How to use the libraries

Libraries and examples are coded with the high-level C++ like psC language. Whatever you can do in VHDL or Verilog can be done in psC, with the same quality of result: low resources and high speed.

psC is to VHDL what C++ is to assembly language.

Novakod Studio is the integrated development environment (IDE) for the ***psC language***.

- Compiling the psC code with the IDE generates quality VHDL code.
- The generated VHDL code can be used in any FPGA project.
- You can use the cores as is, or easily create your own customized core.
- Novakod Studio can be used to design any FPGA applications, parallel or sequential.

Material

You need ***Novakod Studio*** and the ***DE1SoC BSP*** (Board Support Package):

- Download at: https://icitechno.com/download.
- Licenses at: https://icitechno.com/licenses.

If you want to experiment with a real board, you need the DE1SoC board. This board is fully integrated into Novakod Studio. ***psC programs run without modification on the real board***.

- Buy at: Terasic.

Very important

Folder paths and file names you create must not contain spaces or special characters.

To begin with…

Always copy the libraries to *C:\Novakod_Studio\FreeCoresLib* with a double-click on *CopyLib.bat*.

Basic coding rules

Print this page...

The psC language is based on C++ syntax and everything you learned about designing, coding and documenting C++ programs can be used with psC. You should look at the examples to get a good feeling for the coding style.

— Naming convention

As in C++, carefully choose names for variables, ports, functions, components, and so on, to reflect their usage. This must be done as early as possible as it greatly improves readability. In psC, the recommended naming convention is capitalized first word letter, like ExeOpr.

— Indentation

Indentation of 4 spaces, no tabulation, is recommended for compatibility between editors.

— Suffixes and prefixes

Here is a list of prefixes and suffixes specific to psC. You should use them systematically.

<table>
<tr><th>Prefix</th><th>Suffix</th><th>Apply to</th><th>Example</th></tr>
<tr><td>C</td><td></td><td>Component</td><td rowspan="4"><pre>//   Component   Ports
component CTest (in int iP,
                 out int oP)
{ };

CTest PTest;   // Process</pre></td></tr>
<tr><td>P</td><td></td><td>Process</td></tr>
<tr><td>i</td><td></td><td>Input port</td></tr>
<tr><td>o</td><td></td><td>Output port</td></tr>
<tr><td></td><td>_t</td><td>Type</td><td>typedef int:3 int3_t;</td></tr>
<tr><td>c</td><td></td><td>Constant</td><td><pre>const int cLines[] = 1 to 2;
const int cCols [] = 1 to 3;
const identifier cId[] = { A, B, C };
enum Color_t { cRed, cGreen, cBlue };</pre></td></tr>
<tr><td>t</td><td></td><td>Temp variable</td><td rowspan="2"><pre>temp tAdd = V1 + V2;
temp fix8 tAdd(fix8 pF0, fix8 pF1) = pF0 + pF1;
function fct(int pVal, ubyte pTyp) { };</pre></td></tr>
<tr><td>p</td><td></td><td>Parameters</td></tr>
<tr><td></td><td>_T</td><td>Template</td><td rowspan="3"><pre>template< int NVAL, identifier NAME >
    function Add_T( ) { };

function Add_T<8, oPort> Add_I;</pre></td></tr>
<tr><td></td><td>_I</td><td>Template instance</td></tr>
<tr><td colspan="2">All Caps</td><td>Template parameters</td></tr>
<tr><td colspan="2">One to three capital letters</td><td>For…end parameters</td><td><pre>for I in <cRange>
    CInc PInc##I;
end</pre></td></tr>
<tr><td>s_ g_</td><td></td><td colspan="2">Reserved, do not use</td></tr>
</table>