Mathematical Algorithm Hardware Description Languages for System Level Modeling

Ryo Hikawa, Ryuji Kishimoto, Takashi Kambe†
Graduate School of Science and Engineering, Kindai University
3-4-1 Kowakae, Higashi-Osaka City, Osaka, Japan
†Depart. of Electric and Electronic Engineering, Kindai University

Abstract— Mathematical modeling is an important approach for both solving problems and visualizing the abstract concepts involved in system and/or products. Thus a mathematical algorithm description language (HDLMath) should be capable of describing and verifying the entire behavior of electronic systems using mathematical algorithms.

In this paper, the functional requirements of HDLMaths are proposed and several current HDLMaths are compared from a design viewpoint.

I. Introduction

Around the world, engineers in industries such as electronics and automobiles are developing many kinds of systems and products. However, these are developed based on conventional design processes and suffer from many design problems and long design times. Because the laws of nature can be expressed mathematically, mathematics is a good algorithmic method for the description and modeling of such systems. Mathematical modeling is also an important approach for both solving problems and visualizing the abstract concepts involved.

A mathematical algorithm description language (HDLMath) should be able to describe and verify the entire behavior of systems and/or products using mathematical algorithms of electronic systems. It is a higher level language than conventional HDL such as VHDL and System Verilog. HDLMath and its design environment should support the design of many domains and applications such as digital signal processing, computer graphics, digital communications techniques, and so on.

Recently, HDLMath languages such as MATLAB/SIMULINK[1], FinSimMath[2], and System C-AMS [3] are already being used to design the mathematical algorithms in electronic systems. In this paper, we call them HDLMath1,2,and 3 respectively.

The paper describes nine functional requirements for an HDLMath and compares current HDLMath languages from a design viewpoint. It is hoped that this will contribute to accelerating the development and utilization of a common mathematical algorithm design language and to establish a good system modeling environment in the industry.

II. Definition and positioning of HDLMath

HDLMath is defined as a language for describing and verifying the behavior of an entire system or product using mathematical algorithms. When compared with the number of lines of code written using a HDLMath and the length of the C-code generated automatically from its HDLMath description, the number of lines of C-code is several hundred times larger than that of the HDLMath descriptions. This indicates how HDLMath languages can be used to design at a higher level of design abstraction and hence how design productivity is higher than C level design.

III. Functional requirements of HDLMath

When designing mathematical algorithms for system level modeling with an HDLMath, the HDLMath should cover the following functional requirements in order to achieve utmost precision.

A. *Mathematical expressions*

The mathematical operators (for example, $+$, $-$, $*$, $**$, and $/$) should be applicable to any combination of the following operand and result formats: arbitrary-precision fixed-point, arbitrary-precision floating-point, integer, real, register, and constants. Trigonometric and hyperbolic (direct and inverse) functions should also be supported for any precision and power, logarithm, and square root operations are also needed.

B. *Various kinds of precision computation*

Data should include scalar, complex numbers in Cartesian co-ordinates, and complex numbers in Polar coordinates. The high level support should be bit-accurate, i.e. the result of computations performed during simulation should match the results produced by the actual hardware. The formats (floating or fixed point) of high level data, as well as the number of bits in their respective fields should be modifiable during algorithm design. Modifying formats and their respective fields allows for a more efficient design space exploration.

C. *Exception and error handling*

To help optimize the implementation of mathematical algorithms, errors should be minimized using only the necessary number of bits. Access should also be provided to information regarding the occurrence of overflow, underflow, maximum number of bits required, and cumulative error.

D. *Multi-dimensional arrays*

One and two-dimensional arrays of any kind of data, including sparse arrays, and arithmetic and logical operations on any kind of legal combination of data should be supported. This capability allows the implementation of all mathematical algorithms.

E. *Mathematical functions*

There should be support for a large number of mathematical functions, such as differential equations, FFT, DFT, finding eigenvalues and eigenvectors, norms and distances, finding roots of polynomials. Although all mathematical functionality can be written using the arithmetic operators, such an implementation would be slow.

F. *Mixed numerical and symbolic computations*

All the necessary processing should be performed in one execution. Users should not be burdened with passing data from a symbolic environment to a numeric environment. When performing symbolic simulation using strings, a string should be evaluated in the current context, as if it were an expression in the numeric environment.

G. *Feedback process*

One important aspect in control system design is to analyze the effects of feedback loops on the overall system.

H. *User-defined functions in C-code*

Support for extending simulation functionality by having the capability to incorporate C code execution in the simulation in a standard manner is required to enhance performance. The rationale behind this requirement is that many design teams have their own mathematical libraries and nothing else can work as well for them. In such cases, the designers can use their own libraries.

I. *Verification environment*

Test benches are an essential tool in the circuit design environment. They provide a virtual environment used to verify the correctness of the design or model. The test bench capability of an HDLMath language should include four components: input, circuit, check functions, and output.

TABLE I
COMPARISON OF CURRENT HDLMATH LANGUAGES

Requirements	HDLMath1	HDLMath2	HDLMath3
Mathematical expressions	✓	✓	(∗)
Variable precision	(∗1)	✓	(∗)
Exception handling	(∗2)	✓	(∗)
Multi-dimensional arrays	(∗3)	✓	(∗)
Mixed computations		✓	(∗)
Mathematical functions	✓	✓	(∗)
Feedback process	✓	(∗)	(∗)
User-defined C-code(*)	✓	✓	✓
Verification environment	✓	✓	✓

(∗) Implemented using additional C coding.
(∗1) The computation should be preceded by digits(n).
(∗2) HDLMath1 does not support flag setting.
(∗3) Limited to small array sizes.

IV. COMPARISON OF CURRENT HDLMATH LANGUAGES

Table 1 shows a comparison between current HDLMath languages based on the functional requirements for an HDLMath. These languages support most of the functional requirements, but they still require further descriptive capabilities for larger scale designs. HDLmath1 has bit length limitations (maximum 64 bits) and several problems from a hardware design perspective such as limitations for functional requirements 2, 3, and 4. HDLmath2 requires more functionality such as the ability to handle feedback processes without additional C coding. HDLmath3 is able to describe mathematical algorithms, but it requires a lot of C coding and debugging.

V. CONCLUSION

In this paper we have described the nine functional requirements that an HDLMath language specification should cover in order to design mathematical algorithms for system level models precisely and concisely. Current HDLMath languages have certain of the functionalities required for mathematical algorithm design. However, they still require more extensive descriptive capabilities in order to handle larger scale design.

REFERENCES

[1] A. K. Tyagi : "MATLAB and Simulink for Engineers," Oxford University Press, (2012).

[2] http://www.fintronic.com/finmath.html

[3] Vachoux, A., et al. : "Extending SystemC to support mixed discrete-continuous system modeling and simulation," pp5166-5169, Vol.5, the proceeding of IEEE International Symposium on Circuits and Systems (2005).