System on Chip Design of a Linear System 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*—This paper is focused on hardware error-free solution of dense linear systems using residual arithmetic on a System on Chip Modular System. The designed Modular System uses Residual Processors (RP)s for solving independent linear systems in residue arithmetic and combines RP solutions into solution of the linear system. In order to efficiently exploit parallel processing and cooperation of the individual components, a System on Chip architecture of the Modular System with several RPs is designed, each with a large memory unit used for data transfer and storage. A Xilinx FPGA architecture with a MicroBlaze processor is used to verify the proposed architecture. The experimental results are obtained for an evaluation FPGA board with Virtex 6 and a 1 GiB DDR memory and serve for further theoretical analysis of the system performance for various linear system sizes and the architecture of the system. The proposed system can be useful as a special hardware peripheral or a part of an embedded system.**

***Index Terms*—system of linear equations; system of linear congruences; residue number system; error-free computation; FPGA; System on Chip**

I. Introduction and Method

Solving a regular system of linear equations (SLE) is a common task in numerical mathematics and its difficulty depends on many aspects. The input is an augmented matrix of the linear system, while the output is a solution vector. The difficulty of the solution process depends on many aspects, such as an input matrix dimension, its density, conditioning, accuracy requirements, and properties of the matrix. Solution is performed using different numerical methods and algorithms on all kinds of computational resources including PCs, GPUs, clusters, specialized hardware, and even supercomputers.

Solution occurs frequently in a floating-point arithmetic comitting errors and thus there should be a question of numerical stability. Using floating-point (FP) arithmetic, as defined in the IEEE 754:2008 Standard [?], requires the result of each immediate operation rounded to a representable FP number, committing a roundoff error. Input error magnification and the accumulation of rounding errors committed during solution may even destroy the result. For this reason, SLE solution shall always induce a question of numerical stability, esp. in case of a large, dense, and/or ill-conditioned system.

Non-rounding arithmetics can go around rounding problems and such SLE solution processes were already implemented in special hardware. To avoid undesired rounding effects, it is possible to use a non-rounding arithmetic such as the arithmetic of the Residue Number System (RNS) [?], which, in addition to non-rounding, offers natural parallelism. Parallel RNS-SLE solution algorithms were proposed in [?][?][?] and are further exploited by a dedicated hardware Modular System (MS). Such MS was discussed in [?][?][?][?].

When studying SLE solution methods of in RNS arithmetic, a gap in the publication activity can be observed since the mid 1990s. This can be explained by limited technology resources available at that 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 solution is based on the fact that FP arithmetic has its limitations, and at the same time, progress in technology enables creating hardware solvers that can effectively use RNS to solve SLEs for real applications.

RNS is currently used to accelerate computation in areas such as the digital image processing [?][?], digital signal processing [?][?], and cryptography [?][?].

SLE solution in RNS is built on the Chinese Remainder Theorem (CRT), and the process proceeds in 3 stages [?]:

Method 1

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

Using Method 1 has both its pros and cons. An obvious disadvantage is the increased time and/or space complexity. An advantage is gained by exploiting the built-in parallelism, ergo lowering the time complexity by implementing the method in hardware. Although parallel processing increases spatial complexity, computation units at the 2nd stage of Method 1 are identical and provide a time-space complexity tradeoff.

The paper continues with design of a dedicated SLE solution hardware in RNS and provides important results for further development and verification. Architectures so far designed and testing of its implementation in FPGA and ASIC technologies were covered in papers [?][?] and [?]. These papers focused on design and implementation of an SLC solver as

[1]The matrix $\mathbf{A}|\mathbf{y}$ can generally be consisting of floating-point numbers or integers.

978-1-4799-6890-9/14/$31.00 ©2014 IEEE

the most important and computationally intensive part of the solution process (the 2nd and part of the 1st stage of Method 1), while this paper deals with SLC solvers composed into a System on Chip including interconnection and integration into an SLE solving system. Such system can be useful as a special hardware peripheral attached to a Host System, or a part of an embedded system requiring SLE solution. The SoC design provides a platform for analysis of a practical and functional system. The results of this analysis are important for subsequent development of the system and confirmation of the validity of the method and design process.

The paper is organized as follows: Section I (Introduction and Method) introduced the reader into the context of the paper. Section II (Previous Work) discusses so far designed components (SLC solvers) of the designed SLE solver. Section III (Architecture of Modular System SoC FPGA) describes the system architecture of the SLE solver using SLC solvers (Residual Processors) described earlier. Section IV (Implementation and Experimental Results) presents the results of an implementation of the system with multiple residual processors on FPGA with the ML605 prototyping board. Section V (Analysis of the Experimental Results) compares the implementation's performance in comparison with a model. Section VI (Perspectives of the Modular System) discusses possible extensions and improvement of the system, while Section VII (Conclusion) summarizes the paper.

II. Previous Work

SLE solution in RNS performed in dedicated hardware [?] requires that the hardware is able to perform all Method 1 stages. The architecture of such system is depicted in Fig. 1:

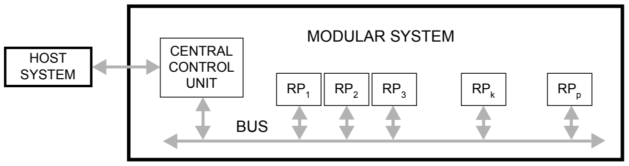

Fig. 1. The Architecture of the Modular System [?]. The Central Control Unit controls processes required by Method 1 and communicates with the Host System. RP_1 through RP_p are individual Residual Processors, each with its own modulus m_i for $i = 1 : p$. RPs support all Method 1 stages i.e. perform transformation into RNS (stage 1), perform SLC solution mod m_i (stage 2), and back-transformation from RNS (stage 3). The Bus denotes an internal interconnection of all units within the Modular System.

The Modular System (MS) consists of a Central Control Unit (CCU), p hardware identical Residual Processors (RP)s, and an interconnection Bus. CCU communicates with an external Host System such as a computer, coordinates RP's work, and dispatches data to and from RPs. Each RP works with its own distinct prime number modulus m_i and in [?] was designed to support all 3 Method 1 stages. The Bus, not necessarily implemented as a data/control bus, denotes necessary interconnection of all units within MS. The following paragraphs recapitulate so far achieved progress in papers [?][?] and [?], all dealing with an RP architecture.

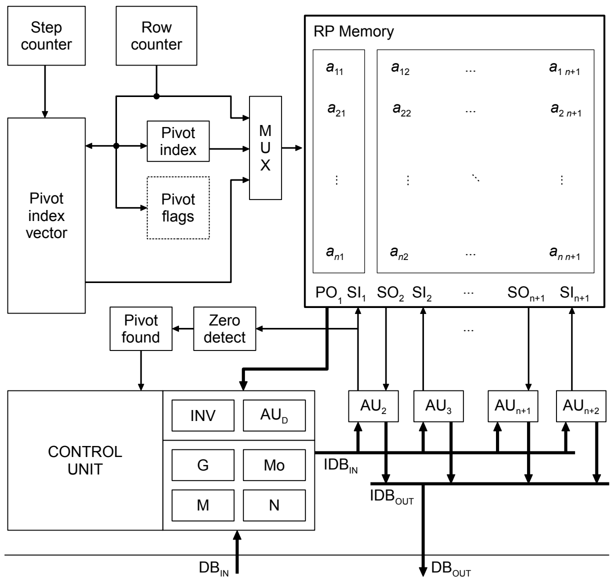

Fig. 2. The Architecture of a Residual Processor. a_{ij}s represent elements of the augmented SLC matrix, AU_i stand for individual arithmetic units, PO_1 is a parallel output, $\mathrm{SI_i}/\mathrm{SO_i}$ denote serial inputs/outputs, $\mathrm{IDB_{IN}}/\mathrm{IDB_{OUT}}$ represent internal data buses, while $\mathrm{DB_{IN}}/\mathrm{DB_{OUT}}$ represent external data buses. INV, $\mathrm{AU_D}$, G, Mo, M, and N are auxiliary units described in [?]. Pivot index vector, Pivot flags, Pivot index, Zero detect, and Pivot found registers are used to support the non-zero residual pivoting [?].

Paper [?] presents an initial RP architecture (Fig. 2) that was designed to perform a Gauss-Jordan-Rutishauser (GJR) elimination upon individual SLCs mod m_i stored within the Residual Processor Memory. RPs contain specialized Arithmetic Units (AU)s interconnected with the Residual Processor Memory. This allows performing vector (SIMD) operations corresponding to the GJR elimination, which is controlled by the Control Unit. The dedicated hardware for residual pivoting also solves the zero pivot occurrence problem. Next, there are a control unit and auxilliary units (see Fig. 2), primarily for computing a multiplicative modular inverse mod m_i (INV) and the determinant (DET) of the SLC, both needed during the back-transformation performed in stage 3 of Method 1.

The Memory, implemented in FPGA as a block RAM, 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, $\ldots$, SI_{n+1}. Loading of values of rows from Memory to AUs is done via Serial Outputs SO_2, SO_3, $\ldots$, 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:

$$\mathbf{A}\mathbf{x}_k \equiv \mathbf{b} \pmod{m_k}. \qquad (1)$$

RPs together with the Control Unit of the MS also fully support all conversion operations from the integer set to RNS and also partially back from RNS to the integer set.

The Arithmetic Units AU_2, AU_3, $\ldots$, AU_{n+2} and $\mathrm{AU_D}$ design is modified from the original circuit in [?] by using a strictly synchronous design supporting computation of the modulo operation on multi-word inputs, which is used during loading of a new matrix into MS. Modular multiplication and addition operations are calculated during elimination. Multiplication operations are carried out using a shift-add algorithm with interleaved modulus subtraction, while the modular inverse is calculated with the left-shift algorithm [?].

The Control Unit contains a finite state machine using a memory-based transition and output functions. It implements a GJR elimination algorithm as well as data input and output and allows flexibility with regard to modification and extensions.

The proposed architecture was implemented in FPGA, tested for various matrix dimensions, and the amount of used block RAM and speed were evaluated. The architecture was implemented as a post place and route model in FPGA for various SLC dimensions up to $n = 1000$ and with 24-bit m_i. Implementation results performed on Xilinx Virtex 6 FPGA identified a bottleneck in the memory subsystem because of massive data accesses and pivoting. The maximum $n = 1000$ sized matrix took approximately $90\,\%$ of the available block RAM (BRAM) and approximately $60\,\%$ of all available slices. The implementation in FPGA was approximately 2 times faster than its GCC-compiled software counterpart running on an Intel T9400 CPU running at $2.53\,\mathrm{GHz}$.

The FPGA implementation was reimplemented in ASIC and the memory size and speed were again compared. Paper [?] focuses on the aforementioned bottleneck, implements the design in ASIC, and compares it to the FPGA implementation. The fact that the design was implemented in ASIC allowed a larger memory subsystem than it was possible in FPGA. In ASIC, the memory blocks are composed of generated static RAM blocks. The memory size is determined by a generic parameter that is used to generate the corresponding memory blocks. RP design for SLCs with $n > 1000$ produces a chip with an area larger than $100\,\mathrm{mm}^2$ in a $130\,\mathrm{nm}$ ASIC technology using static RAMs. Both designs were compared and in the $n = 1000$ case the ASIC implementation was about 4 times faster than its FPGA analogue.

Next, three ASIC implementations in different technologies were compared and evaluated. Paper [?] describes an ASIC RP implementation in three different standard cell libraries and compares them together. The architecture remained the same and the libraries were $130\,\mathrm{nm}$, $110\,\mathrm{nm}$ high-speed, and $55\,\mathrm{nm}$ low-power. The maximum dimension that fitted into an area of about $1\,\mathrm{cm}^2$ was an RP with $n = 1000$ in the $130\,\mathrm{nm}$ and $110\,\mathrm{nm}$ technologies, while $n = 2000$ in the $55\,\mathrm{nm}$ technology.

The purpose of this paper is to integrate Residual Processors with the Central Control Unit and design the communication of the MS with the Host System.

III. Architecture of Modular System SoC FPGA

This section deals with design of an interconnection of the units within MS. The architecture, which is depicted at Fig. 3, is designed to allow interconnection of RPs with the Host System (HS), Main Memory, and other units to perform Stages 1 and 2 of Method 1.

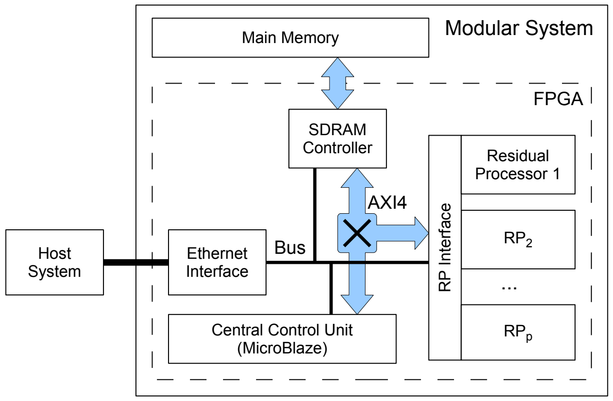

Fig. 3. The Modular System as a System on Chip in FPGA. The Central Control Unit is a MicroBlaze soft-processor. RP_1 through RP_p are individual Residual Processors, each with its own modulus m_1 to m_p. RPs are connected using an internal bus and form a common AXI4 bus master peripheral. The AXI4 crossbar switch is used for a high speed interconnect, while the Bus is used for a lower speed programmed I/O.

In Stage 1, the augmented matrix is sent to the MS from the Host System using the Ethernet Interface. It provides a good flexibility and is supported in FPGA development systems by several performance options. The matrix in form of multi-word integer numbers is transferred and stored into the Main Memory and is therefore prepared for loading and (possibly parallel) conversion to modulus m_i. Main Memory will hold the augmented matrix of the SLE to be solved and is connected to the FPGA with a (DDR3) SDRAM Controller supporting burst transfers in between the memory and other parts of the system. A high-throughput channel is created using a high performance interconnect (AXI4 in crossbar configuration) between the SDRAM Controller and a set of RPs. RPs share a common data channel from Main Memory since they always get the same data. During loading of a matrix, each RP applies the modulo operation on the data with its modulus m_i. The set of RPs is connected using a common bus-master AXI4 peripheral. The system contains a Xilinx MicroBlaze processor as the Central Control Unit (CCU) for overall communication, data loading, and synchronization control. The MicroBlaze processor was chosen as a convenient way to create a prototype implementation including network protocol handling. Although MicroBlaze is not a final solution, it is sufficient for performing experiments and evaluation of basic properties of the proposed system.

The processor transfers the data over the Ethernet Interface to RP memories and once loaded, they automatically start solving the SLCs. The processor uses a network protocol for communication and data transfer from the Ethernet Inteface to the memory. After receiving the matrix data, it starts the process of loading this data from memory to the RPs. This is done by initiating a bus master read by the RP peripheral from a specified address in memory. After loading and converting all the data (Stage 1 of Method 1), the RPs automatically start solving their SLCs independently (Stage 2). The CCU can either wait for the RPs to complete or perform other tasks such as communication to load another instance from the HS.

The SLE solving process according to Method 1 can be implemented on the architecture in Fig. 3 using two algorithms synchronized by data communication over the Ethernet Interface (Eth). The algorithms, Algorithm 1 for the Host System, and Algorithm 2 for the Modular System, follow here:

ALGORITHM 1. **Host System – HS**.

1) Send the SLE augmented matrix to the MS over Eth
2) Wait until MS finishes computing the results
3) Receive the results from MS over Eth
4) Perform a partial back conversion
5) Enough moduli for a whole back conversion? ⇒ END.
6) Goto 2

ALGORITHM 2. **Modular System – MS**.

1) Receive the SLE aug. matrix from the HS over Eth
2) Load the matrix in all available RPs
3) Perform elimination on all available RPs
4) Read the results
5) Send the results to the HS over Eth
6) Load the matrix in all available RPs with new moduli
7) Goto 3

The system is now ready to perform experiments and evaluate the experimental results.

IV. Implementation and Experimental Results

The designed architecture of the MS (see Fig. 3) was implemented in FPGA and verified by simulation and hardware prototyping. The implementation and testing platform used for the development was a Xilinx ML605 board with a Virtex-6 LX240T FPGA, 1 GiB DDR3 SDRAM memory, gigabit Ethernet interface, and other peripherals. The MS is controlled using a MicroBlaze soft-processor inside the FPGA.

The MicroBlaze processor controls the transfer of data between Main Memory and the RPs. External DDR3 SODIMM was the Main Memory used to store the augmented matrix of the SLE, and the processor program code.

The hardware (RPs, the RP bus-master and slave interface) was described in VHDL, while the software running on MicroBlaze was written in C. The MicroBlaze processor is an IP core included in the development tool. Xilinx ISE and Embedded Development Kit (EDK) development tools were used to describe, synthesize, and implement the SoC architecture into FPGA.

The selected FPGA platform allowed for a system with $p = 1, 2, \ldots, 5$ RP units and matrix dimensions $n = 20, 100, 200$. Each matrix element is $q = 3$ words long, where each word is $z = 24$ bits long. Each modulus m_i is also 24-bit all internal computation is done in a 24-bit modular arithmetic, and the following text assumes these values. Parameters p, n, q, z are configurable at synthesis time. The maximum attainable matrix dimension n depends on the number of RPs p implemented on one FPGA. For up to $n = 200$, we could attain the default clock frequency of the MicroBlaze processor of 100 MHz.

The RP peripheral attachment to the AXI4 crossbar was created using a bus master template from Xilinx EDK. The RPs are connected to the rest of the system via a FIFO. The networking code for the MicroBlaze processor was based on a TCP echo server example contained in the EDK and was extended to enable data transfer in between the Main Memory and an RP peripheral. Time measurement was done using a dedicated timer peripheral and reported using a serial interface.

The testing data for our experiments were generated using Wolfram Mathematica. We have generated several SLE instances together with their solutions for verification. Data were transfered between PC and the tested system over Ethernet using the `ncat`[2] tool.

The times needed for processing of individual steps, mainly for load, elimination, and result read are collected. The resulting measurements are presented in Table I. The column **Load** describes time needed for loading data from the Main Memory to all RP units including reduction modulo m_i, **Elim** describes time needed for computing the solution vector modulo $m_1, \ldots, m_p$ by GJR elimination in all p RPs, and **Read** denotes time needed to retreive the results form the corresponding number of RPs into the Main Memory. Loading and elimination are performed with all RPs in parallel, while the read operation is performed sequentially. Table I shows that most of the time is spent in elimination. The chosen platform (Xilinx Virtex-6 LX240T) allowed us to implement a maximum of 2 RPs for $n = 200$.

TABLE I
Load, elimination, and read times for the MS architecture with multiple RPs.

n	Load [ms]	Elim [ms]	Read [ms] for a number of RPs				
			1	2	3	4	5
20	0.13	0.51	0.015	0.025	0.036	0.047	0.058
100	2.18	12.60	0.054	0.104	0.154	0.204	0.254
200	7.46	50.41	0.103	0.202	-	-	-

The gathered performance data from a real implementation are important for evaluation of the whole modular system operation as it reflects the real behavior including external parts (e.g. the DDR3 memory). The measured time and area complexity are used for subsequent analysis, which is the topic of the next section.

V. Analysis of the Experimental Results

The number of clock cycles needed for load and elimination process can be calculated by the equations (2) and (3) which were presented in [**?**]. These equations express the behavior of a model of the RP without accounting for its surroundings, i.e. interface to other parts of the MS.

$$\mathrm{load}(n,\, z,\, q) = ((2n + 2z + 5)q + 7)n + 1, \tag{2}$$
$$\mathrm{elim}(n,\, z) = ((5z - 2)n + 3)n + 14. \tag{3}$$

The $\mathrm{load}(n,\, z,\, q)$ equation describes the number of cycles needed to accept the matrix of n by $n{+}1$ elements, each having

[2]Ncat 5.51 (http://nmap.org/ncat)

q words z bits long. It includes the cycles needed to reduce the matrix modulo a z-bit modulus, and storing the matrix in the RP's internal Memory. The $\mathrm{elim}(n,\, z)$ equation describes the number of cycles an RP needs for the GJR elimination to compute the solution vector, assuming the data already loaded and stored in the internal Memory.

TABLE II
LOAD AND ELIMINATION TIME FOR OUR MS ARCHITECTURE.

n	Estimated			Measured		
	Load [ms]	Elim [ms]	Load [%]	Load [ms]	Elim [ms]	Load [%]
20	0.06	0.47	11.3	0.13	0.51	21.2
100	0.77	11.8	6.1	2.18	12.6	14.7
200	2.73	47.2	5.5	7.46	50.4	12.9

The experimental measured times were compared the with the theoretical estimation form equations (2) and (3). The clock period was assumed to be 10 ns (corresponding with the implementation). The comparison is shown in Table II indicating a match of the measured elimination time with the prediction. The elimination process is largely independent of the control by the MicroBlaze Central Control Unit and the time measurement was burdened only with a small error caused by reading state of an external timer.

The measured load times however did not match the prediction. Load times were ≈ 2.7 times longer than the estimation. Load times were first estimated with an assumption that all data were available when needed. In the real system, control and synchronization overhead was added to the loading times, causing this load time growth. Even though data were read from Main Memory using AXI4 master read transfers, it was necessary to read them in blocks (of 32 words) under MicroBlaze software control. When loading data, the RP(s) compute remainders of loaded data modulo the assigned modulus. Thus some form of throttling or flow control had to be implemented and in our case, this was done in software. The associated overhead could be decreased by improving the master read data throttling without the need for software control.

Comparing the predicted results from analyzes with measured values (c.f. Tab. II) provided a good match of predicted values with the measured values for the elimination. Such match was not found for the load part, which was predicted by equation (2), not involving a communication and synchronization overhead. The load overhead was proportional to the predicted values and therefore can be extrapolated for larger instances. Even with the current overhead, the load portion of the time was small compared to the elimination.

VI. Perspectives of the Modular System

In this section, a performance analysis of a technically realistic hypothetical Modular System configuration is presented. The experimental data gathered in the previous sections will be used to evaluate performance of the overall projected system.

The number of z-bit moduli can be derived from Hadamard's inequality. Let us have a set of k distinct prime number moduli $\beta = \{m_1,\, m_2,\, \ldots,\, m_p, \ldots,\, m_k\}$. The product of moduli $M = \prod m_i$ needed to express the solution follows from the Hadamard's inequality [**?**] giving an upper bound of the maximum possible absolute value of a matrix determinant. From this bound, a number of z-bit prime moduli needed to express the solution can be derived. The number of moduli k is generally higher than the number of RPs p in MS, and thus it is necessary to process k by p moduli in parallel.

It is assumed a modulus size $z = 24$ bits, input integer size $qz = 72$ bits ($q = 3$), SLE size of $n = 20$ to 2000, the number of RPs in the system $p = 25$ to 1000. The operating frequency of the MS is $f_{cl} = 100$ MHz (taken from the implementation), and data throughput between the HS and MS is 1 Gibit/s.

TABLE III
MEASURED AND EXTRAPOLATED TIMES FOR 1 RP

n	Load [ms]	Elim [ms]	Read [ms]	Total [ms]
20	**0.13**	**0.51**	**0.015**	0.66
100	**2.18**	**12.60**	**0.054**	14.84
200	**7.46**	**50.41**	**0.103**	57.96
1000	174	1260	0.5	1434
2000	688	5039	1	5728

Table III contains the times of loading, elimination, and result read of a single RP. Numbers printed in bold signify measured values, while the remaining numbers (for $n = 1000$ and 2000) were extrapolated from model equations (2) and (3) with respect to the actually measured times.

In order to analyze the performance of the system, it was necessary to fractionate the time in a communication and computation part. First, the communication part is analyzed.

In Table IV we present the amount of data that is sent from the Host System to the Modular System to transfer the SLE augmented matrix, and also the amount of data received as the solution, i.e. MS to HS. The received data depends on the number of moduli actually used that is always less than the maximum number of moduli k, which is considered.

TABLE IV
ETHERNET TRANSFER DATA SIZE AND ESTIMATED TIME (1 GIBIT/S)

		Max Size [KiB]			**Data Transfer**
n	k	**HS to MS**	**MS to HS**	**Total**	**Time [ms]**
100	314	89	93	182	1
200	632	353	372	725	6
1000	3208	8798	9408	18206	149
2000	6457	35174	37853	73027	598

From Table IV we can see that for the considered cases, and thus the required data amounts, the estimated times are in the order of hundreds of milliseconds. If needed, the times could be improved by using a faster data connection (e.g. a 10 Gibit/s Ethernet or PCI Express).

Next, we analyze the time taken by the computation part. We use the values for a single RP from Table III, and extrapolate based on the considered a MS with the number of RPs p from 25 to 1000 and n from 100 to 2000. The results of the extrapolation are presented in Table V.

The k column in Table V denotes the maximum number of moduli needed for the solution. The loading and elimination

TABLE V
SOLUTION TIME FOR DIFFERENT SIZES AND NUMBERS OF RP UNITS

		Solution time for a MS with p RP [s]					
n	k	$p = 25$	$p = 50$	$p = 100$	$p = 200$	$p = 500$	$p = 1000$
100	314	0.203	0.110	0.06	0.04	0.04	0.04
200	632	1.53	0.80	0.43	0.25	0.14	0.14
1000	3208	185.5	93.55	47.57	24.58	10.78	6.19
2000	6457	1485.6	746.0	376.2	191.3	80.3	43.3

process is done in parallel by p RPs. If more than p moduli is needed, RPs are assigned a new moduli set and the process is repeated (c.f. Algorithms 1 and 2). The results show that solving larger systems ($n \geq 1000$) in reasonable time (units and tens of seconds) can be achieved by using parallel systems with $p = 100$ and more RPs. Although the times may seem high for the considered SLE dimensions, it must be noted that the solution occurs without rounding errors. By comparing the computation and communication parts of the time, it is evident that even for $p = 1000$ RPs, and $n = 2000$, the communication time is in orders of magnitude less than computation.

The performance can be further improved by fine-tuning the design of the architecture having speedup potential in the order of a few units. The architecture can be implemented in ASIC, which could speed up the system up to 10 times. By employing a larger word size, the performance can be further improved. The number of moduli needed is always less than the theoretical upper bound considered in our data. In most cases, around a half of the maximum number of moduli is needed. Considering the mentioned possible improvements, the SLE $n = 2000$ could be solved on a MS with $p = 500$ to 1000 RPs in a time under 1 second. Such a result would be useful for a number of applications requiring solving SLEs without rounding errors.

The backward conversion from the remainder representation into the rational number set, which is performed by the HS, also contributes (not necessarily significantly) to the overall time and it is therefore important to perform the conversion efficiently. One such parallel approach is described in [**?**].

VII. CONCLUSION

Solving dense regular systems of linear equations (SLE) without loss of precision is a demanded task of numerical mathematics. The Modular System (MS) as a System on Chip (SoC) architecture of an SLE solver using residue arithmetic to avoid rounding errors is presented. MS was described in VHDL, designed, and a prototype was implemented on a Xilinx ML605 development board with a Virtex 6 FPGA with a 1 GiB DDR3 memory and a gigabit Ethernet interface, which was used to transfer data between the Host System and the evaluation board.

The implemented design was used to verify correctness of the architecture and to gather time and performance data with a limited number of a maximum of 5 parallel Residual Processors (RP) and 200 equation SLEs. Measured time also contained a control overhead of starting data transfers and start/end of the elimination process. Elimination times agreed with the estimation and high load time overhead was identified and future improvements suggested. Time and area complexity of the real implementation including external memory is useful for future development of a complete system with many residual processors. Such a system can be useful as a hardware peripheral attached to a Host System, or part of an embedded system needing SLE solution.

The measured time and area complexity were used for subsequent analysis and extrapolation to a larger number (up to 1000) RPs and larger SLE instances (up to 2000 equations). Considering several possible improvements, solution times of 2000 equation SLEs under 1 second could be attainable.

ACKNOWLEDGMENT

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