Accelerating Scalar Conversion for Koblitz Curve Cryptoprocessors on Hardware Platforms

Sujoy Sinha Roy, Junfeng Fan and Ingrid Verbauwhede

Abstract

Koblitz curves are a class of computationally efficient elliptic curves where scalar multiplications can be accelerated using τNAF representations of scalars. However conversion from an integer scalar to a short τNAF is costly and thus restricts speed. In this paper we present acceleration techniques for the recently proposed scalar conversion hardware based on division by τ^2. Acceleration is achieved in two steps. First we perform computational optimizations to reduce the number of long subtraction operations during the conversion of scalar. This helps in reducing the number of integer adder/subtracter circuits from the critical paths of the conversion architecture. In the second step, we perform pipelining in the conversion architecture in such a way that the pipeline stages are always utilized. Due to bubble free nature of the pipelining, clock cycle requirement of the conversion architecture remains same, while operating frequency increases drastically. We present detailed experimental results to support our claims made in this paper.

I. Introduction

Elliptic curve cryptography (ECC) is the modern standard for asymmetric key cryptography as it provides more security per key bit compared to other asymmetric key cryptography standards. Security in ECC based cryptosystems is based on the difficulty of solving discrete logarithm problems over the elliptic curve groups (ECDLP). Elliptic curve scalar multiplication is the soul of any ECC processor. In scalar multiplication, a point P on an elliptic curve is multiplied by a large scalar k to get kP. The standard way to perform scalar multiplication is to use the *double and add* algorithm [1], [2], [3] where, point doubling operations are performed for every key bit

The authors are with the Department of Electrical Engineering, ESAT/COSIC, KU Leuven. Email : {Sujoy.SinhaRoy, Junfeng.Fan, Ingrid.Verbauwhede}@esat.kuleuven.be. This paper is under review in IEEE TVLSI.

and point additions are performed for every nonzero key bit. Both point addition and doubling operations are computation intensive due to the presence of complicated finite field operations. As the number of point additions is determined by the Hamming weight of the scalar, computation time of the scalar multiplication depends on the scalar k. Numerous works are present in literature on improving the scalar multiplication time by applying various acceleration techniques such as efficient representation of scalars, optimization of elliptic curve group operations, use of projective coordinate systems, and use of special computation friendly class of elliptic curves etc. Architectural optimizations depending on the implementation platforms add further acceleration [4], [5], [6], [7].

Koblitz curves [8] are a special class of elliptic curves, where the Frobenius endomorphism can be utilized to represent an integer scalar in a $\tau-$adic form. With this special type of representation of a scalar, costly point doubling operations are replaced by cheaper point squaring operations. Solinas extended the work and proposed $\tau-$adic non-adjacent-form known as τNAF [9]. However, the length of both $\tau-$adic and τNAF representations of a scalar are almost twice the length of a binary representation of the integer scalar. Increase in the length (thus increase in the number of point additions) becomes a negating factor in achieving the acceleration offered by the Frobenius endomorphism. Length reduction schemes were first introduced by Meier and Staffelback in [10] and were later improved by Solinas in [9]. The length reduction algorithms proposed by Solinas are efficient on software platforms, but are not amiable to hardware implementations due to the presence of multi-precision integer multiplications and divisions.

On the hardware side, very few research work exists in literature on designing efficient scalar conversion algorithms. In [11], [12], the conversions are performed in a software processor, while only the scalar multiplications are performed on dedicated hardwares. Such an approach is well suited when the scalar multiplier architecture is slow. However, the present speed records for scalar multiplications on hardware platforms have made such an approach a bottleneck [13], [7]. The first hardware implementation of conversion architecture was presented in [14]. In [15], [16], the implementations of Koblitz curve processors keep the conversion units in the hardware along with the scalar multipliers. Still the hardware versions of the converters are slow and have a large area count.

Brumley and Järvinen in [17] proposed efficient conversion architecture for hardware platforms. Their algorithm repeatedly divides the scalar by τ for m number of times in F_{2^m} to generate the length reduced scalar. Due to its sequential nature, the authors call the algorithm *lazy reduction*. The algorithm uses only integer addition/subtraction and shifting operations. No multi-precision divisions or multiplications are used. The authors present a hardware architecture for the lazy reduction [17] and then reuse the architecture to generate the τNAF from the reduced scalar.

An accelerated version of the lazy reduction, known as the *double lazy reduction* was proposed in [18] by Adikari, Dimitrov, and Järvinen. The authors observe that division by τ^2 is also cheap on hardware platforms. Repeated divisions by τ^2 are used instead of divisions by τ to reduce the number of iterations from m to nearly $m/2$ in F_{2^m}. Similar to [17], the hardware architecture for reduction is also reused to generate the τNAF from the reduced scalar. As divisions by τ^2 are performed, the hardware architecture is able to generate two consecutive τNAF digits in every clock cycle.

In this paper we propose acceleration techniques for the double lazy reduction algorithm [18]. We observe that several addition and subtraction operations can be eliminated during the scalar reduction and the τNAF generation operations. Subtraction or addition of the nonzero remainders are replaced by alteration of the low order bits in the operands, use of one's complement of the operands and by considering borrow/carry inputs in subtracter and adder circuits. We eliminate unnecessary subtractions from zero using iterative property of the conversion steps. Such optimizations help in reducing the number of integer adder and subtracter circuits from the critical paths of the conversion architecture without affecting the clock cycle requirement. With the proposed acceleration techniques, we achieve improvements in the computation time by atleast 12.5% and 17.9% compared to [18] for the fields $F_{2^{233}}$ and $F_{2^{283}}$ respectively

Next we perform efficient pipelining in the proposed conversion architecture. Using the iterative property of the conversion steps, we pipeline the conversion architecture in such a way that the pipelined stages are always utilized. Due to bubble free nature of the pipelined architecture, almost no clock cycles are wasted to satisfy the data dependencies between different pipeline stages. Our pipelined conversion architecture achieves acceleration by 35.5% and 40% compared to [18] for the fields $F_{2^{233}}$ and $F_{2^{283}}$ respectively.

The organization of the paper is : Section II has a brief mathematical background on the Koblitz curves and scalar conversion techniques. In Section III, computational optimizations for the double lazy reduction algorithm are discussed. Section IV shows optimizations during the τNAF generation process. A hardware architecture for the scalar conversion is designed in Section V. Efficient pipeline strategy for the conversion architecture is presented in Section VI. The section also describes a two stage pipelined architecture for the conversion hardware. Experimental results are presented in Section VII. The final section draws the conclusions.

II. Preliminaries

The Koblitz curves are a class of binary field elliptic curves and are of the following form.

$$E_a \ : y^2 + xy \ = \ x^3 + a \cdot x^2 + 1, \qquad a \in \{0, 1\}$$

An elliptic curve group over a binary field F_{2^m} is the union of all points $P(x, y)$ satisfying the curve equation and the point at infinity $\mathcal{O}$. We denote the group by $E_a(F_{2^m})$. The group operations are elliptic curve point addition and point doubling. In scalar multiplication, a base point $P(x, y)$ on an elliptic curve is multiplied by an integer scalar k. Scalar multiplication can be performed using the well known *double and add algorithm* [1] which uses the elliptic curve group operations (point addition and doubling). Complexity of the scalar multiplication is determined by the number of group operations required.

A. Frobenius Endomorphism

On Koblitz curves, the Frobenius map can be applied to reduce the number of elliptic curve group operations significantly during scalar multiplications. The Frobenius map $\tau : E_a(F_{2^m}) \rightarrow E_a(F_{2^m})$ is defined below.

$$\tau(\mathcal{O}) = \mathcal{O}, \qquad \tau P(x, y) = Q(x^2, y^2)$$

Application of the map τ on any point P squares the coordinates and gives another point Q on the curve. Computing the Frobenius map is an easy operation as it involves only squaring, which is cheap over binary fields [1], [19]. The map operator τ satisfies the relation $\tau^2 + 2 = \mu\tau$, where $\mu = (-1)^{1-a}$ depends on the elliptic curve.

Ring of polynomials in τ with integer coefficients is denoted by $\mathbb{Z}[\tau]$. For any polynomial $u_{l-1}\tau^{l-1} + \ldots + u_1\tau + u_0 \in \mathbb{Z}[\tau]$ with $u_i \in \{0, 1\}$, and any base point P on a Koblitz curve, we see the following relation.

$$[u_{l-1}\tau^{l-1} + \ldots + u_0]P = [u_{l-1}]\tau^{l-1}P + \ldots + [u_0]P \tag{1}$$

In the above equation, the base point P is multiplied by a scalar which is an element of $\mathbb{Z}[\tau]$. The scalar multiplication involves only point addition and point squaring operations. No point doubling operations are required.

Solinas in [9] proposed algorithms to convert integer scalars into polynomials in τ with coefficients $u_i \in \{0,1\}$ ($\tau-$adic form). He also showed that number of point additions can be reduced using non-adjacent form which is known as the τNAF of a scalar. Calculation of the τNAF from a scalar requires iterative divisions by τ. Here are two theorems related to the division by τ.

Theorem1 : Any element $\alpha = (d_0 + d_1\tau) \in \mathbb{Z}[\tau]$ is divisible by τ if and only if d_0 is even. The result of the division when stored in (d_0, d_1), becomes

$$(d_0, d_1) \leftarrow (\mu d_0/2 + d_1, -d_0/2).$$

Theorem2 : Any element $\alpha = (d_0 + d_1\tau) \in \mathbb{Z}[\tau]$ is divisible by τ^2 if and only if $d_0 \equiv 2d_1 \ (mod\ 4)$. The result of the division when stored in (d_0, d_1), becomes

$$(d_0, d_1) \leftarrow ((-d_0 + 2\mu d_1)/4, -(\mu d_0 + 2d_1)/4).$$

B. Length of τNAF and Reduction Schemes

The length l of the τNAF (or $\tau-$adic) for any integer scalar k is approximately $2log_2k$, which is almost double the length of the binary representation of the scalar. Expansion in length of the τNAF increases the number of point additions during scalar multiplication and thus restricts the acceleration achieved using the Frobenius endomorphism.

Length reduction schemes proposed by Solinas in [9] are based on the observation that $(\tau^m - 1)P(x,y) = \mathcal{O}$. For any scalar k, one can get the following relation.

$$k = \lambda(\tau^m - 1) + \gamma$$

Thus, for any point $P(x,y)$ on the Koblitz curve, we have $kP(x,y) = \gamma P(x,y)$. In the length reduction scheme [9], an integer scalar k is first reduced to $k \ (mod\ \delta)$, where $\delta = \frac{\tau^m - 1}{\tau - 1}$ and τNAF is generated from the reduced scalar. The maximum length of the τNAF generated is $m + a$ in F_{2^m}. Computation of the reduced scalar involves multi-precision integer division. In another reduction scheme [9], scalar is partially reduced to avoid multi-precision integer division at the cost of multiplication. Since multi-precision division and multiplication operations are complicated, implementations of the conversion algorithms on hardware platform are inefficient.

Algorithm 1: *Double Lazy Reduction*

```
Input: integer $k$
Output: reduced scalar $\gamma$
1  begin
2      $(a_0, a_1) \leftarrow (1, 0),\ (b_0, b_1) \leftarrow (0, 0),\ (d_0, d_1) \leftarrow (k, 0)$
3      for $i = 1$ to $(m-1)/2$ do
4          $u \leftarrow (d_0 - 2d_1)\ mod\ 4$
5          $u_0 \leftarrow u\ mod\ 2,\ u_1 \leftarrow \lfloor u/2 \rfloor$
6          $d_0 \leftarrow d_0 - u_0,\ d_1 \leftarrow d_1 - u_1$
7          $(d_0, d_1) \leftarrow ((-d_0 + 2\mu d_1)/4, -(\mu d_0 + 2d_1)/4)$
8          if $u > 0$ then
9              $b_0 \leftarrow (b_0 + u_0 a_0 - 2u_1 a_1)$
10             $b_1 \leftarrow\ b_1 + u_0 a_1 + u_1(a_0 + \mu a_1)$
11         end
12         $(a_0, a_1) \leftarrow (-2(a_0 + \mu a_1),\ \mu a_0 - a_1)$
13     end
14     if $d_0 \equiv 1 (mod\ 2)$ then
15         $u \leftarrow 1,\ d_0 \leftarrow d_0 - 1$
16         $(b_0, b_1) \leftarrow (b_0 + a_0, b_1 + a_1)$
17     end
18     $(d_0, d_1) \leftarrow (\mu d_0/2 + d_1, -d_0/2)$
19     $\gamma \leftarrow (b_0 + d_0, b_1 + d_1)$
20 end
```

In [17], Brumley and Järvinen presented the *lazy reduction* algorithm, where the scalar k is repeatedly divided by τ for m number of times to get the following relation.

$$\begin{aligned} k &= (d_0 + d_1\tau)\tau^m + (b_0 + b_1\tau) \\ &= (d_0 + d_1\tau)(\tau^m - 1) + (b_0 + d_0) + (b_1 + d_1)\tau \\ &= \lambda(\tau^m - 1) + \gamma \end{aligned}$$

The authors use γ as the reduced scalar and show that the τNAF generated from γ has length at most $m + 4$ in F_{2^m}. Since division by τ is a simple operation (Theorem 1), the algorithm is suitable for hardware platforms.

In [18], Adikari, Dimitrov and Järvinen proposed an accelerated version of the lazy reduction which is called the *double lazy reduction*. The authors observe that division by τ^2 is also easy to perform (Theorem 2). In the double lazy reduction, the scalar is iteratively divided by τ^2 for $(m-1)/2$ number of times. At the end, one division by τ is performed to obtain the reduced scalar. Clock cycle requirement for the scalar reduction scheme reduces to nearly half compared to the lazy reduction. The computational steps in the double lazy reduction are shown in Algorithm 1. We advise the readers of this paper to study the double lazy reduction algorithm from [18], as this will be helpful to understand the acceleration techniques proposed in our paper.

III. Improved Reduction Algorithm

In this section, we propose computational optimizations for the double lazy reduction algorithm to reduce the number of long addition and subtraction operations during the scalar reduction steps. Improvements are achieved following the steps discussed below. Throughout the discussion we consider $\mu = -1$. Similar techniques can also be applied when $\mu = 1$ (shown in the appendix).

A. *Elimination of Long Subtractions for Nonzero Remainders*

In line 6 of Algorithm 1, remainders u_0 and $u_1 \in \{0, 1\}$ are subtracted from d_0 and d_1. We observe that the subtractions are *easy* in some cases. For example, when $d_0 \equiv 1 \ (mod\ 4)$ and $2d_1 \equiv 0 \ (mod\ 4)$ (i.e. $u_0 = 1$ and $u_1 = 0$), the subtraction of u_0 from d_0 is equivalent to changing the least significant bit of d_0 from 1 to 0. Hence, in this case the long subtraction can be replaced by a bit alteration. However, when carry propagations are involved with long subtractions, alteration of few specific bits do not work as a replacement. For example, when $d_0 \equiv 3 \ (mod\ 4)$ and $2d_1 \equiv 0 \ (mod\ 4)$ (i.e. when $u_0 = 1$ and $u_1 = 1$), a long subtraction appears. Use of signed remainder set u_0 and $u_1 \in \{0, \pm 1\}$ helps to a certain extent in eliminating the long subtractions of nonzero remainders for such cases. Table I shows how the signed remainders are generated during the reduction steps depending on the low bits of d_0 and $2d_1$. It can be noticed from the table that, except Case 4, subtractions of the u_0 and u_1 from d_0 and d_1 involve no carry propagation and thus can be replaced by alterations of the low bits in d_0 and d_1.

For Case 4, if we perform the subtraction of $u_0 = -1$ in line 7 of Algorithm 1 instead of line

TABLE I

SIGNED REMAINDERS DURING REDUCTION OF SCALAR

Cases	$d_0\ (mod\ 4)$	$2d_1\ (mod\ 4)$	u_0	u_1
1	0	0	0	0
2	1	0	1	0
3	2	0	0	-1
4	3	0	-1	0
5	0	2	0	1
6	1	2	-1	0
7	2	2	0	0
8	3	2	1	0

6 (i.e. we put $d_0 + 1$ in place of d_0), then we have the following observation.

$$\begin{aligned} d_0 &\leftarrow -\frac{2d_1 + (d_0 + 1)}{4} \\ d_1 &\leftarrow -\frac{2d_1 - (d_0 + 1)}{4} \end{aligned}$$

This is equivalent to taking carry/borrow inputs in the adder/subtracter circuits during the computations of d_0 and d_1. This is shown below.

$$\begin{aligned} d_0 &\leftarrow -\frac{2d_1 + d_0 + (Carry\ input\ = \ 1)}{4} \\ d_1 &\leftarrow -\frac{2d_1 - d_0 - (Borrow\ input\ = \ 1)}{4} \end{aligned}$$

From the observations presented in this subsection, we draw the conclusion that the long subtractions of the nonzero remainders can be eliminated by changing the low order bits of d_0 and d_1 or by considering carry/borrow inputs to the adder/subtracter circuits.

B. Elimination of Subtractions from Zero

In line 7 of Algorithm 1, a subtraction from zero is required for d_0 after computing $2d_1 + d_0$ (Equation 2).

$$(d_0, d_1) \leftarrow (-\frac{2d_1 + d_0}{4}, -\frac{2d_1 - d_0}{4}) \tag{2}$$

We eliminate the subtraction from zero using the following scheme. Instead of Equation 2, we compute Equation 3.

$$(d_0, d_1) \quad \leftarrow \quad (\frac{2d_1 + d_0}{4}, \frac{2d_1 - d_0}{4}) \tag{3}$$

The results from Equation 2 and 3 have opposite signs, but same magnitudes. So, when Equations 2 and 3 are applied iteratively for an even number of times, the results of the equations are same. For an odd number of iterations, the results only have opposite signs.

We use the iterative property (for-loop) present in the reduction algorithm to eliminate the subtractions from zero. We replace the computation in line 7 by Equation 3. Thus (d_0, d_1) has wrong sign after any odd number of iterations and correct sign after any even number of iterations of the for-loop. The loop iterates for $(m-1)/2$ number of times. So, when $(m-1)/2$ is odd, only one subtraction from zero is required at the end to make (d_0, d_1) of correct sign. We can also compensate this subtraction in line 18 or 19 by performing subtractions in place of additions or vice-versa.

The same trick is applied to eliminate the subtractions from zero during the computation of (a_0, a_1) in line 12 of Algorithm 1. Instead of computing Equation 4

$$(a_0, a_1) \leftarrow (-2(a_0 - a_1),\ -(a_0 + a_1)) \tag{4}$$

we compute Equation 5.

$$(a_0, a_1) \leftarrow (2(a_0 - a_1),\ (a_0 + a_1)) \tag{5}$$

After completion of the for-loop, one subtraction from zero is required to make the signs correct for (a_0, a_1) when $(m-1)/2$ is odd. This subtraction from zero can be eliminated if we compute Equation 6

$$(b_0, b_1) \leftarrow (b_0 - a_0, b_1 - a_1) \tag{6}$$

in line 16 of Algorithm 1.

Since the remainders are generated by observing the low order bits of d_0 and d_1 (Table I), use of (d_0, d_1) which has wrong sign should be justified for the correctness of the reduction algorithm. Let after any odd number of iterations of the for-loop in Algorithm 1, we have the pairs $(a_{i,0}, a_{i,1})$, $(b_{i,0}, b_{i,1})$ and $(d_{i,0}, d_{i,1})$. Since wrong sign is assigned to $(d_{i,0}, d_{i,1})$ after any odd number of iterations, we have the following relation.

$$k = -(d_{i,0} + \tau d_{i,1})\tau^{2i} + (b_{i,0} + \tau b_{i,1}) \tag{7}$$

In the next iteration, remainders u_0 and u_1 are generated from $d_{i,0}$ and $d_{i,1}$ (instead of the correct values -$d_{i,0}$ and -$d_{i,1}$). If $(d_{i+1,0}, d_{i+1,1})$ are the quotients after the division by τ^2, then we have the following relation between $(d_{i,0}, d_{i,1})$ and $(d_{i+1,0}, d_{i+1,1})$.

$$(d_{i,0} + \tau d_{i,1}) - (u_0 + \tau u_1) = -(d_{i+1,0} + \tau d_{i+1,1})\tau^2 \tag{8}$$

After plugging in Equation 8 in Equation 7, we get the following equation.

$$k = (d_{i+1,0} + \tau d_{i+1,1})\tau^{2i+2} - (u_0 + \tau u_1)\tau^{2i} + (b_{i,0} + \tau b_{i,1}) \tag{9}$$

From Equation 9, we find that the actual remainders are $(-u_0, -u_1)$. Interestingly, after any odd number of iterations, wrong sign is also assigned to (a_0, a_1). So, we have $(a_{i,0}, a_{i,1}) = -\tau^{2i}$. Since, both (u_0, u_1) and (a_0, a_1) are of same sign in any iteration, computation of (b_0, b_1) is insensitive to the wrong sign of operands. Thus, $(b_{i+1,0}, b_{i+1,1})$ has the following relation.

$$(b_{i+1,0}, b_{i+1,1}) \leftarrow (u_0 + \tau u_1)(a_{i,0} + \tau a_{i,1}) + (b_{i,0} + \tau b_{i,1})$$

This justifies correctness of the proposed reduction algorithm during assignment of wrong sign to the variables (d_0, d_1) and (a_0, a_1).

Throughout this section we have discussed how the number of long addition/subtraction operations can be reduced during the scalar reduction steps. Our proposed improvements over the double lazy reduction are described in Algorithm 2. We see that only one addition or subtraction operations are performed during the computations of d_0, d_1, a_0, a_1 and b_0 in every iterations. For b_1, at most two addition/subtraction operations are performed per iteration. Thus, if implemented on hardware platform, critical path contains only one adder/subtracter circuit of width $m+1$ bit. In the previous reduction architectures [17], [18], critical paths are through two cascaded adder and subtracter circuits of data width $m+1$. Since integer adder and subtracter circuits have large delay due to carry propagation, removal of such circuits from critical paths help in improving the delay of conversion architecture.

In the next section, we further look into τNAF generation algorithm and discuss how long subtractions of nonzero remainders can be eliminated during the τNAF generation steps.

Algorithm 2: *New Reduction Algorithm*

```
Input: integer $k$
Output: reduced scalar $\gamma$
1  begin
2      $(a_0, a_1) \leftarrow (1, 0),\ (b_0, b_1) \leftarrow (0, 0),\ (d_0, d_1) \leftarrow (k, 0)$
3      /* Iterative divisions by $\tau^2$ start here */
4      for $i = 1$ to $(m-1)/2$ do
5          $u \leftarrow (d_0 - 2d_1)\ mod\ 4$
6          $(u_0, u_1) \leftarrow Table\ 1$
7          $(d_0, d_1) \leftarrow AlterLowBits(d_0, d_1)$
8          if $Case\ 4\ is\ True$ then
9              $(B, C) \leftarrow (1, 1)$ /* Borrow and Carry Inputs */
10         end
11         else
12             $(B, C) \leftarrow (0, 0)$
13         end
14         $(d_0, d_1) \leftarrow ((2d_1 + d_0 + C)/4, (2d_1 - d_0 - B)/4)$
15         if $u > 0$ then
16             $b_0 \leftarrow (b_0 + u_0a_0 - 2u_1a_1)$
17             $b_1 \leftarrow (b_1 + u_0a_1 + u_1(a_0 - a_1))$
18         end
19         $(a_0, a_1) \leftarrow (2(a_0 - a_1),\ a_0 + a_1)$ ;
20     end
21     /* Iterative divisions by $\tau^2$ finish here */
22     if $d_0 \equiv 1(mod\ 2)$ then
23         $d_0 \leftarrow AlterLeastBit(d_0)$
24         if $\frac{m-1}{2} \equiv 0(mod\ 2)$ then
25             $(b_0, b_1) \leftarrow (b_0 + a_0, b_1 + a_1)$
26         end
27         else
28             $(b_0, b_1) \leftarrow (b_0 - a_0, b_1 - a_1)$
29         end
30     end
31     $(d_0, d_1) \leftarrow ((2d_1 - d_0)/2, d_0/2)$ /* Final division by $\tau$ */
32     if $\frac{m-1}{2} \equiv 0(mod\ 2)$ then
33         $\gamma \leftarrow (b_0 + d_0, b_1 - d_1)$
34     end
35     else
36         $\gamma \leftarrow (b_0 - d_0, b_1 + d_1)$
37     end
38 end
```

TABLE II

NAF GENERATION FOR $\mu = -1$

Cases	$d_0(mod 4)$	$2d_1(mod 4)$	$d_0(mod 8)$	$2d_1(mod 8)$	r_0	r_1
1	0	0			0	0
2	1	0			1	0
3.A	2	0	2	0	0	1
3.B	2	0	6	0	0	-1
3.C	2	0	2	4	0	-1
3.D	2	0	6	4	0	1
4	3	0			-1	0
5.A	0	2	0	2	0	1
5.B	0	2	4	2	0	-1
5.C	0	2	0	6	0	-1
5.D	0	2	4	6	0	1
6	1	2			-1	0
7	2	2			0	0
8	3	2			1	0

IV. IMPROVED DOUBLE DIGIT τNAF GENERATION

In [18], two consecutive τNAF digits are generated in a single step from the reduced scalar $d_0 + \tau d_1$ by performing divisions by τ^2. The authors call the NAF as *double τNAF*. Table II shows how the consecutive τNAF digits r_0 and r_1 are generated by observing the low order bits of d_0 and d_1. Similar to Section III, we eliminate the subtractions of nonzero remainders from d_0 and d_1 during the τNAF generation process.

From Table II, we see that for the cases 2, 3.B, 3.C, 3.D, 5.A, 5.B, 5.D, 6 and 8, the subtractions of nonzero remainders from d_0 or d_1 affect only the low order bits of d_0 and d_1. For the above cases, the long subtractions are replaced by cheaper bit alterations in d_0 and d_1. Subtraction of $r_0 = -1$ in Case 4 in Table II can be handled in the same way we did for Case 4 in Table I (Section III-A).

In Case 3.A, the subtraction of $r_1 = 1$ from d_1 involves borrow propagation and thus may affect all the bits of d_1. If we incorporate this subtraction in the next step where we perform the division by τ^2, then by putting $d_1 - 1$ in place of d_1 in Equation 2, we have the following observation.

$$\begin{aligned}(d_0, d_1) &\leftarrow (-\frac{2(d_1-1)+d_0}{4}, -\frac{2(d_1-1)-d_0}{4}) \\ &\leftarrow (-\frac{2d_1+(d_0-2)}{4}, -\frac{2d_1-(d_0+2)}{4}) \end{aligned} \tag{10}$$

Thus we find that the subtraction of r_1 from d_1 is equivalent to the addition or subtraction of two with d_0. As $d_0 \equiv 2 \ (mod\ 8)$, the subtraction or addition of 2 changes only the three low bits of d_0.

In Case 5.C, the subtraction of $r_1 = -1$ from d_1 involves carry propagation. When we put $d_1 + 1$ in place of d_1 in Equation 2, we have the following observation.

$$\begin{aligned}(d_0, d_1) &\leftarrow (-\frac{2(d_1+1)+d_0}{4}, -\frac{2(d_1+1)-d_0}{4}) \\ &\leftarrow (\frac{2\bar{d}_1-d_0}{4}, \frac{2\bar{d}_1+d_0}{4}) \end{aligned} \tag{11}$$

So, using one's complement of d_1 in Case 5.C, we eliminate the long subtraction of r_1 from d_1. Computing one's complement in hardware platform is easy as all the bits of d_1 can altered in parallel.

Algorithm 3 describes the steps of the new τNAF generation technique. Only one addition or subtraction operations are performed on d_0 and d_1 during division by τ^2 in any iteration. Thus, for the τNAF generation part of the scalar conversion, presence of only one adder/subtracter circuits in the critical paths of d_0 and d_1 is sufficient.

V. Hardware Architecture

The hardware architecture for performing scalar conversion using the proposed acceleration techniques is shown in Figure 1. The architecture is constructed for the curve parameter $\mu = -1$. Similar to [17], [18], our conversion architecture is designed to be used for the scalar reduction and the double digit τNAF generation. Such resource sharing helps to keep the design compact.

In any iteration of Algorithm 2 and 3, the variables a_0, a_1, b_0, b_1, d_0 and d_1 have data dependencies on their values in the previous iteration. So, storage registers are used in the architecture for each of the variables. Data paths of the variables are kept in parallel to each other as there are no immediate data dependencies between them in any particular iteration.

Algorithm 3: *New τNAF Generation Algorithm*

Input: Reduced Scalar $\gamma = d_0 + \tau d_1$

Output: τNAF(γ)

```
begin
    $S \leftarrow \langle\rangle$                /* Used to store $\tau$NAF */
    $Sign \leftarrow 0$            /* Used to keep sign of $(d_0, d_1)$ */
    while $d_0 \neq 0$ or $d_1 \neq 0$ do
        $(r_0, r_1) \leftarrow Table\ 2$
        $(d_0, d_1) \leftarrow AlterLowBits(d_0, d_1)\ in\ Section\ IV$
        $(B, C) \leftarrow (0, 0)$          /* Borrow and Carry Inputs */
        if $Sign = 1$ then
            $(r_0, r_1) \leftarrow (-r_0, -r_1)$
        end
        $Prepend\ (r_1, r_0)\ to\ S$          /* $\tau$NAF digits */
        if $Case\ 4\ True$ then
            $(B, C) \leftarrow (1, 1)$
        end
        if $Case\ 5.C\ True$ then
            $(d_0, d_1) \leftarrow (\frac{2\bar{d}_1 - d_0}{4}, \frac{2\bar{d}_1 + d_0}{4})$
            $Sign \leftarrow Sign$
        end
        else
            $(d_0, d_1) \leftarrow (\frac{2d_1 + d_0 + C}{4}, \frac{2d_1 - d_0 - B}{4})$
            $Sign \leftarrow 1 \oplus Sign$
        end
    end
end
```

The component *Bit Alteration and Remainder Generation* is a combinational circuit which scans the low order three bits of d_0 and two bits of d_1. Remainder digits are generated as per Table I and II during the reduction and the τNAF generation phases of the scalar conversion. This component also performs the subtractions of the nonzero remainders from d_0 and d_1 using the bit alteration technique discussed in Section III and IV. Input signal *mode* is used to distinguish between the scalar reduction and the τNAF generation phases of the conversion. After a subtraction of u_0 from d_0 as bit alteration, two different outputs $d_0\#A1$ and $d_0\#A2$ are generated for the two different data paths through A1 and A2 adder/subtracter circuits. This happens because of Case 3.A in Table II, where 2 is subtracted and added for the two different data paths through A1 and A2 respectively.

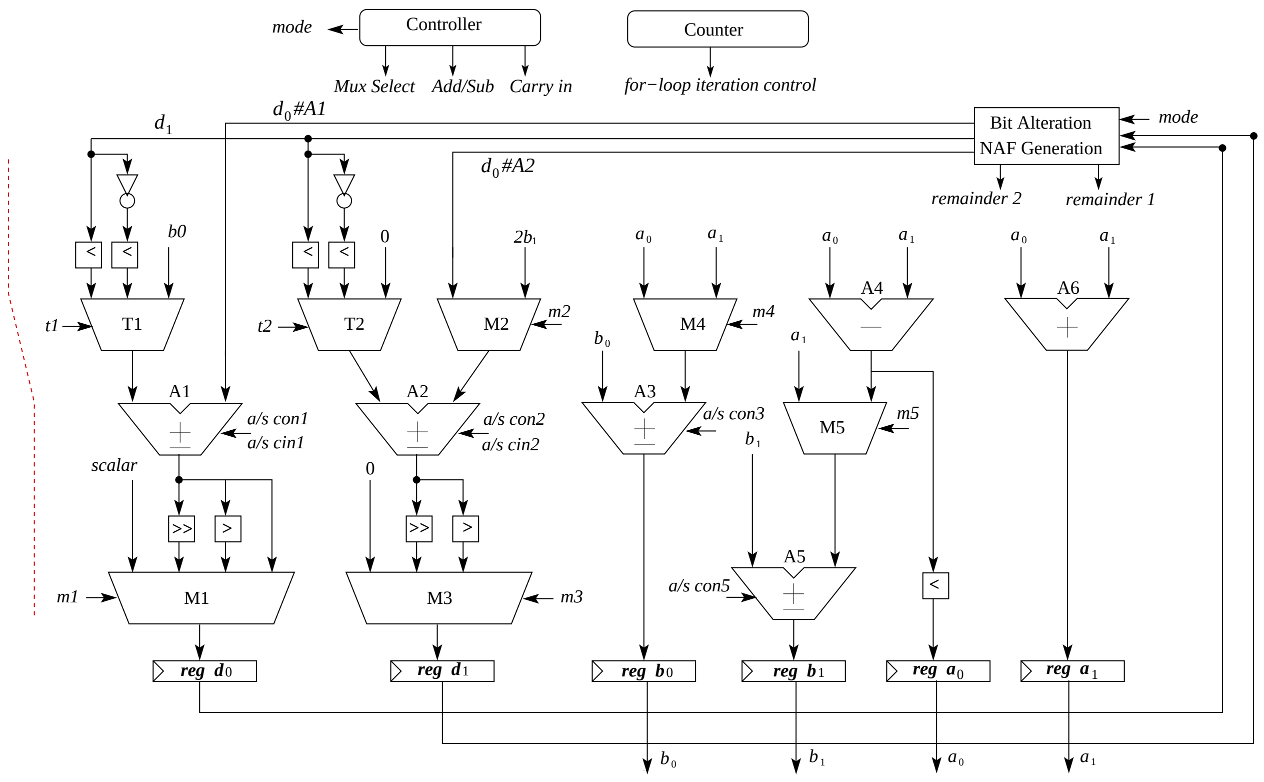

Fig. 1. Koblitz Curve Scalar Conversion Architecture for $\mu = -1$

A finite state machine (FSM) is used to drive the conversion hardware according to Algorithm 2 during the scalar reduction and Algorithm 3 during the double digit τNAF generation parts. During the reduction phase, all data paths in the conversion hardware remain active. A counter is used to perform division by τ^2 for $(m-1)/2$ number of times. After completion of the iterative divisions by τ^2, the FSM moves to the state where only one division by τ is performed. In the next state, the final step of reduction (Line 33 or 36 of Algorithm 2) is performed using the data paths for d_0 and d_1. The reduced scalar is kept in d_0 and d_1.

After completion of the scalar reduction, the FSM moves to the double digit τNAF generation state. During this process, only the data paths for d_0 and d_1 remain active. The sign of d_0 and d_1 are taken into account in a one-bit register $sign$ (Algorithm 3). When the sign of d_0 and d_1 are correct, the remainders generated by the *Bit Alteration and Remainder Generation* unit are the two consecutive τNAF digits. For, the other case, sign of the remainders are changed and then expressed as the two consecutive τNAF digits.

A controller is used to generate the control signals for the multiplexers and the adder/subtracter circuits. The control block also generates the carry and borrow inputs for the adder/subtracter circuits A1 and A2 when required. In the figure, T1 and T2 are special categories of multiplexers which produce x, $\bar{x}$ and y from the inputs x and y as per equation $((x \oplus s_0) \cdot \bar{s_1})|(y \cdot s_1)$ when the selection inputs (s_1, s_0) are 00, 01 and 10 respectively. For LUT base FPGA platforms, this special construction for T1 and T2 achieves better LUT utilization [20] and thus saves area. The counter circuit is used to calculate the number of τNAF digits generated. Completion of the τNAF generation is indicated when $m + 4$ number of τNAF digits are generated.

The critical path of the conversion architecture is indicated by the dotted line. The delay of the architecture is determined by a single $m + 1$ bit integer adder/subtracter circuit present in the critical path. Since the integer adders are slow, even after the computational optimizations mentioned in this paper, the proposed conversion architecture do not match the speed of the binary field primitives used in a Koblitz curve processor. To increase speed of the conversion architecture, we implement pipelines in the conversion architecture. The next section discusses how pipeline helps in achieving high operating frequency without affecting the clock cycle requirement.

VI. Pipelining the Conversion Hardware

When we integrate the conversion architecture with the binary field primitives of a Koblitz curve scalar multiplier, large delay of the integer adders restricts the operating frequency of the scalar multiplier. Use of faster adder circuits increase frequency at the cost of area. However for long operand size, such adders are also slower compared to the binary field primitives specially when pipelining is applied over the field primitives. It should be noted that pipelining is a common practice in designing fast elliptic curve scalar multipliers over binary fields [21], [22], [23], [6]. In this section, we propose a solution to this problem by applying pipeline strategy in the conversion architecture.

A. Pipelining Iterative Addition and Subtraction Operations

The central operations in the scalar conversion hardware are additions and subtractions. To pipeline the conversion hardware, we focus on pipelining the addition and subtraction operations in every iterations. Before pipelining the complicated conversion hardware, we discuss pipelining of the iterative addition and subtraction operations with a simple example. Let two variables c_0 and c_1 have some initial values and are later updated iteratively as per the following equations.

$$\begin{aligned} c_0 &\leftarrow c_0 + c_1 \\ c_1 &\leftarrow c_0 - c_1 \end{aligned} \tag{12}$$

Let the data width for both c_0 and c_1 are maximum m. Thus, to perform iterative addition and subtraction operations as per the above equations, one adder and one subtracter of width m bits are required. The two stage pipelined architecture for computation of c_0 and c_1 is shown in Figure 2. The adder and subtracter circuits are split in two equal stages of width $m/2$ by putting registers in the carry and borrow propagation paths.

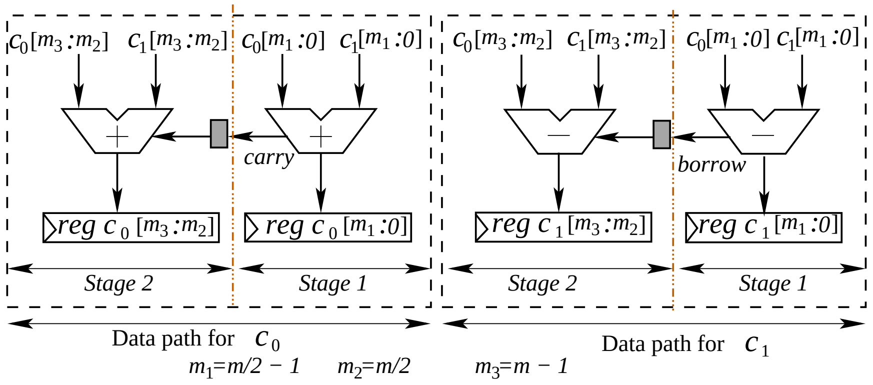

Fig. 2. Two stage pipelined data path for iterative addition and subtraction

Computations in the stage 2 have data dependencies on the stage 1 by the carry and borrow outputs from the stage 1. Due to this dependency, computations in the stage 2 are delayed by one clock cycle. Timing diagram of the computational steps in the two stages are described in Figure 3 for the first five clock cycles. Iteration numbers are indicated by the superscripts. Computations in the stage 1 and 2 run in parallel in any clock cycle (except the first one). As per the timing diagram, first four iterations of the consecutive addition and subtraction operations complete after the fifth clock cycle. It is straight forward to understand that for I number of iterations (Equation 12), the two stage architecture takes $I + 1$ clock cycles. In comparison, a non-pipelined architecture takes I number of clock cycles to finish I rounds. The advantage of the pipelined architecture is in the reduction of overall delay by half (ideally) compared to the non-pipelined architecture at the cost of only two flip-flops and one clock cycle.

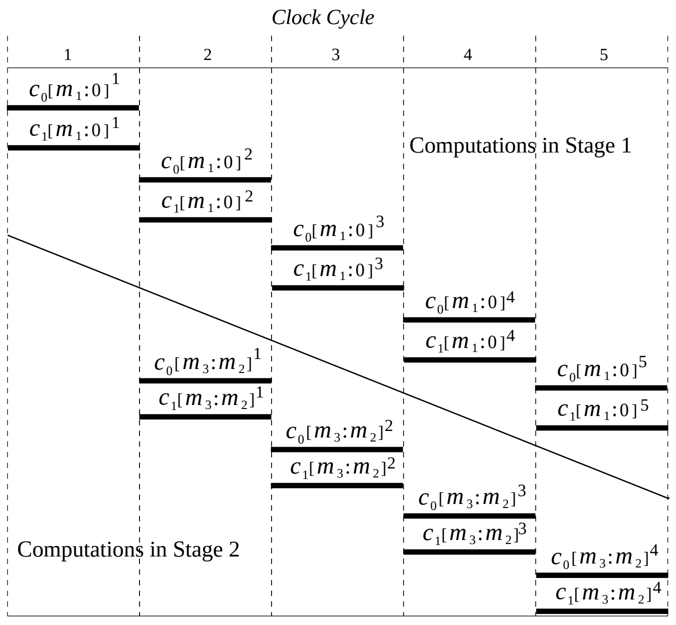

Fig. 3. Timing diagram of the two stage pipelined data path for iterative addition and subtraction

B. Pipelining the conversion architecture

We apply the same concept in pipelining the conversion hardware. However data dependencies get more complicated due to the presence of shifter circuits and due to the different data widths of the registers present in the conversion architecture (Figure 1). Additionally, synchronization between the parallel data paths is essential to maintain functional correctness of the conversion hardware.

Figure 4 shows the two stage pipelined conversion architecture for $\mu = -1$. Data paths are split in almost symmetric stages to achieve best operating frequency for the two stage architecture. In the figure, suffix $\#1$ and $\#2$ indicate the parts of different components in the first and second stages of the pipelined architecture respectively. The registers a_0, a_1, b_0, b_1, d_0 and d_1 are split into two equal halves between the two stages. The lower half of a register gets updated by the

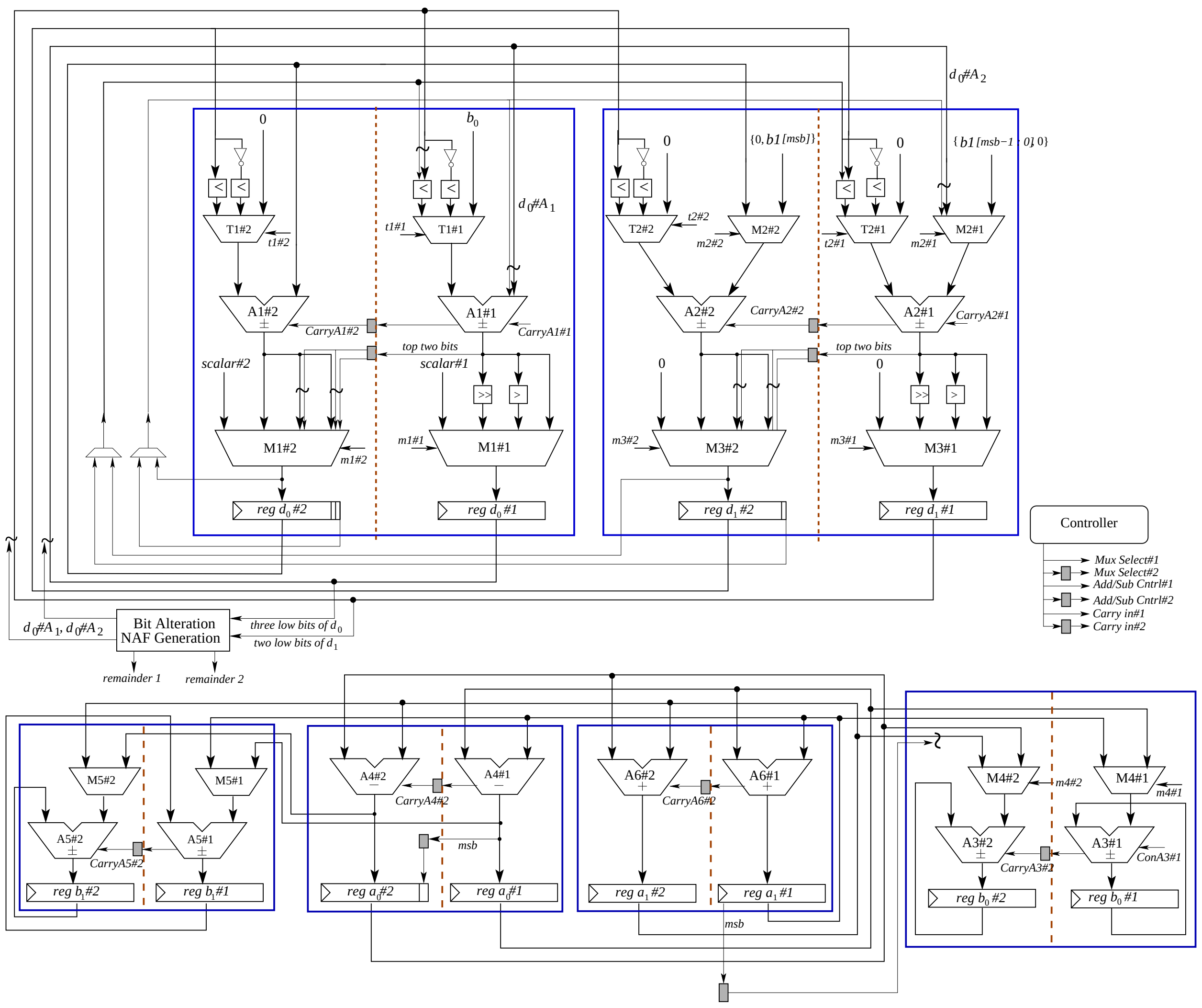

Fig. 4. Pipelined Conversion Architecture for $\mu = -1$

computations in the first stage of the pipeline and the upper half gets updated by the second stage of the pipeline. In the first stage of the data path for d_0, the adder/subtracter A1#1 has width $\frac{m+1}{2} + 2$ bits due to the presence of division by four and two (right shift) circuits. After a division by four, the most significant bit, i.e, the $(\frac{m+1}{2} + 2)^{th}$ bit of the output from A1#1 is written into the $(\frac{m+1}{2})^{th}$ bit, i.e. the most significant bit position of $d_0\#1$. To match the data width requirement for A1#1, the bits from position $\frac{m+1}{2} + 1$ and $\frac{m+1}{2} + 2$ of d_0 are needed. However, these two bits belong to the upper half of d_0 register ($d_0\#2$) present in the second stage. As the second stage lags the first stage by one clock cycle, we can not use the two least significant bits from $d_0\#2$. The output from the multiplexer M1#2 leads by one clock cycle over $d_0\#2$ (because in any positive clock cycle transition, data in the output of M1#2 gets written into $d_0\#2$). We perform data forwarding of the two least significant bits from the output of the multiplexer M1#2 to the input of A1#1. Similar data forwarding strategies are applied in the first stage of data path for d_1 register due to the presence of division by two and four circuits. Merging of wires are indicated by the horizontal and vertical $\sim$ symbols in Figure 4.

Control signals for the adder/subtracter circuits and the multiplexers present in the second stage are lagged by one clock cycle to maintain the lag of the second stage in the pipelined data path. Data paths for a_0, a_1, b_0 and b_1 are also split in two stages to maintain synchronization between all parallel data paths present in the conversion hardware. The second stage of the data path for b_0 has data dependency on the bit position $\lceil \frac{m}{2} \rceil$ of register a_1. This particular bit is the most significant bit of the register $a_1\#1$. Due to the lag of the second stage of register b_0, we apply data lagging of the required bit using an edge triggered flip-flop.

VII. Experimental Results

We have evaluated the proposed acceleration techniques for the NIST recommended Koblitz curves [24] K-233 and K-283 on Xilinx Virtex 4 FPGA xcvlx200-11ff1513. All these curves have $\mu = -1$ and support the present security standards. Table III shows performance results of the proposed pipelined and non-pipelined conversion architectures. Results are obtained from Xilinx ISEv12.2 tool after place and route analysis with optimization for speed. Comparisons with other reported conversion architectures are also presented in the table.

The conversion time is the total time required for the scalar reduction and the complete τNAF generation. Our non-pipelined conversion architecture and the architecture in [18] have same clock cycle requirement of $m + 6$ for the complete scalar conversion in F_{2^m}. The pipelined conversion architecture takes only two extra clock cycles and thus requires $m + 8$ clock cycles in F_{2^m}. In [17], the conversion architecture uses division by τ and takes $2m + 7$ clock cycles to complete the conversion of scalar.

Frequency of the conversion architectures depend on the type of integer adders used and also on the optimization done by the synthesis tool. The computational optimizations proposed in the paper consider use of generic adder and subtracter circuits in the conversion architecture. The results shown in the table for our architectures were obtained using carry propagation adder and

TABLE III

COMPARISON OF OUR PROPOSED CONVERSION HARDWARE WITH PUBLISHED RESULTS ON XILINX VIRTEX 4 FPGA

Work	Curve	Slices	Freq MHz	Reduction Time (μs)	Conversion Time (μs)
Brumley [17]	K-233	1380	76	3	6
Adikari [18]		1777	75.3	1.55	3.1
Non-pipelined*		1661	81.5	1.4	2.8
Pipelined*		1582	119	1.0	2.0
Brumley [17]	K-283	1671	65.9	4.3	8.6
Adikari [18]		1998	65.1	2.2	4.4
Non-pipelined*		1910	70.2	2.0	4.1
Pipelined*		1814	107	1.3	2.6

subtracter circuits. However, no description about the type of adder and subtracter circuits were found in [17] and [18]. For fare comparison of the operating frequencies, we have implemented a small circuit which is same as the data path for d_0 register in [18] and uses carry propagation subtracter circuits [1]. With the same optimization parameters, we achieved frequencies 72MHz and 59.5MHz for the fields $F_{2^{233}}$ and $F_{2^{283}}$ respectively. When we consider implementation of the conversion hardware in [18] using carry propagation adder and subtracter circuits, operating frequencies will be limited by the above mentioned values due to the increased circuit complexities. Thus, under this fare comparison, our non-pipelined conversion architectures achieve improvement in frequencies by atleast 12.5% and 17.9% for the mentioned fields respectively. Area requirements of the non-pipelined conversion architectures are slightly lesser than the architectures in [18]. The proposed computational optimizations reduce the number of adder and subtracter circuits but increase the number of multiplexers (T1 and T2).

Use of the pipeline strategy helps in improving operating frequency drastically. Absence of bubbles in the pipelined data path keeps the clock cycle requirement almost same as the non-pipelined architecture. We achieve 35.5% and 40% improvements in the overall conversion time compared to [18] for the curves K-233 and K-283 respectively. It is interesting to observe that the pipelined architectures have lesser area compared to the non-pipelined architectures.

[1]The circuit has inputs u_0, u_1, k and d_1. The most significant bit of d_0 register is the output from the circuit. The critical path consists of two $m+1$ bit subtracter circuits and one 4:1 multiplexer.

The pipelines are implemented by placing only one flip-flop between the two stages of any data path. Thus, the pipelined architecture requires 10 extra flip-flops compared to the non-pipelined architecture. With this small extra cost, the critical paths of the pipelined conversion architecture are nearly nearly half of the non-pipelined architecture. Due to shorter critical paths, the optimization tool performs lesser number of logic replications during synthesis of the pipelined design to meet timing constraints [25]. This results in lesser area requirement for the pipelined architectures.

VIII. Conclusion

The paper presents acceleration techniques for scalar conversions required in the Koblitz curve based cryptoprocessors. Acceleration is achieved by reducing the number of costly addition and subtraction operations during the reduction and the τNAF generation steps. Optimization in the number of addition and subtraction operations reduces the critical paths of the conversion architecture and thus helps in achieving higher operating frequency. Further, the paper proposes architecture level improvements using pipeline strategy. Efficient pipelines are implemented for the conversion architecture which are free from bubbles. This improves the operating frequency of the architecture without affecting the clock cycle requirement.

With the improvements proposed in this paper, the conversion architecture achieves high operating frequency and thus becomes suitable for integration with the binary field components of a Koblitz curve scalar multiplier. Due to cost effectiveness of the proposed pipeline strategy, it is expected that more number of pipelined stages will increase the operating frequency from the present values. The number of stages in the conversion architecture should be fixed to match the speed of the binary field primitives present in the Koblitz curve processor.

References

[1] D. Hankerson, A.J. Menezes and S.A. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., 2003.

[2] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[3] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman & Hal, 2005.

[4] A. R.-M. R. Azarderakhsh, “High-Performance Implementation of Point Multiplication on Koblitz Curves,” *Circuits and Systems-II, IEEE Transactions on*, pp. pre–print, 2012.

[5] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, “A Parallel Architecture for Koblitz Curve Scalar Multiplications on FPGA Platforms,” in *Digital System Design (DSD), 2012 15th Euromicro Conference on*. IEEE, 2012, pp. 553–559.

[6] S. Roy, C. Rebeiro, and D. Mukhopadhyay, “Theoretical Modeling of Elliptic Curve Scalar Multiplier on LUT-Based FPGAs for Area and Speed,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 901–909, 2013.

[7] K.U. Järvinen, and J.O. Skyttä,, “Fast Point Multiplication on Koblitz Curves: Parallelization Method and Implementations,” *Microprocessors and Microsystems*, vol. 33, pp. 106–116, 2009.

[8] N. Koblitz, “CM Curves with Good Cryptographic Properties,” *Proc. Crypto ’91*, pp. 279–287, 1991.

[9] J.A. Solinas, “Effecient Arithmetic on Koblitz Curves,” *Design, Codes and Cryptography*, vol. 19, pp. 195–249, 2000.

[10] W. Meier and O. Staffelbach, “Efficient Multiplication on Certain Nonsupersingular Elliptic Curves,” *Advances in Cryptology-CRYPTO ’92*, pp. 333–344, 1992.

[11] S. Okada, N. Torii, K. Itoh, and M. Takenaka, “Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA,” *In Proc. Intl Workshop Cryptographic Hardware and Embedded Systems (CHES 00)*, pp. 25–40, 2000.

[12] J. Lutz and A. Hasan, “High Performance FPGA based Elliptic Curve Cryptographic Coprocessor,” *In Proc. Int. Conf. Information Technology: Coading and Computing, ITCC 2004*, vol. 2, pp. 486–492, 2004.

[13] K.U. Järvinen, and J.O. Skyttä, “High-Speed Elliptic Curve Cryptography Accelerator for Koblitz Curves,” in *Field-Programmable Custom Computing Machines, 2008. FCCM ’08. 16th International Symposium on*, April 2008, pp. 109 –118.

[14] K.U. Järvinen, J. Forsten, and J.O. Skyttä, “Efficient Circuitry for Computing τ-adic Non-Adjacent Form,” *Proc. IEEE Intl Conf. Electronics, Circuits and Systems (ICECS 06)*, pp. 232–235, 2006.

[15] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson, W. F. Chan, and Z. Huang, “FPGA Implementation of Point Multiplication on Koblitz Curves using Kleinian Integers,” in *Cryptographic Hardware and Embedded Systems*, ser. CHES’06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 445–459.

[16] V.S. Dimitrov, K.U. Järvinen, M.J. Jacobson, W.F. Chan, and Z. Huang,, “Provably Sublinear Point Multiplication on Koblitz Curves and Its Hardware Implementation,” in *Computers, IEEE Transactions on*, vol. 57, no. 11, Nov. 2008, pp. 1469–1481.

[17] B.B. Brumley, and K.U. Järvinen, “Conversion Algorithms and Implementations for Koblitz Curve Cryptography,” *Computers, IEEE Transactions on*, vol. 59, no. 1, pp. 81 –92, jan. 2010.

[18] J. Adikari, V.S. Dimitrov, and K.U Järvinen, “A Fast Hardware Architecture for Integer to τNAF Conversion for Koblitz Curves,” *Computers, IEEE Transactions on*, vol. 61, no. 5, pp. 732 –737, may 2012.

[19] K.U Järvinen, “On Repeated Squarings in Binary Fields,” in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, M. Jacobson, V. Rijmen, and R. Safavi-Naini, Eds., vol. 5867. Springer Berlin / Heidelberg, 2009, pp. 331–349.

[20] C. Rebeiro and D. Mukhopadhyay, “High Speed Compact Elliptic Curve Cryptoprocessor for FPGA Platforms,” in *INDOCRYPT*, 2008, pp. 376–388.

[21] W.N Chelton, and M. Benaissa, “Fast Elliptic Curve Cryptography on FPGA,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 198 –205, feb. 2008.

[22] K.U. Järvinen, “Optimized FPGA-based Elliptic Curve Cryptography Processor for High Speed Applications,” *Integration, the VLSI Journal*, vol. 44, no. 4, pp. 270 – 279, 2011, hardware Architectures for Algebra, Cryptology and Number Theory.

[23] C. Rebeiro, S.S. Roy, and D. Mukhopadhyay, “Pushing the Limits of High-Speed $GF(2^m)$ Elliptic Curve Scalar

Multiplication on FPGAs," in *Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, vol. 7428. Springer Berlin Heidelberg, 2012, pp. 494–511.

[24] National Institute of Standard and Technology, "FIPS 186-2, Digital Signature Standard," Federal Information Processing Standards Publication, 2000.

[25] Xilinx Inc, "Performance strategies," 2009.

APPENDIX A

Here we present computational optimizations for the curve parameter $\mu = 1$. We compute (d_0, d_1) as per Equation (13) to avoid long subtractions from zero (Section III-B).

$$(d_0, d_1) \leftarrow \Big(\frac{2d_1 + d_0}{4}, \frac{2d_1 - d_0}{4}\Big) \tag{13}$$

During the iterative divisions by τ^2, wrong sign is assigned to either d_0 or d_1 in any iteration. Assignment of the wrong sign alternates in every consecutive iteration. We find Equation 13 is same as Equation 3, only with the difference in the relative positions of d_0 and d_1 in the left-hand-side. During the reduction of scalar, the nonzero remainders are generated as per Table I for both $\mu = 1$ and $\mu = -1$. Thus, the computational optimizations we followed in Section III for $\mu = -1$, are also applicable for $\mu = 1$.

TABLE IV

DOUBLE DIGIT τNAF GENERATION FOR $\mu = 1$

Cases	$d_0(mod4)$	$2d_1(mod4)$	$d_0(mod8)$	$2d_1(mod8)$	r_0	r_1
1	0	0			0	0
2	1	0			1	0
3.A	2	0	2	0	0	-1
3.B	2	0	6	0	0	1
3.C	2	0	2	4	0	1
3.D	2	0	6	4	0	-1
4	3	0			-1	0
5.A	0	2	0	2	0	1
5.B	0	2	4	2	0	-1
5.C	0	2	0	6	0	-1
5.D	0	2	4	6	0	1
6	1	2			-1	0
7	2	2			0	0
8	3	2			1	0

Double digit τNAF is generated as per Table VIII for $\mu = 1$. Comparing with Table II, we see only the cases 3.A-3.D are different in Table VIII. For the cases which are same in both the tables, we apply the same computational optimizations discussed in Section IV for $\mu = -1$. Subtractions of remainders from d_1 are performed by altering low order bits of d_1 for the cases 3.A, 3.C and 3.D. However the subtraction of remainder in case 3.B involves carry propagation. We eliminate this long subtraction by incorporating it in the next step where we perform division by τ^2. This is shown in Equation 14. Subtraction of 2 or addition of 1 with d_0 is easy as it requires only alteration of low order bits of d_0. We also consider a borrow input to the adder/subtracter circuit in the critical path of d_1 (Equation 14).

$$(d_0, d_1) \leftarrow \Big(\frac{2d_1 + (d_0 - 2)}{4}, \frac{2d_1 - (d_0 + 1) - 1}{4}\Big) \tag{14}$$