SCALABLE SHIFTER SYNTHESIS FOR A FINITE FIELD ARITHMETIC UNIT

Jan Schmidt, Martin Novotný
Dept. of Computer Science and Computer Engineering, Czech Technical University
Karlovo nám. 13, 212 35 Prague 2, Czech Republic
Email: {schmidt,novotnym}@fel.cvut.cz

Abstract *We present a process that optimizes a sub-block of a parametric design which is strongly dependent on another, dominating, sub-block. The optimization is done by genetic algorithm, which in turn uses a dynamic programming approach to evaluate tentative designs. We demonstrate that problem-specific proofs can dramatically reduce the search space.*

1 Introduction

The problem presented here surfaced during the design of finite field arithmetic unit (AU) for a comparison study [1], which discussed trade-offs in hardware for Elliptic Curve Cryptography [4]. The unit used normal basis representation. Therefore, the inversion algorithm of Itoh, Teechaji, and Tsujii (ITT) [3] was chosen, based on repeated multiplication and squaring. In normal basis representation, squaring is realized as rotation (circular shift). Series of squarings of varying length are required, which can be implemented as shorter series of longer rotations. The arithmetic unit then comprises a multiplier and a shifter.

To achieve fair comparison in the cited study, the arithmetic unit had to be scalable. As area and performance were the main measures, we needed to adjust the area of the AU to obtain given performance and vice versa.

Besides this motivation, we also considered system optimization scenarios as in [2], where the top-level optimization process works on a set of blocks with parametric performance and (in that case) power consumption.

Our AU design (Figure 1) is dominated by the multiplier in both area and time. We choose the design of the multiplier to be the *primary* task and the design of the shifter as a *dependent* task, that is, we optimize it in the context of the dominating block. Our aim is to develop a procedure where the digit width of the multiplier controls the design of the entire AU.

Surprisingly, the problem of optimum shifter design exhibits a rich structure and a complex solution space. Although the solution presented here is heuristic, proving some problem-specific theorems played a role in decomposition of the problem.

We tried to solve the task by behavioral synthesis software; however, we never persuaded the synthesizers to decompose the rotations in time and space as described below.

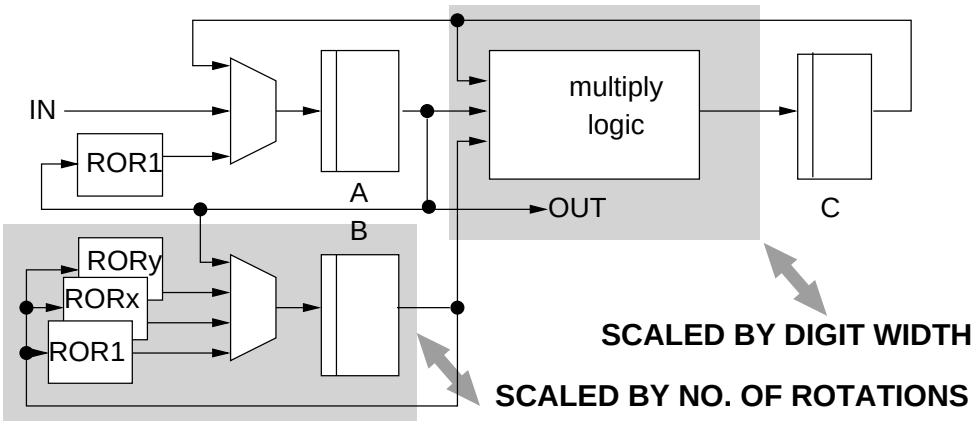

Figure 1: Finite field arithmetic unit

2 Problem Formulation

Let m be the degree of the finite field we work in, $GF(2^m)$. The ITT inversion algorithm calls for r distinct rotations, where $r = \lfloor \log_2 m \rfloor - 1$. The smallest rotation is always 1, the biggest one approximately $m/2$. Each of the rotations is performed exactly once in an inversion operation.

The task is to synthesize a shifter which implements all rotations and which, *when combined with the rest of the AU*, gives optimum performance/area ratio.

Two dependencies between the designs of the shifter and the multiplier exist. Firstly, the ratio of the shifter's area and time must be 'the right one'. Secondly, a long critical path the shifter may slow down the AU clock.

3 Approach Overview

To scale the shifter, we first find a limited set of rotations to be implemented in hardware and their combinations which realize the given set of rotations. This is the *time domain subproblem*. Then, we construct the hardware – the *space domain subproblem*. We approximate the hardware as an n-input multiplexer.

An iterative process (a genetic algorithm in this case) choses a set of rotations. Then, a dynamic programming procedure finds the number of clock cycles required for the whole computation in $O(m^2 \log m)$ time, which is fast enough given the range of m. The rotations compose modulo m, which makes many more rotation sets feasible.

Finally, the area and critical path length of the shifter are estimated, giving the value of the optimization criterion of the iterative process.

3.1 Sub-optimum Rotation Set by a Genetic Algorithm

In the time-domain problem, we seek $n \leq r$ hardware-implemented rotations. A configuration is given by n values in the range $(0, m)$. This is the phenotype of the genetic algorithm. The genotype (chromosome) is a fixed structure with a simple decoder. Constrained optimization is implemented using a penalty for each rotation k_i which the individual in question cannot realize.

Table 1: Shifters Adjusted to Different Multipliers

Multiplier			Shifter		
digit width	area	clock cycles	area	clock cycles	rotations
-	0	0	976	10	1,5,20,81
1	489	1956	244	159	1
6	2934	326	488	24	1,10

The rest of the genetic algorithm is quite classical, with single-point crossover and linear scaling of fitness values. The adaptive nature of linear scaling causes the convergence to remain unchanged even in the presence of large area and time of the multiplier, where the difference in evaluations are relatively small.

3.2 Sub-optimal Rotation Set by a Fast Heuristic

Based on observation of the genetic algorithm result, we designed a much faster heuristic procedure for finding the rotation set:

1 Find n_{max} such that the critical path length of n_{max}-input multiplexer is not bigger than that of the multiplier.
2 For $i = 1 \ldots n_{max}$ do:
  2.1 Implement rotation by 1.
  2.2 If $i < r/2$, implement every $\lceil i/r \rceil$-th rotation.
  2.3 If there are still less than i implemented rotations, pick the rest at random.
  2.4 Obtain the number of clock cycles by dynamic programming
  2.5 Compute the quality measure and record.
3 Choose the best solution.

The random assignment in Step 2.3 is justified by the fact that using more distinct rotations than $r/2$ does not bring much improvement. The complexity of this algorithm is $O(m^2 \log^2 m)$.

4 Results

The optimizer was implemented using the GAlib C++ library [5]. A number of experiments has been performed, with m in range interesting for elliptic curve cryptography, that is, from 160 to 250. The following facts were observed for the procedure using genetic algorithm:

1 The algorithm found an optimum solution where known.
2 Any realized rotation in an optimum solution was identical to some given rotation, although even slightly sub-optimum solutions did not have this property.
3 No optimum and only a few of sub-optimum solutions employed the modularity of rotation composition.
4 When the modularity was not used, the search space became disconnected, and more time was needed to achieve equivalent results.
5 All optimum solutions were best implemented by a multiplexer.

6 With population size of 100, the algorithm required circa 3000 generations to converge at $m = 160$, rising to 4000 at $m = 250$.

Table 1 illustrates the influence of the multiplier size on the shifter. The results were obtained for $m = 163$, where the required set of rotations is $\{1, 2, 5, 10, 20, 40, 81\}$. The area of an n-input multiplexer was $1.5n$ and the critical path of the unit was outside the shifter.

The procedure using the fast heuristics gave results almost identical to the other for 'normal' cases, i.e. for $n \leq 3$. Beyond that, the results were worse, in average by 15%.

5 Future Work

The biggest challenge for the future is certainly Observation 2. If this held in general, the search space of the problem would be drastically reduced. If not, an almost obvious experiment to perform is to *encourage* the genetic algorithm to produce solutions for which better implementations than multiplexers exist.

A weaker assertion that might be eventually proven is Observation 3, however, it does not benefit the solution immediately (cf. Observation 4).

6 Conclusion

We have presented a process, which optimizes a block in the context of another, dominating block with strong interdependences in area and time of computation.

We decomposed the problem into time- and space-domain subproblems. A part of the time-domain problem is solved exactly and repeatedly inside the genetic algorithm. The solution of the space-domain subproblem was approximated and the approximation verified.

The results show that formally proving certain properties of the circuit can dramatically reduce the search space. Such proofs, although problem-dependent and hard to construct, could improve the performance of EDA software.

Acknowledgment

The software for this work used the GAlib genetic algorithm package, written by Matthew Wall at the Massachusetts Institute of Technology.

References

[1] J. Schmidt, M. Novotný, M. Jäger, M. Bečvář and M. Jáchim, ''Exploration of Design Space in ECDSA", *Field-Programmable Logic and Applications – FPL2002*, Berlin, Springer, 2002, pp. 1072–1075.

[2] T. Givargis, F. Vahid, and J. Henkel, "System-Level Exploration for Pareto-Optimal Configurations in Parametrized System-on-a-Chip", *IEEE Transaction on VLSI*, Vol. 10, No. 4, pp. 416–422, 2002

[3] T. Itoh, G. Teechai, S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^t)$ using normal bases", *Journal of the Society for Electronic Communication* (Japan), Vol. 44, pp. 21–36, 1986.

[4] *IEEE P1363. Standard for Public-key Cryptography (Draft Version 8).* IEEE, October 1998

[5] M. Wall, GAlib: A C++ Library of Genetic Algorithm Components [Online] Available from http://sourceforge.net/projects/galib/