International Journal of Engineering Researches and Management Studies

AUTOMATIC GENERATION OF TRIGONOMETRIC HARDWARE WITH HLS TOOLS USING THE CUBEDC HARDWARE COMPILER/OPTIMIZER

Michael Dossis*, Vasilios Hados, Georgios Dimitriou
*Depatment of Informatics Engineering, TEI of Western Macedonia, Kastoria, Greece
Depatment of Informatics Engineering, TEI of Western Macedonia, Kastoria, Greece
Department of Electrical and Computer Engineering, University of Thessaly,Volos, Greece

ABSTRACT

The complexity of contemporary integrated circuits (ICs) creates a productivity and project duration challenge to respective industries. In order to handle this complexity FPGA/ASIC/SoC design projects must employ fully automatic and formal methods for the design of their custom blocks. These blocks are found in numerical applications such as trigonometric functions. This paper discusses fully automatic, abstract and formal design and developent methods for complex trigonometric blocks which are parts of ICs and they accelerate their host computing system. Such custom subroutines are rapidly specified and verified in fully-standard, high-level ANSI-C code. Thus, productivity is increased by orders of magnitude and first-time-right and provably-correct implementations are rapidly and formally generated. Our design methodology is evaluated with a number of basic trigonometric functions but that they prove the argument of increased development productivity and easy to use, in the experimental section of this paper.

KEYWORDS: High-level Synthesis, Electronic DA, E-CAD, hardware arithmetic, formal methods, RTL HDL coding**.**

I. INTRODUCTION

Hardware arithmetic blocks that are found in contemporary integrated circuits (ICs) are very complex, and their design with conventional methods is long, tedious and prone to functional bugs. To deal with this high-level, abstract, and fully behavioural specifications and rapid and automated development and verification techniques for such complex circuit blocks are needed. As an example, hardware arithmetic units that utilize trigonometric and floating point (FP) operations are found in embedded telemetry systems, avionics, computer gaming/graphics, transport control systems, etc. In this paper, the benefits of using a formal and automated High-Level Synthesis (HLS) tool called the CubedC compiler, are discussed for the rapid development of these circuits.

Computing systems that include multiple floating-point operations per cycle have motivated processor designers to incorporate a number of floating-point units per chip. The PowerPC970-FX processor from IBM has two floating-point arithmetic units [1]. The Niagara 2 chip from Sun has one floating-point (FP) unit attached to every one of its eight embedded processor cores [2].

High-level Synthesis (HLS) tools transform high-level behavioural programs into optimized Register-Transfer Level (RTL) hardware implementations of the algorithms which are specified in the source program code. HLS borrows techniques from compiler technology, intermediate data/control-flow descriptions and their optimizations. HLS's optimizing transformations include allocation, binding and scheduling. The first author of this paper has developed the CubedC HLS toolset using formal compiler-compiler and logic programming transformations. These optimizing transformations are based on the above formal techniques in the CubedC tools, [21].

Our formal methodology enables rapid design, verification and implementation of complex numerical operations such as floating point and trigonometric algorithms. High-level, functional, source code verification, based on rapid program compilation and execution, is completed easily, in a fraction of time which is needed for traditional manual RTL coding and simulation. The automatic CubedC approach is orders of magnitude faster, correct and more productive than traditional RTL coding techniques. This is shown in the experimental sections of this work. At the moment, the ADA and ANSI-C programming languages are used to code the high-level specifications, although more language front-ends such as C++ and Java are under development. The complete set of the standard programming

IJERMS
International Journal of Engineering Researches and Management Studies

language constructs, syntactic and semantic programming language elements are accepted and processed by the CubedC tools. These include nested conditionals, complex expressions, hierarchical subroutines, most standard data and control types, all types of loops (e.g. for, while, etc), and complex data objects such as records and multi-dimensional arrays, and others.

Existing work and background in designing trigonometric hardware as well as High-level synthesis is discussed in section II. Section III outlines the author's CubedC synthesis framework. Section IV discusses the rapid functional verification flow of our methodology. Trigonometric numerical hardware implementation experiments, including the four basic trigonometric operations evaluate our tools in Section V. Finally, the last section draws useful conclusions and proposes future work of this fruitful research approach.

II. EXISTING WORK AND BACKGROUND

The development and verification of custom numerical hardware blocks such as floating-point and trigonometric functional units is a cumbersome, tedious, time-consuming and not bug-free task. Errors were discovered in 1994 in the floating-point division of Intel's Pentium processor [3]. An investigation led to the conclusion that some locations in the division function SRT-4 lookup table were zero, instead of being equal to the correct number 2. Consequently, the division algorithm was reading the value 0 instead of 2, and therefore it was producing wrong results. This error caused an unpredicted cost of 500 million dollars to Intel [4]. This constitutes a great motivation for high-level, abstract techniques to design and verify numerical blocks, such as the ones of this work.

Uncaught functional bugs in hardware implementations may generate more than just financial issues. A series of wrong floating-point roundings in the operation of the first Patriot anti-missile system, has cost the life of 28 soldiers. The missile's timer was counting in 0.1 seconds units. However, and since there is no precise representation of 0.1 in the floating-point system, approximations were used instead of the precise value [5]. Consecutive approximations and roundings in the system's timer created a deviation for about 0.34 seconds from the precise time. This made the Patriot rocket to miss the Scud missile for about 500 meters. As a result, the Scud missile hit a military camp, and killed a number of soldiers.

The Ariane 5 mission rocket's acceleration was stored in a floating-point variable. An overflow of this variable caused the explosion of the rocket, leading to the failure of the Ariane 5 space mission [6]. The most interesting part of this accident is that the same software was used in other missions, without the occurrence of this error. This prevented the engineers from tracing the FP mistake, and the resulting failure makes our functional high-level design and rapid verification methodology absolutely necessary. This is why the automated, rapid and high-level approach of the CubedC verification is so important to catch all the functional bugs, early in the design flow.

Better performance with the cost of increased area, is achieved with the Carry-Prediction Adder (CPA) in the core of the floating-point addition and other numerical hardware algorithms. Advanced fixed-point adder implementations such as Carry Lookahead [11], Brent-Kung [12], Kogge-Stone [13] and Sklansky [14] are heavily referenced in the bibliography. Efficient implementation of fixed-point adders and subtractors is found in [15]. In [16], a double-path addition/subtraction structure uses a far-path and a close-path to deliver optimized performance

The contemporary RTL synthesis tools provide libraries with optimized adders and adder trees that are suitable for the purposes of the particular application and the targeted technology. Our CubedC tools rapidly and automatically generate RTL VHDL/Verilog code which is compatible with all of the existing commercial and academic RTL synthesizers, thus the implementation concerns about the fixed-point adder architecture are left to the RTL synthesis experts that have done so much useful work already. Our numerical hardware implementations, which are automatically generated are independent from (and compatible to) any target technology or tool vendor.

IJERMS
International Journal of Engineering Researches and Management Studies

The CubedC framework [19], [20], [21], is used here to code in C, verify and synthesize the four basic trigonometric functions. Our high-level C programs, are cross-tested and cross-verified with the related output of the RTL verification simulations that are reported and discussed near the end of this paper.

The theoretical foundations for hardware trigonometry experiments are found in [22], [23], [24]. We use the Taylor series approximation of the sine, cosine, tangent and contagent functions and we code these algorithms in CubedC ANSI-C. A high-level functional testbench is built by our team in C to rapidly verify the trigonometric algorithms at the abstract and functional level or regular ANSI-C code. We cross-check the C testbench results with the RTL VHDL simulation results, to prove the argument of correctness, although due to the formality of the CubedC tools the setup and run of RTL simulations are not necessary. This saves considerable amount of verification effort in most design projects. The simulation results are correct as expected, as shown by all of the experimental RTL simulations that we run.

Algorithms and applications that demonstrate the usefulness and utility of hardware trigonometric computation units is found in [25], [26], [27]. Reference [25] explains how to design trigonometric calculation hardware algorithms for laser diode displacement sensor using the limited Taylor series as well as the CORDIC (for COordinate Rotation Digital Computer) method. [26] reports single precision hardware applications which are based on the evaluation of truncated Taylor series using the difference method. The authors of [26] call this approach the ATA (Add-Table lookup-Add) method. In [27] a nontrivial modification of the well-known CORDIC technique is discussed. This applies to computation additional to this paper's work, which we plan to pursue in the future.

III. THE CUBEDC DEVELOPMENT FRAMEWORK

The CubedC framework consists of front-end and back-end compilers. These two compilation phases constitute the synthesis flow and they exchange information with the Intermediate Tables Format (ITF) database [29]. The internal transformations of the back-end compiler, as well as the ITF files are implemented with logic programming predicates and Prolog facts [28]. These predicates implement the formal transformations of the CubedC synthesis tools and they are the underlying technology for our formal HLS approach. ITF includes a set of homogeneously grouped Prolog facts that capture the complete algorithmic, typing, control, structure and hierarchy information of the source ADA and/or ANSI-C specification code. The front-end compiler is based on compiler-compiler techniques and intelligent front-end parsers with XML validation facilities. It compiles the input ADA/C code into the ITF database. The back-end compiler, loads the ITF facts into its logic rules inference engine and it "concludes" to an equivalent (to the input ADA/C subroutines) number of RTL hardware coprocessor models (numerical hardware implementation modules). These automatically and rapidly generated coprocessor modules can be used to accelerate certain functions of their host computing systems, thus they act as the system's accelerators.

Our CubedC HLS synthesis flow is shown in Fig. 1. First the application is coded using standard programming constructs in either standard ADA or ANSI-C. The input program control-flow and subroutine hierarchy is maintained through the CubedC synthesis, unless the user decides to implement it in a flat structure in the source code. A number of target HDL (Hardware Description Language) hardware coprocessors, equivalent to the number of input code subprograms, is automatically generated by the formal optimizing synthesis of the CubedC system. Additionally, there is a matching correspondence of the generated code signal, variable and operator names with the names of corresponding data objects of the source programs, as well as of the data type names. This makes debugging and object tracing on the generated code very easy, with regard to its source code naming correspondence and functionality equivalence, thus high quality in the generated RTL HDL code is achieved. Moreover, the generated HDL code is highly readable and structured so the user that is familiar with HDL coding can investigate it and verify it without any difficulty.

IJERMS
International Journal of Engineering Researches and Management Studies

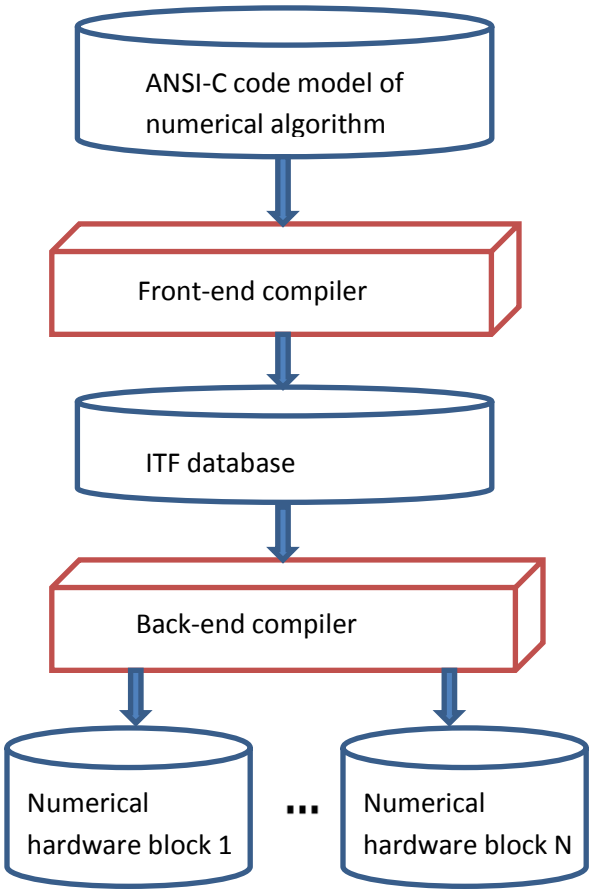

Figure 1. ***The CubedC framework design flow***

A near complete description of the ITF structure, syntax, statements and semantics can be found in [29], and an extensive analysis of the CubedC tools in [30]. The CubedC framework generates synthesizable RTL code in either VHDL or Verilog (chosen by the user via command line parameter) which is independent from any vendor or technology – specific templates and library instantiations. Therefore, our synthesis flow can be "plugged" easily, on any established, academic or industrial ASIC and FPGA tools flow. The generated HDL code files constitute complete and standalone FSM+datapath, or massively parallel hardware implementations [30]. There are many options to guide the synthesis process, such as (local or global) resource constraints, and the targeted microarchitecture template. One choice for the designer and user of the CubedC tools is that complex, sizable and multidimensional data objects such as large arrays can be located locally in the coprocessor or on external, shared memories. The preferred HDL language can be (at the moment) VHDL or Verilog. More detailed descriptions of the CubedC custom options can be read in [31] and [31]

All of the so far generated RTL modules were simulated and they matched the correct behavior of the input C code, as expected, due to the formal nature of our synthesis transformations. The source code lines are less at least by 10-times compared to the generated RTL code lines, thus one more way for increased productivity and ease to model and

IJERMS
International Journal of Engineering Researches and Management Studies

develop ability is given to the designers. The PARCS scheduler [30], included in the back-end compiler optimizes drastically the input operation schedule into a set of optimized parallel operation FSM states.

IV. THE CUBEDC VERIFICATION FLOW

The CubedC verification methodology is characterized by rapid, high-level program code compile-and-test approach. The configuration of the ADA or C code and testbench are shown in Fig. 2.

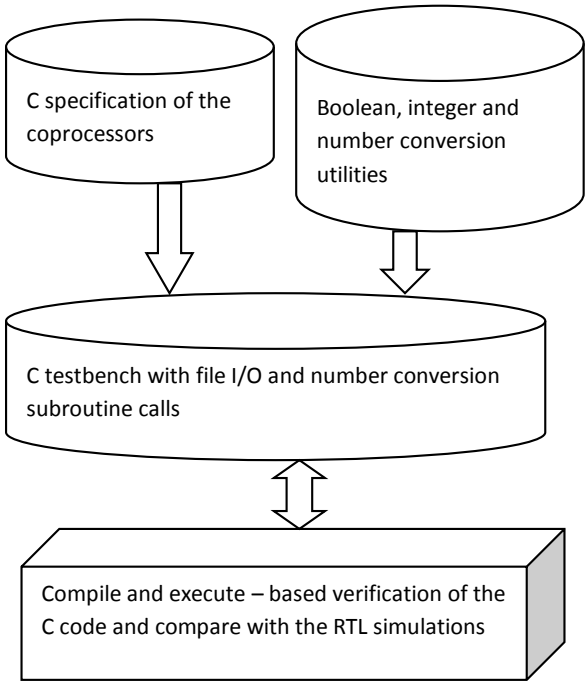

Figure 2. ***CubedC verification flow***

The trigonometric benchmarks were coded in pure, unaltered algorithmic ANSI-C subroutines, in the same manner that a software program is constructed. For clear design correspondence of the custom block structure, the Boolean and array of Boolean types were used in the core C function. Certain conversion functions were placed in a separate file (C program module library) to convert between integer, boolean and real values for test purposes. These subroutines are called by a single C testbench, compiled as such and run to test the model's behavior. The verification results were cross-checked with the RTL simulations and in all cases the functionality of the generated hardware matched that of the C models, which was expected due to the formality of the synthesis flow.

Our high-level verification technique is orders-of-magnitude faster and more robust than RTL or gate-level simulations, since it is based on mature and rapid compile and execute of functional program code. Due to CubedC's formal transformations the synthesis results are provably-correct and they always match the behavior of the C source programs, therefore, time-consuming and prone to bugs RTL simulations are avoided for large designs.

The trigonometric numerical algorithms of this work were implemented in the CubedC ANSI-C front-end, used for this purpose. The basic four trigonometric functions were specified in C, while more benchmarks are currently being

IJERMS
International Journal of Engineering Researches and Management Studies

developed. The Taylor series were utilized to approximate the theoretical values for the trigonometric functions. Using our rapid C-to-HDL transformation, first-time-right trigonometric RTL is formally realized.

```
Give me a number to calculate sine: 90
The result of sine will be: 1000

Give me a number to calculate cosine: 180
The result of cosine will be: -1000

Give me a number to calculate tangent: 45
The result of tangent will be: 1000

Give me a number to calculate cotangent: 45
The result of cotangent will be: 1000
```

Figure 3. ***Execution of the integrated trigonometric testbench in C***

Fig. 3 demonstrates the run of the C testbench for the four trigonometric functions in an integrated fashion that tests all of functions in a single go. Due to scaling, the results are multiples of the theoretical ones, so the output numbers in Fig. 3 need to be divided by 10^3.

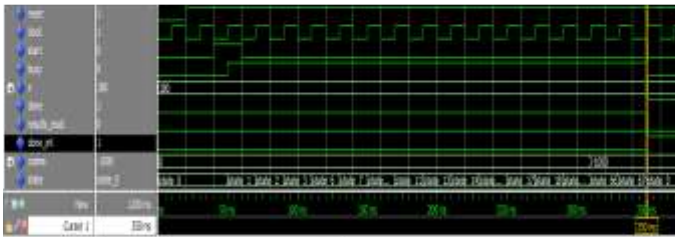

Figure 4. ***RTL simulation of the optimized by PARCS cosine function***

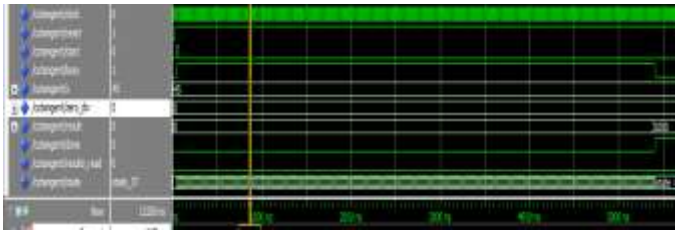

Figure 5. ***RTL simulation of the PARCS contangent implementation***

Fig. 4 and Fig. 5 show the cosine and cotangent function RTL simulation results for the optimized by PARCS. They clearly match (as expected) the results of the high-level functional C testbench again in this case.

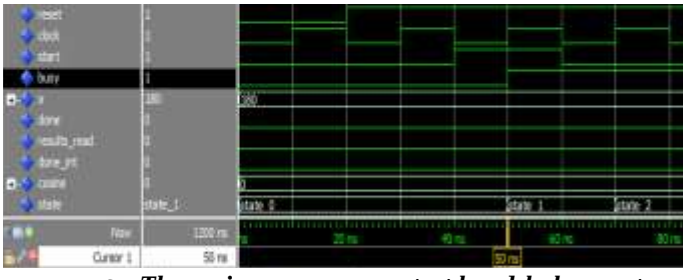

Figure 6. ***The cosine coprocessor start handshake event***

IJERMS
International Journal of Engineering Researches and Management Studies

The waveforms in Fig. 6 and Fig. 7 show the START (beginning) and END (completion) handshake events for the cosine calculation coprocessor RTL simulations. The start/end handshake interface events used by CubedC is a robust way to control the functions of the generated coprocessors and they indicate the beginning and completion of the coprocessor's function, occupying just one clock cycle each.

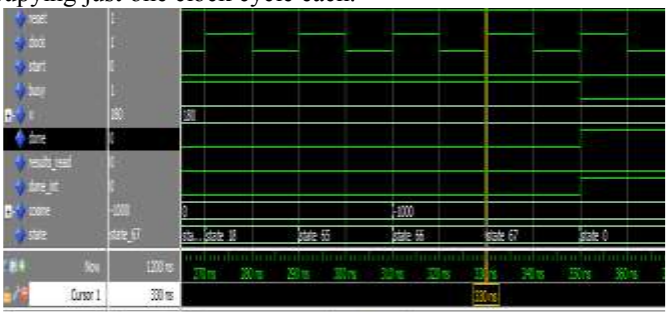

Figure 7. ***The cosine coprocessor end handshake event***

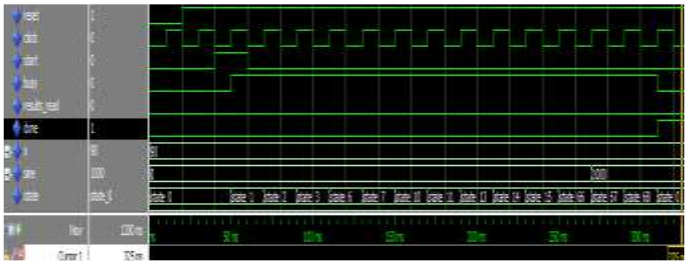

Figure 8. ***The sine calculation RTL simulation***

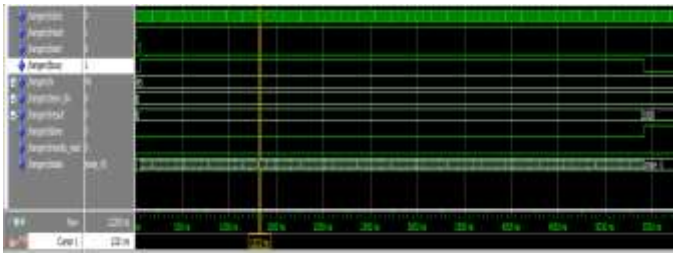

Figure 9. ***The tangent coprocessor RTL simulation***

Fig. 8 and Fig. 9, show the functionality match between the high-level C model (specification test in Fig. 3) and the generated RTL functional simulation for the sine and tangent functions. The argument of this work is proven once again, and since the CubedC tool transformations are formal, there is practically no need for exhaustive RTL or gate-netlist simulations, and verification can be concluded in a rapid manner at the high-level C or ADA program level.

Many more RTL simulations were run in this work to prove the correctness of the generated hardware implementations, but showing them here is outside the space/purpose of this paper.

V. SYNTHESIS AND IMPLEMENTATION RESULTS

The numerical benchmarks and a broader number of hardware arithmetic blocks surpass in some cases more than 100 generated RTL schedule states. Due to the complexities of such designs, it is impossible to develop and verify such circuits directly in RTL without letting functional bugs to penetrate the design verification methodology. This is because of the vast test-case set of these circuits if the designers try to achieve exhaustive simulation coverage. Using

IJERMS
International Journal of Engineering Researches and Management Studies

our synthesis methodology the above designs were verified and synthesized in terms of a few hours, even by not experienced informatics students. Therefore, our formal and rapid high-level ADA/C compile-and-execute-based verification is extremely faster than conventional RTL simulation and debugging.

The basic trigonometric functions were implemented in CubedC ANSI-C code and verified as discussed earlier in this paper. Subsequently, they were all synthesized and implemented using the Xilinx ISE 14.7 64-bit version RTL implementation tool. The Spartan-3 xc3s400-4FT256 device was targeted, which is one of the low-end cost Xilinx device.

```
        do
        {
           counter = counter + 1;
           n = n + 2;
           calc = 1;

           for (i = 0; i < n; i ++) {

              calc = calc * radians;
              calc = calc / (i + 1);

             if (i > 0) calc = calc / 1000;

              if (i == n - 1) {
                 sinterm = calc;
              }
           }
           if (counter % 2 == 1)
           {
              sinx = sinx - sinterm;
           } //if
           else
           {
              sinx = sinx + sinterm;
           }//else
        } while ((sinterm > 0));
        result = sinx;
     }
```

Figure 10. ***Core ANSI-C code for the sine function***

The core part of the ANSI-C sine function is shown in the listing of Fig. 10, where all of the standard code structures are used and accepted by the CubedC compiler for hardware modeling. From this part of code it can be confirmed that the use of the ANSI-C language in our synthesis flow is seamless.

TABLE I. ***NUMERICAL BENCHMARK STATE OPTIMIZATION***

benchmark	**Unoptimized states**	**Optimized (PARCS) states**	**opt. rate**
sine	95	69	38%
cosine	95	68	40%
tangent	138	98	41%
cotangent	137	98	40%

IJERMS
International Journal of Engineering Researches and Management Studies

The implementation statistics for the numerical benchmarks are formulated in Tables I and II

TABLE II. *XILINX IMPLEMENTATION RESULTS FOR THE TRIG. BLOCKS*

	Sine UNOPT	Sine PARCS	Cosine UNOPT	Cosine PARCS	Tangent UNOPT	Tangent PARCS	Cotangent UNOPT	Cotangent PARCS
Slice Flip Flops	544	519	556	533	783	735	835	776
4 input LUTs	3,021	3,027	3,060	3,059	4,758	4,746	4,738	4,726
occupied Slices	1,916	1,902	1,947	1,935	2,842	2,790	2,835	2,797
LUTs used as a route-thru	444	444	445	444	474	468	468	470
bonded IOBs	70	70	70	70	70	70	70	70
MULT18X18s	7	7	7	7	7	7	7	7
Minimum clock period	220 ns	220 ns	210 ns	220 ns	221 ns	227 ns	227 ns	222

Table I shows the reduction (optimization) of the benchmarks' number of FSM states by the PARCS scheduler. Table II shows the Xilinx implementation statistics for the 4 basic trigonometric functions.

VI. CONCLUSIONS AND FUTURE WORK

The CubedC tools are used in this work for the rapid design, development and verification of numerical hardware functions. Experimental results and an analysis of our method prove that this method is very efficient, useable and provably-correct. The Taylor series mathematical base was used for the coding of the trigonometric functions. The learning curve of the CubedC tools, even for not experienced informatics students was steep and the total experiments' flow took less than a few weeks. Regarding the produced chip area, the CubedC synthesis of the numerical benchmarks are not competing with full-custom chip design, but considering the development effort and time, the CubedC methodology is more than orders of magnitude efficient, faster, formal and first-time-correct. Moreover, a few lines of ADA or C program code produce hundreds of lines of RTL code, thus even in terms of simply program code writing, the high-level coding effort is much shorter than RTL coding.The CubedC synthesis approach is formal and extensive simulations of the produced RTL code are not needed. Thus, the method is extremely faster and bug-free compared to custom or manual RTL design. This was proven in practice by all of our so far synthesis and implementation experiments. The main contribution of this work is the use of rapid synthesis-based development flows for the implementation of trigonometric functions in the form of hardware coprocessors. These coprocessors are controlled by robust FSMs that are also automatically and rapidly produced from high-level functional program code.

Future work includes the experiments with more complex hardware arithmetic functions. Such applications are found in computer graphics and image/DSP processing of big data. The flexibility of the CubedC framework is enhanced with the number of available user options, in order to quickly deliver complex circuits and custom arithmetic blocks that can be easily incorporated as silicon-proved IP in large hardware design projects.

IJERMS

International Journal of Engineering Researches and Management Studies

VII. REFERENCES

[1] *IBM PowerPC 970FX RISC Microprocessor User's Manual*, IBM, Online Document (22/11/2013). Available at: https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/DC3D43B729FDAD2C00257419006FB955/$file/970FX_user_manual.v1.7.2008MAR14_pub.pdf, 2006.

[2] *Sun Niagara 2 Processor*. Online Document (22/11/2013), Available at: http://bnrg.eecs.berkeley.edu/~randy/Courses/CS294.F07/SunNiagara2.pdf , 2103.

[3] *Statistical analysis of floating point flaw*, Intel, Online Document (22/11/2013). Available at: http://users.minet.uni-jena.de/~nez/rechnerarithmetik_5/fdiv_bug/intel_white11.pdf , 30th November 1994.

[4] D. Deley, *THE PENTIUM DIVISION FLAW*, Online Document. Available at: http://en.wikipedia.org/wiki/Pentium_FDIV_bug, 22/11/2013.

[5] GAO/IMTEC-92-26, Patriot Missile Software Problem, Online Document (22/11/2013). Available: http://www.fas.org/spp/starwars/gao/im92026.htm, 1992.

[6] *ARIANE 5 Flight 501 Failure*. Online Document. Available at: http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html, 1996.

[7] M. Ercegovac, and T. Lang, Digital Arithmetic, Morgan Kaufmann, 2003.

[8] I. Koren, Computer Arithmetic Algorithms, 2nd ed. A K Peters Ltd, 2001

[9] B. Parhami, Computer Arithmetic: Algorithms and Hardware Designs, 1st ed. Oxford University Press, 1999.

[10] Proceedings of the Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05), pp. 198-203, 2005.

[11] A. Weinberger, and J. Smith, "A logic for high-speed addition", National Bureau of Standards Circular, vol. 591, pp. 3-12, 1958.

[12] R. Brent, and H. Kung, "A regular layout for parallel adders", IEEE Trans. Comput., vol. C-31, pp. 260-264, 1982.

[13] P. Kogge, and H. Stone, "A Parallel algorithm for the efficient solution of a general class of recurrence equations", IEEE Trans. Comput., vol. 22, pp. 786-793, 1973.

[14] J. Sklansky, "Conditional sum addition logic," IEEE Trans. Electron. Comput., vol. EC-9, no. 6, pp. 226-231, Jun. I960.

[15] V. G. Oklobdzija, "An algoritmic and novel design of a leading zero detector circuit: Comparison with logic synthesis", IEEE Trans. VLSI Syst., vol. 2, no. 1, pp. 124-128, 1993.

[16] M. Farmwald, On the design of high performance digital arithmetic circuits, Ph.D. dissertation, Stanford University, 1981.

[17] J. Hennessy, and D. Patterson, Computer Architecture: A Quantitative Approach, 3rd ed. MORGAN KAUFFMAN, ch. Appendix H, 2002.

[18] *Pentium II math bug?*, Online Document, Available at: http://www.rcollins.org/ddj/Aug97/Aug97.html, 22/11/2013.

[19] M. Dossis, *CCC (Custom Coprocessor Compilation), Custom Coprocessors on the Web*, Online tools. Available at: http://195.130.87.55/ccc/index.html, 1/5/2014.

[20] M. Dossis, Patent number 1005308, 5/10/2006 by the Greek Industrial Property Organisation, 2006.

[21] M. Dossis, "Modeling and Simulation in a Formal Design Framework," ACM Proceedings of the 6th Balkan Conference in Informatics, Thessaloniki, Greece, pp. 31-38, September 19-21, 2013

[22] Michael Greenberg, 1998. Advanced Engineering Mathematics (2nd ed.), Prentice Hall, ISBN 0-13-321431-1

[23] George B. Jr. Thomas, and Ross L. Finney, 1996. Calculus and Analytic Geometry (9th ed.), Addison Wesley, ISBN 0-201-53174-7

[24] D. J. Struik, 1969. A Source Book in Mathematics 1200-1800. Cambridge, Massachusetts: Harvard University Press, 1969, pp. 329-332.

[25] U. Zabit, F. Bony and T. Bosch, 2008. "Implementation of optimized trigonometric functions for a self-mixing laser diode displacement sensor under moderate feedback". Proceedings of the 2008 IEEE Sensors conference, Lecce, 26-29 Oct. 2008, pp. 988-991.

IJERMS

International Journal of Engineering Researches and Management Studies

[26] W.F. Wong, and E. Goto, 1995. "Fast evaluation of the elementary functions in single precision". IEEE Transactions on Computers, vol. 44, issue 3, pp. 453-457.
[27] V. Kantabutra, 1996. "On hardware for computing exponential and trigonometric functions". IEEE Transactions on Computers, vol. 45, issue 3, pp. 328-339.
[28] U. Nilsson, and J. Maluszynski, Logic Programming and Prolog, John Wiley & Sons Ltd., 2nd Edition, 2000.
[29] M. Dossis, "Intermediate Predicate Format for design automation tools," Journal of Next Generation Information Technology (JNIT), Vol. 1, No. 1pp. 100-117, 2010.
[30] M.F. Dossis, "A Formal Design Framework to Generate Coprocessors with Implementation Options," International Journal of Research and Reviews in Computer Science (IJRRCS), August 2011, ISSN: 2079-2557, Science Academy Publisher, United Kingdom, www.sciacademypublisher.com, Vol 2, No 4, pp. 929-936, 2011.
[31] M. Dossis, "Rapid Modelling and Verification in the Intelligent CCC Synthesis Flow," International Journal of Information Science and Intelligent System (IJISIS), vol. 2, no.1, June 2013, pp. 7-25, 2013.
[32] M. Dossis, "Custom Options for Custom Processors," Proceedings of the 1st International Virtual Scientific Conference, Slovakia, Zilina, June 10-14, 2013, pp. 370-375, 2013.