

Ever considered SystemC ?

Harald Devos, Hendrik Eeckhaut, Benjamin Schrauwen, Mark Christiaens, Dirk Stroobandt
Ghent University ELIS-PARIS
St.-Pietersnieuwstraat 41, 9000 Gent, Belgium
{hdevos,heekchau,bschrauwm,christi,dstr}@elis.ugent.be

Abstract—In recent years a lot of new C-based design languages have been developed. They all promise a smoother transition from a high level to a low level description of a hardware system.

A disadvantage of these new languages is that a lot of simulation models of e.g. FPGA-cores are only available in standard languages like VHDL or Verilog. This makes it hard to develop a complete system with one of those new languages.

However in a modular design approach this should not cause a problem. Here a distinction is made between application specific and hardware platform specific modules. It is possible to describe the application at a high level with e.g. SystemC and refine its modules with the same language, using simple models of the underlying hardware, until a level where translation to VHDL/Verilog is trivial. The platform specific modules can then be developed bottom up using the available VHDL/Verilog models of the used cores.

In this paper the design of an IDWT is taken as a test case. The design trajectory starts from a C-description and uses SystemC to arrive at a synthesizable level where an automatic translation to VHDL is done, which allows implementation on an FPGA.

One can conclude that although tool support is only emerging recently for these new C-based languages their benefits can already be exploited when using the right language at the right place.

Keywords— SystemC, inverse wavelet transformation, modular design, scalable wavelet video decoder, reconfigurable hardware.

I. INTRODUCTION

A lot of new C-based design languages have been created during the last years. They allow modelling at a higher level of abstraction and try this way to deal with the growing complexity of electronic systems (cfr. Systems On a Chip, SOC). Architecture exploration early in the design process becomes possible and can sometimes avoid unnecessary design iterations.

In the past, hardware (HW) and software (SW) were mostly developed separately and combined late in the design cycle. The results of a bad HW/SW partitioning showed up late, at a point where a new partitioning and design cycle has a high cost. HW/SW codesign becomes feasible with those C-based languages. The separation between what will be done in HW and what in SW can be

delayed. Also software developers do not have to wait for the full hardware specification to start working.

Using an old design methodology with a new language will not bring any benefit. Simulation at RT-level in VHDL will be as fast or faster than in SystemC as the tools are more optimized.

A new syntax can introduce new possibilities. The use of SystemC for developing a part of a system starting from a C program can benefit from the lack of a syntax gap and result in a smoother design trajectory. In a classical language a large translation step from an algorithm specification to a high level description in e.g. VHDL would be needed.

The large tool support is a big advantage of classical design languages like VHDL and Verilog. Therefore it is not likely that they will disappear. The more because the reuse of existing (IP-)blocks, which are currently mostly written in those languages, has become very important.

This paper describes how in a modular design the benefits of both classical design languages as VHDL or Verilog and modern C-based languages as SystemC can be combined. This was done even without cosimulating SystemC and VHDL as the tools were not mature at the time the work described in this paper was done. Cosimulation is only possible when there is a low level interface (signals) between the blocks of the different languages. Therefore it can only be used as a verification when the SystemC is refined to a lower level. It does not allow high level architecture exploration.

The implementation of an IDWT on an FPGA is used as a test case for using SystemC. The IDWT is part of a scalable video decoder. The other parts of the decoder are still only available in software. A video decoder partially running in software partially in hardware is the preliminary result. The partial conversion to hardware already speeds up the total decoder and gives indications for the future design stages.

Section II gives a short introduction to SystemC. The context of this work and the video decoder principles are described in III. The used design flow is explained in IV. The SystemC specific part of this design flow can be found in V.

II. THE BASICS OF SYSTEMC

SystemC is a C++ class library that provides constructs that allow to model system architectures including hardware timing, concurrency, and reactive behavior that are missing in standard C++. An event-driven simulation kernel is part of the class library. With nothing more than a regular C++ compiler an executable can be produced that simulates the described design.

At a low level (RT level) SystemC looks little more than a syntactic variant of languages like VHDL and Verilog. Fig. 1 shows an example taken from [2]. The words in capitals like SC_MODULE and SC_METHOD are macros that hide the real C++ syntax and provide a cleaner syntax. Almost every VHDL construct has a SystemC variant.

At a higher level SystemC offers something else. Signals in VHDL are less more than a representation of an electric wire. In SystemC signals can be much more. They can vary from fifo buffers to large bus structures with buffers and calculation units.

For more information the reader should be referred to e.g. [11], [3], [2], [12].

III. A SCALABLE VIDEOCODEC

A. The RESUME project

The work described in this paper is related to the RESUME project (Reconfigurable Embedded Systems for Use in scalable Multimedia Environment)[1].

The main research topic is how reconfigurable (scalable) hardware (FPGAs) can be used for scalable applications (i.e. with varying QoS, Quality of Service). FPGAs offer a great computational power with flexibility that is not provided by an ASIC. An implementation of a scalable video decoder on an FPGA platform will be made. Scalable video means that with one encoded bitstream a lot of different bitstreams for different QoS (resolution, framerate, color depth, image quality, ...) can be created only by dropping the unnecessary part in the encoded stream without extra decoding and encoding steps.

Such a scalable video decoder has advantages for both the server (providing video streams) and the clients. The server only needs to produce one video stream for all clients and the client can easily change the decoding parameters to optimize the use of the display, the required processing power, the required memory, etc... A scalable video decoder is a perfect fit for the use of FPGAs since it can be reconfigured each time the QoS requirements change, while still performing in real-time.

On a FPGA platform a modular basic design was developed (Fig. 3). It serves as a structure in which hardware modules can be plugged in and out and provides several

SystemC model of a flipflop.

```
// dff.h
#include "systemc.h"
SC_MODULE(dff) {
    sc_in<bool> din;
    sc_in<bool> clock;
    sc_out<bool> dout;

    void doit() {
        dout = din;
    };

    SC_CTOR(dff) {
        SC_METHOD(doit);
        sensitive_pos << clock;
    }
};
```

VHDL model of a flipflop.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
    port(clock : in std_logic;
         din : in std_logic;
         dout : out std_logic);
end dff;

architecture rtl of dff is
begin
    process
    begin
        wait until clock'event
            and clock = '1';
        dout <= din;
    end process;
end rtl;
```

Fig. 1. A simple example of SystemC and the corresponding VHDL.

communication facilities (IV-A). It will be used as the basic structure for implementing the video decoder.

III-B gives a short overview of the video decoder. The implementation of the IDWT is taken as a testcase for the C to SystemC to VHDL to FPGA design flow and as a test for the modular basic design and its communication with the software.

B. Scheme of the video decoder

A block scheme of the video decoder is shown in Fig. 2. It consists of the following parts:

P : De packetizer receives the incoming bitstream and

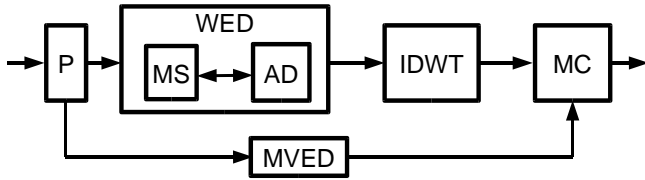


Fig. 2. Block scheme of the scalable wavelet based video decoder.

drops parts that are not necessary for the desired QoS.

WED : The Wavelet Entropy Decoder consists of a model selector and an arithmetic decoder that are closely coupled.
MS : The Model Selector selects a decompression model for the next bit based on some properties of the surrounding pixels.

AD : The Arithmetic Decoder decodes the incoming bit-stream bit by bit using the models generated by the model selector [7], [9], [8].

IDWT : The Inverse Discrete Wavelet Transformation produces reference and error frames.

MVED : The motion vector Entropy Decoder decodes the motion vectors.

MC : The Motion Compensation takes blocks of 1 or 2 reference frames and translates them as described by the motion vectors to generate an estimation of a frame. The final frame is produced by adding an error frame to this estimation. Translation over a fractional number of pixels is possible by interpolation, which can be calculation intensive.

A more detailed description of the video codec can be found in [5], [10] and [6].

IV. A MODULAR DESIGN FLOW

A. The modular basic design

Currently we use an Altera Stratix (EP1S25) on a PCI-card with 256 MiB memory DDR SDRAM. The card is placed into a standard PC. To make it possible to place different tasks on this board a basic design was made in which modules can easily be plugged in and out. The basic design takes care of sharing the different IO (Input and Output) facilities among the different tasks (modules) in the design (Fig. 3). Each module can work at its own clock rate as long as the communication interface (clock frequency, bus protocol, ...) with the basic design is respected.

B. The IDWT as a new IP-block

We want to develop a new IP-block that performs an IDWT and can be plugged into different FPGA-architectures not only the one described above. Therefore

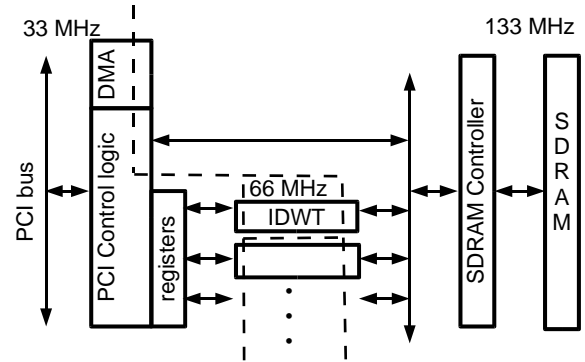


Fig. 3. The modular basic design with the IDWT plugged in. The dashed lines represent the separation of clock domains.

we split the design in a basic module which performs the IDWT and surrounding blocks which take care of the IO. The former can be platform independent while the latter will have to be adopted to the HW platform. The principle is shown in Fig. 4.

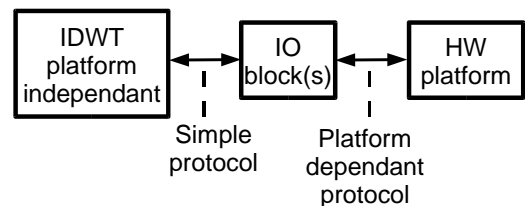


Fig. 4. Principle of separating calculation and communication interface.

It is not realistic to make a design totally platform independent. We make some constraints that most modern FPGAs will meet.

The FPGA should have:

- Embedded multipliers.
- RAM blocks to form little on chip memory with low latency (1 clock cycle)
- Dual-port RAM blocks which allow to buffer data and pass it from one clock domain to another.

The simple protocol between the IDWT and the IO blocks will consist of e.g. signals asking to load or store a line of a frame. The IO blocks will then initiate one or more bus transactions to transfer the needed data.

The modular basic design was written in VHDL. We want to use SystemC for the implementation of the IDWT. Somewhere the border between VHDL and SystemC has to be put. Our choice was to make 2 implementations of the IO blocks, one in each language.

C. The design traject

The design traject is illustrated in Fig.5.

The starting point of the design traject is a C function that executes an IDWT on an array. It is tested by a main function which reads a file into the array and dumps the result to another file.

First the C/C++ code is transformed into a SystemC design. In fact this is the real IDWT design process and forms the largest step. It consists of a lot of small steps described in V. The communication and calculation are seperated. The calculation part is refined to synthesizable RT level. The IO block only serves as a testbench and can use any usefull C++ construct as it does not need to be synthesized. One of the results of these manipulations is that a simple protocol between the IDWT and the IO is defined.

Afterwards this protocol is used as a specification for new IO modules written in VHDL and connected to the modular basic design. These modules were tested on the hardware by connecting the ports normally connected to the IDWT to a very simple testbench: a dual-port RAM block for the data path and registers, readable and writable from the PC, for the control signals.

Finally, the IDWT modules were translated to VHDL with the Synopsys CoCentric SystemC compiler, included in the modular design and given to the Stratix II tool to place it onto the FPGA.

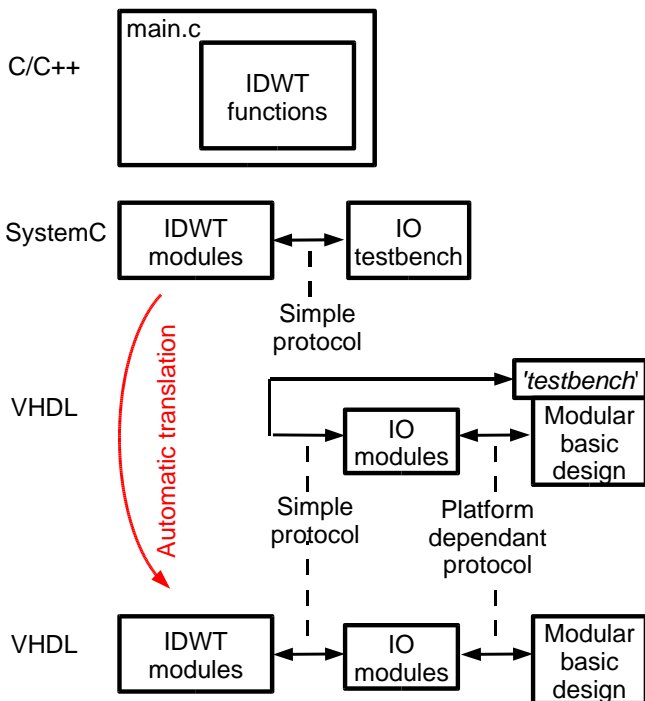


Fig. 5. The different stages in the design traject.

V. IMPLEMENTATION OF AN IDWT

This section describes the transition from C source code to a SystemC design with synthesizable modules and a testbench. During the whole design process always a totally working system was kept.

A. From floating to fixed point

The C program made use of floating point types. These are hard to implement in hardware. They were replaced by SystemC fixed point types. By parameterizing the wordlength (number of bits) and integer wordlength (number of bits before the binary point) of the several types an exploration of the needed accuracy of the variables was possible. It turned out that 18 bit, the standard word length for multipliers on FPGAs was sufficient for lossless decoding. 9 bit is the basic word length for on-chip RAMs in the FPGA. By (un)commenting one `#define` statement it was made easy to swap between floating and fixed point datatypes. The usage of fixed points slows down the compilation and execution time. Therefore fixed points were only used for the exploration of the accuracy and floats for other steps.

B. Optimization and profilation

Redundant calculations were removed. e.g. multiplications by zero as a result of upsampling of the input were removed by splitting the even and odd samples. Exploiting the symmetry of the wavelet filter coefficients also reduced the number of calculations.

Dynamic memory allocation was changed to static memory usage. This was no big problem because the memory usage is deterministic in function of the frame size.

All these optimizations were on the algorithm and not hardware or software specific. It reduces both the software execution time and the hardware requirements.

After these steps a profilation was made. The number of calculations per frame appears to be given by Eq. 1. R and C are the height and width of a frame. A and B are constants that depend on the kind of operation and the filter length. They are shown in Table I for a 9/7 biorthogonal wavelet filter pair. K is the number of transformation levels.

$$\frac{\#oper.}{frame} = ARC \frac{1 - (\frac{1}{4})^K}{\frac{3}{4}} + B(R + C) \frac{1 - (\frac{1}{2})^K}{\frac{1}{2}} \quad (1)$$

The number of calculations per second is calculated by multiplication with the framerate, i.e. the number of frames per second, and a factor 3/2 as a result of the YUV

TABLE I
PARAMETERS FOR EQ. 1.

Mul		Add	
A	B	A	B
9	82	16	98

4:2:0 format (Eq. 2). The pixel intensity (Y) is given at full resolution while the color information (U and V) is only available at a quarter of the resolution ($1 + 1/4 + 1/4 = 3/2$).

$$\frac{\#oper.}{s} = \frac{3 \#oper.}{2 frame} \times framerate \quad (2)$$

For a framerate of 30 Hz and CIF format (288×352 pixels) we get $60 \cdot 10^6$ multiplications and $103 \cdot 10^6$ additions per second. At a modest clock frequency of 50 MHz this would mean an average rate of 1.2 multiplications and 2.1 additions per clock cycle. It is clear that the calculation cost will not be the limiting factor, certainly not for an FPGA with embedded multipliers.

The IO bandwidth might become a bigger problem. It is impossible to store one frame totally in the internal FPGA memory. It has to be stored in the external memory, that is shared by different modules. One of the goals of this design was a fast exploration of the C to SystemC to VHDL design traject. Therefore a simple IDWT algorithm, in this case a row-column based one, was taken. The manipulation of columns prevents the use of burst memory transactions. If needed a line-based implementation with a much more friendly memory access pattern and lower IO bandwidth will be implemented [4].

C. Going to HW

In a VHDL based design flow there would now be a large translation step from the C-code to high-level VHDL, including the writing of a new test bench. This step was not needed here because SystemC is totally based on C++ and uses its syntax.

From here the SystemC and VHDL design methodology are quite similar. It consists of partitioning and refinement steps. The targeted structure was introduced by first separating the functionality in functions, then turning them into concurrent processes (SC_METHOD or SC_THREAD in SytemC) and finally modules. Starting from untimed code, passing through event triggered, finally synchronous processes were created. During these transformations the refinement level could be totally different in different parts of the design. The calculation was separated form the input and output. Dual-port RAM blocks needed for buffering data serve as borders between the IDWT and its IO and

between the IDWT clock domain and the IO clock domain. Fig. 6 shows the final structure.

A nice property of SystemC is that (in intermediate steps or for non-synthesizable parts) not only signals can be used for communication (as in VHDL) but also more abstract objects, class instances, pointers and references, ...

Finally the fixed points were replaced with integers inserting shifts where needed. The accuracy of the numbers remained parameterized.

The IDWT modules were refined to the synthesizable subset of SystemC. The IO modules were not and could use the full C++/SystemC functionality e.g. file IO, which is much nicer in C than in VHDL.

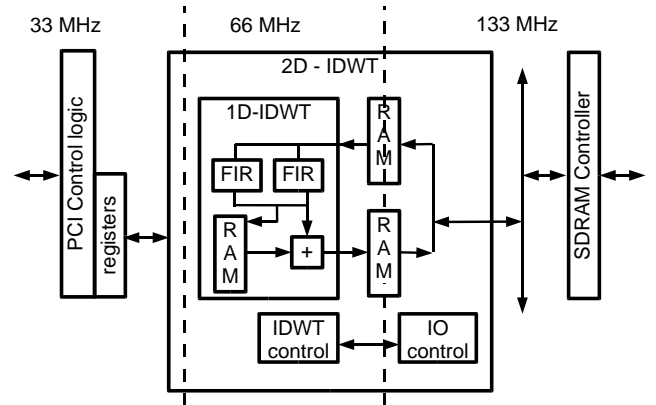


Fig. 6. A closer look at the IDWT in the modular basic design

D. Translation to VHDL

The VHDL code produced by the CoCentric SystemC compiler was not totally functionally equivalent with the SystemC code. Some variables were initialized to 0 by the C++ compiler and to U (unknown value) by the VHDL simulator. Some small adaptations of the SystemC code removed these and other problems.

When writing SystemC code that has to be translated into VHDL it is worth first examining the behavior of the translation tool.

VI. RESULTS AND FUTURE WORK

The previous sections described the hardware implementation of the IDWT. The other parts of the decoder were only available in software. With the use of some hardware drivers HW and SW could be glued together. Data is transferred between the PC and the FPGA board by use of DMA. A working video decoder was the result. Because only the IDWT was implemented in hardware real-time decoding is not achieved yet.

The major result until now is the qualification of the modular design flow, and the use of SystemC. It will be used to transform the other decoder blocks with real-time decoding as target.

VII. CONCLUSIONS

In recent years new C-based design languages have emerged. They try to raise the abstraction level that is used for digital design.

This paper described how an IDWT was implemented on a FPGA as a test-case for the use of SystemC and a modular design flow in which VHDL and SystemC can both show their benefits.

The real high-level constructs of SystemC were not used yet, but even without those the use of SystemC can reduce the design cost when starting from a description in the same syntax.

The tool support for SystemC is increasing. Therefore it is likely that its disadvantages will become smaller and smaller.

VIII. ACKNOWLEDGEMENTS

This research is supported by I.W.T. grant 020174, F.W.O. grant G.0021.03 and by GOA project 12.51B.02 of Ghent University. Harald Devos is supported by the fund for scientific research Flanders (F.W.O.).

REFERENCES

- [1] The RESUME project: Reconfigurable Embedded Systems for Use in Scalable Multimedia Environments. <http://www.elis.UGent.be/resume>.
- [2] SystemC Version 2.0 User's Guide Update for SystemC 2.0.1, 2002. Available from the Open SystemC Initiative (OSCI) <http://www.systemc.org>.
- [3] SystemC 2.0.1 Language Reference Manual, 2003. Available from the Open SystemC Initiative (OSCI) <http://www.systemc.org>.
- [4] C. Chrysafis and A. Ortega. Line-based, reduced memory, wavelet image compression. *IEEE Transactions on Image Processing*, 9(3):378–389, March 2000.
- [5] H. Devos, H. Eeckhaut, M. Christiaens, F. Verdicchio, D. Stroobandt, and P. Schelkens. Performance requirements for reconfigurable hardware for a scalable wavelet video decoder. In *CD-ROM Proceedings of the ProRISC / IEEE Benelux Workshop on Circuits, Systems and Signal Processing*. STW, Utrecht, November 2003.
- [6] H. Eeckhaut, M. Christiaens, H. Devos, B. Schrauwen, and D. Stroobandt. Scalable and hardware-friendly wavelet entropy coding. In *CD-ROM Proceedings of the ProRISC / IEEE Benelux Workshop on Circuits, Systems and Signal Processing*. STW, Utrecht, November 2004.
- [7] D. Marpe, H. Schwarz, G. Blättermann, G. Heising, and T. Wiegand. Context-based adaptive binary arithmetic coding in JVT/H.26L. *Proc. IEEE International Conference on Image Processing (ICIP'02)*, 2:513–516, September 2002.
- [8] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):620–636, July 2003.
- [9] D. Marpe and T. Wiegand. A highly efficient multiplication-free binary arithmetic coder and its application in video coding. In *Proc. IEEE International Conference on Image Processing (ICIP'03)*, pages 263–266, Barcelona, Spain, October 2003.
- [10] D. Stroobandt, H. Eeckhaut, H. Devos, M. Christiaens, F. Verdicchio, and P. Schelkens. Reconfigurable hardware for a scalable wavelet video decoder and its performance requirements. *Computer Systems: Architectures, Modeling, and Simulation*, 3133:203–212, July 2004.
- [11] The Open SystemC Initiative (OSCI). <http://www.systemc.org>.
- [12] G. Thorsten, L. Stan, M. Grant, and S. Stuart. *System Design with SystemC*. Kluwer Academic Publishers, 2002.