Proceedings

Sixteenth IEEE International Workshop on Rapid System Prototyping

Shortening the Path from Specification to Prototype

8–10 June 2005
Montreal, Canada

Sponsored by
IEEE Computer Society Technical Committee on Simulation
IEEE Computer Society Technical Committee on Design Automation

IEEE COMPUTER SOCIETY

IEEE

KoVer : A Sophisticated Residue Arithmetic Core Generator

Nikolaos Kostaras[1] and H. T. Vergos[1,2]
[1]Dept. of Computer Engineering & Informatics, University of Patras, 26 500 Rio, Greece
[2]Computer Technology Institute 3 Kolokotroni St., 26 221 Patras, Greece
E-mail : {kostaran,vergos}@ceid.upatras.gr

Abstract

Numerous architectures have been recently proposed for residue arithmetic components, each with its own speed, area and power consumption characteristics. In this paper, we present KoVer, a novel software tool that gives a designer the opportunity to explore several architectures for implementing his residue arithmetic blocks, select the one that best suits his goals and instantly get the HDL level description of the selected architecture.

1. Introduction

Pre-verified cores, available in technology independent descriptions such as in hardware description languages (HDLs) reduce time-to-market by providing a vehicle for faster design cycles. Since arithmetic components are one of the most commonly used design blocks, several arithmetic core generators have been presented during the last years. Most of these generators however only attack the problem of producing cores for the weighted binary / 2's complement / IEEE 754 floating point representations.

On the other hand, arithmetic modulo d, where d is a positive integer, has found great applicability in digital computing systems. Its areas of application include the implementation of residue, inverse residue, product and checksum arithmetic codes which are used extensively in TCP/IP network error detection and in traffic monitoring and statistics gathering in high-speed networks, in pseudo-random number generation by hardware, in the implementation of cryptographic algorithms that use modulo multiplications and exponentiations and in the Fermat number transform, which is used to compute convolutions without round off errors.

Furthermore, non-positional Residue Number Systems (RNS) that make parallel use of arithmetic modulo d components, have been proposed as an attractive alternative to the binary system for applications whose arithmetic operations are limited to addition, subtraction, multiplication and squaring [1]. In particular, the scientific community has shown great interest for d of the $2^n \pm 1$ forms. Apart from their use in TCP/IP networking hardware and Fermat number transform, these moduli have been extensively used in RNSs that exploit the commonly used three moduli set $\{2^n, 2^n-1, 2^n+1\}$. Therefore, a great number of distinct very fast architectures have been proposed for arithmetic components modulo $2^n \pm 1$.

In this paper , we present KoVer, a core generator that attacks the problem of producing reusable and pre-verified residue arithmetic cores. KoVer integrates a significant number of different architectures proposed for modulo $2^n \pm 1$ arithmetic components. In the case of the 2^n+1 architectures, components that either use the weighted or the diminished-1 number system [2] have been considered.

By integrating a variety of architectures, KoVer provides the designer with the ability to perform an architectural exploration of the possible solutions and choose the one that best suits his area, time and power consumption goals. KoVer produces structural HDL descriptions of the chosen cores, along with a testbench comprised of random simulation vectors for the generated cores. Since no encryption is performed on the output, the designer can easily further customize the produced HDL code and straightforwardly integrate it with the rest of his design.

Since the number of architectures proposed for modulo $2^n \pm 1$ components is constantly rising, an open software architecture was a prerequisite for designing KoVer. Therefore KoVer's design follows a completely modular architetcure and gives a user the ability of adding completely new architectures or extending any of the existing ones in a straightforward manner.

2. Supported Architectures & Features

Considering that the main applications of residue arithmetic target performance enhancement, we chose to integrate into KoVer only the fastest known architectures of each generated arithmetic core and their derivatives.

For modulo 2^n-1 adders KoVer supports the architec-

Proceedings of the 16th International Workshop on Rapid System Prototyping (RSP'05)
1074-6005/05 $20.00 © 2005 **IEEE**

tures proposed in [3-7], along with a proprietary architecture. Adders that support either both or only a single representation for the zero operand can be generated. The adders of [7] are based on a CLA carry computation unit and offer an execution time similar to the integer CLA adders. All the rest architectures are either very fast parallel-prefix ones or hybrid ones , that is, they use both parallel-prefix blocks and CLA parts between them [4]. The totally parallel-prefix architecture of [3] is only applicable to n-bit adders with n of the 2^k form. On the other hand, when $n \neq 2^k$, several simplifications can be applied to the parallel-prefix structure as proposed in [5, 6].

For modulo 2^n+1 adders KoVer supports both weighted and diminished-1 representation of operands. In the last case, the user can further select whether the handling of zero operands will be done by another block or within the adder description produced by KoVer, as proposed in [8]. In the case of weighted representations, KoVer provides support for both architectures proposed in [9]. For diminished-one operands KoVer supports the totally parallel-prefix architecture proposed in [10], which is only applicable to n-bit adders with n of the 2^k form, the hybrid architecture proposed in [4] and a proprietary one.

In the case of modulo 2^n-1 multipliers, KoVer provides support for both array [11] and modified Booth [12] multipliers. The last adder of the multiplier can be further selected from the supported architectures. KoVer also supports three different baseline architectures for modulo 2^n+1 multiplication. For weighted representations the array architecture proposed in [13] can be selected along with any of the two weighted adder architectures as the last stage adder. In the case of diminished-1 operands, either an array [14] or a modified-Booth architecture [15] can be followed. The last stage adder can be selected among the offered architectures of diminished-1 adders.

Finally, KoVer supports modulo 2^n-1 [16] and weighted / diminished-1 modulo 2^n+1 squarers [17] as well as several different proprietary architectures for combined multiplication / sum-of-squares units.

Table 1 summarizes the number of baseline architectures supported for each type of arithmetic components as well as the number of variant architectures that can be constructed according to the user preferences. As we can see, the number of different cores that KoVer can generate is very large and gives the designer the opportunity for architectural exploration of the best solution. On the other hand, architectural exploration requires a significant amount of time. To this end, KoVer includes some sophisticated features that can be of significant help to the designer.

We have currently implemented the following two sophisticated features :

- *Past results knowledge* and

Table 1. Different architectures supported

Component type	# of Baseline Architectures	# of Variants
2^n-1 adders	6	12
2^n+1 adders	5	8
2^n-1 multipliers	2	24
2^n+1 multipliers	3	16
2^n-1 squarers	1	12
2^n+1 squarers	2	10
Combined multiplication / sum of squares units	3	40

- *Automated RNS datapath construction.*

The *past results knowledge* refers to the incorporation into KoVer's database of implementation area, execution delay and power consumption measurements. These measurements were gathered by analysis and simulation of layout extracted data for each of the supported variants, utilizing two distinct implementation technologies. These measurements are readily available to the user and are further used in KoVer's database for classifying the possible variants. As a result the end user may only choose to select a category instead of a specific architecture and therefore the required exploration is narrowed significantly. For example, once the user asks the tool for the fastest modulo 2^n-1 adder with a single representation of zero, taking into account the past knowledge, the tool will limit the architectural exploration to the architecture of [3] if $n=2^k$, or to the architectures of [5,6] instead.

The automated RNS datapath construction is an integrated environment for a user that wants the minimum interaction with the tool and the less integration effort. The user first needs to specify the arithmetic components of his target RNS datapath and then the required precision of his application, for example 64 bits. KoVer will then provide a set of alternative RNS moduli sets comprised of up to five moduli that can offer the required precision. In our example the moduli sets $\{2^{22}, 2^{22}-1, 2^{22}+1\}$ and $\{2^{17}, 2^{17}-1, 2^{17}+1, 2^{13}-1\}$ are two of the possibilities considered by KoVer. Based on the past results knowledge, KoVer categorizes these possibilities in terms of area, delay and power consumption.

3. Comparison results

In this section, we compare KoVer against a commercial core generator, using as test vehicles a few residue arithmetic cores.

All the required HDL descriptions were produced by KoVer in less than 2 seconds of run time in a Pentium 4

personal computer. After extensive simulation of the HDL codes, the designs were synthesized in a 0.25μm static CMOS technology with the restriction of a maximum fan-out of 4 and optimized following a standard optimization script. The synthesized netlists along with the design constraints to achieve them were then used for placing and routing the designs. All design constraints, such as output load, floorplan initialization information and pin placement were held constant for each architecture. Final timing analysis was performed after all RC parasitic information was extracted from the layout and back-annotated to the gate-level netlist. Typical voltage and temperature operating conditions are assumed along with average gate delays.

The results obtained are indicated in Table 2. As we can see, the cores produced by KoVer outperform the ones of the commercial generator in both the required implementation area as well as the execution latency. The savings observed are in the average of the examined cases 30.4% with respect to the implementation area and 17.5% with respect to the execution delay. If we restrict our observations only to modulo multipliers, which is considered an even more specific module than a modulo adder, we can see that the savings offered by KoVer's components raise to 35.5% and 21.3% in implementation area and execution delay respectively.

Table 2 : Comparison results

	KoVer generated cores			Cores produced by commercial generator	
Component	***Architecture followed***	***Area (μm^2)***	***Delay (ns)***	***Area (μm^2)***	***Delay (ns)***
2^{16}-1 adder	[26]	23012.7	1.24	31875.1	1.63
2^{16}+1 adder	[32]	26765.3	1.28	34667.0	1.67
2^{8}-1 multiplier	[35]	33891.3	3.06	51982.9	3.88
2^{8}+1 multiplier	[37]	35641.3	3.09	55873.2	3.93

4. Conclusions

In this paper, we have presented KoVer, a new core generator for residue arithmetic components. KoVer can produce a large number of distinct very fast architectures for each considered residue arithmetic component and many variants of each, enabling a designer to find the one that best suits his needs by architectural exploration. The sophisticated features embedded in KoVer can further help the designer narrow his search space. The generated cores are offered in synthesizable HDL, enabling further customization by the user. KoVer follows an open software architecture, that allows straightforward upgrades with new design templates and features.

5. References

[1] N. Szabo and R. Tanaka, *Residue Arithmetic and its Aplications to Computer Technology*, McGraw-Hill, 1967.

[2] L. M. Leibowitz, "A simplified binary arithmetic for the Fermat number transform", IEEE Trans. Acoust. Speech, Signal Processing, 1976, pp. 356-359.

[3] L. Kalampoukas et al., "High-Speed Parallel-Prefix Modulo 2^n-1 Adders", IEEE Trans. on Computers, July 2000, pp. 673-680.

[4] C. Efstathiou et al., "Modulo 2^n±1 Adder Design Using Select-Prefix Blocks", IEEE Trans. on Computers, November 2003, pp. 1399-1406.

[5] G. Dimitrakopoulos et al., "A Systematic Methodology for Designing Area-Time Efficient Parallel-Prefix Modulo 2^n-1 Adders", Proc. of ISCAS 2003, pp. 225-228.

[6] G. Dimitrakopoulos et al., "A family of Parallel-Prefix Modulo 2^n-1 Adders", Proc. of ASAP 2003, pp. 326-336.

[7] C. Efstathiou et al., "Area–Time Efficient Modulo 2^n-1 Adder Design", IEEE Trans. on Circuits and Systems–II, 1994, pp. 463–467.

[8] C. Efstathiou, et. al, "Handling Zero in Diminished-One Modulo 2^n+1 Adders", Int. J. of Electronics, Feb. 2003, pp. 133-144.

[9] C. Efstathiou et al., "Fast Parallel-Prefix Modulo 2^n+1 Adders", IEEE Trans. on Computers, Sep. 2004, pp. 1211-1216.

[10] H. T. Vergos et al., "Diminished-One Modulo 2^n+1 Adder Design", IEEE Trans. on Computers, Dec. 2002, pp. 1389-1399.

[11] Z. Wang et al., "An algorithm for multiplication modulo (2^N-1)", Proc. of the 39th Midwest Symp. on Circuits and Systems, 1997, pp. 1301-1304.

[12] C. Efstathiou et al., "Modified Booth Modulo 2^n-1 Multipliers", IEEE Trans. on Computers, March 2004, pp. 370-374.

[13] A. Wrzyszcz and D. Milford, "A New Modulo 2^a + 1 Multiplier", Proc. of ICCD 1993, pp. 614–617.

[14] C. Efstathiou et al., "Efficient Diminished-1 Modulo 2^n+1 Multipliers", IEEE Trans. on Computers, April 2005, pp. 491 – 496.

[15] Y. Ma, "A Simplified Architecture for Modulo (2^n+1) Multiplication", IEEE Trans. on Computers, March 1998, pp. 333-337.

[16] S. J. Piestrak, "Design of squarers modulo A with low-level pipelining", IEEE Trans. on Computers, 2002, pp. 31–41.

[17] H. T. Vergos and C. Efstathiou, "Diminished-1 Modulo 2^n+1 Squarer Design", Proc. of DSD 2004, pp. 380-386.