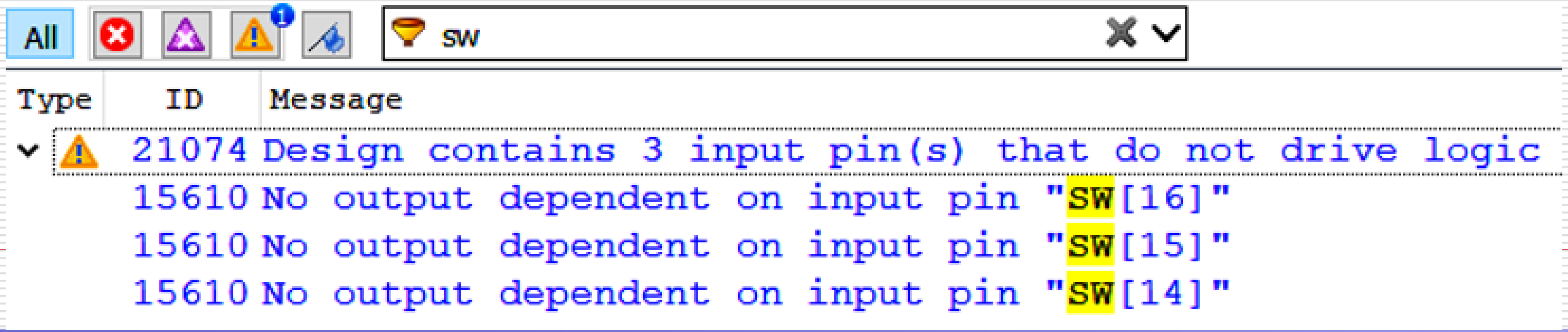
DCE/library Multiplexer2x1bus



Parameter	Value	Type
WBUS	7	Signed Integer

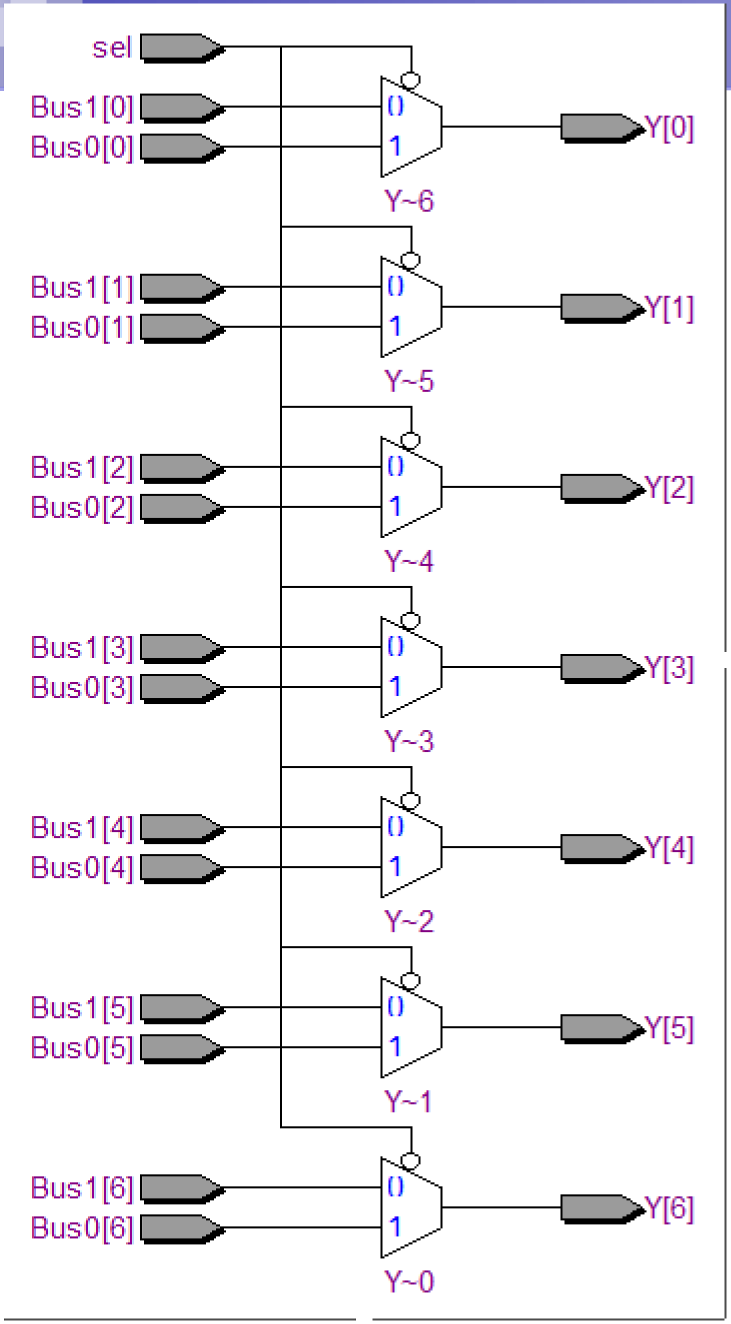
we add unconnected SW to continuous used series





```
Type  ID     Message
      21074  Design contains 3 input pin(s) that do not drive logic
      15610  No output dependent on input pin "SW[16]"
      15610  No output dependent on input pin "SW[15]"
      15610  No output dependent on input pin "SW[14]"
```

RTL Viewer

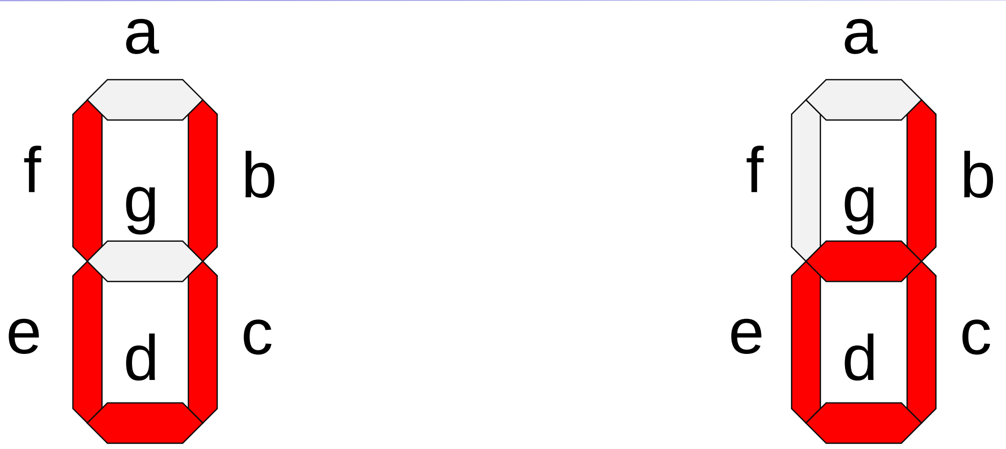


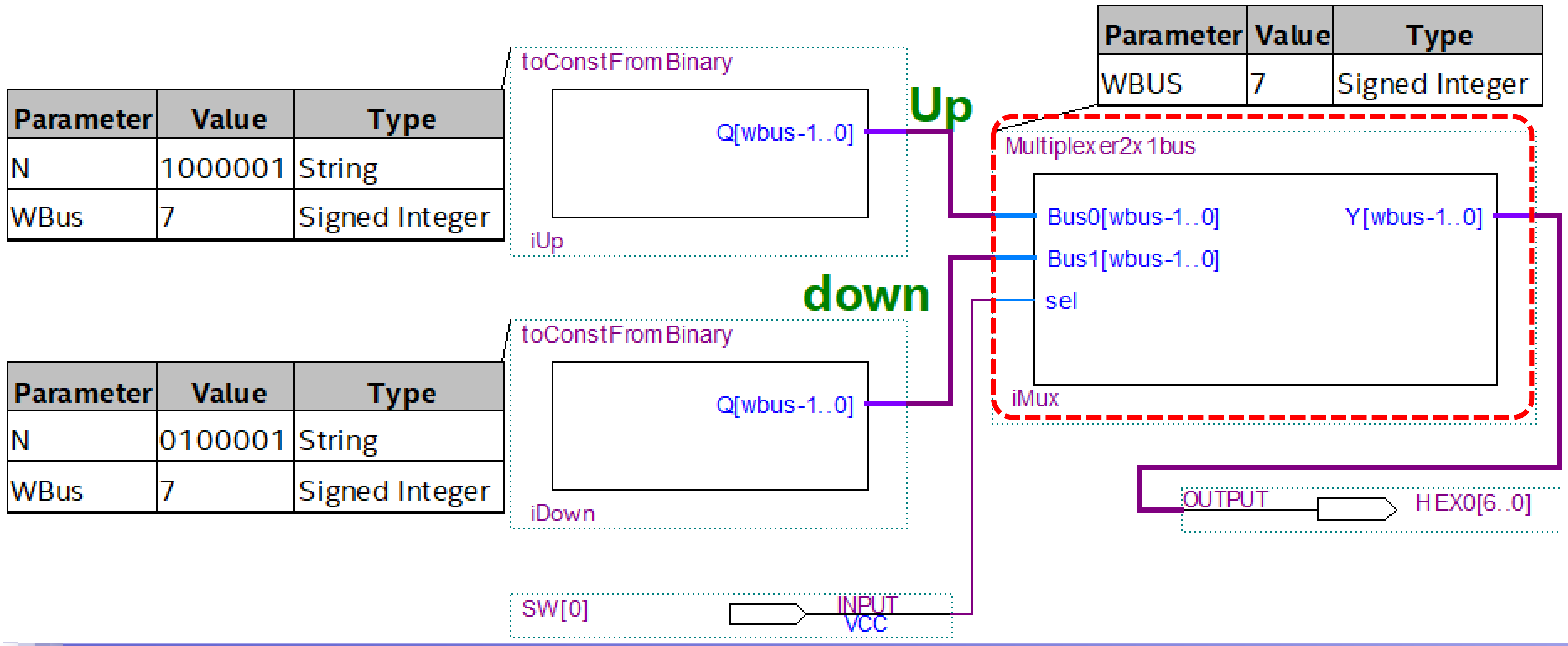
```
y <= Bus1 when sel else Bus0;
```

Example

Up Down on 7-segment

UpDownToHex.vhd





Parameter	Value	Type
N	1000001	String
WBus	7	Signed Integer

Parameter	Value	Type
N	0100001	String
WBus	7	Signed Integer

Parameter	Value	Type
WBUS	7	Signed Integer

Port map

- Our simple block diagram could be written directly in VHDL, as we show at the end, but we will use it to demonstrate creating instances in VHDL.

Component from Multiplexer2x1bus

```
component Multiplexer2x1bus is
  generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
  port (Bus0, Bus1 : in std_logic_vector((WBUS-1) downto 0);
          sel : in std_logic;
          Y : out std_logic_vector((WBUS-1) downto 0) );
end component;
```

```
library ieee; use ieee.std_logic_1164.all;
entity Multiplexer2x1bus is
generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
port (Bus0, Bus1 : in std_logic_vector((WBUS-1) downto 0);
                 sel : in std_logic;
                 Y : out std_logic_vector((WBUS-1) downto 0));
  begin
    assert WBus>0 report "Excepted the width of buses WBus>0"
    severity ERROR;
end entity;
```

UpDown_VHDL.vhd

```
library ieee; use ieee.std_logic_1164.all;
entity UpDown_VHDL is  port  ( SW : in std_logic_vector(0 downto 0):="0";
                                        HEX0 : out std_logic_vector(6 downto 0));
end entity;
architecture behavioral of UpDown_VHDL is
  component Multiplexer2x1bus is
     generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
     port -- inputs and outputs
     (    Bus0, Bus1    : in std_logic_vector((WBUS-1) downto 0);
        sel     : in std_logic;
        Y       : out std_logic_vector((WBUS-1) downto 0));
  end component;
 begin
  iMux  : Multiplexer2x1bus generic map(7)
                    port map("1000001", "0100001", SW(0), HEX0);
 end architecture;
```

More reliable ***named associations***
can be given in any order

```
iMux  : Multiplexer2x1bus
          generic map(WBUS => 7)
           port map(sel  => SW(0), Y  => HEX0,
                               Bus0 => "1000001",
                               Bus1 => "0100001" );
```

Shorter **positional associations**
must exactly follow the entity order

```
iMux  : Multiplexer2x1bus
         generic map( 7 )
         port map("1000001", "0100001", SW(0), HEX0);
```

UpDown_VHDL.vhd

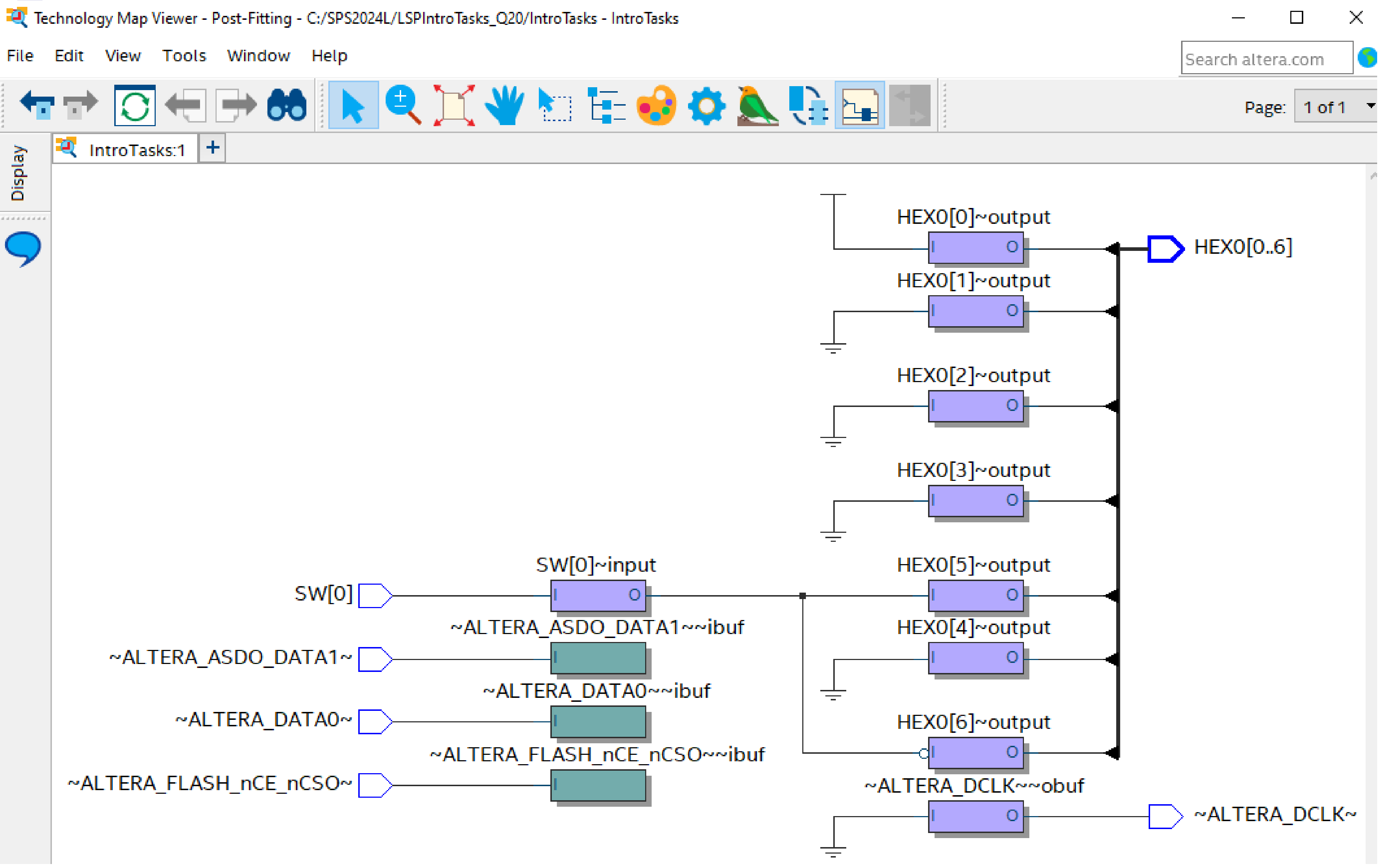
```
architecture behavioral2 of UpDown_VHDL is
component Multiplexer2x1bus is
    generic ( WBUS : natural := 16 ); -- WIDTH (LENGTH) of data Bus
    port -- inputs and outputs
    (   Bus0, Bus1    : in std_logic_vector((WBUS-1) downto 0);
        sel     : in std_logic;
        Y       : out std_logic_vector((WBUS-1) downto 0));
end component;
begin iMux  : Multiplexer2x1bus generic map(WBUS => 7)
                        port map(sel  => SW(0), Y  => HEX0,
                                        Bus0 => "1000001", Bus1 => "0100001");
end architecture;
```

Directly written UpDown_VHDL.vhd

```
library ieee; use ieee.std_logic_1164.all;
entity UpDown_VHDL is
    port  ( SW : in std_logic_vector(0 downto 0):="0";
            HEX0 : out std_logic_vector(6 downto 0));
end entity;

architecture behavioral3 of UpDown_VHDL is
begin
    HEX0 <= "1000001" when SW(0)='0' else
            "0100001";
end architecture;
```

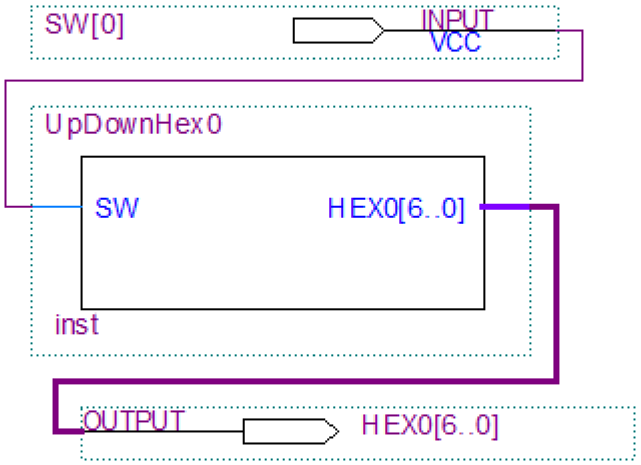
As a top-level entity, it connects to Veek-MT2 pins.

Technology Map Viewer



Differences std_logic vs std_logic_vector

As top-level entity, it does not directly use SW Veek-MT2 pin.



```
library ieee;
use ieee.std_logic_1164.all;
entity UpDownHex0 is
     port  (SW : in std_logic:='0';
            HEX0 : out std_logic_vector(6 downto 0));
end entity;
architecture behavioral3 of UpDownHex0 is
begin
      HEX0 <= "1000001" when  SW='0'  else
              "0100001";
end architecture;
```