

Reconfigurable Computing Applied to Problems in Communications Security

Duncan A. Buell, James P. Davis, Gang Quan
Department of Computer Science and Engineering
University of South Carolina
Columbia, South Carolina 29208
{buell|jimdavis|gquan}@cse.sc.edu

Abstract— We describe the application of a reconfigurable computing platform to problems in communications security. Specifically, we describe the implementation of multiprecision multiplication as a kernel needed for certain NIST elliptic curve cryptographic standards. Our target architecture is the Star Bridge Systems Hypercomputer¹. By converting what would have been software routines into synthesized logic for the programmable resources of the Hypercomputer, significant speedups can be obtained in execution time. The key problems faced in implementing the multiplication kernels are the choice of a multiplication algorithm that effectively balances the various costs of computation and the partitioning of that algorithm among the several FPGAs on the Hypercomputer.

I. INTRODUCTION

Many of the major computational problems in modern communications center around the problem of secure communications. For a number of years NIST has promulgated standards for cryptographic protocols, some of which involve public key algorithms [JM99], [Men93], [NIS99]. At the heart of public key computations such as those of the NIST standards are the basic arithmetic operations either for multiprecision integers or for elements in finite fields of characteristic 2 (“binary fields”) [Gra01], [LN94]. What we will describe are library routines that speed up the basic arithmetic without sacrificing flexibility.

Our primary parallel computing platform for this study is the HC-36m Hypercomputer from Star Bridge Systems, Inc. This machine contains an array of five Xilinx Virtex-II XC2V6000 FPGA devices², configured in a “crosspoint” structure, with copious amounts of RAM. Data movement to and from the FPGAs is via a PCIX bus 64 bits wide running at a target clock speed of 134MHz. The issue for realizing efficient arithmetic architectures on this platform has to do with managing the tradeoff of area and speed—specifically in the form of minimizing the communications overhead of keeping the arithmetic units loaded with data while also insuring that structures can be realized that can be partitioned to Xilinx devices in such a way to minimize off-chip communications.

This work was sponsored by the Department of Defense under the LUCITE contract #MDA904-98-C-A081. Research of the first author is also partly supported by the National Science Foundation under grant #0112874.

¹Hypercomputer is a trademark of Star Bridge Systems, Inc.

²Xilinx and Virtex are trademarks of Xilinx, Inc.

II. ARITHMETIC KERNELS FOR CRYPTOGRAPHY

For pedagogical purposes we will use a canonical representation of an elliptic curve proposed by NIST for cryptographic applications, and we will deal only with *prime fields* $GF(p)$ defined modulo a large prime p . Such a curve can be written in homogeneous form as

$$Y^2Z = X^3 + AXZ^2 + BZ^3$$

for constants A and B . The fundamental operation to be explored is multiplication of a point $P = (x, y, z)$ on the group of the curve by a scalar M . The reason that this can be a computationally intensive process is due to the size of the operands.

There are 192-, 224-, 256-, 384-, and 521-bit versions of the NIST curve, with each of p, x, y, z and M being that many bits in length and requiring multiprecision modular multiplications of those sizes. Multiplication on the elliptic curve of a point by a scalar is done using the additive analog of the standard recursive doubling method for exponentiation. For a 192-bit multiplier, this will require 191 doublings and on average 96 additions (analogous to squarings and multiplications) on the curve. With the above canonical homogeneous representation of an elliptic curve modulo p , for p one of the five primes proposed by NIST, a single elliptic addition can be done using two squarings, 12 multiplications, seven additions, and two bit-shifts, all done modulo p .

The squarings and multiplications are the expensive operations here, in part because multiplication is much more expensive than addition and in part due to the modular reduction that follows either a multiplication or an addition. A sum of two operands is at most one bit longer than the operands and can be reduced modulo p with at most one subtraction. Reduction of a double-length product modulo p , however, has nearly the same cost as a multiprecision division.

The modular reduction can be eliminated by using Montgomery multiplication [Mon85] instead of ordinary multiplication and reduction; by using this method, the reduction is eliminated at the cost of two additional multiplications and two bit-shifts.

Beyond the actual rewriting of routines for attached reconfigurable hardware, the real problem to be addressed in moving software into reconfigurable logic is the balance of data movement and the granularity of computation.

If we implement only the multiplication in the reconfigurable hardware, then we are likely to be underutilizing the FPGAs. Even if we implemented Montgomery multiplication on the FPGAs, we would need to pass in three multiprecise operands (a , b , and modulus p) as input to the FPGAs and receive one ($a \cdot b$ modulo p) as output. It is unlikely that the improved speed of the multiply would compensate for the cost of moving data to and from the FPGAs. The solution is to move more and more of the computation onto the FPGA. If the entire elliptic addition were a single operation on the FPGA resource, then one would need to move four operands (x_n , y_n , z_n , and p) as input to the FPGAs and receive three (x_{n+1} , y_{n+1} , z_{n+1}) as output, but as many as 42 modular multiplications could be computed with the speed improvement of the hardware (counting squaring as a multiply, we have 14 multiplications modulo p , each of which in Montgomery mode has the cost of three multiprecise unsigned integer multiplications).

A final test would be whether one could implement the entire multiplication of a point by M as a reconfigurable design. The parameters to be explored involve the relative costs of data movement and compute time and the size of the computation that can effectively be placed on the reconfigurable resource.

Our problem thus begins with finding the most effective way to implement a multiprecise multiplication on the reconfigurable hardware, with the additional constraint that we will eventually want to implement a significant number of multiplications, in a pipelined fashion, and on a reconfigurable resource for which designs must eventually be partitioned over several FPGAs.

We note here the best-case limitations of the target hardware. The Hypercomputer is targeted at 64 bits per clock with a clock rate of 134 MHz. A 256-bit operand thus takes four clock cycles and the best we can hope for is 33.5 million such operands per second, or one operand every 30 ns. Four such operands as needed for a Montgomery multiplication means that we have a maximum rate of one multiplication every 120 ns.

III. RELATED WORK

In addition to the work on implementations for the NIST elliptic curve standards, we are doing similar work on the Advanced Encryption Standard (AES, also known as Rijndael) [DR01], [NIS01]. Once again, the kernel computations are not the standard floating point operations for which standard processors have been designed, and the kernels are highly amenable to implementation as synthesized logic.

We mention also our eventual goal, which is that libraries for the reconfigurable hardware should be transparently substitutable for software libraries in packages like Mathematica, Matlab, or Maple. This would permit scientists and engineers doing research to move from software-only code in a very high-level scientist-friendly style to software and hardware execution on a faster platform without changes to the source code. Experimentation on a

desktop could become more serious experimentation on a department-wide server and then intensive testing on a large machine without requiring costly and bug-ridden rewriting of code.

IV. PATHS FROM DESIGN TO SYNTHESIZED LOGIC

There are several pathways by which a 256-bit multiplication could be synthesized for a reconfigurable machine.

First, one could use the Xilinx ISE tools to design, synthesize, and do the placement and routing of a multiplication architecture on a target Xilinx FPGA of one's choosing.

The ISE tools are, presumably, close to optimal in terms of their usage of the chip resources and the timing of the resulting system, since these tools are quite specifically aimed at Xilinx FPGAs. We note that with the Xilinx tools we can directly synthesize a 32-bit or a 64-bit multiplier using templates from Xilinx to which we provide a bit-length parameter.

As an alternative to the Xilinx ISE, one could use other synthesis tools, such as those from Synopsys, to design a multiplication, synthesize the logic, and then target the Xilinx FPGA. Place-and-route would then be performed by Xilinx tools. The price, if any, that will be paid for more generality in the synthesis would be in the mapping to the FPGAs.

Finally, in this case in which the target hardware is the Star Bridge Systems Hypercomputer, one path is the use of the proprietary Star Bridge Viva³ software. This package performs synthesis targeted at the Xilinx FPGAs on the Star Bridge hardware, and what we can expect from Viva is that the Star Bridge hardware will be utilized effectively. What is yet to be determined is how well Viva matches up as a synthesis tool against the more established (and more expensive) tools.

V. DESIGNS FOR MULTIPRECISE MULTIPLICATION

The cost of multiplication is inherently quadratic in the number of bits of the operands. It is impractical to construct monolithic circuits of such large widths, so we would normally expect to construct a multiprecise multiplication (of 256 bits, for example) by staging a multiply from smaller building blocks. A reasonable first version would be to use 32-bit or 64-bit multiplication units. Several different algorithms for multiplication on operands of this size exist in the literature [Car01], [FO01], [Knu97], including a scalable algorithm for Montgomery multiplication [TK99].

Having modelled these basic multiplication units, we subject them to benchmarking. As pointed out in Chen and Mead [CM85], what might be considered an optimal or efficient algorithm in a mathematical or software sense may not be so from the standpoint of its VLSI architecture because of physical cost considerations. We must therefore subject each model to cost evaluation. For synthesized circuits, this is generally done in terms of area, speed and power characteristics. Since this is an implementation on

³Viva is a trademark of Star Bridge Systems, Inc.

approach from 32-bit multipliers and that uses 504 slices and 12 of the 144 internal multiplier units.

E. A 128-bit multiplier built up with a divide-and-conquer approach from 64-bit multipliers and that uses 1814 slices and 36 of the 144 internal multiplier units.

F. A 256-bit multiplier built up with a divide-and-conquer approach from 64-bit multipliers and that uses 6025 slices and 108 of the 144 internal multiplier units.

G. A 256-bit multiplier built with the broadcast approach from eight 32-bit multipliers and that uses 1809 slices and 32 of the 144 internal multiplier units.

Method	Slices	Delay (ns)	Mults
32-bit shift-add (Synopsys)	197	6.2	
32-bit Knuth (Synopsys)	192	6.7	
32-bit Booth (Synopsys)	387	9.3	
64-bit shift-add (Synopsys)	412	10.0	
64-bit Knuth (Synopsys)	814	21.0	
64-bit Booth (Synopsys)	424	10.5	
(A) 32-bit Xilinx ISE	119	4.4	4
(B) 64-bit Xilinx ISE	484	6.4	16
(C) 64-bit Xilinx ISE	2268	7.1	0
(D) 64-bit Xilinx ISE divide-and-conquer	504	7.7	12
(E) 128-bit Xilinx ISE divide-and-conquer	1814	16.6	36
(F) 256-bit Xilinx ISE divide-and-conquer	6025	18.1	108
(G) 256-bit Xilinx ISE broadcast-32	1809	*14.8	32

Table 1: Resources needed for multiplier units

From this, we can begin to estimate the ability of the XC2V6000 chip to support long integer multiplication. The divide-and-conquer implementations D, E, and F, for 64-, 128-, and 256-bit multiplication use, as expected, exactly 3, 9, and 27 times the number of multipliers needed for the Xilinx 32-bit multiplier. The slice utilization ratios $504/119 = 4.2$, $1814/504 = 3.6$, and $6025/1814 = 3.3$ suggest that the resource needed in excess of that for the multiplication units is decreasing as a fraction of the total resource needed (remembering, of course, that the 32-bit unit requires no additional adders and that increasingly larger multipliers will actually need relatively more and not fewer adders).

Slice utilization does not preclude the use of the XC2V6000 for long multiplication using divide and conquer. A divide-and-conquer 512×512 multiplier will require three 256-bit multipliers. If the slice utilization ratios hold, then the $6025 \times (6025/1814) \approx 20,000$ slices needed would still be comfortably less than the 33,792 slices on

the XC2V6000. However, three multipliers would require 324 on-chip multiply units, but only 144 are available, and even the largest Virtex FPGA, the XC2V8000, has only 168 multipliers. Clearly, if a 512-bit multiply is to be put on the XC2V6000, a hybrid multiplication architecture will be necessary. Also clear is that the next larger multiplication architecture needed to support the NIST standard, which would be 384 bits long, would be a very tight fit indeed, and that the 544-bit arithmetic must be split over multiple chips if the multiplications are done in this way.

Having seen the need for a hybrid algorithm, we present as multiplier G data for a 256-bit multiplication unit built from 32-bit units and done in eight multiplication steps. The total slice utilization is only 1809, of which just over half is for the eight 32-bit multiplication units. This version splits the multiplication into four stages: multiply in parallel, sum the the two partial products P_0 and P_1 , sum the result into the running accumulator, and shift out the low 32 bits into the final result register. The time per stage comes out to just under 15 ns. At a cost in speed, the pipelined multiplications can be done on a single chip for 256-bit operands. (We are not sure at this point, however, why there is such a large discrepancy between the 4.4 ns of the 32-bit multiplication and the 14.8 ns of the eight parallel 32-bit multiplications for the broadcast algorithm.)

A 512×512 multiplication unit done with the broadcast method on 128-bit operands would require four 128-bit multiplication units. These might just fit on a single chip; the problem is that they would require all 144 of the internal multiplication units. Although it is usually extremely difficult to use *all* of any resource on an FPGA and still have enough wiggle room for place-and-route to succeed, the four multipliers and the attendant adders should use less than half the slices on the chip, so there is some reason to believe that the design might be feasible. The slightly longer 521-bit multiply would probably require reuse of some of the silicon resources at a cost of longer delay or more clock cycles, but the slowdown would probably be less than what one would face if it were necessary to partition the multiply across two FPGAs.

The bottom line at this stage of the research is the following. We have identified the internal multiplier units on the Xilinx chips as the crucial resource if the Xilinx fast multiplication designs are to be used. We do not know whether direct synthesis of long multiplication is feasible, but we would not expect it to be anywhere near competitive in speed with use of the Xilinx designs, and hence the number of internal multiplier units is probably the limiting factor in the overall design of the long multiply. Given that conclusion, it would seem that the breakpoint is at 256 bit operands. For the 192-, 224-, and 256-bit operands of the NIST proposal, a divide-and-conquer multiplication design should be feasible and fit on a single chip. For the larger 384- and 521-bit operands, hybrid approaches will be necessary, using fewer resources but requiring more time. With five XC2V6000 chips on the Hypercomputer, a complete Montgomery multiplication should be achievable for the smaller operands; for the larger operands, this is likely

to be possible only through the partial re-use of resources.

REFERENCES

- [Car01] J. D. Carpinelli. *Computer Systems Organization and Architecture*. Addison-Wesley, 2001.
- [CM85] M. C. Chen and C. A. Mead. Concurrent algorithms as space-time recursion equations. In S. Y. Kung, H. J. Whitehouse, and T. Kailath, editors, *VLSI and Modern Signal Processing*. Prentice-Hall, 1985.
- [DR01] Joan Daemen and Vincent Rijmen. *The Design of Rijndael : AES, The Advanced Encryption Standard*. Springer Verlag, Berlin, 2001.
- [FO01] M. J. Flynn and S. F. Oberman. *Advanced Computer Arithmetic Design*. John Wiley and Sons, 2001.
- [Gra01] Torbjörn Granlund. *GNU MP: The GNU multiple precision arithmetic library*. Free Software Foundation, Inc., 2001.
- [JM99] Don Johnson and Alfred Menezes. The elliptic curve digital signature algorithm. Technical Report CORR 99-34, University of Waterloo Center for Applied Cryptographic Research, 1999.
- [Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 2, Seminumerical Algorithms. Addison-Wesley, 3 edition, 1997.
- [LN94] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and Their Applications*. Cambridge University Press, 1994.
- [Men93] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Boston, 1993.
- [Mon85] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
- [NIS99] NIST. Recommended elliptic curves for federal government use, July 1999. csrc.nist.gov/csrc/fedstandards.html, csrc.nist.gov/encryption/dss/ecdsa/NISTReCur.pdf, csrc.nist.gov/publications/fips/fips186-2/fips186-2.pdf.
- [NIS01] NIST. FIP 197: Announcing the Advanced Encryption Standard (AES), November 26 2001. csrc.nist.gov/encryption/aes/index.html.
- [Par00] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.
- [Syn95] Synopsys, Inc. Application note: Flattening and structuring: A look at optimization strategies, 1995. DS031-2.
- [TK99] A. F. Tenca and C. K. Koc. A scalable architecture for Montgomery multiplication. In C. K. Koc and C. Paar, editors, *Proceedings, CHES'99, Lecture Notes in Computer Science 1717*, pages 94–108. Springer Verlag, 1999.