Abstract—Heterogeneous architectures require custom intellectual property (IP) blocks, named ‘accelerators’, that perform an operation efficiently. Residue Number System (RNS), a non binary compatible arithmetic system, can be used in such accelerators. One fundamental challenge in the design of RNS circuits is the selection of the moduli set. Here, we present our algorithm that results in very good moduli set for the given requirements. Experimental results and comparisons with similar algorithms, prove the efficiency of it. The algorithm has been implemented into a public web tool and is available via a web browser.

I. INTRODUCTION

The computing industry strives to deliver more energy, performance and cost efficient systems, in order to support the continuous increase in user requirements. It has reached a point where the traditional homogenous architectures consisting of the same type of processing units (PU), are not the wildcard solution for any problem. To facilitate higher gains, the computing industry has started investigating heterogeneous architectures, which consist of more than one type of PUs, or processing units paired with different accelerators. Except the conventional techniques of increasing the speed in a circuit, which are based on the principles of the binary arithmetic system, there are special architectures that operate accordingly to other arithmetic principles, like residue number system (RNS) [4]. RNS has been proposed as a viable alternative to conventional binary systems for achieving a very high throughput, in cases where the dominant operations are multiplication and addition, and the reason is the carry free operation.

Computations in RNS are done in parallel, inside specific data flow datapaths, called channels. Every channel is using a unique number as a base for it, named modulo. Every RNS circuit is using a number of special selected moduli, called the moduli set. The first and most important step in designing an RNS based circuit, is the moduli selection, which forms the base of the computations, because it affects every RNS operation of the system. In the RNS circuit design, the selection of the moduli plays a very crucial role. Our major contribution is: (i) we present a novel RNS moduli selection algorithm for a given set of constraints. Also, our minor contributions resulting from (i) are: (ii) we make available the algorithm as a web tool that can be used by anyone and reproduce our research, (iii) we provide another web tool that can create hardware language description (HDL) of binary to residue converters for the optimum moduli set for the given input requirements.

The tools are available at: http://arch.icte.uowm.gr/hdl/bin2rnsFA-multimodulus.php

II. RELATED WORK

One of the interesting research works in the ‘80s about RNS moduli selection, was the work of Guest et al [1], in which they detailed an algorithm to select a moduli set in a number of passes. This algorithm favors many small moduli, because the authors target to minimize the memory requirements in the implementation. We have implemented this algorithm in order to test its correctness. The complexity of this algorithm is high, and has been computed to $O(N \times \ln \ln N) + O(N^2)$, while the memory requirements are $O(N^2) \times 4$ Bytes A recent algorithm for moduli selection was also presented by Setiaarif et al [3]. Here, moduli sets of the form $2^{n-i+k-2}$ are created, where $n, i, k$ are integers and are specified according to some equations. This algorithm has very low complexity, only $O(1)$, because it performs a fixed number of computations. Also, the memory requirements are minimal, because only some scalar variables are stored. The main drawbacks of it are the high variation on the moduli channels, because one of the moduli is always 2, and that the dynamic range of the selected modulo does not cover the requested dynamic range.

Compared with the research of other authors, our work differentiates at three key aspects. First, it is a general moduli selection technique that bears no restrictions as to the general type of the moduli. In our algorithm, the designer can define the maximum allowed variation of the moduli, giving him a unique flexibility not present in other author’s algorithms. Second, it uses a heuristic to hone in quickly towards the best set selection, in contrast with other authors, who illustrate high complexity due to the brute forcing selection and high memory requirements, due to the precomputations. Third, it is the only one that has been implemented into a web tool and can be used anonymously by anyone to either reproduce our research or use it on his own circuits.

III. OUR ALGORITHM

We solved the problem of an efficient moduli selection, given a set of constraints, using the Algorithm 1. The algorithm performs a small number of iterations in order to locate the best moduli set, consisting of $nr$ prime numbers. First, it computes the $nr$ order of root, in order to hone in towards the central modulo of the set. If this number is not prime, it is increased by one and is re-examined. This is repeated as needed, until a prime, called $nrRoot$ is found. When the first prime is found, we initiate in parallel two searches: one to locate the upper half of the set (high subset), consisting of the immediate $ceil\left(\frac{nr}{2}\right)$ higher primes from $nrRoot$, where $ceil()$ denotes...
Algorithm 1: Moduli Selection Pseudocode.

Input: maximum number $x$, number of moduli in set $n_r$, maximum moduli difference $d$

Output: ModuliSet

1. begin
2. $nr$Root = ceil(compute $nr$ order of root($x$))
3. while !is_prime($nr$Root) do
4.       $nr$Root++
5. $moduliset = Array[]$
6. while $moduliset$ < $nr$ do
7.       locate_highest_moduli(ceil($nr$/2), $nr$Root)
8.       locate_lower_moduli(floor($nr$/2), $nr$Root)
9. initialize_moduliset(moduliset)
10. while $computed$ range($moduliset$) <= $x$ and $maximum$ moduli_difference > $d$ do
11.       shift_moduli_numbers(moduliset)
12. modify_moduli_set(moduliset)
13. return moduliset

the ceiling() function of rounding to the next integer. The same is repeated for the immediate floor($\frac{nr}{d}$) lower primers from $nr$Root, where floor() denotes the floor() function of rounding to the previous integer, to form the low subset.

If the dynamic range of the selected moduli set is higher than the requested, and the number of bits of every moduli are either equal or do not vary with a higher value than $d$, then this moduli set is returned to the user. If the variation is higher than $d$, then the moduli set is shifted by one prime first towards lower prime numbers. It is evident that our heuristic has very low complexity, and it guarantees to deliver a solution for the given parameters (product of the selected moduli to be larger than requested range and number of bits differ at most by $d$) and to the best of our knowledge this is a novel algorithm that can be used to locate very good moduli sets. The complexity of our algorithm for $n$ moduli is always $O(n)$, because it performs 1 computation of the $nr$ order of the root, $n$ computations of the number of input bits and up to $n$ shift operations.

IV. EXPERIMENTAL RESULTS

In order to evaluate our moduli selection algorithm we perform an extensive number of experiments for different input bits requirements (from 4 input bits to 64 input bits, annotated in the x-axis, with the label 'requested range'), for moduli sets consisting of 2 moduli (denoted with $n2$ in the figures) up to 6 moduli (denoted with $n6$ in the figures), and for moduli bitwidth variations $d$ varying from 0 (completely balanced channels) to 4.

To better evaluate the efficiency of our moduli selection, we decided to compare them with suggestions from other residue number system researchers. We divided the moduli selection algorithms in two domains. In the first domain, exist the algorithms that have only one variable in the definitions, and this is an integer value named ‘$n$’, while in the second domain, we placed the algorithms that are more complex and require a number of operations. In the first domain, we selected a recent publication [2], in which the authors propose the usage of the following moduli sets: the setA, which is $\{2^{n+2}+1, 2^n+1, 2\}$, and the setB $\{2^{n+2}+1, 2^{n+2} 2^{n+1}+1, 2\}$, with $n \in \mathbb{Z}, n > 1$. In both cases, our algorithm outperforms computationally these generic templates, because it has both lower memory requirements and lower complexity, resulting in a much faster solution. Our algorithm not only locates a solution much faster, but this solution is much better in most of the cases, than the generic templates. We performed a thorough analysis of these moduli sets for input bits ranging from 4 up to 64.

In the second domain, we decided to compare our algorithm with the algorithms of [1] and [3]. The experimental results illustrate (Figure 1 and 2) that our algorithm always finds solutions that cover the requested range, while the algorithm of [3] locates solutions that some times do not cover it, something which is a major shortcoming. Also, the computed range of our selected moduli set is very close to the requested range, while the other algorithm some time produces solutions with much higher computed range, something that is inefficient.

V. CONCLUSIONS

The residue number system can be used in heterogeneous architectures in their accelerators, further increasing the performance and lowering energy budget of contemporary architectures. Here, we provide our contribution to this area, by proposing a novel generic moduli selection algorithm. The experimental results prove that our moduli sets are very efficient with either minimum or none deviation from the requested range, and are closer to the requested dynamic range than other published moduli sets. Our algorithm is implemented as a web EDA tool and is available via a typical browser.

REFERENCES