IEC NWIP for DTR on FinSimMath as model for HDLMath

1) Introduction

On October 11, 2013, the US National Committee for IEC TC91 WG 13 submitted a New Work Item Proposal (NWIP) for a Technical Report on HDLMath, its requirements and how these requirements are met by FinSimMath.

HDLMath is intended as a standard that supports the implementation of mathematical algorithms into ASICs or FPGAs. The standardization process is currently planned to begin within the IEEE. Subsequently, HDLMath is planned to become a dual logo IEEE/IEC Standard.

The IEC TC91 WG 13 is seeking participants with background in mathematics, engineering, electronic design, hardware description languages, compiler technology, or in standardization work.

The work to be performed by participants consists of reviewing the requirements of HDLMath and how these requirements can be met by language constructs. Qualified individuals will receive free one-year licenses of FinSim with support for FinSimMath. FinSimMath is currently the only language that supports the all the existing requirements of HDLMath.

Individuals who wish to participate in the standardization work are invited to e-mail Dr. Alec Stanculescu their name, a short description of their interest in the implementation of mathematical algorithms into ASICs and FPGAs and their contact information to alec@fintronic.com and alec.stanculescu@iectc91dawg.org.

The official IEC voting procedure for the final TR consists of one vote per P member country. In order to check whether a given country is a full IEC member please search for “IEC National Committee MyCountry”, where MyCountry is the country of interest. In order to find whether a given country is a P member of TC91 please search for “IEC TC91 P members”. If you want to get involved with the official IEC voting please contact your IEC National Committee.

2) Problem being addressed

There is pent-up demand for high-speed, low power computational processing in many fields: UAV’s, miniature drones, automobiles, medical electronics, mobile applications, oil and gas exploration, just to name a few. Many of the mathematical algorithms involved are very mature, having been implemented as software running on general purpose processors.

Professor Mark Horowitz, from Stanford University, discussed the pent-up demand for ASICs at the CANDE workshop in November 2010. He concluded that lack of tools is the cause for the inability to create the needed ASICs. Specifically, there is a lot of pent-up demand for implementing mathematical algorithms in ASICs, which are smaller, faster and use less power than general purpose processors. Even FPGA’s are more efficient than general purpose processors, but lack of tools is also felt in large FPGA designs.

In general, the design must begin at the highest level of abstraction and must continue all the way to the lowest levels of abstraction that describe the gate level implementation, including all the intermediary

levels of abstraction. The verification of the implementation must be performed at a low enough level to ensure that the particular description of the implementation can actually be implemented.

3) Solution

The solution consists of developing HDLMath, a language that supports both the mathematical level of abstraction where the design process starts and the gate-level of abstraction where the design process ends and the verification is performed.

The productivity is increased by more than 1000x if both the high level and the low levels of abstraction involved in the design process are supported by the same language and the same simulator. This productivity increase was obtained with the introduction of Verilog and VHDL, which supported both RTL and Gate level descriptions. Verilog and VHDL increased the productivity of designing circuits at the RTL level and verifying the implementation at the gate level.

The level describing the implementation (RTL and Gate) is currently supported by standard HDLs, such as Verilog, VHDL, and SystemC. Therefore, an extension of any of these languages for mathematical descriptions (to be used for design as well as for mixed assertions) would be a good language to consider for the task of designing and verifying mathematical algorithms implemented as ASICs or FPGAs. Verilog, VHDL and SystemC are looking more and more like each other. We chose Verilog because it still has better support for the implementation level of abstraction (SDF, timing, etc.).

4) Requirements an HDLMath-like language

An HDLMath-like language must have the following features:

a) Support both the design at the high mathematical level and the verification of the implementation at the RTL/Gate levels. Note that the high level support must be bit-accurate, i.e. the result should match the result produced by the actual hardware and not a result obtained by assuming that infinite resources are available.

b) The formats (floating or fixed point) of high level data containers, as well as the number of bits of their respective fields must be modifiable during the execution of the simulation.

c) Support the following data containers:

i) All data containers supported by the HDL

ii) Data containers with modifiable formats, size of fields, and rounding options, known also as variable precision data containers.

iii) Scalar, Cartesian and Polar data containers.

iv) One and two dimensional arrays of any data container supported.

d) Support arithmetic operations of any combination of data containers, matrices or sparse matrices thereof.

e) Support exception handling (overflow, underflow).

f) Support tracking of cumulative errors.

g) Support tracking of peak number of bits used.

h) Support for populating one and two dimensional arrays.

i) Support for sparse one and two dimensional arrays.

j) Support for printing multiple values stored in one and two dimensional arrays.

k) Support for displaying graphically values stored in one and two dimensional arrays.

l) Support bit level interfaces of modules containing high level mathematical descriptions. As a result one can easily replace high level descriptions with their low level implementation for fast simulations and get support from existing waveform viewers.

m) Support mixed numerical and symbolic computations. For an example, look at www.fintronic.com/eval_dif_fin_lap.html. During a simulation a symbolic expression is numerically evaluated base on the current values of its variables.

n) Support information necessary for synthesis, e.g.

1) resources available along with their cost, latency, geometrical parameters, number of bits, etc.

2) address in memory (including memory block) of given variables.

3) specification of variables to be implemented as registers or memory elements.

4) Specification of sampling rates for input data, clock rates for clocks supplied to various memory blocks, as well as to computational resources.

o) A large number of system functions, such as FFT, DFT, finding eigenvalues and eigenvectors, norms and distances, finding roots of polynomials, must be supported as well.

The standardization committee may revise this list of required features and perhaps modify some of them and add new ones. The standardization committee will review FInSimMath based on the list of agreed upon required features and define HDLMath.

5) Industry awareness of the need for unified design and verification.

How did it happen that, currently, the design of mathematical algorithms is done in one language and their implementation as ASICs or FPGAs is done in another language, leading to a very low productivity, therefore limiting the designs that can be attempted?

It is not because this problem was not noticed. In a panel at DVCon 2013 titled “Where design ends and Verification begins” all panelists, including Gary Smith, agreed that the wall between design and verification should be reduced in order to increase productivity. Proposals by panelists included bringing the design and verification teams (yes, in many places these are different teams!) in close proximity.

The question that begs to be asked is “What better decrease in the wall between design and verification than having both the design and the verification use the same language and the same simulator?” This approach would allow the design and verification teams to talk to each other in terms of the same language and would even allow the same team to perform both design and verification, which is seldom

the case now due to the difficulty of mastering both the language describing mathematics and the language describing the network of resources.

6) What has been the state of the art in the absence of FinSimMath?

a) Mathematical algorithms where designed in C/C++ or m-language (MatLab).

b) Bit accurate models were developed with substantial larger efforts than when using FinSimmath. In case of using C/C++/m-language, it was not practical to explore the format and size of individual data containers. Typically, all data containers used had the same format and size of fields. In FinSimMath, each data container can have its own format and its own field sizes and the modification of formats and sizes can be programmed to occur within one simulation.

c) Verilog, VHDL or directly FPGA bit streams were produced by synthesis tools or manually.

d) Debugging the implementation was often done by a different team from the one having designed the mathematical algorithms. Having one language for design and one for implementation maintains a huge wall between the two activities, thus reducing productivity.

Whereas designing in C/C++/SystemC may be a good solution in case the target architecture is a processor running code which is generated out of C/C++ code, such solutions are not appropriate for the case in which the target architecture is an ASIC or an FPGA, because in the latter the target language is Verilog or VHDL, and therefore the source language should be an extension of Verilog or VHDL, such as FinSimMath.

7) How to integrate FinSimMath in an existing Design Flow?

a) Mathematical algorithms can be developed in FinSimMath or converted to FinSimMath from C/C++/m-language. For a comparison of FinSimMath code and m-language code please look at www.fintronic.com/lsim1.html. It is easy to convert to FinSimMath C/C++ code that describes mathematical computations supported by Synthesis tools, since FinSimMath supports Verilog constructs such as function, task, for/while loops, case/if statements, fork-join statement, as well as high level mathematical descriptions.

Once the mathematical algorithms are described in bit-accurate form in FinSimMath and the implementation is described in Verilog, one can develop a test bench in Verilg/FinSimMath that can use mathematics in order to:

(i) create the appropriate stimulus,

(ii) process results (e.g. FFT, DFT, etc.) and display the resulting information,

(iii) compare the high level computations with their implementation counterparts using mixed level assertions, modify the implementation in small steps in order to achieve additional functionality, less cost, better performance.

(iv) debug the implementation using the test bench written in FinSimMath. It helps a lot that one can perform fast simulations, because most of the resources are modeled at the highest level (yet bit-accurate) and the interconnection of the resources is modeled at the Verilog bit-level being easy to view in any Waveform Viewer.

For an example of how FinSimMath is used in implementing a mathematical algorithm in ASIC or FPGA visit www.fintronic.com/Integrating_FinSimMath_in_an_Existing_Design_Flow.pdf. Other examples of using FinSimMath are available in links present in www.fintronic.com/finmath.html.