Area-optimized Montgomery multiplication on IGLOO 2 FPGAs

Pedro Maat C. Massolino*, Lejla Batina*, Ricardo Chaves† and Nele Mentens‡
*iCIS - Digital Security Group, Radboud University, The Netherlands
Email: {P.Massolino,lejla}@cs.ru.nl
†INESC-ID, IST, Universidade de Lisboa, Portugal
Email:ricardo.chaves@inesc-id.pt
‡imec-COSIC, KU Leuven, Belgium
Email:nele.mentens@kuleuven.be

Abstract—**This paper presents the first area-optimized Montgomery modular multiplication module on low–power reconfigurable IGLOO® 2 FPGAs, from Microsemi. In order to obtain a good response time with few resources, the FPGA pipelined Math blocks and the embedded memory blocks are fully leveraged. As a result, 256-bit modular multiplications can be done in 2.33 µs, at a cost of 505 LUT4 cells, 257 Flip Flops, 1 Math block and 1 64x18 RAM block. If more area resources are considered, a modular multiplication can be performed in 1.25 µs at a cost of 680 LUT4s, 341 Flip Flops, 2 Math blocks and 2 64x18 RAM blocks. This work is the first fundamental step towards area-efficient public-key cryptography on the Microsemi IGLOO® 2 FPGAs.**

1. Introduction

Modular multiplication of large integers can be efficiently implemented using the Montgomery multiplication algorithm [1]. This algorithm can be applied for the computation of RSA [2] and Elliptic Curve Cryptography (ECC) [3] with prime field operations. Montgomery multiplicaiton manages to perform modular multiplication by trading a prime n division, for one with an easier divisor r, usually a power of 2, and addition of multiples of n. Also, it is possible to tweak the algorithm by increasing the value r to avoid the final subtraction [4] and avoid modular reductions during additions and/or subtractions [5].

In the literature, there is a great number of co-processors for modular multiplication with the Montgomery algorithm [6]–[9]. Some of these papers show results for modular multiplication, while others only for the complete implementation of the public-key cryptosystem. Also, the same can be split into the ones that aim for compact structures or take full advantage of the existing resources.

This paper proposes two hardware co-processors performing Montgomery multiplication and integer addition/subtraction. Both structures strive for a low-area footprint, with the second one using more resources in order to provide a better performance.

The presented implementations target the IGLOO® 2 [10] FPGA from Microsemi. This FPGA is based on flash technology, as opposed to SRAM technology commonly seen in Xilinx and Altera FPGAs. Since it is a flash based technology, it is labeled as more energy-efficient and more secure [10].

Note that Microsemi offers ECC embedded cores on higher-end IGLOO 2 and SmartFusion 2 devices, starting from M2GL060. However, these are only available in a limited set of devices labeled as TS and under restricted exportation rules.

This paper is organized as follows. Section 2 presents background information on the optimized Montgomery multiplication algorithms implemented, describes the FPGA main features, and details the proposed structures and respective implementations. Section 3 shows the obtained results and analyzes the related state of the art in regard to the proposed solution. Section 4 concludes this paper with some final considerations and future work directions.

2. Background and Implementation

Koç et al. [11] proposed Coarsely Integrated Operate Operand Scanning (CIOS) and Finely Integrated Operate Operand Scanning (FIOS) algorithms as an optimization for Montgomery multiplication targeting word size processors or multiplication units. The CIOS algorithm, as shown in Algorithm 1 computes one entire partial product, lines 2–6, before performing a reduction, lines 8–12. FIOS on the other hand, computes the multiplication of a word of each input, lines 2 and 6, and then proceeds with the reduction, lines 4, 7 and 9. While the CIOS algorithm operates directly on partial products, the iterative word-based structure of the FIOS algorithm allows for a design with two parallel units, one doing the multiplication, line 6, and the other one the reduction, line 7.

In the proposed implementations, the IGLOO 2 FPGA family from Microsemi is targeted. These FPGAs include

This work was supported in part by the Technology Foundation STW (project 13499 - TYPHOON & ASPASIA), from the Dutch government, and †partially supported by national funds through Fundacão para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

Algorithm 1 CIOS Montgomery multiplication algorithm [11]

```
Require: $a, b \leq 2n$, $r = 2^{(\lceil(\lceil log_2(n)+2\rceil/\text{word size})\rceil\cdot\text{word size})}$,
    $n' = -n^{-1} \pmod r$, $w = 2^{\text{word size}}$, $l = \lceil log_2(r/w)\rceil$
Ensure: $o = a \cdot b/r \pmod n$
 1: $o \leftarrow 0$
 2: for $i \leftarrow 0$ to $l-1$ by $1$ do
 3:     $c, p_0 \leftarrow a_0 \cdot b_i + o_0$
 4:     for $j \leftarrow 1$ to $l-1$ by $1$ do
 5:         $c, p_j \leftarrow a_j \cdot b_i + o_j + c$
 6:     end for
 7:     $p_{j+1} \leftarrow c$
 8:     $m \leftarrow n' \cdot p_0 \pmod w$
 9:     $c, null \leftarrow (p_0 + m \cdot n_0)$
10:     for $j \leftarrow 1$ to $l-1$ by $1$ do
11:         $c, o_{j-1} \leftarrow (p_j + m \cdot n_j + c)$
12:     end for
13:     $o_j \leftarrow (p_{j+1} + c)$
14: end for
15: return $o$
```

Algorithm 2 FIOS Montgomery multiplication algorithm [11]. Same inputs as Algorithm 1

```
 1: $o \leftarrow 0$
 2: for $i \leftarrow 0$ to $l-1$ by $1$ do
 3:     $c1, p_0 \leftarrow a_0 \cdot b_i + o_0$
 4:     $m \leftarrow n' \cdot p_0 \pmod w$
 5:     $c2, null \leftarrow a_0 \cdot b_i + o_0$
 6:     for $j \leftarrow 1$ to $l-1$ by $1$ do
 7:         $c1, p_j \leftarrow a_j \cdot b_i + o_j + c1$
 8:         $c2, o_{j-1} \leftarrow (p_j + m \cdot n_j + c2)$
 9:     end for
10:     $o_l, o_{l-1} \leftarrow (o_l + c1 + c2)$
11: end for
12: return $o$
```

dual-port SRAM with two inputs and two outputs and a three-port SRAM, which has two outputs and one input. The dual port SRAM can do 2 reads, 1 read and 1 write, or 2 writes at the same time. And the three-port SRAM can do 2 reads and 1 write at the same time, as long as the writing address is different from the reading addresses. Therefore, loads and stores on the three-port memory needs to be carefully scheduled.

Another FPGA component is the Math Block. This block is able to perform $z \leftarrow t \pm x \cdot y + (z >> 17)$, where x and y are unsigned 17-bits words and t and z are unsigned 43-bits words. This operation can be summed into a multiplication plus an addition and an optional carry of the previous computed value. By analyzing the main Math Block operation and the lines in CIOS Algorithm 1, it is very easy to map them. Lines 3 and 9 can be done by setting the previous z to 0, and in line 13, x or y has to be set to 0. Lines 5 and 11 just fit exactly into $z \leftarrow t+x\cdot y+(z >> 17)$, since in our case the words are 17 bits wide and the c is the previous computed value shifted by the word size. Finally, line 8 can be done by setting t to 0.

Just like CIOS, Algorithm 2 FIOS can also be done in one Math Block, however it is more interesting to pipeline the operations. Lines 7 and 8 can be pipelined because the operation in 7 does not depend of the result in 8, but only the other way around. Therefore the most inner loop in Algorithm 2 is split into first stage computes line 7 and outputs to the next stage to compute line 8. Line 10 can be split into $t \leftarrow o_l + c1$ and $o_l, o_{l-1} \leftarrow t + c2$, then each part happens on a different stage. Line 3 is the same as line 7, but lines 4 and 5 happens in the second stage. Therefore, the pipeline cannot operate non stop, because there is a bubble to wait for line 4 to be resolved.

The previous mapping results on Figure 1 structures, where the left is the CIOS version and the right is the FIOS version. Because CIOS does only one operation per cycle, one three-port SRAM is enough for all operations. However, FIOS requires at least two three-port SRAM, since up to 3 values are read at once and one value is written. Because FIOS has two memories, it supports primes up to 539 bits, on the other hand CIOS structure supports only up to 267 bits primes because of the memory size. The memories itself are not fully registered, so values are directly feed into the circuit.

For the addition and subtraction operations, the applied algorithm is the same as described by Batina et al. [5]. In this method, additions are done without applying reduction, because the reduction can be postponed for the next multiplication. Which in the Math Block with $z \leftarrow t + x \cdot y + (z >> 17)$, is done by setting x as 1 while the addition operands are in t and y. Subtractions also do not require reduction, but the output needs to be added with a multiple of the prime to guarantee the output is non negative. The subtraction then is $o \leftarrow c - b + 2n$, which in the CIOS circuit is done by computing $t \leftarrow c - 1b$, and then $o \leftarrow t + 2n$. FIOS circuit can be used in pipeline mode where the first stage computes $t \leftarrow c - 1b$, and the second $o \leftarrow t + 2n$. This delayed reduction has a small price to pay in the Montgomery multiplication, where the number of bits that are added in r is not only 2, but instead 4 [5]. To support even more additions and subtractions we implement our solution with a minimum of 5 extra bits, instead of only 4. Because one of the multiplications inputs has to be 1 in additions and subtractions, it was opted to have a register outside of the Math Block that can be directly set to 1.

3. Results and Related Work

In order to properly evaluate the proposed structures, experimental results were compared with the state-of-the-art in Table 1. However, only previous works with multiplication timing results are depicted. Other works considering low latency and/or high throughput are not discussed, such as [12], [13]. Since they require significantly more FPGA resources, thus not resulting in a meaningful comparison.

Considering an efficiency metric of the inverse of *area* (LUT + Register) times *latency* (µs), the second structure is able to achieve an efficiency of 784 being 40% more

Figure 1. Montgomery multiplier structure for CIOS with 1 Math Block and 1 memory (left) and for FIOS with 2 Math Blocks and 2 memories (right)

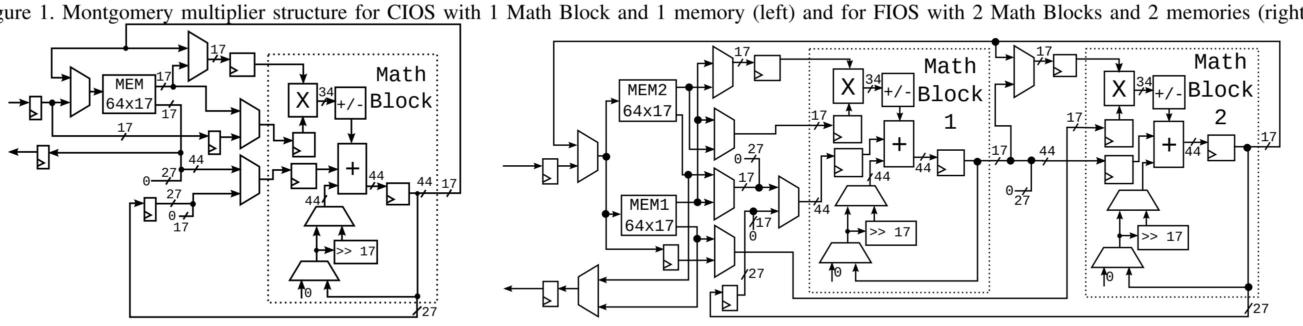

Table 1. Comparison of our results to the literature on hardware implementations for ECC. The speed results are for one modular multiplication.

Work	Field	FPGA	LUT4	FF	Emb. Mult.	BRAM 64x18	BRAM 1kx18	Frequency (MHz)	Add. Cycles	Sub. Cycles	Mult. Cycles	Time (µs)
Only Finite Field Multiplication												
[6]	256	Virtex II Pro®	2866⋆	2866⋆	4	0	0	101.87	–	–	582	5.7
[14]	256	Virtex® E	3096⋆	3096⋆	0	0	0	100.44	–	–	772†	7.69
[15]	512	Virtex® 5	40⋆	40⋆	4	0	4		-	-		8.67
[16]	512	Artix® 7	1789	2164	57	0	0	106	-	-	66	0.62
[17]	256	Virtex® 7	1917		9	0	0	225	-	-	114	0.51
Finite Field Addition, Subtraction and Multiplication												
Our 1	251-267	IGLOO® 2	505	257	1	1	0	250	21	37	583	2.33
Our 2	251-267	IGLOO® 2	680	341	2	2	0	250	23	23	312	1.25
Our 2	506-522	IGLOO® 2	680	341	2	2	0	250	38	38	1062	4.25
Finite Field with Inversion												
[18]	256	Virtex II®	6218⋆	6218⋆	0	0	0	31.92	1	1	257	8.05
Full ECC Processors												
[19]	256	SmartFusion®	3690	3690	0	0	12	109	46	46	401	3.7
[19]	256	Virtex II Pro®	1546⋆	1546⋆	1	0	3	210	46	46	401	1.9
[19]	256	Virtex II Pro®	2316⋆	2316⋆	4	0	3	210	28	28	157	0.75
[7]	256	Virtex II Pro®	3664⋆	3664⋆	2	0	9	108.2	44	44	637	5.89

⋆ Maximum possible value assumed from the number of slices. † Values estimated by multiplying time with frequency.

efficient and supporting fields up to 522 bits in comparison with first structure. It should be noted that both structures are able to operate at a frequency of 250MHz, the maximum frequency imposed by the three port Memory Block [10].

In the literature, McIvor et al. [6] considered the SOS, CIOS and FIOS algorithms, proposed in [11] targeting a Virtex II Pro® FPGA. The proposed design is identical for the 3 algorithms, using an ALU to perform the multiplication or addition. This approach is similar to our more compact one, implementing CIOS, in the aspect that it only uses one ALU, iterating over the intermediate values. However, this structure requires multiple cycles to perform the operation $t+x\cdot y+carry$, while ours does it in a single cycle. Also, the implementation of McIvor et al. requires significantly more area, 4 DSPs to implement the ALU, and only operates at 100MHz. The multiple cycle and lower frequency are related to a more older technology, and not design choice.

Örs et al. [14] propose a systolic array architecture for Montgomery modular multiplication. The systolic array approach has an array of cells, where each cell computes one bit of the modular multiplication. This strategy is more suitable when embedded multipliers are not available, such as in ASICs. Nevertheless, it still requires more cycles to perform the computation.

Also not using the embedded primitives of the FPGA nor other memories to store the data, there is the work of Daly et al. [18]. They propose a structure with an adder of the size of the input and perform the multiplication iteratively, requiring as many cycles as the prime length. This way, a multiplication can be computed in each iteration by multiplying one bit of one of the inputs by the other entire input. This approach allows to reduce the number of computational cycles, but results in a significantly larger data path and consequently lower frequencies. Moreover, the amount of occupied LUTs is also very high, imposing a very high area cost.

Varchola et al. [19] proposed a small ECC co-processor only supporting NIST primes curves. This work also has the main goal of achieving an ECC design as small as possible, therefore using only a small number of DSPs. Their structure follows a similar approach as ours, using a single ALU that keeps receiving values to be processed. However, their ALU is composed of one multiplier, using the built-in DSP in the FPGA, followed by a final adder, implemented using FPGA LUTs. The separation between multiplier and adder allows then to simplify the operation scheduling, and thus to use less cycles to compute the result, at the cost of more LUTs and routing. The authors present three implementations, one supported by SmartFusion® FPGA, which does not include embedded multipliers, and two others supported by a Xilinx®

Virtex II Pro®. In Virtex II Pro® based structures, one employs a single embedded multiplier, achieving a computation delay of 1.9 µs, and other employs 4 multipliers, achieving a computation delay of 0.75 µs. Since the presented area is for the entire ECC co-processor with scalar point multiplication, a fair analysis cannot be done. However, the less compact structure suggests a higher area cost potentially allowing a faster computation.

Another approach also with a minimization strategy, was proposed by Vliegen et al. [7], also targeting a Virtex II Pro® FPGA. The authors mapped the CIOS algorithm in a more straightforward manner, not worrying so much about the resource mapping and operation scheduling. As a consequence, the resulting structure requires 2 embedded multipliers and significantly more LUTs and memory blocks. Despite this, the achieved computational delay is twice as long as ours, mostly due to the much lower achievable frequency, resulting in a less efficient design.

From this, it can be concluded that with a careful mapping of the resources available in the FPGA devices, along with a careful scheduling and reuse of the needed operations, significant improvements and area savings can be achieved.

4. Final Considerations

The work presented in this paper proposes two area-optimized FPGA structures for the computation of the Montgomery modular multiplication algorithm for generic primes, targeting low–power IGLOO® 2 FPGAs from Microsemi. The proposed structures impose a very low usage of the available FPGA resources while still achieving good performances. To achieve this performance with a low area, the Math Blocks and embedded memories were used together with a careful scheduling to assure a full pipeline usage and a low number of computation cycles. While the first structure achieves the lowest area, the second one allows to approximately half the computation time at the cost of twice the amount of embedded memories and Math Blocks and only 35% more LUTs and registers.

Future work will consist of adding wrapping and control logic to allow for the full computation of the ECC scalar multiplication. The main challenge will be to minimize the additional memory and control resources needed to store and process the intermediate values.

References

[1] P. L. Montgomery, "Modular Multiplication without Trial Division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, feb 1978.

[3] V. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology - CRYPTO 85 Proceedings*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer Berlin / Heidelberg, 1986, vol. 218, pp. 417–426.

[4] C. Walter, "Montgomery exponentiation needs no final subtractions," *Electronics Letters*, vol. 35, no. 21, pp. 1831–1832, Oct 1999.

[5] L. Batina, G. Bruin-Muurling, and S. B. Örs, "Flexible Hardware Design for RSA and Elliptic Curve Cryptosystems," in *Topics in Cryptology – CT-RSA 2004*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, vol. 2964, pp. 250–263.

[6] C. McIvor, M. McLoone, and J. V. McCanny, "FPGA Montgomery multiplier architectures - a comparison," in *Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on*, April 2004, pp. 279–282.

[7] J. Vliegen, N. Mentens, J. Genoe, A. Braeken, S. Kubera, A. Touhafi, and I. Verbauwhede, "A compact FPGA-based architecture for elliptic curve cryptography over prime fields," in *Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on*, July 2010, pp. 313–316.

[8] H. Alrimeih and D. Rakhmatov, "Fast and Flexible Hardware Support for ECC Over Multiple Standard Prime Fields," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 12, pp. 2661–2674, Dec 2014.

[9] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, "Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks," *Computers & Electrical Engineering*, vol. 35, no. 2, pp. 329 – 338, 2009, circuits and Systems for Real-Time Security and Copyright Protection of Multimedia.

[10] Microsemi, "IGLOO2 Product Information Brochure," Microsemi, Tech. Rep., 2014. [Online]. Available: http://www.microsemi.com/document-portal/doc_download/132013-igloo2-product-information-brochure

[11] Ç. K. Koç, T. Acar, and B. S. Kaliski Jr., "Analyzing and comparing Montgomery multiplication algorithms," *Micro, IEEE*, vol. 16, no. 3, pp. 26–33, Jun 1996.

[12] G. Orlando and C. Paar, "A Scalable GF(p) Elliptic Curve Processor Architecture for Programmable Hardware," in *Cryptographic Hardware and Embedded Systems — CHES 2001*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, vol. 2162, pp. 348–363.

[13] K. Javeed and X. Wang, "Efficient Montgomery Multiplier for Pairing and Elliptic Curve Based Cryptography," in *Communication Systems, Networks Digital Signal Processing (CSNDSP), 2014 9th International Symposium on*, July 2014, pp. 255–260.

[14] S. B. Örs, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of a Montgomery modular multiplier in a systolic array," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, April 2003, p. 8.

[15] M. M. Sandoval and A. D. Perez, "Novel algorithms and hardware architectures for montgomery multiplication over gf(p)," Cryptology ePrint Archive, Report 2015/696, 2015.

[16] A. Mrabet, N. El-Mrabet, R. Lashermes, J. B. Rigaud, B. Bouallegue, S. Mesnager, and M. Machhout, "A systolic hardware architectures of montgomery modular multiplication for public key cryptosystems," Cryptology ePrint Archive, Report 2016/487, 2016.

[17] D. Amiet, A. Curiger, and P. Zbinden, "Flexible fpga-based architectures for curve point multiplication over gf(p)," in *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, pp. 107–114.

[18] A. Daly, W. Marnane, T. Kerins, and E. Popovici, "An FPGA implementation of a GF(p) ALU for encryption processors," *Microprocessors and Microsystems*, vol. 28, no. 5–6, pp. 253 – 260, 2004, special Issue on FPGAs: Applications and Designs.

[19] M. Varchola, T. Güneysu, and O. Mischke, "MicroECC: A Lightweight Reconfigurable Elliptic Curve Crypto-processor," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 204–210.