# Optimised Montgomery Domain Inversion on FPGA

Francis Crowe * Alan Daly William Marnane

*Abstract* — **Modular inversion is a critical operation in Elliptic Curve Cryptosystems (ECC). This paper presents a hardware optimised modular inversion algorithm targeted towards an FPGA implementation. It exploits the underlying structure of the device, leading to a fast and efficient design. Arithmetic is performed in the Montgomery domain, which allows an inversion result to be an input to further operations without the need for domain conversion. Results presented show an increase in throughput over existing inverter designs on reconfigurable logic.**

## 1 INTRODUCTION

Elliptic Curve Cryptosystems were proposed independently by Miller [1] and Koblitz [2] in the mid-eighties. They require much smaller key sizes than traditional public key schemes such as RSA, due to the relative difficulty of solving the underlying mathematical problems on which they are based. Point operations in ECC over the field $GF(p)$ are comprised of a series of modular additions, subtractions, multiplications and inversions. Modular inversion is the most time consuming of these field operations.

The first reported modular inverter architecture over $GF(p)$ on reconfigurable logic was presented by Daly in [3], based on the Montgomery inverse algorithm defined in [4]. An improvement to this architecture was made by Dormale in [5], using an algorithm with negative variables and sign rather than full magnitude comparisons.

In this paper a fast and efficient inverter design is proposed, based on a new version of the original Montgomery inverse algorithm. The improved design operates in the Montgomery domain, making it suitable for applications such as ECC, where a large number of operations are performed in series in the same domain. For example, a 256-bit ECC point scaler multiplication requires an average of 128 point additions and 256 point doublings [6]. Each point addition requires 3 modular multiplications and 1 inversion, while there are 4 multiplications and 1 inversion per point doubling.

The flexibility and speed of FPGAs make them a suitable choice in cryptographic applications. They are structured as an array of configurable logic blocks (CLBs) interconnected by a programmable routing network. The basic building block of the Xilinx Virtex CLB is the logic cell (LC) [7]. Each LC contains a 4-input look-up table (LUT), a storage element and dedicated carry logic for high-speed arithmetic. The carry logic is routed vertically upwards through the CLBs within a single column. It is important that the carry chain does not exceed the column height as it will have to be routed from the top of one column to the bottom of the next. This will cause considerable routing delays and increase the critical path of the design.

While adder structures such as carry-save and carry-lookahead are popular in VLSI designs, carry-propagate adders are most efficient on FPGAs due to the dedicated carry routing [8]. For efficient use of resources, one of the adder inputs should be directly connected to the serial carry chain multiplexor to enable the full adder function [5]. Also combinational logic between the adder and memory element should be kept to a minimum.

## 2 MODULAR INVERSION

An efficient method of performing modular multiplication was proposed by Montgomery in 1985 [9]. The need for trial division by the modulus is removed and replaced by a series of additions and right shifts. If the inputs are first converted to the Montgomery domain by Montgomery multiplying by a constant, the output will also be in the Montgomery domain. This conversion overhead is negligible when a large number of computations are performed in series.

The modular multiplicative inverse of an integer $a \in [1,M-1]$ modulo $M$ is defined as the integer $x$, such that $a.x \bmod M = 1$, where $x$ exists if $a$ and $M$ are relatively prime. The Montgomery inverse was defined by Kaliski as the integer $x = a^{-1}2^m \bmod M$, where $m$ is the bitlength of the modulus [4]. The algorithm to compute the Montgomery inverse is based on the extended Euclidean algorithm (EEA) and is broken into two phases. Improved versions of these are given in algorithms 1 and 2. The new algorithms are motivated by the FPGA design constraints and lead to a more efficient hardware circuit realisation.

Algorithm 1 first performs the EEA (steps 01→18) followed by a modular correction stage (steps 19, 20). One of the inputs to each subtraction can be fixed by comparing $\frac{U}{2}$ and $\frac{V}{2}$ in each

*Department of Electrical & Electronic Engineering, University College Cork, Ireland, e-mail: francisc@rennes.ucc.ie

iteration when $U$ and $V$ are odd (i.e. $U_0, V_0 = 1$). A *switch* variable is introduced to determine whether $RS_a$ or $RS_b$ should be doubled in each iteration. At the end of the EEA, $RS_b = -a^{-1}2^k \bmod M$ and $RS_a = M$, where $RS_b \in [1, 2M-1]$ and $k$ is the number of iterations. Modular negation and correction are performed by first subtracting $RS_b$ from $RS_a$ and then adding $M$ to the result if this is negative. The output of phase 1 is given by $R = a^{-1}2^k \bmod M$.

The algorithm choice for phase 2 depends on whether the input operand and output result are in the integer or Montgomery domain. In algorithm 2, the input is doubled $(2m-k)$ times modulo $M$ to give a final output of $R = a^{-1}2^{2m} \bmod M$. An advantage of using this algorithm is that when the input operand is in the Montgomery domain, the output will also be in the Montgomery domain. This allows the inverse result to be used in further operations without the need for domain conversion.

**Algorithm 1:** Montgomery Domain Inv (Phase 1)

**Input**: $a \in [1, M-1]$ and $M$, $\gcd(a, M) = 1$;
**Output**: $R = a^{-1}2^k \bmod M$, $m \leq k < 2m$;

$U \Leftarrow M$; $V \Leftarrow a$; $RS_a \Leftarrow 1$; $RS_b \Leftarrow 0$; $k \Leftarrow 0$; $switch \Leftarrow 0$;
01. **while** $(V > 0)$ **do**
02. **if** $(U_0 = 0)$ **then** $U \Leftarrow \frac{U}{2} - 0$;
03. **if** $(switch = 0)$ **then** $RS_b \Leftarrow 2(RS_a)$;
04. **else** $RS_b \Leftarrow 2(RS_b)$;
05. $switch \Leftarrow 1$;
06. **else if** $(V_0 = 0)$ **then** $V \Leftarrow \frac{V}{2} - 0$;
07. **if** $(switch = 0)$ **then** $RS_b \Leftarrow 2(RS_b)$;
08. **else** $RS_b \Leftarrow 2(RS_a)$;
09. $switch \Leftarrow 0$;
10. **else if** $(\frac{U}{2} > \frac{V}{2})$ **then** $U \Leftarrow \frac{U}{2} - \frac{V}{2}$;
11. **if** $(switch = 0)$ **then** $RS_b \Leftarrow 2(RS_a)$;
12. **else** $RS_b \Leftarrow 2(RS_b)$;
13. $RS_a \Leftarrow RS_a + RS_b$; $switch \Leftarrow 1$;
14. **else if** $(\frac{V}{2} \geqslant \frac{U}{2})$ **then** $V \Leftarrow \frac{V}{2} - \frac{U}{2}$;
15. **if** $(switch = 0)$ **then** $RS_b \Leftarrow 2(RS_b)$;
16. **else** $RS_b \Leftarrow 2(RS_a)$;
17. $RS_a \Leftarrow RS_a + RS_b$; $switch \Leftarrow 0$;
18. $k \Leftarrow k + 1$;
19. $RS_a \Leftarrow RS_a - RS_b$;
20. **if** $(RS_a < 0)$ **then** $RS_a \Leftarrow RS_a + M$;
**return** $R = RS_a$ and $k$;

## 3 ARCHITECTURE

### 3.1 Optimised Inverter Design

The architecture of the new inverter is shown in Fig. 1. Three carry-propagate adders are required for operations on $U$, $V$ and $RS_a$. Parity checking in hardware is achieved by inspecting the least significant bit (LSB), while right-shifting by one bit divides by two. Two's complement subtraction is performed by inverting one of the adder inputs and setting the carry-in to 1. The carry-outs of the adders are used to indicate the relative magnitude and sign of the operands. The *MODE* signal is set in a state machine to 00 for the EEA, 01 and 10 for the modular correction steps and 11 for phase 2.

**Algorithm 2:** Montgomery Domain Inv (Phase 2)

**Input**: $R$ and $k$ from Phase 1
**Output**: $R = a^{-1}2^{2m} \bmod M$
$RS_a \Leftarrow R$;
01. **for** $i = 1$ **to** $2m - k$ **do**
02. $RS_a \Leftarrow 2(RS_a)$;
03. **if** $(RS_a \geqslant M)$ **then** $RS_a \Leftarrow RS_a - M$;
**return** $R = RS_a$;

One of the inputs to each of the adders is fixed and there is only a small combinational stage between the $RS_a$ adder and memory element. This is required for phase 2, when either $2(RS_a)$ or $2(RS_a - M)$ are selected for the next iteration of the algorithm depending on the carry-out of the $RS_a$ adder. Both addition and subtraction are performed by this adder, with the carry-in set to the LSB of the $MODE$ signal ($MODE_0$). The output registers are loaded initially with the input operands and then updated on each clock cycle when their clock enable ($CE$) signals are asserted. Each $CE$ is dependent on the $MODE$, the LSB of $U$ and $V$, and the carry-outs of the adders.

It takes 1 clock cycle for loading, $k$ for the EEA, 2 for modular negation and correction and $(2m-k)$ for phase 2, giving a total of (2m+3) clock cycles per inversion. This is the fastest method of performing modular inversion when both the input and output are in the Montgomery domain. Also the risk of timing analysis attacks is reduced as the number of clock cycles per inversion is independent of the input operand value.

### 3.2 Carry-Select Design

To prevent the carry chain exceeding the maximum column height of the FPGA and to reduce the carry-propagation delay, carry-select adders can be used to break the carry chain into smaller units. It is not possible to pipeline the architecture as each iteration of the algorithm is dependent on a magnitude comparison from the previous iteration. To perform a carry-select addition of $A$ and $B$, the $m$-bit inputs are split into upper and lower $(\frac{m}{2})$-bit blocks $A_H, B_H, A_L, B_L$ and the following are computed: $(A_L + B_L)$, $(A_H + B_H)$ and $(A_H + B_H + 1)$. The carry-out of the $(A_L + B_L)$ addition selects

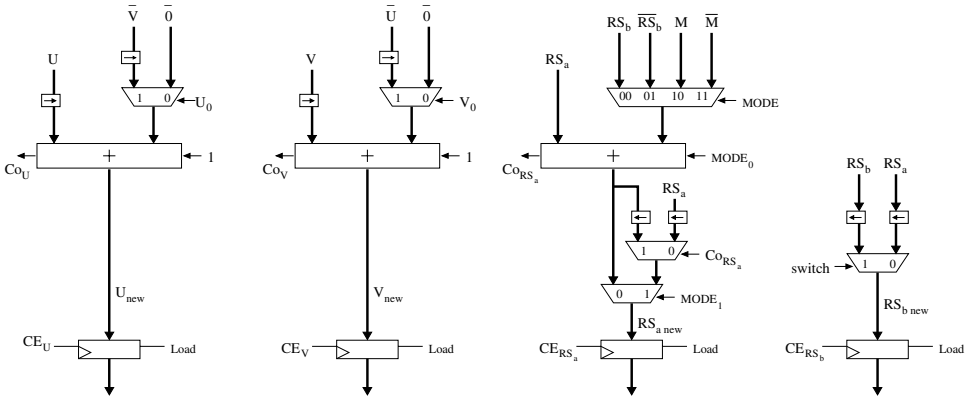


Figure 1: New inverter architecture

the correct output for the upper half of the result $(A+B)_H$.

For a carry-select subtraction, $(A_L - B_L)$, $(A_H - B_H)$ and $(B_H - A_H)$ are computed. In this case the correct output for the upper half of the result is either $(A_H - B_H)$ or $(A_H - B_H - 1)$, depending on the carry-out of the $(A_L - B_L)$ subtraction. In two's complement representation, $(A_H - B_H - 1)$ is equivalent to $\overline{(B_H - A_H)}$, which is easily calculated by inverting the $(B_H - A_H)$ subtraction result.

The $RS_a$ carry-select architecture is shown in Fig. 2. There are 3 $(\frac{m}{2})$-bit adders and control logic to determine the upper half of the result. The extra $(\frac{m}{2})$-bit adder and multiplexors between the adders and memory element will inevitably lead to an increase in the area of the carry-select design. The $UV$ carry-select architecture shown in Fig. 3 has 4 $(\frac{m}{2})$-bit adders and logic to select the upper half of $U_{new}$ and $V_{new}$. No extra adders are required for this part as there is an overlap in calculations when both $(\frac{U}{2} - \frac{V}{2})$ and $(\frac{V}{2} - \frac{U}{2})$ are performed in parallel.

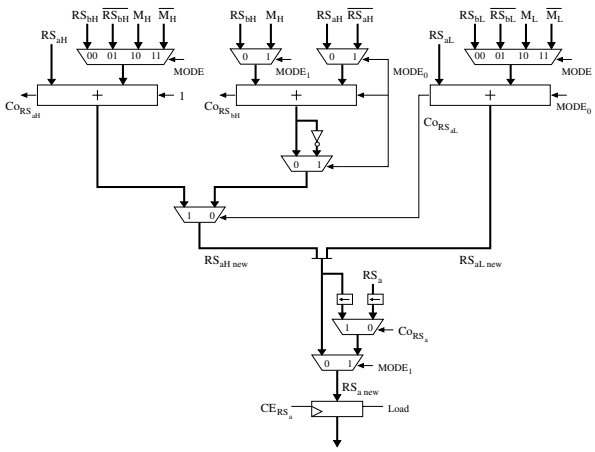


Figure 2: $RS_a$ carry-select architecture

## 4 RESULTS

The Xilinx Virtex-E xcv2000e-6bg560 device was targeted to allow fair comparison with existing inverter designs. There are 2 bits of carry logic per CLB and 80 CLBs per column, meaning a maximum unbroken carry chain length of 160 bits. Table 1 gives results for various bit lengths of the basic and carry-select designs. Throughput rates are computed based on a maximum of $(2m + 3)$ clock cycles per inversion. The benefit of the carry-select design is more evident for bitlengths greater than the column height. The percentage increase in speed is greatest for the (128×2)-bit carry-select design. At low bitlengths the basic architecture outperforms the carry-select in terms of the throughput/slice ratio, due to the overhead of the extra control logic. A 32-bit interface is included in the results, with the input operand and inverse result loaded into $(\frac{m}{32})$ 32-bit shift registers.

| **Design** bits | **Speed** MHz | **Area** slices | **Thr.** Mbps | **Thr/area** kbps/slice |
|---|---|---|---|---|
| 64 | 63.18 | 347 | 30.87 | 88.95 |
| 32×2 | 72.50 | 454 | 35.42 | 78.02 |
| 128 | 45.20 | 693 | 22.34 | 32.23 |
| 64×2 | 57.57 | 882 | 28.45 | 32.26 |
| 160 | 40.52 | 858 | 20.07 | 23.39 |
| 80×2 | 51.44 | 1094 | 25.48 | 23.29 |
| 256 | 25.58 | 1363 | 12.72 | 9.33 |
| 128×2 | 39.04 | 1722 | 19.41 | 11.27 |

Table 1: Results for basic and carry-select designs.

A comparison with the throughput/slice ratios of the only two known inverter architectures on FPGA is made in Fig. 4. Significant reductions in area have been achieved over the original architecture in [3]. Although the design in [5] consumes less area, the throughput/slice ratios are higher for the work

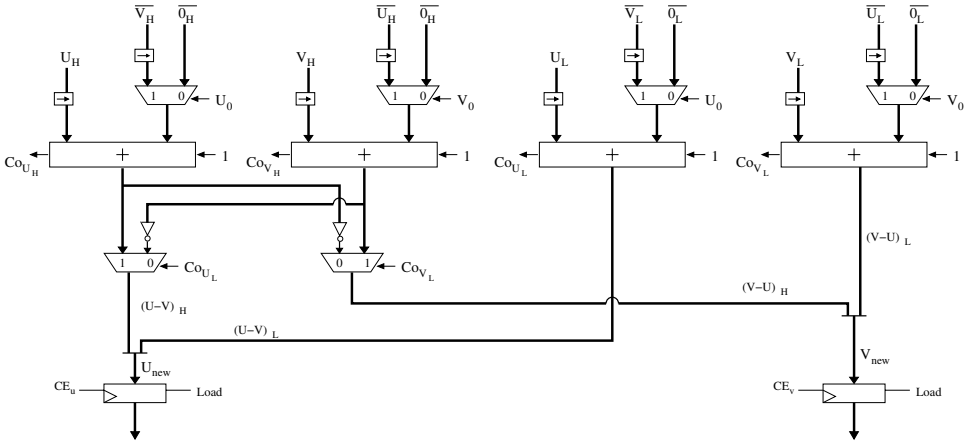


Figure 3: $UV$ carry-select architecture

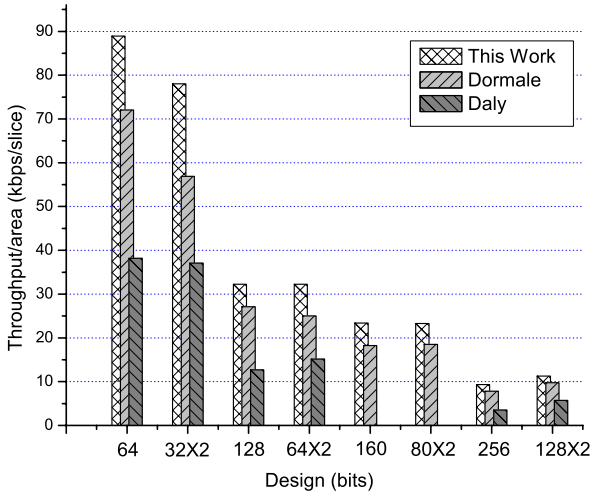


Figure 4: Inverter comparison

in this paper. Both existing designs use a phase 2 algorithm based on modular halving that outputs a result in the Montgomery domain. However if the inputs are in the Montgomery domain, the outputs will not be in the same domain. Modular multiplication will be required if the result is to be used in further operations, taking an extra $m$ clock cycles to compute.

## 5 CONCLUSIONS

The inverter architecture presented in this paper is suitable for use in ECC applications. A new version of the Montgomery inverse algorithm was seen to result in a significant saving in FPGA resources. The problem of carry-chain overflow was addressed with a carry-select adder design. Improvements over existing architectures were made, using the throughput/slice metric for comparison.


## Acknowledgments

This work is funded by a research innovation project from Enterprise Ireland. F. Crowe is supported by the Boole Centre for Research in Informatics (BCRI) in UCC.


## References


[1] V. S. Miller. Use of elliptic curves in cryptography. *Advances in Cryptology – CRYPTO '85*, 218 of Lecture Notes in Computer Science:417–426, 1985.

[2] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[3] A. Daly, W. Marnane, and E. Popovici. Fast Modular Inversion in the Montgomery Domain on Reconfigurable Logic. *Proceedings of the Irish Signals and Systems Conference (ISSC 2003)*, pages 363–367, July 2003.

[4] B. S. Kaliski. The Montgomery Inverse and it's applications. *IEEE Transactions on Computers*, 44(8):1064–1065, August 1995.

[5] G. M. Dormale, P. Bulens, and J. J. Quisquater. An Improved Montgomery Modular Inversion Targeted for Efficient Implementation on FPGA. *International Conference on Field-Programmable Technology - FPT 2004*, pages 441–444, Dec 2004.

[6] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography*. 265 of London Mathematical Society Lecture Notes. Cambridge University Press, 2000.

[7] Xilinx. *Virtex-E 1.8 V Field Programmable Gate Arrays*. Product Specification, Sep 2002.

[8] S. Xing and W. H. Yu. FPGA Adders: Performance Evaluation and Optimal Design. *IEEE Design and Test of Computers*, 15(1):24–29, 1998.

[9] P. L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, April 1985.