From Behavioral to RTL[1] Design Flow in SystemC

LLR[2]– PROSILOG[3] scientific collaboration

by
Emmanuel Vaumorin – PROSILOG
Thierry Romanteau – LLR

Contacts for PROSILOG: Emmanuel Vaumorin - vaumorin@prosilog.com
Contacts for LLR: Thierry Romanteau - Romanteau@poly.in2p3.fr

[1] Register Transfer Level description, depicts a textual description that can be processed by a synthesis tool

[2] Laboratoire Leprince Ringuet, Ecole Polytechnique, route de Saclay 91128 PALAISEAU (FRANCE)

[3] PROSILOG, 8 rue Traversière 95000 CERGY PREFECTURE (FRANCE)

Abstract

This paper reports the scientific collaboration between LLR and PROSILOG. The aim of this collaboration was to show the possibility to quickly implement a system into a FPGA, using SystemC[4] as the unique description language. Starting from behavioral abstraction level, the model, before hardware synthesis, is refined down to RTL then automatically translated to the equivalent model into VHDL or Verilog.

It will be shown that this design flow is less time consuming, more efficient and more reliable than the traditional C++ to HDL flow.

Introduction

The LLR is part of IN2P3[5], a physics research institute of the French scientific research center CNRS. The lab is currently involved in the Large Hadron Collider (LHC) project, a proton accelerator, currently being built at CERN[6]. Four detectors will study the very high energy events produced by the LHC. One of them is the Compact Muon Solenoid (CMS[7]). LLR teams are currently designing electronics systems for the Electromagnetic Calorimeter (ECAL) one of the four sub-detectors which will be mounted on CMS.

LLR started collaboration with PROSILOG in order to evaluate the SystemC modeling possibilities and the associated design flow. PROSILOG is a SystemC specialist: its engineers teams have been working with this language since its first release, developing and providing dedicated models, tools and solutions.

Goal of the collaboration

The challenge for the two partners was to prove the possibility to use SystemC for a full design and verification flow, from specification to implementation into an electronics device. The electronics system chosen as an example had already been implemented, following a classical top-down approach, and synthesized from VHDL code. The main purpose of this study was to use a smoother design flow, based on SystemC, and obtain by another way the RTL description. The final goal was to prove the equivalence of this last model with the reference model, and analyze the advantages of the new method.

Standard approach

The traditional flow used to implement this kind of electronics system is described in figure 1. It consists of the following steps:

- the required electronics system is first specified by the potential user.
- a behavioral model is then written in C++ and tested with a test bench created inside a classical or specific framework[8].
- this material (behavioral model + test bench) is then given to an electronics engineer team, who will write the equivalent VHDL or Verilog RTL model.
- after a verification phase, where the RTL model is validated, the prototype can be synthesized.

An evident disadvantage of this standard method is that the manual abstraction refinement from behavioral to RTL level is made at the same time as the manual language translation from C++ to HDL. It is proven that this procedure is time consuming and error prone. Moreover, in order to verify the RTL model, the C++ test bench has also to be manually translated in HDL.

Other issues appear when modifications have to be carried out to the system. For example, if a new functionality is added to the C++ model then the electronics designer has to spend time propagating this modification into the HDL code. Sometimes it can be done directly, in most cases, it is necessary to iterate between the behavioral and the RTL models, which is time consuming. In the worst case, the modification is made directly on the HDL code, forgetting the C++ model, which stops being up-to-date and as a consequence becomes obsolete.

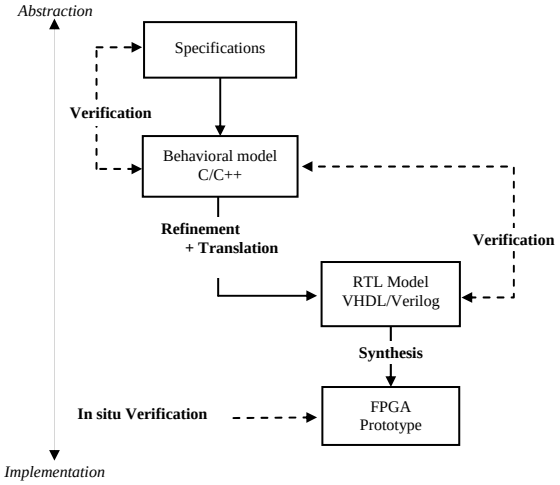

figure 1 : Traditional design flow

In the next section, it will be shown how the issues relative to this kind of design flow can be avoided using SystemC.

Collaboration protocol

A collaboration protocol was written in order to distribute the roles of this project. The role of LLR was to write the behavioral model and the test bench in SystemC; this model was then used as a golden reference. Then, the role of PROSILOG was to refine the SystemC model and create an RTL model in the same language. With the delivered test bench, a verification step was necessary. Then, the RTL SystemC had to be translated using the PROSILOG's SystemC to HDL compiler, which generates the equivalent RTL VHDL or Verilog model. The verification of this model was done using the same SystemC test bench and a final behavioral comparison with the RTL SystemC model was performed.

Figure 2 below describes the SystemC flow used in this project for the design, the refinement and the verification:

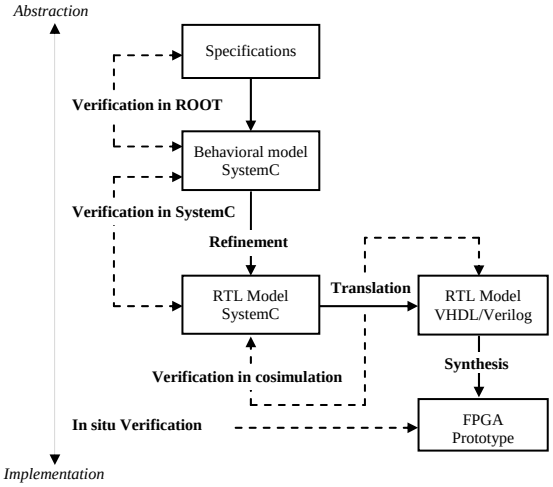

figure 2 : SystemC based design flow

---

[4] High level and HDL modeling description language from Open SystemC Initiative (OSCI) : http://www.systemc.org/

[5] Institut National de Physique Nucléaire et de Physique des Particules (France)

[6] European Organization for Nuclear Research located in Geneva, Switzerland

[7] Compact Muon Solenoid physic detector project from CERN collaboration: http://cmsinfo.cern.ch/Welcome.html/

[8] ROOT, an object oriented data analysis framework for physic experiment: http://root.cern.ch/**.** It is particularly useful to graphically analyze physics events simulation results and make decision for behavioral model description.

The first advantage of this methodology is that the design process language remains the same from behavioral to RTL level; thus, the delicate task of refinement becomes easier and consequently the errors due to language translation can be avoided.

Another advantage is brought by the SystemC to HDL translator offering the possibility to automatically translate the RTL synthesizable SystemC code into VHDL or Verilog language. Thus, the SystemC code remains the only one that must be maintained. After modification of the behavioral model, the electronics designer must only check if the added code is synthesizable or not. If it's true, the new code is naturally added to the RTL model, otherwise, before translation, the modification must be manually performed.

For the verification steps, the present design flow allows to reuse the same test bench for the behavioral model, the SystemC RTL model and the translated HDL model (using SystemC-VHDL/Verilog cosimulation). Even if the behavioral model and consecutively the test bench are described with unsynthesizable data types, it is always much more efficient to use an abstraction wrapper in order to adapt the test bench for the RTL models, in SystemC or HDL, than to translate it manually. Using the same test bench at each level of abstraction allows to obtain more accurate verification level.

TPG Strip behavioral SystemC model

The system under study is located on a front end board integrated on the LHC/CMS/ECAL detector of CMS. The model provided by LLR is a component of the so called Trigger Primitive Generation (TPG) system, integrated on an ASIC device, and used as a trigger vector provider. These vectors are sent to the central trigger logic selecting the event to be recorded.
The inputs of the system are variable amplitude pulses, digitally coded with 12 bits of mantissa plus 2 bits of gain scale[9], coming from each detector channel and sampled at 40MHz frequency.

TPG[10] Strip architecture

A diagram of the internal architecture of the TPG Strip[11] is shown in figure 3

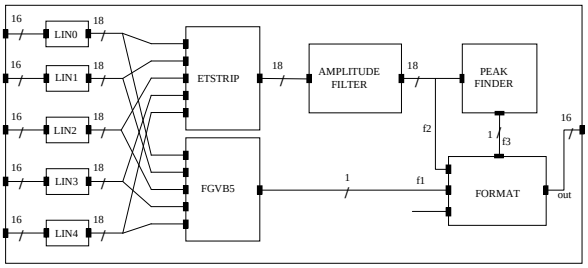

figure 3 : TPG Strip internal architecture

The five inputs are decoded using an integer representation by the five LIN blocks, then added by ETSTRIP adder. The output of the adder is then digitally filtered through an AMPLITUDE FILTER module, in order to get the amplitude.PEAK FINDER block detects the maximum of the pulse (output is set to one if the current sample reach the maximum); in parallel, the FGVB5 generates a veto bit if more than n inputs are greater than a threshold (this value and n programmable). Finally, the FORMAT module combines all these signals in order to generate the trigger vector output.

The TPG Strip was written in SystemC by a LLR physicist without any hardware implementation consideration[12]. This behavioral model was then sent to PROSILOG. The code could not be synthesized as is, and had to be adapted in order to produce a synthesizable RTL model.

Modeling style for the behavioral model

The TPG Strip block can be seen as a hierarchical module, what induce that all the sub-blocks presented before are modeled as separated SystemC module. A macro-module considered as the top-level of the TPG Strip model, instantiates and connects sub-modules to each other and to its ports through SystemC signals.

As the TPG Strip manipulates positive integer data with variable length, the data types used in the behavioral model were *sc_uint<n>* or *sc_int<n>*. These SystemC types are template so it is possible to set the number of bits that codes the integer value (up to 64), and use specific methods as bit part selection, operators, informational, settings or conversion methods. At all description levels, it was important to use this kind of predefined data types, because the specification of behavior integrated bits calculation notions. Others data types like *bool*, *unsigned short* or *unsigned int* were also used in the design.

The type of processes used for each block was *SC_METHOD*. This type of process can be sensitive on one or more input ports and executed on each signal value change inside the sensitivity list.

The only block connected to the clock is the FORMAT module. The others, are combinatory and only sensitive to their inputs.

The AMPLITUDE FILTER, the PEAK FINDER and the FORMAT modules compute their output as a function of the current and the past inputs. Their inputs are stored using a C++ class such as *deque* or *queue*, which allow stacks manipulation (to push or pop a data for instance). Initialization of these stacks is done directly in the constructor[13] of the concerned modules.

Most of the modules in the TPG Strip use coefficients for output evaluation. In the behavioral model, these coefficients are stored in tabs and initialized using a function called by the constructor.

Refinement down to SystemC RTL

Getting the behavioral model and its associated test bench, the work of PROSILOG was to refine it down to the synthesizable RTL level. This level is defined by the SystemC synthesizable subset accepted as an entry by the PROSILOG's SystemC to HDL compiler and which is the industry's largest.

In order to start the refinement phase, we firstly launched the compiler on the behavioral model to know which of its code elements were not synthesizable and one modified them to obtain a whole synthesizable model. Thus, all missing elements could be identified and an alternative good description was reported to LLR in the final collaboration documentation.

The data types used in the behavioral model (*sc_uint<n>*, *sc_int<n>*, *bool*, *unsigned short*, *unsigned int*) are synthesizable and can be still used in the RTL model. Moreover, the usage of the *.range()* and shift >> operators is allowed for the translation in RTL.

The *SC_METHOD* process type is also synthesizable.

It has been seen that in the behavioral model, the initialization is performed for each block by using a function, in which the coefficients are set to their value, and called from the constructor's module. This kind of function call is accepted by the compiler.

TPG behavioral model is hierarchical, by this means, all sub-blocks are instantiated in a top-level macro-module. References of these sub-blocks are done using pointers. This kind of hierarchical architecture is allowed by the translator and can remain the same in the RTL model.

---

[9] Requested by the very large dynamic range of the amplitude excursion of input signal

[10] Trigger Primitive Generation

[11] More details are available at URL: http://polywww.in2p3.fr/work/busson/cms/ECALElectronics/FrontEnd/FENIX_STRIP.html

[12] Originally this model was written for simulation purpose

[13] Each SystemC module has an initialization fonction that is called a constructor in C++ terminology

It was shown that AMPLITUDE FILTER, PEAK FINDER and FORMAT modules use C++ proprietary classes for the stacks management (*deque* and *queue*). These stacks are used by modules in order to store in a pipe the N past inputs values and compute the output values following a defined algorithm (respectively filtering, peak detection and formatting). The use of these specific C++ classes is obviously not synthesizable and for each of these blocks, we had to introduce the description of a synthesizable pipe device. This was done as follow[14]:

- addition of clock reset ports to the module
- creation of a tab (*sample_[N]*) in order to store values in the pipe
- declaration of the *do_pipe*() method, clock and reset sensitive (respectively positive and negative)
- implementation of this method : for reset, initialization of *samples_ tab* values, as push the new input and shift the stored values
- access to the pipe values by other methods is not changed and done as reference of the same indexes

These modifications guarantee the same functional behavior for the internal blocs and for the TPG Macro-module, but they also introduce a difference in the temporal behavior.
Because the pipe model, using C++ classes for stack operation, doesn't take into account the one clock period delay between data apparition on the input port and storage in the pipe, a difference appears. We had to model this behavior in the RTL model and that's why a temporal delay was introduced.

This difference produces a big issue for the FORMAT RTL model. This module receives, as inputs, signals coming from FGVBG5 (f1), AMPLITUDE FILTER (f2) and PEAK FINDER (f3) (see figure 3). We have seen that the RTL modeling introduces a one clock period time-lag for the output generated by AMPLITUDE FILTER. This output is used as input by PEAK FINDER, which also introduces a one clock period interval, and then the output of this module will obviously present a two clock period delay. Let's see what this difference modifies in the design:

The FORMAT behavioral model confirms the following equation[15]:

- $out(kT) = F[f1(kT), f2(kT), f3(kT)] \quad ,k \in N$

If the same functional behavior is requested for the TPG RTL model, the FORMAT module must verify the following equation:

- $out(kT) = F[f1[\ (k-2)T\ ], f2[\ (k-1)T\ ], f3(kT)] \quad , k \in N$

By taking in account these delays in the FORMAT RTL module, the problem could be solved.

Thus the RTL TPG output produces its output with a two clock period delay compared to the behavioral model, this delay neither change the output samples values nor their sequence. This is what can be called an equivalent functional behavior, with a different functional behavior.

In order to avoid the modification of the FORMAT module design and respect the behavioral model as golden reference, one can introduce a one period clock shift registers, as shown in the figure 4. In this figure, the modules that introduce a T delay are marked with the T symbol.

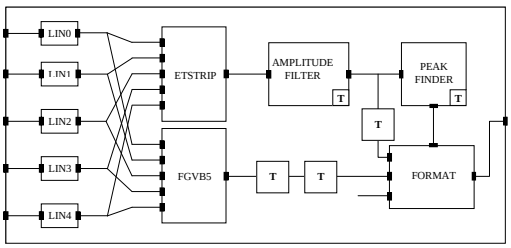

figure 4 : Validation test bench for the SystemC RTL model

[14] you can refer to the sources available in the appendix

[15] T=clock period

The advantage of such a method is that each module behavior is independent of its environment. Thus, if one considers that the FORMAT module generates its outputs taking into account synchronized inputs and moreover if one has to modify the TPG design by integrating for instance a module upstream the FORMAT module, finally one has only to insert the needed shift registers, in order to keep the system synchronism.

Validation method for the SystemC RTL model

When the synthesizable model is available, one has to verify that its functional behavior is the same as the one issued from the behavioral model. To do this, one creates a SystemC verification project which includes firstly the two models to be compare and secondly the originally delivered test bench. The comparison is performed by several verification modules connected to the both models. A diagram of this verification test bench is presented in figure 5:

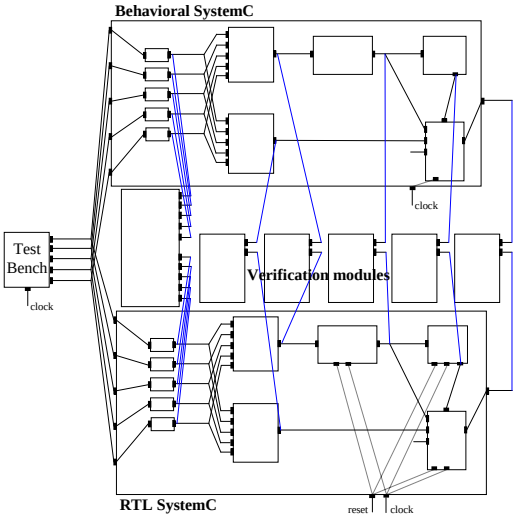

figure 5 : Validation test bench to compare two model's view

The LLR's test bench is used to generate the stimuli to be sent to the two SystemC models (behavioral and RTL).

Using this test bench, one is able to compare the outputs of several sub-modules from the RTL and the behavioral models. To achieve this goal, it was created the generic validation modules in SystemC that generate *.raw* format trace files directly visualized with the PROSILOG's waveform viewer. This view allows to quickly detect differences between signals and to find out the mistakes in the source code. These modules are template style, which allow to specify the type of data to be compared. With them, it's also possible to configure the number of inputs and enable a comparison mode where a computed difference signal is generated for each pair of signals to be compared.
As explained in the previous section, some of the sub-modules described in the RTL introduce a time gap on their output compared to the behavioral model. To compensate these delays, it was also added functionality to these validation modules by specifying a number of clock period delays. Doing so, errors could be easily detected.

Using these validation modules, and the ".vcd" trace generated by SystemC, one can also visualized it in the PROSILOG's viewer. Then it is possible to validate, block by block, the equivalence of the functional behaviors for the two models.

Translation into VHDL and Verilog

After the RTL SystemC model has been approved, it can be translate into VHDL or Verilog language. In this paper the model generated in VHDL is only considered, but all the considerations made for the VHDL could also be applied for the Verilog.

SystemC to HDL compiler

In order to translate automatically the SystemC to VHDL, one uses the PROSILOG's SC2HDL compiler. A SystemC synthesizable subset is defined, which describes the code accepted as input; the description of this subset is available on the PROSILOG's website[16].

The most important features of the synthesizable subset are the following:

- Hierarchical modules
- Template SystemC modules
- Conditional (*if* ; *switch*)
- Static type conversion
- Array (ports; signals; member variables; sub-modules)
- Loops (*for*; *while*)
- Member functions
- Enumeration
- Constructor functions

The tab of translation, used for SystemC to VHDL correspondence type, was the followings:

bool	→	*bit*
sc_int<N>	→	*signed(N downto 0)*
sc_uint<N>	→	*unsigned(N downto 0)*
unsigned short	→	*unsigned(15 downto 0)*

When the translation is launched, only the top level module of TPG has to be specified. The compiler analyzes which components and sub-blocks have to be translated and generates a VHDL file (.vhd) or a Verilog file (.v) for each SystemC description module (.h and .cpp files). As example, the different PEAK FINDER's model codes are available in the Appendix.

If a code part is refused by the translator, because it does not belong to the SystemC synthesizable subset, or if an error is detected in the source code, a well documented report appears, explaining the translation issue that occurred.

Validation method for the HDL RTL model

One of the requirements of the LLR was to keep the same test bench for each TPG model's view. In order to validate the automatically generated RTL VHDL model, we took advantage of the cosimulation possibilities provided by the PROSILOG's environment called *Nepsys*. Its cosimulation assistant helps the designer to quickly encapsulate a SystemC top-level module in a dll[17]. This dll is FLI[18] or PLI[19] protocol compatible. A cosimulation assistant generates also a translation of the module's interface to be translated, into an entity written in VHDL or Verilog.
So, the provided test-bench, the RTL SystemC TPG model, and the bank of validation modules has been encapsulated in the same "dll".

As shown in figure 6, a validation bench in VHDL has been realized, by instantiating the generated entity, corresponding to the cosimulation dll, and connecting it to the RTL VHDL TPG model, delivered from the automated translation. The clock and reset signal had to be driven from VHDL.

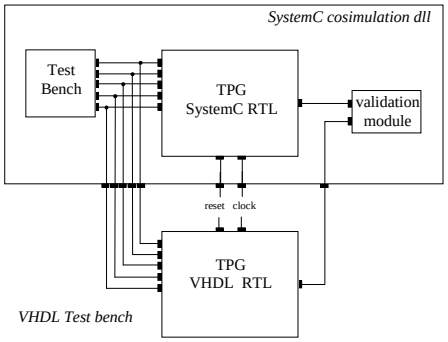

figure 6 : Validation test bench for HDL RTL model

Other signals have been added, that don't appear on the scheme, and which are used to compare outputs of each sub-module in SystemC and VHDL. This technique allowed us to use VHDL simulator's debug and trace tools, in addition of the ".vcd" trace files generated by SystemC, in order to verify behaviors equivalence of the both RTL models.

The results of this validation step were positives and it has been shown that both models present the same functional and temporal behavior.

The VHDL code could then be used for the prototype synthesis.

Conclusion

Success story

The LLR laboratory and PROSILOG collaboration was successful and allowed to show the possibility to use SystemC as a modeling language used for digital hardware systems implementation, from behavioral down to RTL level. It has been proven that the use of the same language at different abstraction level reduces refinement phases time. Moreover, this language could be used by no hardware specialists to rapidly define the behavioral of an electronic system. At this step, application of a minimum of design rules (see appendix) can also greatly reduce the overall time necessary to refine a pure behavioral model by an electronics designer. Thus, by using the same language, the communication between two different kinds of designer can be greatly facilitated.

It has been show that the proposed method reduces the design time and design errors by automating translation tasks. This task needs a SystemC to HDL compiler, in order to translate the system description in a language accepted by classical synthesizers.

This experience didn't target the hardware synthesis of the system, which could allow to verify the prototype behavior on an FPGA. It proposes nevertheless verification methods and tools in order to help designers to validate models at several abstraction levels, using the same test bench all along the design flow.

A recommendations list for SystemC coding style is available in the appendix. These recommendations can help designers team during model refinement in SystemC from behavioral to RTL level.

Acknowledgements TO LLR:

Special thank to Mr. Romanteau for his collaboration initiative and technical management, Pascal Paganini, Nicolas Regnault for their work on the SystemC code, Philippe Busson and Ludwik Dobrzynski for the management of the project.

Appendix

Recommendations list for model refinement in SystemC

- Pointers use is not recommended as reference to data exchange variables between modules. Pointers haven't any hardware equivalent.

- Avoid the use of proprietary C++ functions (stack management for instance). This is certainly very useful, for functional modeling, but they are not synthesizable in most of the cases and could require big design modification during refinement down to RTL.

- Try to integrate, as soon as functional level, the clock and reset ports, in all modules which may work in synchronous mode. If they are introduced later in the design flow or not taken into consideration in the algorithms, big modification in the module design will be necessary when moving down to RTL.

- Hardware implementation constraint must be taken in account when one create a model at the temporal behavioral level. This will avoid to take in account different temporal behavior during verification phases and allow to reuse the same test bench at all abstraction levels.

---

16 http://www.prosilog.com

17 Dynamic Linked Library used on PC platform

18 Foreign Language Interface, a procedural access protocol usable in VHDL simulator

19 Programming Language Interface, a C procedural interface usable in Verilog simulator

Behavioural model for PEAK FINDER module

// BEH_TPG_peak_finder.h

```
class BEH_TPG_peak_finder : public sc_module {

public:
	sc_in< sc_uint<18> >	in ;
	sc_out<bool>		out ;

	deque< sc_uint<18> >	samples_ ;

	void check() ;

SC_CTOR( BEH_TPG_peak_finder ) {
	SC_METHOD( check ) ;
		sensitive << in ;
		for (int i=0;i<3;i++)
			samples_.push_back(0) ;
	}
} ;
```

// BEH_TPG_peak_finder.cpp

```
void BEH_TPG_peak_finder::check() {

	bool peak = 0 ;
	samples_.pop_front() ;
	samples_.push_back(in) ;

	if( samples_[1] > samples_[0] &&  samples_[1] > samples_[2] )
		peak = 1 ;

	out = peak ;
}
```

SystemC RTL model for PEAK FINDER module

// RTL_TPG_peak_finder.h

```
#include <systemc.h>

class RTL_TPG_peak_finder : public sc_module {

public:
	sc_in< bool >		reset ;
	sc_in< bool >		clk ;
	sc_in< sc_uint<18> >	in ;
	sc_out<bool>		out ;

	sc_signal<sc_uint<18> > samples_ [3];

	void do_check() ;
	void do_pip() ;

SC_CTOR( RTL_TPG_peak_finder ) {
	SC_METHOD( do_pip ) ;
		sensitive_pos << clk ;
		sensitive_neg << reset ;

	SC_METHOD( do_check );
		for (int i = 0; i < 3; i++) {
			sensitive << samples_[i];
		}
	}
} ;
```

// RTL_TPG_peak_finder.cpp

```
#include "RTL_TPG_peak_finder.h"

void RTL_TPG_peak_finder::do_check()
{
	out = (samples_[1].read() > samples_[0].read()) &&
		(samples_[1].read() > samples_[2].read());
}

void RTL_TPG_peak_finder::do_pip()
{
	if (!reset.read()) {
		for (int i=0 ; i<3 ; i++) {
			samples_[i] = 0 ;
		}
	} else {
		for (int i=0 ; i<2 ; i++) {
			samples_[i] = samples_[i + 1] ;
		}
		samples_[2] = in;
	}
}
```

VHDL RTL model for PEAK FINDER module

--RTL_TPG_peak_finder.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity RTL_TPG_peak_finder is
	port(
		reset: in bit;
		clk: in bit;
		in_k: in unsigned(17 downto 0);
		out_k: out bit
	);
end RTL_TPG_peak_finder;

architecture rtl of RTL_TPG_peak_finder is

-------------------------------------------------------------------------
-- The functions below are used by VHDL code generated by the Compiler
-------------------------------------------------------------------------

function prosilog_conv_from_bool_to_bit(bool_value : boolean) return bit is
	variable res : bit;
	begin
		if (bool_value) then
			res := '1';
		else
			res := '0';
		end if;
		return res;
end prosilog_conv_from_bool_to_bit;

function prosilog_conv_from_bit_to_std_logic(bit_value : bit) return std_logic is
		variable res : std_logic;
	begin
		if (bit_value = '1') then
			res := '1';
		else
			res := '0';
		end if;
		return res;
end prosilog_conv_from_bit_to_std_logic;

function	prosilog_conv_from_std_logic_to_std_logic_vector(logic_value	:
std_logic) return std_logic_vector is
		variable res : std_logic_vector(0 downto 0);
	begin
		res(0) := logic_value;
		return res;
end prosilog_conv_from_std_logic_to_std_logic_vector;

	function prosilog_conv_from_std_logic_to_bit(logic_value : std_logic) return
bit is
		variable res : bit;
	begin
		if (logic_value = '1') then
			res := '1';
		else
			res := '0';
		end if;
		return res;
end prosilog_conv_from_std_logic_to_bit;

function prosilog_conv_from_lv_to_lowlv(lv_value : std_logic_vector; size : integer)
return std_logic_vector is
	begin
		return lv_value(size-1 downto 0);
end prosilog_conv_from_lv_to_lowlv;

function prosilog_conv_from_bv_to_lowbv(bv_value : bit_vector; size : integer)
return bit_vector is
	begin
		return bv_value(size-1 downto 0);
end prosilog_conv_from_bv_to_lowbv;

function prosilog_conv_lv_to_bv(A : std_logic_vector) return bit_vector is
		variable B : bit_vector(A'range);
	begin
		for j in A'range loop
			if (A(j) = '1') then
				B(j) := '1';
			else
				B(j) := '0';
			end if;
		end loop;
		return B;
end prosilog_conv_lv_to_bv;

function prosilog_conv_bv_to_lv(A : bit_vector) return std_logic_vector is
		variable B : std_logic_vector(A'range);
	begin
		for j in A'range loop
			if (A(j) = '1') then
				B(j) := '1';
			else
				B(j) := '0';
			end if;
		end loop;
		return B;
end prosilog_conv_bv_to_lv;
-------------------------------------------------------------------------
-- Module implementation
-------------------------------------------------------------------------
type samples_s_idx_9_mem_type is array( 0 to 2 ) of unsigned(17 downto 0);
signal samples_s_idx_9 : samples_s_idx_9_mem_type;

begin

do_pip : process(in_k, samples_s_idx_9, clk, reset)
	begin
		if (((not reset)) = '1') then
					samples_s_idx_9(0) <= "000000000000000000";
					samples_s_idx_9(1) <= "000000000000000000";
					samples_s_idx_9(2) <= "000000000000000000";
		else
			if (clk'event and clk = '1') then
					samples_s_idx_9(0) <= samples_s_idx_9(1);
```

```
                                samples_s_idx_9(1) <= samples_s_idx_9(2);
                        samples_s_idx_9(2) <= in_k;
                    end if;
                end if;
end process do_pip;

do_check : process(samples_s_idx_9)
        begin
        out_k       <=      (prosilog_conv_from_bool_to_bit((samples_s_idx_9(1)      >
        samples_s_idx_9(0)))                                                        and
        prosilog_conv_from_bool_to_bit((samples_s_idx_9(1)                          >
        samples_s_idx_9(2))));
end process do_check;
end rtl;
```

Verilog RTL model for PEAK FINDER module

//RTL_TPG_peak_finder.v

```
module RTL_TPG_peak_finder (reset,clk,in,out);

        input               reset;
        input               clk;
        input [17:0]        in;
        output              out;
        reg                 out;

        reg   [17:0]        samples__idx_9[0:2];

always @ ( posedge clk or negedge reset )
begin : do_pip
        if ((! reset))
        begin
                samples__idx_9[0] <= 18'b000000000000000000;
                samples__idx_9[1] <= 18'b000000000000000000;
                samples__idx_9[2] <= 18'b000000000000000000;
        end
        else
        begin
                samples__idx_9[0] <= samples__idx_9[1];
                samples__idx_9[1] <= samples__idx_9[2];
                samples__idx_9[2] <= in;
        end
end

always @ ( samples__idx_9 )
begin : do_check
                out     <=      ((samples__idx_9[1]     >       samples__idx_9[0])      &&
(samples__idx_9[1] > samples__idx_9[2]));
end
```