An ASIC Linear Congruence Solver Synthesized with Three Cell Libraries

Jiří Buček, Pavel Kubalík, Róbert Lórencz, and Tomáš Zahradnický
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00 Prague, Czech Republic
Email: { bucekj | xkubalik | lorencz | zahradt }@fit.cvut.cz

***Abstract*— The paper describes an ASIC implementation of a previously implemented FPGA linear congruence solver, part of a parallel system for solution of linear equations, and presents synthesis results for three different standard cell libraries. The previous VHDL design was adapted to three ASIC technologies (130 nm, 110 nm, and 55 nm) from two different vendors and the synthesized results were mutually compared. The maximum clock frequency and occupied area of the synthesized design were collected and analyzed for several input matrix dimensions and the maximum possible input problem size for each of the technologies was determined. The comparison results were further used to obtain a view of design properties in higher density technologies.**

I. Introduction and Background

Solution of a system of linear equations (SLE) $\mathbf{Ax} = \mathbf{y}$ is a common task of linear algebra and is frequently computed in a floating point arithmetic, which involves rounding. There are numerous methods for solving linear systems of various sizes, for spare or dense systems, for differently conditioned systems, for symmetric, or positive definite systems etc., and all of them need to choose an arithmetic wherein to perform their operations. A common choice is one of the floating point (FP) arithmetics defined in the IEEE 754:2008 Standard [1] and, it is a well known fact, that using an FP arithmetic shall induce a question of numerical stability, especially in the case of large, dense, and/or ill-conditioned systems, where the input error magnification and/or roundoff errors may heavily impact or even destroy the solution of the SLE.

When rounding is undesired, it is possible to use arithmetics that do not round, such as the arithmetic of the Residual Number System (RNS) [2][3], usually at cost of an increased complexity. Parallel algorithms for solving SLEs using congruence techniques were proposed in [4][5][6] converting the input SLE into several to many independent systems of linear congruences (SLC)s, solving them independently, and reconstructing the SLE result with the Chinese Remainder Theorem, summarized in three steps:

1) Transformation of the augmented SLE matrix $\mathbf{A} \mid \mathbf{y}$ into independent linear systems $(\mathbf{A} \mid \mathbf{y}) \bmod m_i$, each with a distinct prime number modulus m_i, for $i = 1 \dots p$ being a number of moduli.
2) Solution of p independent systems of linear congruences (SLC)s in form $\mathbf{Ax} \equiv \mathbf{y} \pmod{m_i}$.
3) Reconstruction of the SLE solution vector $\mathbf{x}$ from its RNS residual representations $\mathbf{x} \pmod{m_i}$.

Using RNS increases the temporal and spatial complexity and one of the ways to go is to use a dedicated hardware Modular System (MS) performing SLE solution and exploiting natural parallelism offered by the RNS. Such MS was discussed in our previous papers [5][6][7][8].

When studying methods of solving SLEs in RNS arithmetic, a gap in the publication activity can be observed since the mid 1990's. This can be explained by the limited technology resources available at the time and thus limited sizes of instances that could be solved. At the same time, better and more sophisticated methods for solving SLEs in FP arithmetic were developed that could be performed on general purpose CPUs. The current demand for precise SLE solving is based on the fact that FP arithmetic has its limitations, and at the same time, progress in technology enables creating HW solvers that can effectively use RNS to solve SLEs for real applications. RNS is currently used in areas of digital image processing [9][10], digital signal processing [11][12][13], and in public-key [14] and elliptic curve [15][16] cryptography.

The goal of this paper is to explore the achievable area consumption and maximum speed for several ASIC technology libraries, and thus establishing the limitations on the maximum SLC instance size that can be solved. Previous work [8] indicated that the internal memory was a limiting factor and it is then interesting to compare the synthesis results in different technologies with their corresponding memory blocks.

The paper is organized as follows: Section I. introduces the reader into the problematics. Section II. provides a brief overview of the previous work. Section III. presents the architecture of the basic SLC solution unit — a Residual Processor. Section IV. summarizes the results, Section V. concludes the paper, while Section VI. describes our future efforts.

II. Previous work

The method of the SLE solution is based on the previous work [5][6] including methods, algorithms, and the corresponding parallel hardware architecture of the MS (Fig. 1). MS solves the SLE in RNS and therefore transforms the input SLE onto multiple independent SLCs, each with its own unique prime number modulus, solved at a distinct Residual Processor (RP). It should be noted that evaluation in each

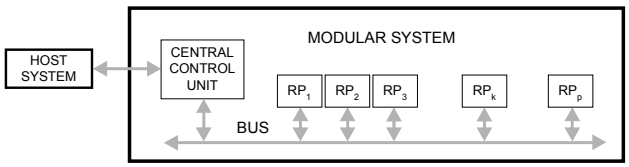

Fig. 1. Architecture of the Modular System [5]. The Central Control Unit controls solution processes and communicates with the Host System. RP_1 through RP_p are individual Residual Processors, each with its own modulus m_i for $i = 1 \ldots p$. RPs perform transformation of the input SLE into RNS, perform SLC solution mod m_i, and also back-transformation from RNS, while The Bus denotes an internal interconnection of all units within the MS.

modulus is performed independently of the others and that the addition and subtraction are carry/borrow-free across the individual moduli, and thus the computation can occur safely in parallel. Once the SLC solutions are available, they are recombined back into a solution of the SLE.

The architecture of the residual processor RP_k is depicted at Fig. 2 consisting of a Memory, Arithmetic Units AU_2, AU_3, …, AU_{n+2} and a Control unit. RP, which was described in [7], was designed with a focus on its effective implementation in FPGA for various problem sizes with special attention to the memory architecture. The memory design was critical to the RP because of massive data access and pivoting. All important parts of the RP architecture, such as data memory, Pivot index, Pivot flags, counters, arithmetic units, inversion unit and control unit were implemented and tested in a Xilinx Virtex 6 FPGA with the largest block RAM capacity of its family, i.e. 38304 Kibits. The results showed that RP architecture with a 1000-row matrix and with a 24-bit word size in Xilinx XC6VSX475T occupied more than 90 % of the available block RAM and approximately 60 % of all available slices. This implementation was the largest possible to fit in the FPGA and ran approximately 2 times faster than a software implementation at a CPU.

The 24-bit word length was chosen as a compromise so that enough prime moduli can be generated to represent the largest number needed during solution of the system of linear equations. This follows from the Hadamard's inequality and its application on solving a linear system exactly [5].

Paper [8] implemented the same RP architecture in ASIC with a 130 nm standard cell library and compared it to the FPGA architecture described in [7]. Results in our previous paper [8] indicated that the ASIC implementation was yet 4 times faster than its FPGA counterpart.

This contribution builds on papers [7][8] and extends the ASIC implementation of our SLC solver to three different standard cell libraries for three ASIC technologies from two different vendors – Synopsys/GlobalFoundries 130 nm and Faraday/UMC 110 nm and 55 nm. The 130 nm and 110 nm libraries are high performance libraries, while the 55 nm library is a low power library. We have chosen these particular libraries mainly because of their availability including memory compilers, which are important in our design and evaluation. The last one being a low power library, it is not directly comparable in terms of speed, but it can be used to evaluate

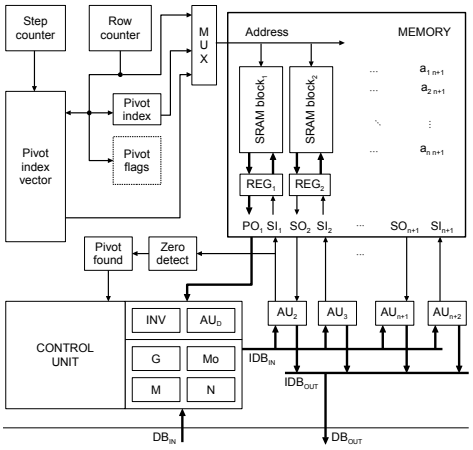

Fig. 2. Architecture of the Residual Processor. a_{ij}s represent elements of the augmented SLC matrix, AU_i stand for individual arithmetic units, PO_1 is a parallel output, $\mathrm{SI_i/SO_i}$ denote serial inputs/outputs, $\mathrm{IDB_{IN}/IDB_{OUT}}$ represent internal data buses, while $\mathrm{DB_{IN}/DB_{OUT}}$ represent external data buses. INV, $\mathrm{AU_D}$, G, Mo, M, and N are auxiliary units described in [6]. Other blocks serve to implement partial pivoting.

the area savings coming from greater integration. The chosen libraries, although not the most recent ones, are sufficient to reveal technology-dependent properties of the architecture, allowing prediction of the behavior of our design in general. The output of these implementations will provide insight into the properties of the design in higher density technologies.

The next section describes the experimental results, compares them together, and analyzes the maximum clock frequency and occupied area of the synthesized RP design.

III. Architecture Overview

Each residual processor RP_k can solve systems of linear congruences (SLC)s $\mathbf{A}\mathbf{x}_k \equiv \mathbf{b} \pmod{m_k}$ and its architecture is depicted at Fig. 2 and consists of Memory, Arithmetic Units AU_2, AU_3, …, AU_{n+2} and the Control unit.

The memory contains an augmented matrix $(\mathbf{A}|\mathbf{b}) \bmod m_k$. It consists of SRAM blocks that are created using the memory compiler specific for the ASIC library and an appropriate size block must be selected for the maximum matrix dimension n.

The arithmetic units AU are connected to the memory via Serial Inputs (SI) and Serial Outputs (SO). The leftmost elements of the matrix are read by the Control unit via the parallel bus PO_1. All AUs and the Control unit are interconnected via Internal Data Buses $\mathrm{IDB_{in}}$ and $\mathrm{IDB_{out}}$. The INV and $\mathrm{AU_D}$ units compute the modular multiplicative inverse and the determinant of $\mathbf{A} \bmod m_k$, respectively.

During elimination, partial pivoting is used, where any non-zero element can be a pivot (Zero detect block). The Pivot index block holds the row address of the pivot for the current elimination step. Pivot flags contain one bit for each row indicating whether the row contained the pivot in past elimination steps, while the Pivot index memory block contains row addresses of all pivots found up to the current elimination step. These addresses are used at the end of the elimination process for result reordering (no row swapping is done during elimination).

IV. Experimental Results

All experiments were conducted on the RP architecture shown at Fig. 2. The design was specified in VHDL, simulated in Mentor Graphics ModelSim, and synthesized using Synopsys Design Compiler.

The maximum matrix dimension n and the word length e were configurable at synthesis time using generics, while the actual matrix dimension, the modulus, and the matrix data were specified at runtime.

The solution designed for FPGA [7] was modified to obtain a solution usable for synthesis process for ASIC. The modification was focused mainly on memory interface and restrictions of the synthesis process such as a number of gate inputs. The designed ASIC architectures for various matrix dimensions were used to conduct a set of SLC solution experiments on a single RP. The SLC solution included input both data modulo reduction and matrix elimination.

In order to verify the design, we added test units to increase testability and observability of the simulated design and to verify the calculation. The test data were generated using Wolfram Mathematica and converted with a Python script into a file format suitable for the simulation. The RP was simulated with Mentor ModelSim to solve SLCs and their solutions were compared to SLCs solutions precomputed in Wolfram Mathematica. The simulation was performed for matrices up to matrix dimension $n = 100$. Matrices with dimension $n > 100$ were not simulated due to high simulation times.

After the verification of simulation correctness we started the ASIC synthesis process. Since block RAMs were not present in the ASIC library as standard cells, we generated them with a special memory generator tool coming with the library. We generated a suitable set of synchronous RAM modules with sizes to cover the expected matrix dimensions. The generated RAM modules were then instantiated according to the generic parameters from the VHDL description to implement memory matrix columns (a_{i1} to $a_{i,n+1}$).

We compared three different standard cell libraries for three ASIC technologies (130 nm, 110 nm, and 55 nm) from two different vendors. The maximum clock frequency and the occupied area of the synthesized design were collected and analyzed for several matrix dimensions. The first two technologies 130 nm and 110 nm were high speed and the last technology, the 55 nm one, was low power. Synopsys Design Compiler tools were used for all ASIC tests. In order to get a good estimate of the best achievable timing while keeping the synthesis run time reasonable, we set the synthesis effort to "medium" and defined the required minimum clock period in a compile script file. The results of our experiments are shown in Table I. The n column denotes the matrix dimension, "Area Utilization" describes an estimation of the final size, while "Frequency" describes an estimation of the maximum frequency. Both previous columns are divided into three technology size: 130 nm, 110 nm, and 55 nm (low power).

The number of clock cycles needed for the elimination process can be calculated from (1), where n stands for a matrix size and z a number of bits per word, which is also the bit length of the modulus and the data path width. For our comparisons and analyses, we chose the word size $z = 24$ and the number of words per element $q = 3$, i.e. input integer element length is 72 bits, as a reasonable compromise regarding the range of input data values. Each input integer (matrix element) is first reduced using a 24-bit modulus and all internal computation is carried out in a 24-bit modular arithmetic.

$$\mathrm{elim}(n, z) = ((z + (4z - 2))n + 3)n + 14. \quad (1)$$

To calculate the elimination time (T_{elim}), we assume that the data is already loaded and stored in memory and the elimination process is in run. The load part takes only a small part of the SLC solution time (about 10 % for the instances considered). The elimination time on a 130 nm ASIC takes 3 ms for an $n = 100$ matrix and 380 ms for an $n = 1000$ one.

The maximum achieved frequency of the design based on matrix dimension n is shown in Fig. 3. The results show that the performance strongly depends on the type of ASIC library used. For small designs, the 55 nm library yields the best speed and the smallest area. However for $n = 300$ and larger, the 55 nm library is the slowest even though it would seem that due to its smaller pitch it should be faster. This is due to the fact that it is a low power library not optimized for performance.

Interesting effect was observed when comparing the maximum frequency between the 130 nm and 110 nm libraries for $n \geq 2000$, where the smaller technology is slower. This

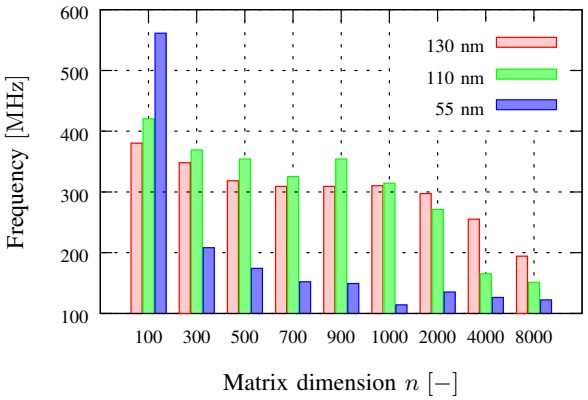

Fig. 3. Achieved frequency of the design based on matrix dimension n.

TABLE I
Synthesis results for the ASIC residual processor architecture.

n	Area Utilization (mm^2)			Frequency (MHz)		
	130 nm	110 nm	55 nm	130 nm	110 nm	55 nm
500	32	42	9	318	354	174
1000	102	117	28	310	314	114
2000	342	367	97	297	271	135
4000	1221	1260	353	255	165	151

is caused by different net delay models and different choice in cell sizes among the two libraries. Further investigation would require analysis at the layout level, however the library backends are not readily available for both libraries.

The same task was implemented on a CPU solving an SLC of dimensions 100, 500, and 1000, it takes approximately 3 ms, 424 ms, and 3.37 s, respectively calculated on Intel T9400 CPU running at 2.53 GHz with a 6144 KiB cache [17]. This shows that for $n = 1000$, our design is approximately 5 times faster.

The results show that our residual processor architecture allows for a maximum reasonable matrix size of approximately 2000 rows by 2001 columns with a word size of 24 bits in the chosen ASIC type. With larger maximum sizes, the frequency drops considerably as a result of high fan-in and fan-out of the array of AUs, and the die size grows as a result of on-chip memory required. The memory area consumption can be addressed either using on-chip dynamic RAM (which was not available in the library design kit), or better yet, by using off-chip external memory for the matrix during the elimination process. The latter option is suitable for significantly larger matrices and requires significant changes in the design.

V. Conclusion

The paper describes an implementation of a solver of systems of linear congruences in ASIC, part of a parallel system for solution of systems of linear equations. We modified the previous FPGA design for ASIC and compared three types of standard cells libraries: 130 nm, 110 nm, and 55 nm. The first two technologies were high speed, while the last one was low power. These implementations provide insight into the properties of the design in higher density technologies.

The occupied area and speed were gathered from synthesis reports. The most significant part was the memory used to store the data for calculation. The time needed to solve one SLC with 4000 congruences was 15 s and the occupied area was $3.5\,\mathrm{cm}^2$ of die. Considering a suitable die size around $1\ \mathrm{cm}^2$, the maximum matrix dimension is 1000 for the 130 and 110 nm technologies, and 2000 for the 55 nm low power technology. The 55 nm technology allowed us to use the same die space for solution of a much larger SLC, but since the technology is low power, the solution time is significantly slower than with the other two technologies.

VI. Future work

The implemented linear congruence solver, a Residual Processor (RP), will be used as a part of the Modular System (MS) for solution of sets of linear equations. Time and area complexity results of the implemented RP obtained in the paper have already been used to extrapolate properties of an MS yet to be synthesized in higher density technologies [18].

In future work we intend to focus on RP's external memory and optimization of the arithmetic unit utilization. Using external memory requires design of a memory interface, possibly limiting data throughput during the parallel read/write operations, and requires further research. The architecture with an external memory also offers opportunity to a more efficient utilization of arithmetic units during the elimination process through efficient memory interface control.

Acknowledgment

This research was supported by the Czech Science Foundation project no. P103/12/2377.

References

[1] IEEE Computer Society Standards Committee., *IEEE Standard for Floating-Point Arithmetic*, ser. ANSI/IEEE STD 754-2008. The Institute of Electrical and Electronics Engineers, Inc., 2008.

[2] A. Omondi and B. Premkumar, *Residue Number Systems: Theory and Implementation*, 1st ed. Imperial College Press, 2007, vol. 2.

[3] M. Lu, *Arithmetic and Logic in Computer Systems*. John Wiley & Sons, Inc., 2004.

[4] Ç. K. Koç, "A parallel algorithm for exact solution of linear equations via congruence technique," *Computers & Mathematics with Applications*, vol. 23, no. 12, pp. 13–24, 1992.

[5] M. Morháč and R. Lórencz, "A modular system for solving linear equations exactly, i. architecture and numerical algorithms," *Computers and Artificial Intelligence*, vol. 11, no. 4, pp. 351–361, 1992.

[6] R. Lórencz and M. Morháč, "A modular system for solving linear equations exactly, ii. hardware realization," *Computers and Artificial Intelligence*, vol. 11, no. 5, pp. 497–507, 1992.

[7] J. Buček, P. Kubalík, R. Lórencz, and T. Zahradnický, "Dedicated Hardware Implementation of a Linear Congruence Solver in FPGA," in *The 19th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2012*. Monterey: IEEE Circuits and Systems Society, 2012, pp. 689–692.

[8] J. Buček, P. Kubalík, R. Lórencz, and T. Zahradnický, "Comparison of FPGA and ASIC Implementation of a Linear Congruence Solver," in *Digital System Design (DSD), 2013 16th Euromicro Conference on*, 2013.

[9] D. Taleshmekaeil and A. Mousavi, "The use of residue number system for improving the digital image processing," in *Signal Processing (ICSP), 2010 IEEE 10th International Conference on*, oct. 2010, pp. 775–780.

[10] D. Younes and P. Steffan, "Efficient image processing application using residue number system," in *Mixed Design of Integrated Circuits and Systems (MIXDES), 2013 Proceedings of the 20th International Conference*. The Institute of Electrical and Electronics Engineers, Inc., 7 2013.

[11] G. Cardarilli, A. Nannarelli, and M. Re, "Residue number system for low-power DSP applications," in *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*, nov. 2007, pp. 1412–1416.

[12] R. Chaves and L. Sousa, "RDSP: a RISC DSP based on residue number system," in *Digital System Design, 2003. Proceedings. Euromicro Symposium on*, sept. 2003, pp. 128 – 135.

[13] A. Mirshekari and M. Mosleh, "Hardware implementation of a fast FIR filter with residue number system," in *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, vol. 2, may 2010, pp. 312 –315.

[14] J.-C. Bajard and L. Imbert, "A full RNS implementation of RSA," *Computers, IEEE Transactions on*, vol. 53, no. 6, pp. 769–774, 7 2004.

[15] D. Schinianakis, A. Kakarountas, and T. Stouraitis, "A new approach to elliptic curve cryptography: an RNS architecture," in *Electrotechnical Conference, 2006. MELECON 2006. IEEE Mediterranean*, may 2006, pp. 1241–1245.

[16] T. Güneysu and C. Paar, "Ultra high performance ECC over NIST primes on commercial FPGAs," in *Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems*, ser. CHES '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 62–78. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-85053-3-5

[17] L. Vondra, "System for solving linear equation systems," Dissertation thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, 2014. [Online]. Available: http://hdl.handle.net/10467/20222

[18] J. Buček, P. Kubalík, R. Lórencz, and T. Zahradnický, "System on chip design of a linear system solver," in *International Symposium on System-on-Chip (SoC)*, 2014 (accepted).