Comparison of Hardware Implementations of Finite Field Inversion Algorithms under GF(2^n)

Amit Thomas, Grace Mary Matson, Soumya Gupta

***Abstract*—In the context of ECC cryptosystems in GF(2^n), unilaterally the most time consuming operation is that of the inversion step. This highly complex operation has been tackled in various ways namely the straightforward Extended Euclidean Algorithm implemented in hardware and the Itoh-Tsujii Algorithm that exploits Fermat's theorem. In this work, we wish to compare the complexity of these two architectures in terms of time, size and power efficiency.**

I. Introduction

Cryptographic systems such as ECC employ operations like scalar multiplication and point addition, both of these tasks are highly complex in nature and require many inversions to complete. And an ECC framework requires multiple point additions and scalar multiplications, each of which require multiple inversion, each of which can take multiple clock cycles. Since inversion is the costliest field operation performed within the ECC system, one benefits highly form optimizing this operation.

Initial works looked at implementing inversion in finite fields via the Binary Extended Euclidean Algorithm including [1], in which we trade the divisions performed in the original EEA method for bit shifts which are much better suited for hardware applications. Next, Kaliski Inversion for Montgomery Domain was proposed in which the inversion is costly in terms of area with a relatively low maximum clocking speed compared to the proposed architectures for multiplication and addition. [5] Finally, the Itoh-Tsujii algorithm exploits Fermat's Little Theorem in order to obtain the inverse via an addition chain. This allows us the flexibility of computing the inverse quickly but at the cost of more area, depending on the number of exponentiation blocks that are cascaded for providing further inputs.

II. Algorithms

In the this section, we provide an overview of the Extended Euclidean Inversion Algorithm and the Itoh-Tsujii Inversion Algorithm as a preliminary for its hardware implementation.

A. Extended Euclidean Inversion Algorithm

The extended Euclidean algorithm is used to find the modular multiplicative inverse of a number. The inverse of x exists if and only if $\gcd(x, n) = 1$. If this is true, there exist integers p and s so that $px + sn = 1$. Notice that regardless of the value of s, if we apply modulo n on both sides of the equation here, we find that the resulting expression is $px = 1$. So, p is the inverse of x under modulo n. This is done based on Euclid' s algorithm which can algorithmically find the gcd of two numbers by succesively finding modulo of one operand with respect to the other and finding the gcd of the result and the latter operand until the result of the gcd operation is trivial. This is extended to find the values of p and s by initialising two other variables u and v that will undergo the same process as the above operands but instead of performing modulo, the quotient of x divided by n is used in lieu of the modulo operation. Once the gcd is found, we will have automatically obtained the values of x and n within the current values of u and v.

B. Itoh-Tsujii Inversion Algorithm

This algorithm is based primarily on Fermat's Little theorem which states that if $a \in GF(2^m)$

$$a^{-1} = a^{2^m - 2} \tag{1}$$

We see that this gives us a method for computing the inverse, albeit, with a very large number of multiplications. Itoh and Tsujii reduced the number of multiplications required within this process by formulating an addition chain which consists of a sequence of intermediate numbers through which it is possible to obtain the required exponent with the minimum number of operations. [2] For example, let us rewrite this in a different form, $a^{-1} = [\beta_{m-1}(a)]^2$, where $\beta_k(a)$ is $a^{2^k - 1} \in GF(2^m)$. By virtue of this formulation, we observe that

$$\beta_{j+k} = \beta_j^{2^k} \beta_k = \beta_k^{2^j} \beta_j \tag{2}$$

Formally, an addition chain is a sequence of numbers where any number can be represented as a sum of the previous numbers within the sequence except for the first number. We need to get $\beta_{m-1}(a)$ and we know that each index of β can be decomposed into the combination of two other β with indices who sum to the required index. Thus, if we can obtain the optimal addition chain for some the order of the finite field, we will be able to optimally compute the inverse of an element within the finite field with the help of binary exponentiation and multiplication.

III. Hardware Implementation

Here, we describe the hardware structures and optimizations used to implement the above algorithms.

```
Inversion:
 $F := F(x)$;
 $S := F(x); V := 0;$ (* $\deg S = m$ *)
 $R := B(x); U := 1;$ (* assume $\deg R = m$ *)
 delta := 0; (* delta = $\deg S - \deg R$ *)
 for $i := 1$ to $2m$ do
  if $r_m = 0$ then
   $R := x \cdot R; U := (x \cdot U) \,\mathrm{MOD}\, F$;
   delta += 1 (* deg $R$−=1 *)
  else (* $r_m = 1$ *)
   if $s_m = 1$ then
    $S := S - R; V := (V - U) \,\mathrm{MOD}\, F$;
   end;
   $S := x \cdot S$; (* deg $S$−=1 *)
   if delta = 0 then
     (* $\deg S < \deg R$ : division done *)
     (* exchange polynomials *)
     $(R \leftrightarrow S); (U \leftrightarrow V)$;
     $U := (x \cdot U) \,\mathrm{MOD}\, F$; delta := 1 (* deg
 $R \leftrightarrow \deg S$ *)
   else
    $U := (U / x) \,\mathrm{MOD}\, F$;
    delta −=1
   end
  end
 end (* $B^{-1}(x) = U = V$ *)
```

Fig. 1. EEIA Algorithm

A. EEIA

The exact algorithm is given by the pseudo-code in Fig. 1 [1] Most surprisingly, the output of the inverse is obtained after a constant number of cycles, which is $2m$, where m is the order of the finite field. Here, R, S, U and V are registers used to hold the inputs and all intermediate quantities too. $F(x)$ denotes the irreducible polynomial and $B(x)$ is the quantity of which we need to find the inverse. Throughout the algorithm, we track the difference in the degree between the polynomials S and R (delta) with a $log_2(m+1)$ bit up/down counter as we finally want to perform S mod R as part of the algorithm. This is done by bit shifting both the polynomials till they are 1 in the MSB, at which point subtraction is done in order to obtain the remainder. Subtraction is implemented via bit-wise xor. Once delta is found to be 0, the order of S is increased, the polynomials are switched, and the same process is continued, till $2m$ iterations have passed. The constant time for completion is due to our assumption in the beginning of the algorithm that the degree of both R and S are m, even if they are not. This leads to extra cycles taken to converge to the actual degrees of both R and S. We don't carry out the S and R operations with respect to the modulus of the irreducible polynomial as we don't care about the exact values of S and R, unlike U and V, which is also the reason why we can liberally bit-shift the two polynomials.

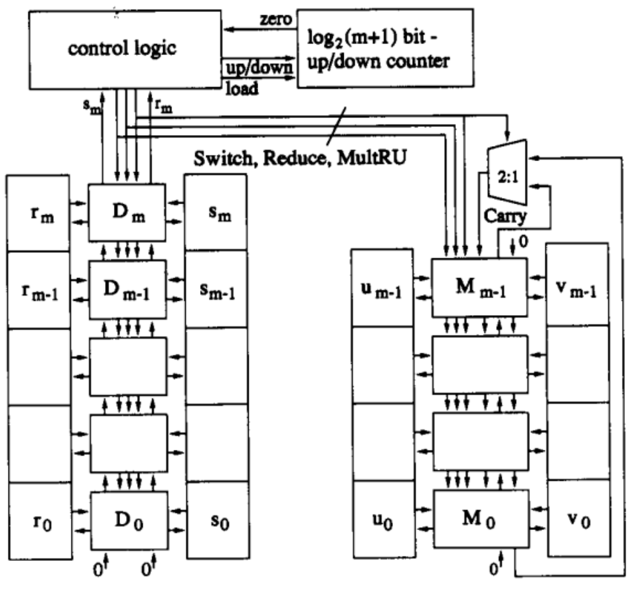


Fig. 2. Hardware Implementation for EEIA

The exact hardware implementation is done via a decoupled cell-array structure. Each array corresponds to either operations done for S and R cells or operations done for U and V cells. This is possible because the operations on the two pairs are independent of each other and the nature of the operation is decided by the various control signals. An illustration of the hardware components in this paper is shown in Fig. 2.

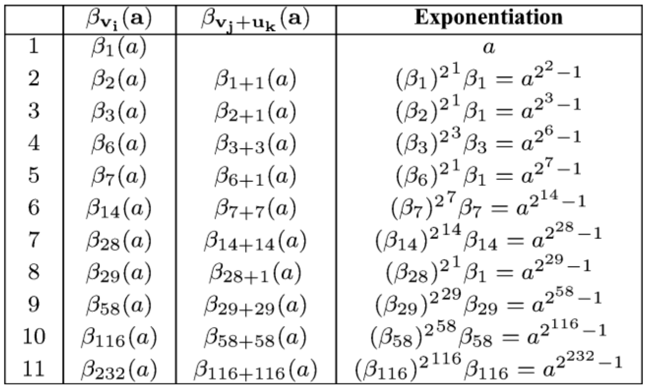
While implementing the individual D and M cell units, we discovered a small errata within the proposed circuit that caused it to not give the correct output, this has been fixed within our implementation.

B. Itoh-Tsujii Algorithm

This architecture as shown in Fig. 4 includes control logic, a cascaded binary exponentiation block, a modular reduction block and a recursive Karatsuba multiplier. [3]The Karatsuba multiplier is implemented via a combinational recursive multiplication module approach. [6] The output of the Karatsuba multiplier undergoes modular reduction and is then either stored for further multiplication or undergoes binary exponentiation. Outputs from the register bank, the cascade exponentiation block and the multiplier output can be selected to be inputs to the multiplier in the next clock cycle. Using these blocks and the knowledge of the addition chain for a given finite field of order m, we will be able to control the inputs to the multiplier block and register bank such that the required outputs will be produced at each output leading us to the required polynomial. An example of an addition chain for $m = 233$ is given in Fig. 3.

IV. RESULTS

We assumed our finite field to be GF(2^7) under the irreducible polynomial $x^7 + x^5 + x^4 + x^3 + x^2 + x^1 + 1$



	$\beta_{\mathbf{v_i}}(\mathbf{a})$	$\beta_{\mathbf{v_j}+\mathbf{u_k}}(\mathbf{a})$	**Exponentiation**
1	$\beta_1(a)$		a
2	$\beta_2(a)$	$\beta_{1+1}(a)$	$(\beta_1)^{2^1}\beta_1 = a^{2^2-1}$
3	$\beta_3(a)$	$\beta_{2+1}(a)$	$(\beta_2)^{2^1}\beta_1 = a^{2^3-1}$
4	$\beta_6(a)$	$\beta_{3+3}(a)$	$(\beta_3)^{2^3}\beta_3 = a^{2^6-1}$
5	$\beta_7(a)$	$\beta_{6+1}(a)$	$(\beta_6)^{2^1}\beta_1 = a^{2^7-1}$
6	$\beta_{14}(a)$	$\beta_{7+7}(a)$	$(\beta_7)^{2^7}\beta_7 = a^{2^{14}-1}$
7	$\beta_{28}(a)$	$\beta_{14+14}(a)$	$(\beta_{14})^{2^{14}}\beta_{14} = a^{2^{28}-1}$
8	$\beta_{29}(a)$	$\beta_{28+1}(a)$	$(\beta_{28})^{2^1}\beta_1 = a^{2^{29}-1}$
9	$\beta_{58}(a)$	$\beta_{29+29}(a)$	$(\beta_{29})^{2^{29}}\beta_{29} = a^{2^{58}-1}$
10	$\beta_{116}(a)$	$\beta_{58+58}(a)$	$(\beta_{58})^{2^{58}}\beta_{58} = a^{2^{116}-1}$
11	$\beta_{232}(a)$	$\beta_{116+116}(a)$	$(\beta_{116})^{2^{116}}\beta_{116} = a^{2^{232}-1}$

Fig. 3. Addition Chain for m = 233

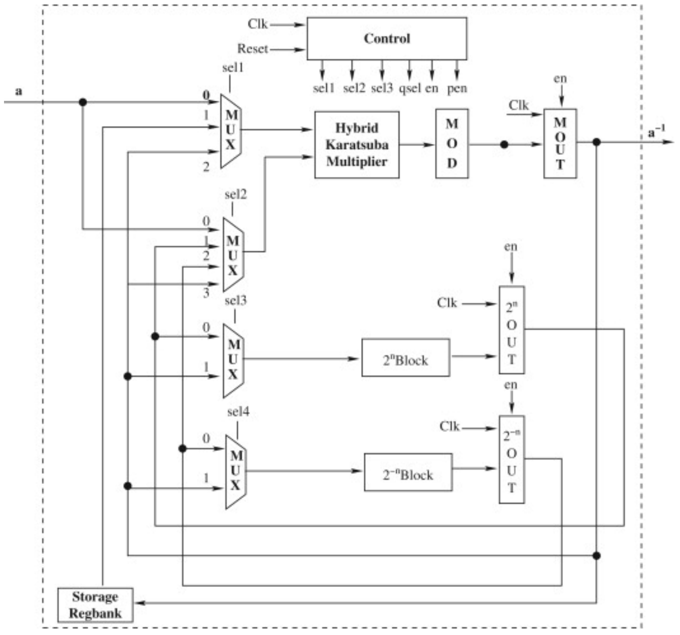


Fig. 4. Itoh-Tsujii Hardware Implementation

for implementations of both inversion algorithms. Both were implemented on the xc7a100tcsg324-1 FPGA board using Vivado. We first notice that the EEI Algorithm is much more flexible to changes in the field itself and much more easier to generalise as we are only required to change the x^{-1} and x^m constant values and the number of registers in the cell-arrays. In contrast, the Itoh-Tsujii algorithm is designed in all aspects to specifically work for a given finite field and irreducible polynomial. This is especially important in order to design the Karatsuba multiplier and modular reduction block which give us the output in a combinatorial fashion. The number of clock cycles taken by the Itoh-Tsujii algorithm to complete inversion is 5 clock cycles and the number of clock cycles required by the EEI algorithm is 14. However, the Itoh-Tsujii algorithm in practice would incur much more propagation delay as compared to the EEI algorithm due to the long combinatorial blocks within its implementation. This means that each clock cycle lasts longer in Itoh-Tsuji. Once we implemented both inversion algorithms on an FPGA, we found that the number of LUTs used in EEI was 2 and that in Itoh-Tsujii was 30. This indicates a large reduction in complexity for the EEI algorithm. Power consumption was found to be of the order of 1.473W for EEI and 10.863W for Itoh-Tsuji.

V. Conclusion

We found that in terms of pure clock cycles, the Itoh-Tsujii algorithm seems to be superior to the EEI algorithm, however, when we compare it in every other respect (power consumption, area, flexibility), the Euclidean implementation seems to be a better option, at least for low order finite fields. Where time complexity of EEI increases linearly with the finite field order, time complexity of Itoh-Tsuji can be approximated as $O(log(n))$ due to the property of addition chains.

References

[1] Brunner, Hannes, Andreas Curiger, and Max Hofstetter. ”On computing multiplicative inverses in GF (2/sup m/).” IEEE Transactions on computers 42.8 (1993): 1010-1015.
[2] Itoh, Toshiya, and Shigeo Tsujii. ”A fast algorithm for computing multiplicative inverses in GF (2m) using normal bases.” *Information and computation* 78.3 (1988): 171-177.
[3] Roy, Sujoy Sinha, Chester Rebeiro, and Debdeep Mukhopadhyay. ”Generalized high speed Itoh–Tsujii multiplicative inversion architecture for FPGAs.” Integration 45.3 (2012): 307-315.
[4] Rebeiro, Chester, and Debdeep Mukhopadhyay. ”High speed compact elliptic curve cryptoprocessor for FPGA platforms.” International Conference on Cryptology in India. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
[5] Kaliski, B.S.: The montgomery inverse and its applications. IEEE Trans. Comput. 44(8), 1064– 1065 (1995)
[6] Machhout, Mohsen, et al. ”Efficient hardware architecture of recursive Karatsuba-Ofman multiplier.” 2008 3rd International Conference on Design and Technology of Integrated Systems in Nanoscale Era. IEEE, 2008.