Comparison of FPGA and ASIC Implementation of a Linear Congruence Solver

Jiří Buček, Pavel Kubalík, Róbert Lórencz, 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*—Residual processor (RP) is a dedicated hardware for solution of sets of linear congruences. RPs are parts of a larger modular system for error-free solution of linear equations in residue arithmetic. We present new FPGA and ASIC RP implementations, focusing mainly on their memory units being a bottleneck of the calculation and therefore determining the efficiency of the system. First, we choose an FPGA to easily test the functionality of our implementation, then we do the same in ASIC, and finally we compare both implementations together. The experimental FPGA results are obtained for Xilinx Virtex 6, while the ASIC results are obtained from Synopsys tools with a 130 nm standard cell library. Results also present a maximum matrix dimension fitting directly into the FPGA and achieved speed as a function of the dimension.**

***Keywords*-system of linear equations; residue number system; error-free computation; FPGA; ASIC**

I. Introduction

Solving systems of linear equations is one of the most frequent tasks in scientific computation. Traditionally, solution of such systems is carried out in floating point arithmetic bringing its associated rounding errors. Although there are algorithms minimizing the impact of rounding errors upon the solution, in some cases this does not need to be enough. An error-free solution of linear systems is often needed in case of large, dense, and possibly ill-conditioned systems, where needless rounding errors can result in longer run times, or even swamp the solution completely. One of the methods to go around rounding errors is to perform an error-free computation by means of residue arithmetic — Residue Number System (RNS).

RNS permits us to represent long integers as independent combinations of small integers based on the Chinese Remainder Theorem. It requires a simple arithmetic unit without any rounding and normalization logic that would be needed for floating point calculation. These properties offer natural parallelism, lead to a simpler hardware, and reduce chip size when compared to a traditional floating point unit implemented in hardware.

RNS is used in areas of digital image processing [1], digital signal processing [2], and in public-key [3] and elliptic curve [4] cryptography. RNS is also used to simulate multiple precision arithmetic and for error-free solution of linear systems [5].

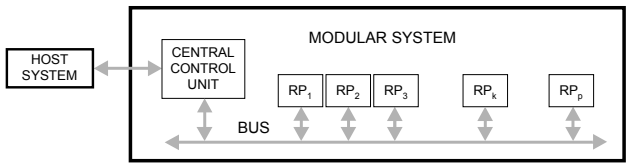

Figure 1. Architecture of the Modular System [7]

Our previous work [6] presents an FPGA implementation of a residual processor (RP) being a dedicated hardware for solution of sets of linear congruences. In this paper, we implement RP in ASIC and compare the achieved time and area complexity to its FPGS implementation.

Papers [7] and [8] design a hardware RNS linear equation solver — Modular System (MS) — whose implementation was very difficult at that time. With current technologies, it is possible to implement the system, either using FPGAs offering a straightforward implementation with reconfiguration possibilities, or in ASIC to achieve higher speeds and possibly a lower price, when produced in larger quantities.

After an extensive redesign in [6] to use block RAM cells found in most recent FPGAs, we implemented the architecture in ASIC using a 130 nm technology with standard cells and compiled memory modules. We also redesigned the addressing and pivoting logic to support efficient implementation of the elimination algorithm used.

After a brief introduction of the MS architecture performing solution of sets of linear equations (SLE)s $\mathbf{Ax} = \mathbf{b}$, the paper focuses on the memory architecture of RPs inside MS. Next, there follow FPGA and ASIC implementation results for various problem sizes, their analyses, and evaluations. Finally, the paper is concluded with FPGA and ASIC RP implementation properties.

II. Architecture of the Modular System

Paper [8] describes the method, the algorithm, and the corresponding parallel hardware architecture of the MS (Fig. 1). Evaluation in each modulus is performed independently of the others with the addition carry-free, subtraction borrow-free across the individual moduli, and thus the computation can occur safely in parallel. Once the computation is done, the result is transformed back into the rational number set either with the Chinese Remainder Theorem or the Mixed Radix Conversion.

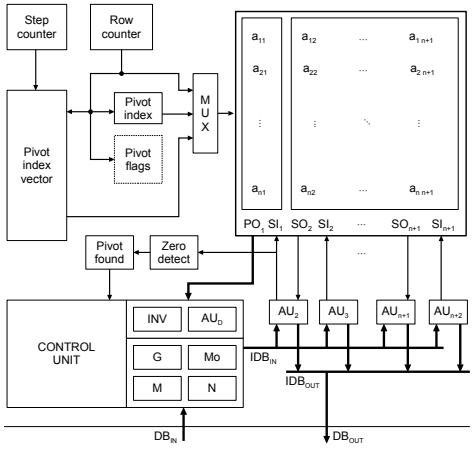

Figure 2. Architecture of the Residual Processor

The error-free solution of an SLE with operations performed in residue arithmetic is implemented in this special MS. The MS typically has a parallel SIMD architecture consisting of a control unit and several processing units — RPs denoted as RP_1, RP_2, …, RP_p interconnected with a BUS (see Fig. 1).

III. Residual Processor Architecture

The architecture of the residual processor RP_k 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 residues of matrix $\mathbf{A}$ and vector $\mathbf{x}$ elements. The storage of values of a row of the matrix from AU registers is performed bitwise via Serial Inputs SI_1, SI_2, …, SI_{n+1}. Loading of values of rows of Memory to AUs is done via Serial Outputs SO_2, SO_3, …, SO_{n+1}. The bits of element values of the first matrix column 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 above RP_k architecture can solve systems of linear congruences (SLC)s $\mathbf{A}\mathbf{x}_k \equiv \mathbf{b} \pmod{m_k}$. RPs together with the Control unit of the MS also support all conversion operations from integer to RNS and vice versa. The INV and DET units compute the modular multiplicative inverse and the determinant of $\mathbf{A}$, respectively.

All SLCs in MS are solved with the Gauss-Jordan elimination with Rutishauser modification [8] (GJR), which is especially suitable for hardware implementation. The elimination process in RNS is specific in a way that it has to perform a so called "nonzero residue pivoting" that was introduced in [7]. Pivoting and massive data access constitute a bottleneck, and therefore the memory architecture design is critical and is dealt with in the next section.

The Rutishauser modification of Gauss-Jordan elimination implies that the column data is shifted by one column to the left during each elimination step. The shift is accomplished by the AUs and the memory interconnection design in Fig. 2. In addition, the first column of the SLC matrix contains values of the elements intensively used during the elimination process and for this reason the output from the first column needs to be parallel (these values are used in the INV and DET units). The values in the first column determine the first multiplication operand in the entire row being processed, both in pivot elimination and row reduction. The other columns a_{i2} to a_{in+1} inclusive are used as the second multiplication operand, and also for addition operations. Assuming serial-parallel (shift-add) multiplication, we need to read individual bits of these values, thus requiring serial access only.

A. Memory Architecture in FPGA and ASIC

In order for a value to be accepted as a pivot, i) it must be nonzero, ii) the row has not contained a pivot yet, and iii) no pivot has yet been found for this elimination step. The pivot is always in the first column of the matrix. Search for a pivot is done sequentially; however, this search can be easily performed concurrently with write operations to the memory. The search is performed while the matrix is loaded or updated during computation. In most cases, the pivot is passed to the inversion unit (INV) long before its inverse is needed.

Once the pivot is found, its row index must be stored in a pivot index vector at the address of the current elimination step. The pivot row must be flagged in order to skip it during the pivot search performed in subsequent elimination steps. If no pivot was found, the matrix is singular in this modulus.

The elimination is performed by rows. The architecture must support addressing of the pivot row first; then sequentially reduce memory matrix rows, with an exception of the pivot row which must be skipped. The first value in each row must be read in parallel. This value is either the pivot, which is inverted, or a value from a different row, which is negated.

The remaining values in each row are read bit-serial (but all values concurrently) from the MSb first. This ensures the correct order for left-shift modular multiplication and addition, and follows from the design depicted at Fig. 2.

Upon completion of the elimination process, the solution vector appears in the first column. The order of its elements corresponds to the pivot row indices and may need to be reordered. The result is therefore read out in correct order by addressing through the pivot index vector.

B. Arithmetic Units, Modular Inverse and Controller

The Arithmetic Units AU_2, AU_3, …, AU_{n+2} and $\mathrm{AU_D}$ design is modified from the original circuit in [8] by using strictly synchronous design. It supports computation of modulo operation on multi-word inputs, which is used when loading a new matrix into the modular system. During elimination, it computes modular multiplication and addition operations. Multiplication operations are performed using a shift-add algorithm with interleaved modulus subtraction.

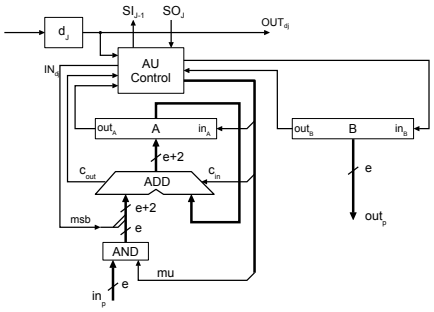

Figure 3. Arithmetic Unit architecture

The block diagram of the arithmetic unit is given in Fig. 3. The main part of the unit is the adder that performs all addition and subtraction operations including elementary additions during multiplication. The intermediate results are stored in the A register. The B register serves as a temporary storage for the values of the multiplied pivot row to be used during row reduction.

For computing the modular inverse in the INV unit, the left-shift modular inverse algorithm [9] is used.

The controller contains a finite state machine using a memory-based transition and output functions. This allows flexibility with regard to modification and future extensions.

C. FPGA and ASIC Implementation

The memory architecture as a critical part of the RP can be divided into two parts: the pivoting unit and data memory containing the matrix. The pivoting unit is always implemented using on-chip logic and memory blocks in FPGA or ASIC. The data memory can be implemented internally using block RAM components, or externally e.g. by a DDR SDRAM. The main implementation differences are in their parameters such as memory capacity, throughput, and latency. On one hand, the internal implementation with FPGA memory has small capacity and low latency, while on the other hand the external memory provides large capacity but also a high latency.

In order to compare the implementations in FPGA and ASIC, we used internal memory in both cases. For a given FPGA type, we can estimate the size of the largest matrix that fits on chip according to maximum size of the block RAM. In ASIC, we do not have such hard limits, but we can observe the occupied chip area as a function of the internal memory capacity. In both cases, we analyze the maximum frequency of the design after implementation (or, in the case of ASIC, after synthesis). Our tested memory architecture consists of two parts: i) the internal memory, and ii) a pivoting control logic to support addressing during the calculation in the RP. The design of our memory architecture is shown in Fig. 2.

The memory matrix consists of the first column a_{i1} and the remaining columns a_{i2} to $a_{i,n+1}$. All columns share a common address. During pivot search, the address is taken from the Row counter and if the pivot is found in the current row, the address is written into the Pivot index vector at the address of current elimination step, and the Pivot flag for the address of the current row is set. At the same time, the pivot address is stored in the Pivot index register for comparison during the next elimination step.

Table I
IMPLEMENTATION RESULTS FOR THE FPGA RESIDUAL PROCESSOR ARCHITECTURE (FPGA IS XILINX XC6VSX475T).

n	Area Utilization					Time
	BRAM	Mem [sl]	Au [sl]	Cntl [sl]	**All [sl]**	**All [MHz]**
100	103	480	2889	77	**4223**	**166**
300	303	1269	8920	87	**12117**	**142**
700	701	2821	21438	73	**29560**	**79**
1000	1001	4068	30095	73	**42368**	**76**

IV. EXPERIMENTAL RESULTS

All experiments were conducted on an RP architecture consisting of data memory, Pivot index, Pivot flags, counters, arithmetic units, an inversion unit, and control units. The final design was written in VHDL. The maximum matrix dimension n and the word length e are configurable at synthesis time using generics, while matrix dimension n, modulus, and the matrix data are set at runtime.

The design was simulated, synthesized and implemented (only for FPGA) (mapped, placed, and routed). The experiments were performed on 1 RP (with a single modulus), including the input data modulo reduction and matrix elimination. A transformation into the rational numbers set was not performed. The experiments were performed separately for FPGA and ASIC.

After verification of a simulation correctness we started an implementation in FPGA. The Xilinx FPGA platform was selected for all FPGA tests in Xilinx ISE. We selected the FPGA with the highest block RAM memory capacity in the Virtex 6 family i.e xc6vsx475t-2-ff1156. The results of our experiments are shown in Table I.

The n column denotes the matrix dimension. In the area utilization part the "All" and "BRAM" columns are the number of slices and Block RAMs for whole residual processor implementation. The "Mem", "Au" and "Control" columns show the number of slices used only for the memory, AUs and for the controller. The "Time" column contains the maximum frequency obtained from the implementation.

For ASIC we performed only the synthesis. Since block RAMs were not in the ASIC library as standard cells, we generated them with a special memory generator tool that comes with the library.

Synopsys Design Compiler was used for all ASIC tests with a 130 nm technology library. The results of our experiments are shown in Table II. The table headings are similar to Table I, but area figures are given in mm^2. The maximum frequency is obtained from synthesis.

Table II
SYNTHESIS RESULTS FOR THE ASIC RESIDUAL PROCESSOR ARCHITECTURE (130 NM LIBRARY).

n	Area Utilization				Time
	Mem [mm^2]	Au [mm^2]	Control [sl]	**All [mm^2]**	**All [MHz]**
100	2,22	0,82	0,015	**3,08**	**380**
300	14,02	2,22	0,016	**16,57**	**348**
700	52,41	5,09	0,017	**58,16**	**309**
1000	93,53	7,34	0,017	**101,74**	**310**

The number of clock cycles needed for load and elimination process can be calculated by (1) and (2), where n is matrix size, with z bits per word and q input words.

$$\mathrm{load}(n,z,q) = ((((2(n+1))+(1+2+2z))q)+7)n+1, \quad (1)$$

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

To calculate the elimination time (T_{elim}) we assume, that the data are already loaded and stored in memory and elimination process is in run. The load part takes only a small part of all entire SLC solution time. For example, for an $n = 100$ matrix, the loading process takes only 12.7 %, while for $n = 1000$ it takes only 11.7 %. As we can calculate, the elimination time for a 100 column matrix is 7 ms for FPGA and 3 ms for ASIC. For a 1000 column matrix, the elimination time is 1552 ms for FPGA and 380 ms for ASIC.

When we run the same task 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 KB cache. 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 matrix size of approximately 1000 rows by 1001 columns with a word size of 24 bits in the chosen FPGA type. Even with the maximum tested matrix dimension of 1000, which uses more than 90 % of the available block RAM, only approximately 60 % of all FPGA available slices are used.

The clock period increases with an increasing matrix dimension. Static time analysis shows that the main parts of the delay in the circuit are in addressing, control and inner data bus signals. The fanout of signals significantly increases when the size of matrix increases.

V. CONCLUSION

We have designed a Residual Processor (RP) architecture performing a solution of a set of linear congruences. RP was designed with a focus on its effective implementation in FPGA and ASIC for various problem sizes with a special attention to the memory architecture.

All important parts of the RP architecture were implemented and tested in Xilinx Virtex 6 FPGA with the largest RAM size available. The ASIC synthesis process was performed to compare our solution with both types of implementation. The results indicate that our RP architecture allows for a maximum matrix size of approximately 1000 rows by 1001 columns with a 24-bit word size in the chosen FPGA type. The maximum tested matrix dimension of 1000 uses more than 90 % of the available block RAM and approximately 60 % of all available slices in the FPGA while being approximately 5 times faster than a software implementation running on a CPU. In a case, when we use ASIC with a 130 nm technology, the elimination time is 4 times faster than in FPGA for a 1000 rows matrix.

Future work will focus on a new RP architecture with an external memory and a limited number of AUs.

ACKNOWLEDGMENT

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

REFERENCES

[1] 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.

[2] 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.

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

[4] 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

[5] R. T. Gregory and E. V. Krishnamurthy, *Methods and Application of Error-free Computation*. Springer Verlag, 1984.

[6] 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.

[7] R. Lórencz and M. Morháč, "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.

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

[9] R. Lórencz, "New Algorithm for Classical Modular Inverse," in *Cryptographic Hardware and Embedded Systems CHES 2002*. New York: Springer, 2002, pp. 57–70.