A High-performance ECC Processor over Curve448 based on a Novel Variant of the Karatsuba Formula for Asymmetric Digit Multiplier

Asep Muhamad Awaludin, Jonguk Park, Rini Wisnu Wardhani, and Howon Kim.

Abstract—In this paper, we present a high-performance architecture for elliptic curve cryptography (ECC) over Curve448, which to the best of our knowledge, is the fastest implementation of ECC point multiplication over Curve448 to date. Firstly, we introduce a novel variant of the Karatsuba formula for asymmetric digit multiplier, suitable for typical DSP primitive with asymmetric input. It reduces the number of required DSPs compared to previous work and preserves the performance via full parallelization and pipelining. We then construct a 244-bit pipelined multiplier and interleaved fast reduction algorithm, yielding a total of 12 stages of pipelined modular multiplication with four stages of input delay. Additionally, we present an efficient Montgomery ladder scheduling with no additional register is required. The implementation on the Xilinx 7-series FPGA: Virtex-7, Kintex-7, Artix-7, and Zynq 7020 yields execution times of 0.12, 0.13, 0.24, and 0.24 ms, respectively. It increases the throughput by 242% compared to the best previous work on Zynq 7020 and by 858% compared to the best previous work on Virtex-7. Furthermore, the proposed architecture optimizes nearly 63% efficiency improvement in terms of Area$\times$Time tradeoff. Lastly, we extend our architecture with well-known side-channel protections such as scalar blinding, base-point randomization, and continuous randomization.

Index Terms—elliptic-curves cryptography (ECC); Curve448; high-speed multiplier; asymmetric Karatsuba; field-programmable gate array (FPGA)

1 Introduction

The performance of Public-Key Cryptography has become one of the main factors of interest in the recently emerging technologies such as the 5G System [1] and Blockchain [2]. At the same time, there is increasing demand to increase the security level against attacks that could compromise the overall performance. In particular, applications enabled on Internet of Things (IoT) devices suffer from performance degradation due to limited resources on the processing unit. Elliptic Curve Cryptography (ECC) has been chosen as the building block in the security protocol of those technologies among asymmetric cryptographic algorithms due to its smaller key size. The Internet Research Task Force (IRTF) [3] recommended Curve25519 [4] and Curve448 [5] for a high level of practical security with 128-bit and 224-bit security levels, respectively, along with inclusion in Transport Layer Security (TLS) standard 1.3 [6]. Afterward, National Institute of Standards and Technology (NIST) [7] also included these curves in their standard.

Curve448 is a conservatively designed elliptic curve with very competitive performance on a wide variety of platforms, leading to ECC construction issues and to advances in strong cryptanalysis and classical attacks [5]. Obviously, this research is being performed not only due to the wide use of high-performance ECC Processors over Curve448 but also to address the need for achieving a high-security and efficient ECC processor architecture as an imperative step for the emergence of Post-Quantum Cryptography (PQC). We must acknowledge, with the invention of the Shor's factorization algorithm [8], that the current state of the cryptographic system will soon likely be compromised by quantum computing; likewise, the classic public key exchange algorithm will soon be replaced. Nevertheless, as long as PQC has not been fully implemented, hybrid mode ECC will still be used in order to sustain the compatibility of industry and government regulations. Since in the hybrid schemes, classic and PQC will work concurrently during the transition to PQC [9]. Classical cryptographic system will still be needed, even though PQC development has come a long way. Consequently, designing an ECC processor architecture with a high level of security, high speed, low latency, and high efficiency in every single processing step is crucial.

Generally, optimization can be done through algorithmic improvement to reduce the number of calculation steps for such expensive primitive operations such as finite field and group operations. Despite that, keeping a short critical delay path on hardware implementation is even more challenging due to being limited by technology, as it determines the maximum working frequency. Thus, critical delay requires more attention than software implementation.

Prior work: To the best of our knowledge, there are only a few published results on hardware implementations targeting ECC with a security level above 128 bits, particularly Curve448.

The first hardware implementation of Curve448 was investigated by Sasdrich and Güneysu in [10]. Their de-

• *The authors are with the School of Computer Sciense and Engineering, Pusan National University, Busan 609735, Korea; (E-mail: {asep.muhamad11, daiula10, rini.wisnu, howonkim}@pusan.ac.kr)*

sign employed schoolbook multiplication with interleaved reduction for the underlying modular multiplier. The implementation results on a Xilinx Zynq 7020 Field Programmable Gate Arrays (FPGA) archived a throughput of 1087 ECC point multiplication (ECPM) per second and consumed 1580 logic slices and 33 Digital Signal Processor (DSP) blocks. Their design also offered basic side-channel protections, such as scalar blinding and base-point randomization. Furthermore, they extended their previous design with additional protection against horizontal attacks in [11] by adding a re-randomization countermeasure. At the same time, they evaluated their countermeasure with scalar- and base-point-dependent leakage side-channel evaluations.

In [12], Shah et al. proposed the hardware design of Curve448 utilizing LookUp Table (LUT) only, which aims to be platform independent. They adopted the redundant-signed-digit (RSD) representation for arithmetic operations and the segmentation approach at the architectural level to reduce the required number of clock cycles for ECPM operations. Their implementation results targeting Virtex-7 achieved a throughput of 869 ECPM per second utilizing 50,143 LUTs.

The proposal by Niasar et al. [13] represents a very recent work hardware implementation of Curve448. They investigated three different implementation strategies (i.e., lightweight, area-time efficient, and high-performance architectures) targeting the Xilinx Zynq 7020 FPGA. Their high-performance architecture increased throughput by 12% by executing 1,219 ECPM per second and increased efficiency by 40% in terms of required clock cycles$\times$utilized area compared to the initial work in [10]. They achieved their speed-up by utilizing 81 DSPs for parallelization in the lowest level of Karatsuba computation. To the best of our knowledge, their high-performance variant is the state-of-the-art of Curve448 hardware implementation in terms of ECPM throughput.

The Karatsuba-Ofman formula [14], also known as Karatsuba formula, has been a widely used method of multiplying two n-bit arbitrary-precision numbers, which reduces the asymptotic complexity to $O(n^{1.585})$ bit operations compared to $O(n^2)$ bit operations for the schoolbook method. However, the nature of its algorithm that uses recursion to construct higher precision numbers leads to the extra overhead of additions. In particular, implementing parallel Karatsuba in hardware is problematic in that it increases the critical delay path due to the addition tree, despite using parallel DSP blocks for digit multipliers at lower levels. Therefore, despite reducing the number of required DSP blocks, the overall operating frequency remains low, as shown in the implementation results in [13] and [15].

Awaludin et al. [16] demonstrated a new way of using the Karatsuba formula for high-speed hardware parallel multiplier without the cost of increasing the critical delay path. The technique employs the combination of the schoolbook method and the Karatsuba algorithm with a compressor circuit (i.e., carry-save-adder tree(CSAT)), despite requiring slightly more DSP blocks than the original Karatsuba method. Apparently, the presented equation is similar to the method discovered earlier by Khachatrian et al. [17], which was then formalized by [18], called the arbitrary degree variant of Karatsuba (ADK). The method was initially intended to avoid overflow during the accumulation of the partial products on a typical word-based processor (i.e., software implementation), which is technically implemented in an iterative way.

Apart from optimizing the cost of extra addition, the method employed by [16] does not leverage the full capability of DSP blocks (i.e., Xilinx DSP48E1), as they use a symmetric 16x16-bit digit multiplier. Thus, the use of the Karatsuba formula with the asymmetric feature of DSP blocks remains unexplored. To the best of our knowledge, Roy et al. [19] represents the most recent work that uses the full capability of asymmetric DSP blocks, which reduces the required DSP blocks in the schoolbook method using the nonstandard tiling method. This method is also used by [20] to construct a 257-bit signed multiplier for the hardware implementation of PQC SIKE.

Our contributions: The contributions of this paper are summarized as follows:

1) We present a novel variant of the Karatsuba formula for asymmetric digit multiplier, which reduces DSP block utilization while offering a high-speed multiplier through parallelization and pipelining. To the best of our knowledge, this is the first work considering the full capability of DSP blocks with the Karatsuba algorithm. Furthermore, it can be generalized for broader use in a cryptographic algorithm that employs multiplication.
2) We then present a high-performance ECC processor architecture over Curve448 that to the best of our knowledge, outperforms the existing architecture in terms of execution time as well as Area$\times$Time efficiency.
3) For the underlying architecture, we propose a 12-stage pipelined modular multiplier with four stages of input delay, which is built from a five-stage 244-bit fully pipelined multiplier with an interleaved fast reduction over the modulus $p = 2^{448} - 2^{224} - 1$.
4) The presented five-stage 244-bit fully pipelined multiplier is constructed from a novel variant of Karatsuba in point 1. At the same time, the interleaved fast reduction is obtained by exploiting the Solinas prime with the golden ratio $\phi = 2^{224}$.
5) We provide an efficient Montgomery ladder scheduling algorithm without the requirement of an additional temporary register.
6) Lastly, the proposed architecture is extended with side-channel attack countermeasures such as scalar blinding, base-point randomization, and continuous randomization, which are expected to resist vertical and horizontal attacks.

The rest of this paper is organized as follows: Section 2 gives a brief introduction to Curve448 with the underlying group arithmetic and field arithmetic. Section 3 describes the proposed novel variant of the Karatsuba formula for asymmetric digit multiplier. Section 4 presents the proposed hardware architecture of the ECC processor over Curve448. Then, in section 6, we present our hardware implementation results and compare them to those of the existing methods. Lastly, Section 7 concludes the paper.

2 PRELIMINARIES

Ed448-Goldilocks is an elliptic curve over prime field $GF(p)$ with a 244-bit security level introduced by Hamburg in [5], which is defined in untwisted Edwards form:

$$E_d : y^2 + x^2 = 1 + dx^2y^2 \mod p \tag{1}$$

with $d = -39081$ and $p = 2^{448} - 2^{224} - 1$. The curve is birationally equivalent to the Montgomery curve defined in RFC 7748 [21] called Curve448, the term we will use for the rest of the paper. Curve448 satisfies the requirement of SafeCurves and is included in TLS standard 1.3 [6].

2.1 ECC Group Law

Let k be a scalar, $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ be two point represented in affine coordinates where $P, Q \in E$ and $x_P, y_P \in GF(p)$. An ECC point multiplication (ECPM), $Q = k \cdot P$, is a k-times additions of point P (i.e., $P + P + ... + P$), which can be performed with group operation of point doubling (PD) and point addition (PA). Typically, the projective coordinate representation is used to avoid modular inversion during intermediate computation, where an affine point $P = (x_P, y_P)$ can be converted to projective point $P = (X, Y, Z)$ such that $x_P = X/Z$ and $y_P = Y/Z$.

The Montgomery ladder was introduced to perform ECPM over the Montgomery curve, which processes point point doubling and addition computation in a single step [22]. A single step of Montgomery ladder is computed with the following formula (taken from [4]):

$$\begin{aligned} X_{PD} &= (X_2 - Z_2)^2 (X_2 + Z_2)^2 \\ Z_{PD} &= \left((X_2 + Z_2)^2 - (X_2 - Z_2)^2\right). \\ &\quad \left((X_2 + Z_2)^2 + a24\left((X_2 + Z_2)^2 - (X_2 - Z_2)^2\right)\right) \\ X_{PA} &= \left((X_2 - Z_2)(X_3 + Z_3) + (X_2 + Z_2)(X_3 - Z_3)\right)^2 \\ Z_{PA} &= \left((X_2 - Z_2)(X_3 + Z_3) - (X_2 + Z_2)(X_3 - Z_3)\right)^2 x_P \end{aligned} \tag{2}$$

where $Q_2 = 2P_2$ and $Q_3 = P_2 + P_3$ with $Q_2 = (X_{PD}, Z_{PD})$, $Q_3 = (X_{PA}, Z_{PA})$, $P_2 = (X_2, Z_2)$, and $P_3 = (X_3, Z_3)$. A constant value $a24 = 39081$ is used specifically for Curve448. Note that this formula needs only x-coordinate of base point P to perform ECPM. The formula requires ten modular multiplications and eight modular additions/subtractions.

2.2 Field Arithmetic

The name "Goldilocks" refers to the prime modulus of Curve448 that is defined as the Solinas trinomial prime with the golden ratio $\phi = 2^{224}$, which offers fast arithmetic in typical (i.e., 32-bit or 64-bit) machines. Moreover, with its golden ratio ϕ, it allows Karatsuba multiplication of two operands $A = (a_1\phi + a_0)$ and $B = (b_1\phi + b_0)$, $A, B \in GF(p)$, to be calculated efficiently as follows:

$$\begin{aligned} C &= (a_1\phi + a_0) \cdot (b_1\phi + b_0) \\ &\equiv (a_1b_1 + a_0b_0) + (a_1b_0 + a_0b_1 + a_0b_0)\phi \pmod p \\ &= (a_1b_1 + a_0b_0) + ((a_0 + a_1)(b_0 + b_1) - a_0b_0)\phi \end{aligned} \tag{3}$$

A modular inversion is required to convert back projective coordinate $Q = (X, Z)$ to affine coordinates representations $Q = (x_Q)$ at the end of ECPM operation. A constant time modular inversion can be implemented using Fermat's Little Theorem (FLT) such that $Z^{-1} \equiv Z^{p-2} \mod p$. Finally, the affine representation of the point Q is calculated as $x_Q = XZ^{-1}$, with Z^{-1} is computed using FLT.

3 NOVEL VARIANT OF KARATSUBA FORMULA FOR ASYMMETRIC DIGIT MULTIPLIER

Consider two n-bit arbitrary-precision numbers A and B represented in asymmetric radixes α and β, where $\alpha \neq \beta$.

$$A = \sum_{i=0}^{u} a_i\alpha^i, \quad B = \sum_{j=0}^{v} b_j\beta^j \tag{4}$$

u and v are the degree of A and B, respectively. The product $C = A \cdot B$ is calculated as follows:

$$C = \sum_{i=0}^{u-1} a_i\alpha^i \sum_{j=0}^{v-1} b_j\beta^j = \sum_{i=0}^{u-1}\sum_{j=0}^{v-1} a_i\alpha^i b_j\beta^j \tag{5}$$

The schoolbook algorithm multiplies u digit and v digit numbers by multiplying each digit of one input by each digit of the other, which takes $O(uv)$ digit multiplications in total. Clearly, it requires un DSP blocks when performing full parallelization on digit multiplication. We investigate a novel variant of the Karatsuba formula for asymmetric digit multiplier, which later reduces the complexity as well as the number of required DSPs compared to other similar works (i.e., [16], [19], [23]).

We rewrite the Equation 5 as follows:

$$C = \sum_{i=0}^{u-1}\sum_{j=0}^{i\frac{\alpha}{\beta}} a_i\alpha^i b_j\beta^j + \sum_{k=0}^{u-1}\sum_{l=k\frac{\alpha}{\beta}}^{v-1} a_k\alpha^k b_l\beta^l \tag{6}$$

where when $\alpha^i\beta^j = \alpha^k\beta^l$, we obtain the following identity:

$$a_i\alpha^i b_j\beta^j + a_k\alpha^k b_l\beta^l = \left[(a_i - a_k)(b_j - b_l) + a_ib_l + a_kb_j\right]\alpha^i\beta^j \tag{7}$$

Equation 7 shows that two multiplications can be reduced into one multiplication for a condition where $\alpha^i\beta^j = \alpha^k\beta^l$. This is similar to the Karatsuba [14] for $\alpha = \beta$ where it reduces four-digit multiplications to three-digit multiplications, as a generalization of our problem illustrated in Fig. 1.

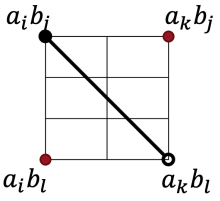

Fig. 1. Karatsuba on Asymmetric Digit Multiplier

The red points are two-digit multiplications that are calculated prior to other digit multiplications. At the same time, the black points, which are connected by a line, are the two-digit multiplications that later can be reduced to one-digit multiplication. In general, the Karatsuba formula can be applied when two-digit multiplications are connected via a diagonal line without being restricted by the used radix. This method works ideally on a radix with a power of two.

TABLE 1
Comparison of required digit multipliers for different operand widths with existing methods

Operand Width	Schoolbook	Nonstandard Tiling [19]	Schoolbook + Karatsuba [16]	Our method
192	96	90	78	60
224	140	120	105	88
256	176	160	136	113
384	368	360	300	216
521	682	660	561	433

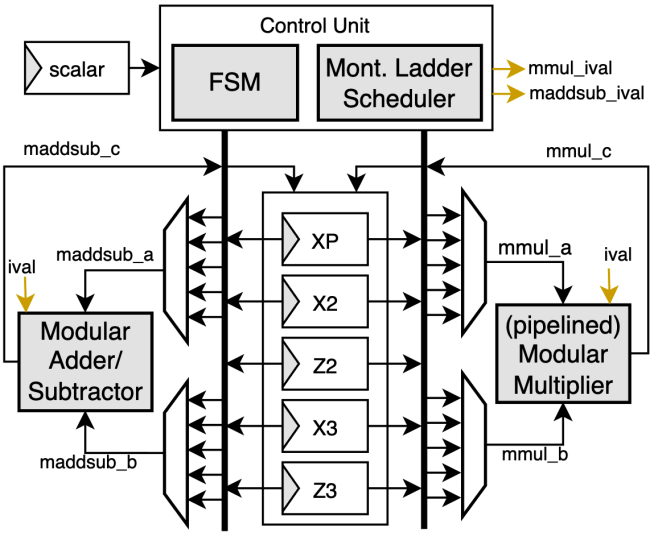

Fig. 2. Top-level Architecture

In particular, if we let $\alpha = 2^{w_1}$ and $\beta = 2^{w_2}$, a greater reduction in complexity can be obtained when $GCD(w_1, w_2) \neq 1$. Table 1 shows the complexity comparison of our method with existing methods. By setting up $\alpha = 2^{24}$ and $\beta = 2^{16}$, our method reduces the DSP utilization compared to the existing methods in the literature.

4 Proposed Hardware Architecture

Fig. 2 depicts the the proposed top-level architecture of Curve448. This is the typical architecture consisting of the control unit, modular multiplier module, and modular adder/subtractor module. In contrast to the architecture proposed in [13], which uses RAM to store the ladder variables, we use register files utilized from flip-flop (FF). This is because in a typical FPGA (i.e., Xilinx FPGA [24]), the availability of FF is higher than that of LUT cells (e.g., in Xilinx, a single slice consists of four LUTs and eight FFs). Therefore, with a design that has higher LUT cell utilization than FF, increasing FF utilization will not drastically increase the slice utilization. Moreover, utilizing FF instead of BRAM preserves the overall performance without introducing overhead on memory read/write access. Apart from performance and utilization considerations, the use of Block Random Access Memory (BRAM) introduces a new opportunity for attackers to extract secret scalar information by recovering information on the BRAM addressing pattern using Differential Power Analysis (DPA). Although it can be protected via address scrambling, such as the design

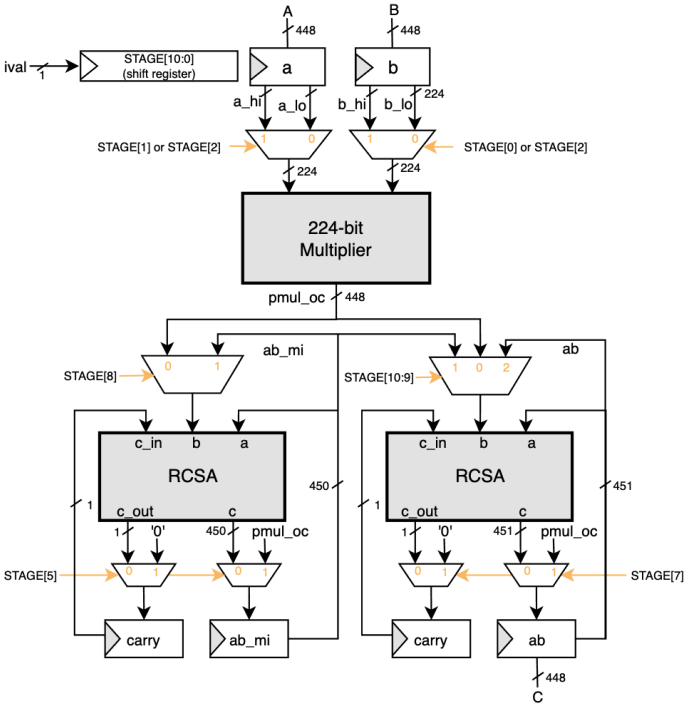

Fig. 3. Proposed 12-stage Pipelined (with four stages of input delay) Modular Multiplier

proposed in [11], clearly, it consumes more area due to the utilization of logic cells. Furthermore, all the computation steps are one-way controlled by precise scheduling of Montgomery ladder without need handshake process such as valid/ready protocol.

4.1 Modular Multiplier

We construct a 12-stage pipelined modular multiplier with four stages of input delay based on a 224-bit pipelined multiplier. We choose a 244-bit width multiplier because the prime number of Curve448 has a golden ratio $\phi = 224$, which later optimizes the reduction step as we propose the reduction algorithm for Curve448.

Fig. 3 shows the overall structure of our modular multiplication design. It consists of a 224-bit pipelined multiplier followed by two Ripple-Carry-Save Adders (RCSAs). An RCSA is actually a pair of adders, which in our case are carry-compact adders (CCAs) [25] that are used to limit the critical delay path of the ripple-carry-adder at some point, as shown in Fig. 4. The carry for the first half is not propagated; instead, it is saved and included as an input for another half in the next stage. This method is suitable for accumulator circuits. Additionally, the pipelined one-hot encoding is used with simple shift register to control the input-output signal between the stages. Note that the output valid and busy signal are not necessary in our design since we use a precise ladder scheduling, considering the restriction in modular multiplier module (i.e., requires four cycles input delay).

4.1.1 *244-bit Pipelined Multiplier*

The construction of the 244-bit pipelined multiplier based on Equation 7 is given in the Fig. 5. As shown in, all the

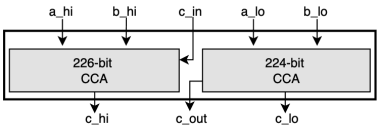

Fig. 4. Ripple-Carry-Save Adder (RCSA). Technically, it limits the carry propagation to a predefined delay, which is the delay of a 244-bit Carry-Compact-Adder (CCA) in our case.

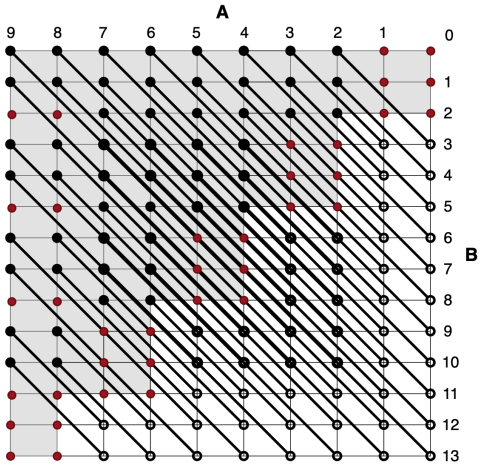

Fig. 5. Construction of 244-bit Multiplication with Asymmetric Digit Multiplier ($\alpha = 2^{24}$ and $\beta = 2^{16}$). Note that since 224 does not divisible by 24, there will be unused bits at the most significant bit of the intermediate results.

red points represent a single-digit multiplication, while the two black points connected by a line represent a single-digit multiplications that are reduced from two-digit multiplications. Note that some lines may pass through multiple points, yet the relation should satisfy the Equation 7. There are some exceptions; when the point is already applied as a red point (i.e., due to the Karatsuba multiplication on the counterpart), it cannot be further reduced, even if it satisfies Equation 7. Thus, it remains as red point (including the counterpart) as in our case, as shown in points (8,2), (9,2), (8,5), (9,5), (8,2), and (9,2) in Fig. 5. Finally, we obtain less complexity, as it requires only 88 DSPs instead of 140 DSPs in the schoolbook method or 105 DSPs in the nonstandard tiling method [19].

Fig. 6 shows the architecture of the 224-bit pipelined multiplier. The architecture contains five fully pipelined stages, which means it can process an input on each cycle. Our calculation steps of $C = A \cdot B$ are described as follows:

- Stages 1 and 2: The parallel 16-bit ripple-carry adder (RCA) is used to compute $b_j - b_l$. The output of the 16-bit RCA is wired to 25x17-bit signed Multiply-Accumulate (MAC) modules, which also have a pre-adder input to compute $a_i - a_k$ before going to the multiplication stage. At the same time, parallel 24x16-bit signed multiplier (MUL) modules are used to compute $a_i b_l$ and $a_k b_j$. Both the 25x17-bit signed MAC with the pre-adder and 24x16-bit signed MUL are utilized from DSP primitive with a three-stage and two-stage pipeline, respectively, to achieve maximum performance, as recommended in [26], which is shown in Fig. 7.
- Stage 3: The output of 24x16-bit signed MUL available in this stage is then used by the 40-bit CCA to calculate $a_i b_l + a_k b_j$. The output 40-bit CCA is routed to the input accumulator of the MAC modules. Note that the output of 24x16-bit signed MUL is also stored in registers, as it will be used in the compression stage (Stage 4).
- Stage 4: Before being processed by the CSAT, all intermediate values are grouped and aligned into 40-bit segments to reduce the number of inputs in the CSAT as well as the depth of the tree. However, while the output of 24x16-bit signed MUL is already in 40-bit width, the calculation of $(a_i - a_k)(b_j - b_l) + a_i b_l + a_k b_j$ obviously produces up to a 41-bit output width. We employ an alignment method similar to that used by [16] to handle the overflow bit (i.e., 41^{st} bit). All intermediates values are compressed using homogeneous 3:2 compression to achieve balanced performance.
- Stage 5: In this stage, a final propagated addition of sum and $carry$ from the output of the CSAT is performed using the CCA proposed in [25]. We obtained the optimal parameter CCA with H = 3 and L = 30 experimentally based on trial and error after synthesis and implemention in FPGA. Furthermore, the input and output of CCA are enclosed by registers to minimize the critical delay path.

4.1.2 *Fast Reduction over* $p = 2^{448} - 2^{224} - 1$

We propose the fast reduction technique interleaving with the intermediate output from the 244-bit pipelined multiplier, which is given in Algorithm 1. The multiplication of A and B, which each have a 448-bit width, can be decomposed into four 244-bit multiplications. Note that we do not take the Karatsuba approach recommended by [5], since it does not give an advantage in our reduction step; rather, we take the additional cost of one clock cycle on the pipelined multiplier. We perform partial reduction for three intermediate results in advance (i.e., $z_4 = (z_1 + z_2 + z_3) \cdot 2^{224} \mod p$), while the second term (i.e., $z_0 + z_3 \mod p$) is accumulated with the first reduction step result and we perform the second reduction accordingly (i.e., $C = z_0 + z_3 + z_4 \mod p$). This technique relies on the following property:

$$(a + b) \mod p = (a + (b \mod p)) \mod p \qquad (8)$$

Considering the advantage of the Goldilocks modulus $p = 2^{448} - 2^{224} - 1$ and the fact that $2^{448} \equiv 2^{224} + 1 \mod p$, the reduction of $z_4 = T \cdot 2^{224} \mod p$, where $T = z_1 + z_2 + z_3$, can be performed efficiently, as mentioned in Step 13 of Algorithm 1. Referring to the structure of the RCSA, the actual addition is performed only in the second adder (i.e., 226-bit CCA). Accordingly, the reduction of $C = G \mod p$, where $G = z_0 + z_3 + z_4$ yields 3 bits of overflow, can be performed efficiently with the RCSA as mentioned in Steps 18–20 of Algorithm 1. Note that the final reduction might produce a carry at the first adder of the RCSA, as this carry needs to be propagated to the second adder at the final step.

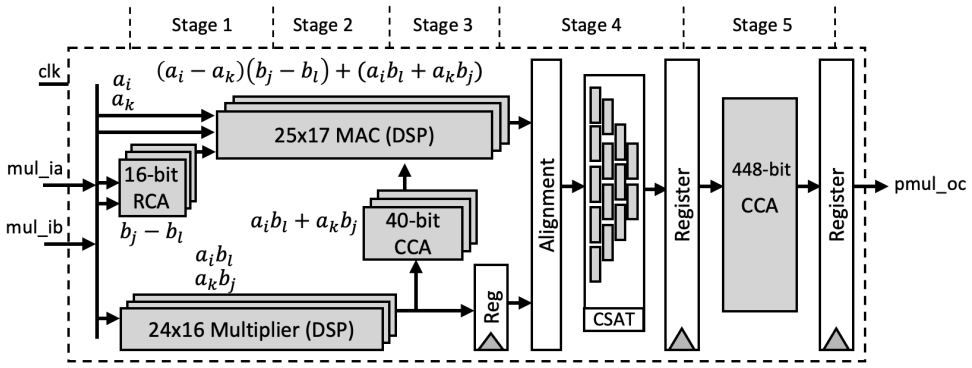

Fig. 6. Proposed Five-stage 244-bit Fully Pipelined Multiplier.

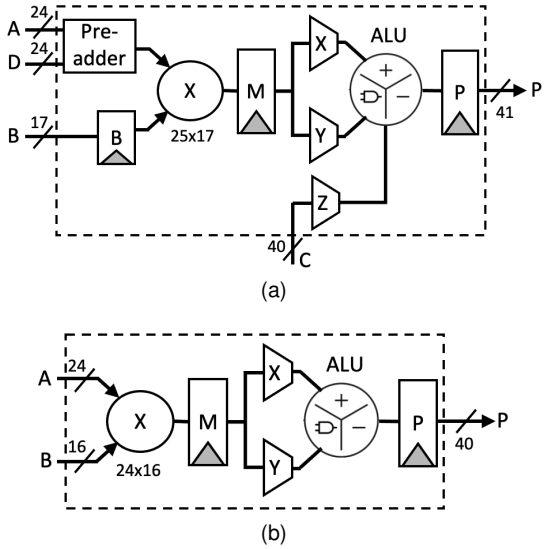

Fig. 7. Digital Signal Processing (DSP) utilization for (a) a three-stage 25x17-bit signed Multiply-Accumulator with pre-adder and (b) a two-stage 24x16-bit signed multiplier.

Algorithm 1 Proposed Interleaved Fast Reduction for $p = 2^{448} - 2^{224} - 1$) modulus

Require: Integer A, B satisfying $0 \leq A, B < p$
Ensure: $C = A \cdot B \mod p$

1:	$a_0 \leftarrow A_{[223:0]}$	
2:	$a_1 \leftarrow A_{[447:224]}$	
3:	$b_0 \leftarrow B_{[223:0]}$	
4:	$b_1 \leftarrow B_{[447:224]}$	
5:	$z_1 \leftarrow a_0.b_1$	244-bit pipelined multiplications
6:	$z_2 \leftarrow a_1.b_0$	
7:	$z_3 \leftarrow a_1.b_1$	
8:	$z_0 \leftarrow a_0.b_0$	
9:	$T \leftarrow z_1 + z_2 + z_3$	{450-bit}
10:	$t_0 \leftarrow T_{[223:0]}$	
11:	$t_1 \leftarrow T_{[449:224]}$	
12:	$t_2 \leftarrow T_{[449:448]}$	
13:	$z_4 \leftarrow (t_0 + t_1 + t_2) \parallel t_{1[223:0]}$	{450-bit}
14:	$G \leftarrow z_3 + z_0 + z_4$	{451-bit}
15:	$g_0 \leftarrow G_{[223:0]}$	interleaved fast reduction
16:	$g_1 \leftarrow G_{[447:224]}$	
17:	$g_2 \leftarrow G_{[450:448]}$	
18:	$U \leftarrow g_0 + g_2$	{225-bit}
19:	$V \leftarrow g_1 + g_2$	{224-bit}
20:	$C \leftarrow (V + U_{[244]}) \parallel U_{[243:0]}$	{448-bit}
21:	return C	

Algorithm 2 Fermat-based inversion for Curve448 ($p = 2^{448} - 2^{224} - 1$)).

Require: Integer z satisfying $0 < z < p$
Ensure: Modular inverse $z^{-1} \equiv z^{p-2} \mod p$

1:	$u \leftarrow z^{2^1} \cdot z$	$z^{(2^2-1)}$
2:	$u \leftarrow u^{2^1} \cdot z$	$z^{(2^3-1)}$
3:	$u \leftarrow u^{2^3} \cdot u$	$z^{(2^6-1)}$
4:	$u \leftarrow u^{2^6} \cdot u$	$z^{(2^{12}-1)}$
5:	$u \leftarrow u^{2^1} \cdot z$	$z^{(2^{13}-1)}$
6:	$u \leftarrow u^{2^{13}} \cdot u$	$z^{(2^{26}-1)}$
7:	$u \leftarrow u^{2^1} \cdot z$	$z^{(2^{27}-1)}$
8:	$u \leftarrow u^{2^{27}} \cdot u$	$z^{(2^{54}-1)}$
9:	$u \leftarrow u^{2^1} \cdot z$	$z^{(2^{55}-1)}$
10:	$u \leftarrow u^{2^{55}} \cdot u$	$z^{(2^{110}-1)}$
11:	$u \leftarrow u^{2^1} \cdot z$	$z^{(2^{111}-1)}$
12:	$v \leftarrow u^{2^{111}} \cdot u$	$z^{(2^{222}-1)}$
13:	$u \leftarrow v^{2^1} \cdot z$	$z^{(2^{223}-1)}$
14:	$u \leftarrow u^{2^{223}} \cdot v$	$z^{(2^{446}-2^{222}-1)}$
15:	$u \leftarrow u^{2^2} \cdot z$	$z^{(2^{448}-2^{224}-3)}$
16:	return u	

The precise scheduling of modular multiplication is presented in Fig. 8. The first four stages are used to calculate z_1, z_2, z_3, and z_0. In these stages, the input A and B are held in the input register, placing the modular multiplier core in a busy state and causing input delay for four cycles. Stages 7 and 8 perform the first accumulation $z_1 + z_2 + z_3$, followed by addition $t_0 + t_1 + t_2$ in Stage 9 using $A1$. At the same time, Stages 8 to 10 perform the second accumulation $z_3 + z_0 + z_4$, followed by two parallel additions $g_0 + g_2$ and $g_1 + g_2$ in Stage 10 using $A2$. Lastly, the output product is available in Stage 12. Therefore, the modular multiplication takes 12 cycles, pipelined with four stages of input delay.

4.2 Modular Adder/Subtractor

A unified modular adder/subtractor is utilized from a single CCA, which calculates $C = A \pm B \pm p$, as shown in Fig. 9. The calculation takes two steps: it first calculates $r1 = A \pm B$ and then calculates $s = r1 \pm p$ with op to control the sign of $\pm B$ and $\pm p$ using masking. The output C is selected between $r1$ and s depending on the value of sel, which is the XOR value of $cout$ of the first step and op. Basically, it detects whether the first step calculation produces a carry/borrow. While the sign of $\pm b$ is converted with two's complement, the sign of $\pm p$ is rather than more efficient due to the special

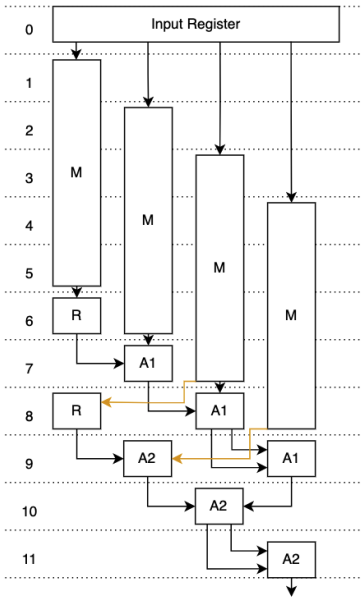

Fig. 8. Modular Multiplication Calculation Steps. M, $A1$, and $A2$ are a 244-bit multiplier, first RCSA, and second RCSA, respectively.

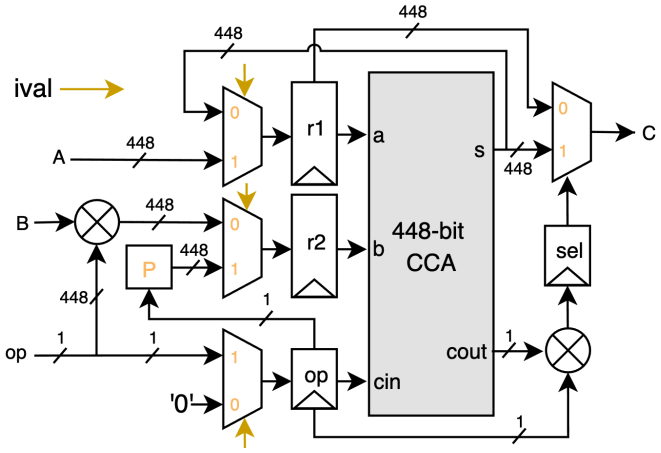

Fig. 9. Proposed Modular Adder/Subtractor Module

form of its prime number. With the value of $2^{448} - 2^{224} - 1$, we can construct its value with the following signal instead of masking (written in Verilog syntax):

```
{223{op}},~op,{223{op}},1'b1}
```

Therefore, it takes two cycles to complete a single modular addition/subtraction. The critical path of this module is defined by the CCA circuit with optimal parameters H = 3 and L = 30 obtained experimentally on FPGA.

4.3 Modular Inverse

A modular inversion is required to transform back from projective coordinates to affine coordinates at the end of the ECPM operation. A fully constant time modular inversion can be performed based on Fermat's Little Theorem (FLT). Let $p = 2^{448} - 2^{224} - 1$ be the prime of Curve448; then, the modular inverse of z^{-1} can be calculated as $z^{-1} \equiv z^{2^{448}-2^{224}-3} (\text{mod } p)$. The modular inversion calculation via exponentiation can be performed with a total of 462 modular multiplications, as given in Algorithm 2, which can be utilized from the modular multiplier module. Therefore, no additional module for inversion is employed.

4.4 Efficient Montgomery Ladder Scheduling

The scheduling of the Montgomery ladder algorithm, Equation 2, is given in Fig. 10. The gray line in the multiplier indicates the busy signal of the 224-bit pipelined multiplier modules, where a single modular multiplication takes four 224-bit multiplications, as mentioned previously. It shows that the 224-bit pipelined modular multiplier module is nearly busy, which yields high usage efficiency for the pipelined architecture. It takes 52 cycles to perform a single Montgomery ladder step. Furthermore, with the pipelined architecture of the modular multiplier, no temporary register in addition to input registers (i.e., x_P, X_2, Z_2, X_3, and Z_3) is required.

The Montgomery ladder in Equation 2 requires conditional swap such that:

$$(X_2, Z_2, X_3, Z_3) = (X_{2+b}, Z_{2+b}, X_{3-b}, Z_{3-b}) \quad (9)$$

with b respect to the most two significant bit values of scalar on each iteration, assuming the scalar register is shifted left. Constant-time conditional swap can be implemented easily on hardware since the update of X_2, Z_2, X_3, and Z_3 naturally are performed in parallel.

5 Side-Channel Attack Countermeasures

In this section, we extend our proposed architecture with side-channel attack protection for both vertical (classical) and horizontal attacks by incorporating several well-known methods from the literature.

5.1 Secure against Vertical Side-channel Attack

Our proposed architecture, presented in section 4, is naturally resistant to timing attacks and simple power analysis due to inherently resistant algorithms (i.e., Montgomery Ladder and FLT). To provide protection against Differential Power Analysis (DPA), additional methods such as base-point randomization and scalar blinding have to be implemented [27]. Enabling these countermeasures provides protection against vertical side-channel attacks, in which the attacker tries to observe multiple runs of ECPM operation.

5.1.1 *Base-Point Randomization*

Point randomization can be achieved by multiplying a random value $\lambda \in Z_{2^{448}} \backslash \{0\}$ to the projective point $P = (X, Z)$ such that $P = (\lambda X, \lambda Z)$. The output of ECPM is not changed in this respect, which can be proven as follows:

$$x_p = \frac{X}{Z} = \frac{\lambda X}{\lambda Z} \quad (10)$$

Base-point randomization provides different point representations corresponding to the entropy given by the random value λ to prevent any information extraction using statistical analysis. In particular, this process initializes the

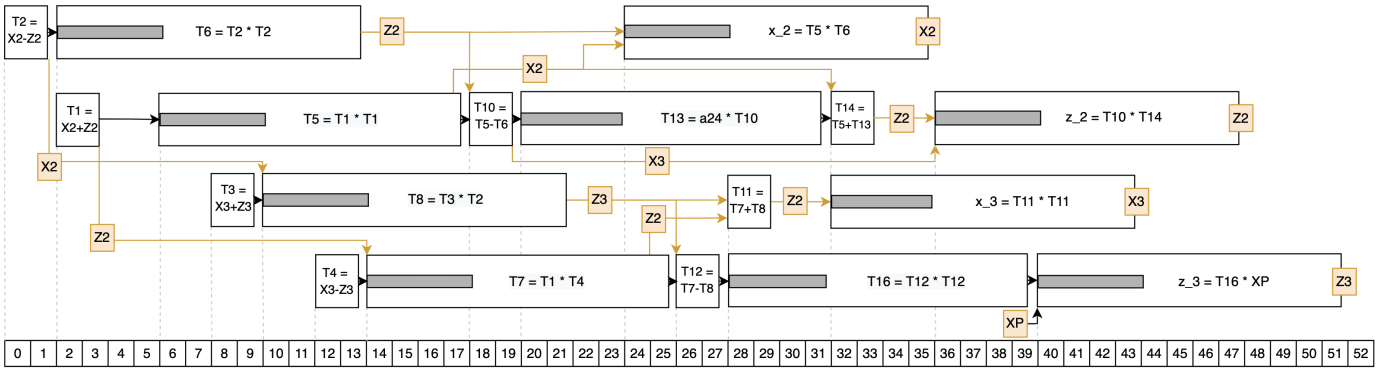

Fig. 10. Pipelined Montgomery Ladder Scheduling of Curve448. The total latency is 52 clock cycles, without the requirement of an additional temporary register. The constant a24 is equal to 39081.

z-coordinate with λ and uses a modular multiplication to update the x-coordinate accordingly. Hence, this countermeasure can be integrated easily by using an additional multiplication call during the initialization phase of the ECPM operation.

5.1.2 Scalar Blinding

Scalar blinding can be achieved by adding multiple group order $\#E$ to scalar k such that $k_r = k + r \times \#E$ where r is a random value. The correctness of this approach can be proven as follows:

$$k_r P = (k + r \times \#E)P = kP + r\Theta = kP \qquad (11)$$

Note that the multiplication of point P and group order $\#E$ results a point at infinity. The computation removes the correlation between the Montgomery ladder swap function and the corresponding bit in scalar k. For ECC with special prime field (i.e., Solinas prime), it is recommended to provide sufficient larger blinding factors r as investigated in [27], which is at least half of the field size. Thus, the blinding factor r with 224-bit length builds k_r with 672-bit length. The latency of ECPM is increased accordingly.

5.2 Secure against Horizontal Side-channel Attack

The horizontal side-channel attack is another type of attack in which the attacker observes leakage within a single run of ECPM operation. Continuous point randomization for each Montgomery ladder within a single ECPM operation can be applied sequentially to prevent horizontal attacks. It requires two more modular multiplications applied on intermediates output (i.e., λX_{PA} and λZ_{PA}) to re-randomized Montgomery ladder computation.

Hence, enabling horizontal attack protection with a continuous point randomization will increase Montgomery ladder time and total latency. In particular, the Montgomery scheduling in Fig. 10 is enlarged to 64 cycles. We assume that the random number is provided externally with sufficient throughput, such as the Random Number Generator (RNG) design proposed by [28].

6 Hardware Implementation Result and Comparison

The proposed design has been described by SystemVerilog HDL. Synthesizing, mapping, placing, and routing were carried out using Xilinx Vivado 2020.1, targeting four modern devices (Xilinx Virtex-7 [XC7VX690T], Kintex-7 [XC7K325T], Artix-7 [XC7A100T], and Zynq 7020 [XC7Z020] FPGA) for a more comprehensive evaluation with other related works. The correctness of implementation was verified by using the testbench with reference to the test vector provided in RFC 7748 [21].

The result of our ECC processor implementation, as well as those of several related papers over Curve448, are presented in Table 2. We achieve the lowest latency among the proposals targeting Xilinx Zynq 7020 FPGA with 0.24 and 0.39 ms for the unprotected and protected designs, respectively.

Additionally, we provide the implementation results on various devices for future reference, such as Artix-7, Kintex-7, and Virtex-7, achieving latency of 0.24, 0.13, and 0.12 ms for the unprotected design, and 0.40, 0.22, and 0.20 ms for the protected design, respectively. For the unprotected design, our fastest implementation (Virtex-7) requires 7,521 slices, while Kintex-7, Artix-7, and Zynq 7020 utilize 7,210, 6,826, and 6,946 slices, respectively. On all four platforms, we utilize 88 DSPs and no BRAM. As can be inferred from the table, our architecture yields the highest efficiency in terms of Area$\times$Time and DSP$\times$Time tradeoff compared to other existing architectures.

To the best of our knowledge, the method by Niasar et al. [13] represents the state-of-the-art high-performance hardware implementation of Curve448. They provide three different designs (i.e., lightweight, area-time efficient, and high-performance); in particular, we compare our proposed design with their high-performance variant. Our proposed design increases the throughput by 242% for the unprotected design and by 259% for the protected design. Their approach is based on the refined Karatsuba formula by Bernstein in [29], employing five levels of Karatsuba computation and parallel multiplication using 81 DSP cores. However, the multilevel Karatsuba approach yields a longer addition tree that increases the critical path delay, limiting their operating frequency to 95 MHz, which is lower than our design.

Table 3 provides the detailed performance analysis with a comparison to their design. The latency of our architecture outperforms the state-of-the-art in all underlying field arithmetic and ECC group operations. The signifi-

TABLE 2
Performance comparison of the proposed High-Performance ECC Processor over Curve448 with existing literatures

Design	Platform	SCA*	Slices	DSP	BRAM	Latency [CCs]	Max. Freq [MHz]	Total Time [ms]	Throughput [OP/s]	Area×Time** $[\times 10^{-3}]$	DSP×Time $[\times 10^{-3}]$
[10]	Zynq 7020	(-)	1,580	33	14	328,286	357	0.92	1,087	4,490	30.36
		(+)	1,648	35	14	473,926	335	1.41	709	7,259	49.35
[11]	Zynq 7020	(+)	1,985	33	14	499,344	341	1.46	685	7,716	48.18
		(++)	2,056	33	14	547,728	341	1.61	621	8,623	53.13
[12]	Virtex-7	(-)	50,143 (LUT)	-	-	372,742	325	1.15	870	14,416***	-
[13]	Zynq 7020	(-)	4,354	81	-	77,702	95	0.82	1,220	10,212	66.42
		(++)	4,424	81	-	133,254	95	1.40	714	17,534	113.40
This work	Zynq 7020	(-)	6,946	88	-	30,469	128	0.24	4,167	3,779	21.12
		(++)	6,984	88	-	49,735	126	0.39	2,564	6,156	34.32
	Artix-7	(-)	6,826	88	-	30,469	127	0.24	4,167	3,750	21.12
		(++)	6,934	88	-	49,735	125	0.40	2,500	6,294	35.20
	Kintex-7	(-)	7,210	88	-	30,469	237	0.13	7,692	2,081	11.44
		(++)	7,269	88	-	49,735	230	0.22	4,545	3,535	19.36
	Virtex-7	(-)	7,521	88	-	30,469	250	0.12	8,333	1,959	10.56
		(++)	7,666	88	-	49,735	245	0.20	5,000	3,293	17.60

* (-): no protection, (+): scalar blinding and point randomization countermeasures,
(++): scalar blinding and point re-randomization countermeasures

** Area = Slices + DSPs×100

*** Area = LUTs/4 (Assume 1 Slice contains 4 LUTs as mentioned in specification [24])

TABLE 3
Performance Analysis of Proposed ECC Processor in comparison with State-of-the-art on Zynq 7020 FPGA

Operations	Clock Cycles	
	Niasar et al. [13] @95 MHz	Our Work @128 MHz
1 x Modular Addition	7	2
1 x Modular Subtraction	7	2
1 x Modular Multiplication	15	12
10 x Modular Multiplication	150	48
1 x Modular Inverse	6,917	5,544
Montgomery Ladder Step	158	52
Single ECC Point Multiplication	77,702	30,469
Total Latency [ms]	**0.82**	**0.24**

cant latency improvement is mainly due to a pipelined modular multiplier, which is constructed from a 244-bit fully pipelined multiplier and proposed fast reduction over $p = 2^{448} - 2^{224} - 1$. Thanks to the novel variant of the Karatsuba formula, we can enable the parallelization at the digit multiplication level without causing large delay propagation caused by additions in the recursion tree while offering relatively low DSP block utilization. Although a single modular multiplication operation does not give significant latency improvement (i.e., 12 cycles compared to 15 cycles), employing multiple operations (i.e., 10 modular multiplications as in Equation 2) results in a significant latency improvement due to pipelining compared to their design (i.e., 52 cycles compared to 158 cycles). Furthermore, all the stages in the 224-bit multiplier are nearly busy during the Montgomery ladder operation with the utilization of $\frac{48}{52} \times 100 \simeq 92\%$, making the use of the pipeline architecture in the highest efficiency. On the other hand, the modular inversion via FLT consumes almost 18% of the total latency and is considered an inefficient method in our design. This is because the exponentiation $z^{2^{448}-2^{224}-3} \mod p$ requires 462 consecutive modular multiplications rather than parallelization through the pipelining architecture.

In terms of area, their design has lower slices utilization (i.e., 4,354 slices for the unprotected design and 4,424 slices for the protected design). However, in terms of Area×Time tradeoff, our design is 63% more efficient for both the unprotected and protected designs. It turns out that the cost of higher utilization is well absorbed by the latency improvement. Note that we use the same assumption as they do where the area is equivalent to slices + DSPs, while each DSP is assumed to be equivalent to 100 slices.

It is worth mentioning that the first hardware implementation of Curve448 was carried out by Sasdrich and Güneysu in [10], who later proposed the protected architecture by considering side-channel attack countermeasures [11]. They demonstrated an evaluation to detect scalar- and base-point-dependable leakage on hardware with side-channel protections (i.e., scalar blinding and point randomization) and proved that their methods are secure against side-channel attacks. Thanks to their results, we also include side-channel protections (i.e., scalar blinding, base-point randomization, and continuous point randomization) in our protected design, yet we present a 313% speed-up compared to their results on the same target device (i.e., Zynq 7020).

Shah et al. [12] proposed a LUT-based implementation

targeting Virtex-7, employing the RSD technique for the arithmetic operations. Their proposed designs aimed to be platform independent by using LUTs only, consuming 50,143 LUTs with a throughput of 870 ECPM operations per second, yet our design is 858% faster than their design.

7 Conclusions

In this paper, we proposed a high-performance ECC processor over Curve448 that outperformed all the previous results in terms of execution time. The implementation on the Xilinx 7-series FPGA Virtex-7, Kintex-7, Artix-7, and Zynq 7020 yielded execution times of 0.12, 0.13, 0.24, and 0.24 ms, respectively. The speed was obtained by utilizing a novel variant of the Karatsuba for asymmetric digit multiplier, constructing a high-throughput 244-bit fully pipelined multiplier. The method combined schoolbook long and Karatsuba multiplication, allowing its digit multiplication to be performed in parallel while leveraging the full capability of asymmetric DSP blocks. It is worth mentioning that the algorithm even works on arbitrary degrees, which means it can be generalized for wider use in a cryptographic algorithm that requires multiplication. In sequence, the interleaved fast reduction over $2^{448} - 2^{224} - 1$ was presented, yields a high throughput 12-stage modular multiplier with four stages of input delay. Furthermore, we also proposed certain components to maximize the speed gain and the overall performance, such as employing a low-latency modular adder/subtractor as well as efficient scheduling of the Montgomery ladder. Finally, the proposed architecture was extended with both vertical and horizontal side-channel protection through well-known countermeasures such as scalar blinding, base-point randomization, and continuous randomization.

Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (2019-0-01343, Regional strategic industry convergence security core talent training business) and supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2021-2020-0-01797) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

References

[1] 3GPP, "Security architecture and procedures for 5g system," *Technical Specification (TS) 3GPP TS 33.501 V17.4.1 (2022–01)*, 2022.

[2] C. Fan, S. Ghaemi, H. Khazaei, and P. Musilek, "Performance evaluation of blockchain systems: A systematic survey," *IEEE Access*, vol. 8, pp. 126 927–126 950, 2020.

[3] A. Langley, M. Hamburg, and S. Turner, "Rfc 7748: Elliptic curves for security," *Internet Research Task Force (IRTF)*, 2016.

[4] D. J. Bernstein, "Curve25519: new diffie-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207–228.

[5] M. Hamburg, "Ed448-goldilocks, a new elliptic curve." *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 625, 2015.

[6] E. Rescorla, "The transport layer security (tls) protocol version 1.3," Internet Requests for Comments, RFC Editor, RFC 8446, August 2018.

[7] L. Chen, D. Moody, A. Regenscheid, and K. Randall, "Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters," National Institute of Standards and Technology, Tech. Rep., 2019.

[8] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[9] N. Bindel, U. Herath, M. McKague, and D. Stebila, "Transitioning to a quantum-resistant public key infrastructure," in *International Workshop on Post-Quantum Cryptography*. Springer, 2017, pp. 384–405.

[10] P. Sasdrich and T. Géneysu, "Cryptography for next generation tls: Implementing the rfc 7748 elliptic curve448 cryptosystem in hardware," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[11] P. Sasdrich and T. Güneysu, "Exploring rfc 7748 for hardware implementation: Curve25519 and curve448 with side-channel protection," *Journal of Hardware and Systems Security*, vol. 2, no. 4, pp. 297–313, 2018.

[12] Y. A. Shah, K. Javeed, M. I. Shehzad, and S. Azmat, "Lut-based high-speed point multiplier for goldilocks-curve448," *IET Computers & Digital Techniques*, vol. 14, no. 4, pp. 149–157, 2020.

[13] M. B. Niasar, R. Azarderakhsh, and M. M. Kermani, "Efficient hardware implementations for elliptic curve cryptography over curve448," in *International Conference on Cryptology in India*. Springer, 2020, pp. 228–247.

[14] A. A. Karatsuba and Y. P. Ofman, "Multiplication of many-digital numbers by automatic computers," in *Doklady Akademii Nauk*, vol. 145, no. 2. Russian Academy of Sciences, 1962, pp. 293–294.

[15] R. Salarifard and S. Bayat-Sarmadi, "An efficient low-latency point-multiplication over curve25519," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 10, pp. 3854–3862, 2019.

[16] A. M. Awaludin, H. T. Larasati, and H. Kim, "High-speed and unified ecc processor for generic weierstrass curves over gf (p) on fpga," *Sensors*, vol. 21, no. 4, p. 1451, 2021.

[17] G. H. Khachatrian, M. K. Kuregian, K. R. Ispiryan, and J. L. Massey, "Fast multiplication of integers for public-key applications," in *International Workshop on Selected Areas in Cryptography*. Springer, 2001, pp. 245–254.

[18] M. Scott, "Missing a trick: Karatsuba variations," *Cryptography and Communications*, vol. 10, no. 1, pp. 5–15, 2018.

[19] D. B. Roy, D. Mukhopadhyay, M. Izumi, and J. Takahashi, "Tile before multiplication: An efficient strategy to optimize dsp multiplier for accelerating prime field ecc for nist curves," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[20] P. M. C. Massolino, P. Longa, J. Renes, and L. Batina, "A Compact and Scalable Hardware/Software Co-design of SIKE," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020.

[21] A. Langley, M. Hamburg, and S. Turner, "Elliptic curves for security," Internet Requests for Comments, RFC Editor, RFC 7748, January 2016.

[22] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[23] B. Devlin, *Blockchain Acceleration Using FPGAs - Elliptic curves, zk-SNARKs, and VDFs*, ZCASH Foundation, 2019.

[24] Xilinx, *7 Series FPGAs Data Sheet: Overview*, 2020 (accessed January 26, 2022), https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[25] T. B. Preußer, M. Zabel, and R. G. Spallek, "Accelerating computations on fpga carry chains by operand compaction," in *2011 IEEE 20th Symposium on Computer Arithmetic*. IEEE, 2011, pp. 95–102.

[26] Xilinx, *7 Series DSP48E1 Slice User Guide*, 2018 (accessed December 28, 2020), https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[27] W. Schindler and A. Wiemers, "Efficient side-channel attacks on scalar blinding on elliptic curves with special structure," in *NIST Workshop on ECC standards*, 2015.

[28] A. M. Awaludin, D. Pratama, and H. Kim, "Anytrng: Generic, high-throughput, low-area true random number generator based on synchronous edge sampling," in *International Conference on Information Security Applications*. Springer, 2021, pp. 157–168.

[29] D. J. Bernstein, "Batch binary edwards," in *Annual International Cryptology Conference*. Springer, 2009, pp. 317–336.