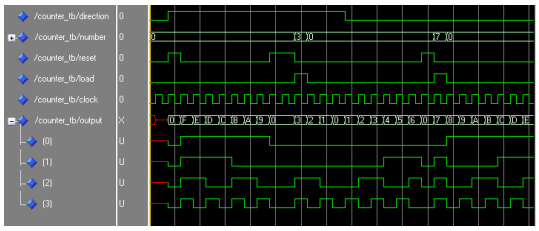
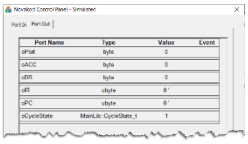
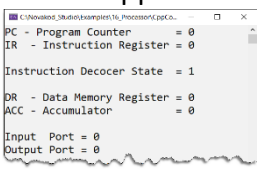
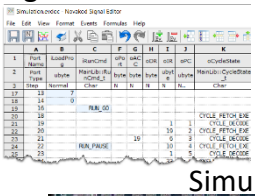
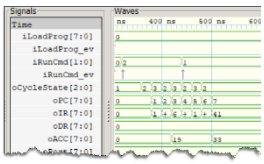
Actual tools vs Novakod Studio

	Coding with assembly languages VHDL-Verilog *The design consists in creating a clocked controlled digital system.* *It is complex, here is why.*	Coding with the high-level C++ like psC language *The design consists in creating an event controlled parallel system.* *It is simpler, here is why.*
Design	*Because of the complexity of implementation, it is difficult to explore many solutions.*	*A high-level language like psC allows the designer to explore the design space and find the optimal solution.*
Signals	Signals are used to interconnect everything: • components, • combinational functions, • registers. ***This makes the code very hard to read.***	Signals are only used for interconnecting components. • Combinational function's signals are ***hidden***. • Register's signals are ***hidden***.
Register value	It often is difficult to design the tree of interconnected combinational functions to provide the value assigned to a register.	As in C++, a *simple* ***assignment statement*** is used to set the register value: `Var = Expression;` The expression ***IS*** the combinational circuit. Assignments may be preceded by, and expressions may include `if` or `switch`.
Register Clock	You need to design a digital circuit, often complex, to manage, for each clock cycle, when a register is loaded.	Execution control is done by sending events to other components, ***clock is hidden***. Events trigger the execution of assignment statements.
Simulation and verification tools	Current tools rely on limited and not very friendly VHDL/Verilog simulation. 	Novakod Studio offers a variety of tools for simulation and test: Manual test  Test with a C++ application  Spreadsheet signal editor and viewer  GTK Wave Signal viewer  Simulated board 
Timing closure	Solved at low-level.	*Solved directly in high-level psC.*

VHDL vs psC

Type system	Types are complex and often not compatibles with each other's: *bit bit_vector boolean character integer std_logic std_logic_vector stc_ulogic signed unsigned real physical positive time*	Types are defined as in C++: *bit bool byte ubyte short ushort int uint long ulong float double time*
Fixed point	*Not supported.*	Fixed point types are native: *fix8 fix16 fix32 fix64*
Literal constants	Use unconventional syntax: *"1100" b"1100" "c"*	Use familiar C++ syntax: Decimal Binary Hex Character *10 01010 0xA '\n'*
Variable precision	Built in *bit_vector* and *std_logic_vector*.	Simple extension to int and uint: *int:5 uint:40*
Expressions and operators	Not conventional.	Almost identical to C++.
Control instructions	Different syntax depending on context: *when … else if … else … elseif, case … when with … select … when*	Only two instructions, used everywhere: *if … else* *switch … case*
Inferred latches	The synthesizer may create unwanted latches!	*Never happens.*
Using distributed memory and memory blocks	Requires the declaration of a component or special code to infer the memory. Using the memory involves connecting and controlling signals.	Declaration and usage are as in C++: *typedef int Arr_t[100];* *Arr_t Arr;* *Arr[77] = 3;* *Var = Arr[X * 2];* ***Implementation uses distributed or block memory***.
C++ like sequential instructions	*Not supported.* The term "sequential" refers to the order of evaluation of statements. They are used in combinational or clocked processes.	***True C++ like sequential instructions****:* *if…else switch…case do do…while break continue function call return* Each instruction takes one clock cycle.
Templates	*Not supported.*	As in C++, you can create templates for components and functions.

Let's not forget the intrinsic advantages of a high-level language, including saving time and $:

- Easier to learn
- Faster to code and debug
- Easier to document and maintain
- Accessible to C++ developpers
- Same quality in terms of resources usage and speed
- Promote usage of coding rules

An high-level language is essential for building a "codeware engineering" discipline for FPGA programming.