Unified Model Generation for FPGA-based LHCb VeLo algorithms

Manfred Muecke[a], Tomasz Szumlak[b]

[a] CERN, Geneva, Switzerland
[b] University of Glasgow, Glasgow, Scotland
Manfred.Muecke@cern.ch

Abstract

We show an alternative design approach for signal processing algorithms implemented on FPGAs. Instead of writing VHDL code for implementation and maintaining a C-model for algorithm simulation, we derive both models from one common source, allowing generation of synthesizeable VHDL and cycle- and bit-accurate C-Code. We have tested our approach on the LHCb VELO pre-processing algorithms and report on experiences gained during the course of our work.

I. Introduction

Considerable digital signal processing takes place in the TELL1 front-end electronics board [1], receiving data from the LHCb Vertex Locator (VELO) subdetector. All digital signal processing on the TELL1 has been implemented on FPGAs which allows for updates of the algorithms even after installation. This situation enables continuous algorithm improvement and therefore tuning of the overall detector performance. Evaluation of algorithms requires C-models to be integrated into the LHCb VELO simulation suite VETRA. Subsequently, matching VHDL-code needs to be developed and tuned for minimal hardware resource usage. As this typically requires modifications to at least the accuracy of the originally evaluated algorithm, one would in principal need to propagate the modifications back to the C-model and iterate again on the evaluation. Experience shows that this is unfeasible with limited manpower. Consequently, small design teams do either not iterate on algorithms, therefore limiting the theoretical performance of the detector, or C- and VHDL-models are not synchronized, making evaluation incomplete and bug tracking very cumbersome and time-consuming.

To improve upon this situation, we have suggested a unified model generation which derives needed models for simulation and implementation from one common source [2]. Parts of the FPGA-based algorithms for the VELO have been developed using the suggested design flow. We will report on experience gained during development and deployment of the derived models.

II. Confluence

Confluence [3] is a functional hardware description language (HDL) for design of digital synchronous systems. Although systems are described at register-transfer level (RTL), descriptions are more generic and less verbose than VHDL. The key reasons for this being:

- focus on synchronous systems, allowing implicit declaration of clock-, reset-, and enable-signals.
- type inference, allowing automatic type propagation through design hierarchies.
- recursion and list type, allowing for a very generic circuit description.

The Confluence compiler generates an independent netlist description (FNF) from which cycle- and bit-accurate C-models, as well as synthesizeable VHDL- and Verilog-models can be derived.

III. C-models

The C-models generated from the Confluence source code are cycle- and bit-accurate with regard to the generated VHDL-models.

The generated C-code defines a data structure containing all of the circuit's observable signals and memory elements. Each signal (including clock-, reset- and enable-signals) or bus is mapped into the least significant bits of an array of `unsigned long`. Functions acting on this data structure are `new_simulator`, `delete_simulator`, `init_simulator` and `calc_simulator`. `calc_simulator` takes the values of all input-signals and current values stored in registers and memories and propagates them through the circuit's combinatorial logic. Cycling the clock input and invoking `calc_simulator`, thus updating the respective values of all registers and memories, achieves a basic simulation cycle.

The provided cycle-accuracy is not necessary for physics simulation, but can complement advanced hardware debugging. Bit-accuracy is crucial to achieve results identical to the ones delivered by final hardware implementation. Physics simulation provides extended statistical routines to obtain overall detector performance parameters. With the integration of bit-accurate models of algorithm variations, it becomes possible to establish a convenient link between hardware design exploration and overall physics impact.

A. *integration*

Digital synchronous hardware is inherently clock-driven and so is the generated C-model. To match existing VETRA interfaces, it is necessary to encapsulate low-level clock-driven functionality such that it can cope with sets of data typically passed on as function parameters in C. This encapsulation is done manually and should be taken care of by the original hardware designer.

The VETRA coding guidelines enforce certain C++ code structures, which are not fulfilled by the automatically generated C-code. This currently requires some manual intervention.

In order to allow for appropriate data probing in between different DSP modules, it is mandatory to keep data structures used in simulation close to their real counterparts implemented in hardware. While this might seem obvious, it is not. Hardware designs benefit from processing data in parallel. Data structures therefore have to allow for easy extraction of the respective data being processed by one processing unit. The same is true for merging of results, received from different processing units. At the same time, they should be simple enough to serve programmers without intimate knowledge of the inner structure of the hardware design.

B. *simulation speed*

When comparing execution speed of generated and hand-written C-code, an overhead can be expected due to the cycle-accuracy, added control-signals and expensive signal propagation. For some modules, execution speed differs up to a factor of 24. Measurements have shown that execution speed of the signal processing algorithms is still only a small fraction of the overall simulation time, though. Modifications to the code generation process are likely to reduce execution time considerably, but have not been investigated. The execution times shown in B. were measured on an AMD Athlon 64 X2 running SLC3 Linux.

Module name	exec. time
VeloPedestalSubtractor	$800ms$
FastVeloPedSubtractor	$30ms$
FIRFilter	$10ms$
VeloBitLimit	$10ms$
VeloChannelReorder	$190ms$
VeloClusterMaker	$< 1ms$

Table 1: Various VELO DSP module's simulation times

IV. VHDL MODELS

The generated VHDL models are fully synthesizeable and can be integrated into existing VHDL designs. Confluence does not keep the original signal names, therefore generated VHDL-code should be considered a black-box in existing designs.

Most VHDL synthesizers can infer RAM from VHDL models, as long as they comply to specific coding requirements. These requirements are however tool-dependent. Consequently Confluence struggles to provide universally synthesizeable VHDL memory descriptions. We have chosen to work around this problem by excluding memories from the common models.

A. *integration*

Due to the lack of memory-modeling, the generated models are quite fine-grained and therefore need to be integrated on a very low level. This makes integration into existing designs a laborious work.

V. CONCLUSION

We have shown that code, generated automatically from Confluence source code, can be integrated into VETRA and into existing VHDL designs. While the design-flow is promising considering future improvements on the algorithms and the improved quality of simulation data, it also became clear that more work is needed to simplify integration of generated models.

It has been observed that definition of interfaces for hardware and software takes very different routes even when encapsulating identical functionality. This seems to be due to the very different background present in hardware and software engineering. We believe that adopting the unified modeling approach suggested, could also improve mutual understanding and therefore overall system design quality.

VI. FUTURE WORK

Many of the Confluence-related problems encountered during our work have stimulated input to discussions on possible language improvements. Out of these discussions, a new design language implementation evolved. It is called HDCaml [4] and has laid the ground to overcome many of the problems described here. It especially eases direct access to different code generators. So far, no VELO algorithms have been implemented in HDCaml, though.

The mismatch between C-models and C++ coding guidelines poses an obstacle to fast integration. Consequently generated models should move to full C++ and Gaudi [5] compliance. The generated C-code should be analyzed for possible execution speed improvements and the code generation process should be modified accordingly. HDCaml shows a promising way to explore possible solutions to these problems, but no results can be reported so far.

The intended fast iteration on more advanced DSP algorithms will only become efficient if the original description allows for enough abstraction to modify algorithms in a convenient way. We are currently extending HDCaml by an abstract fixed-point data type [6].

REFERENCES

[1] Guido Haefeli. Contribution to the development of the acquisition electronics for the lhcb experiment. *available online at: http://cdsweb.cern.ch/record/800810*, 2004.

[2] Manfred Muecke. C/vhdl codesign for lhcb velo zero suppression algorithms. In *Proceedings of the 14th IEEE-NPSS Real Time Conference*, pages 156–157, 2005.

[3] Tom Hawkins. Confluence project. *[Online] http://www.confluent.org/wiki/doku.php?id=confluence*, 2005.

[4] Tom Hawkins. Hdcaml project. *[Online] http://www.confluent.org*, 2006.

[5] CERN. Gaudi project. *[Online]http://proj-gaudi.web.cern.ch/proj-gaudi/*, 2006.

[6] Manfred Muecke. A bitwidth-aware extension to the hdcaml hardware description language. In *Proceedings of the Forum on Specification and Design Languages*, 2006.