LLP-ECCA: A Low-Latency and Programmable Framework for Elliptic Curve Cryptography Accelerators

Yicheng Huang[1], Xueyan Wang*[1], Tianao Dai[1], Jianlei Yang[2], Zhaojun Lu[3], Xiaotao Jia[1], Gang Qu[4] and Weisheng Zhao[1]
[1]School of Integrated Circuit Science and Engineering, Beihang University, Beijing, 100191, China
[2]School of Computer Science and Engineering, Beihang University, Beijing, 100191, China
[3]School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China
[4]Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

2024 IEEE International Test Conference in Asia (ITC-Asia) | 979-8-3315-4033-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ITC-Asia62534.2024.10661323

***Abstract*—Elliptic curve cryptography (ECC) plays a pivotal role in safeguarding data integrity and authentication in contemporary communication contexts, particularly within the domain of Intelligent Transport Systems (ITS). In the realm of ITS, vehicles communicate via the V2X (vehicle-to-everything) protocol, necessitating low-latency responses and minimal power consumption. Given the evolving nature of V2X protocol standards across the globe, programmability becomes a rigid requirement. However, existing strategies cannot meet all these vehicular equipment demands. This paper introduces a novel framework tailored for ECC acceleration to address the issues. Specifically, we propose the design of an Application Specific Instruction Set Processor (ASIP), augmented by pipeline and dual-issue techniques. Furthermore, the envisioned ASIP integrates a hybrid control framework founded on Finite State Machines (FSM), facilitating agile and effective management. Notably, a general GF(p_{256}) Barrett modular multiplier is specially devised to optimize latency and area utilization. Experimental results on Xilinx Kintex Ultrscale+ FPGA demonstrate that the proposed ECC accelerator generates a signature within 131us and verifies a message within 181us, and the performance meets the requirements of today's V2X standard.**

***Index Terms*—Elliptic Curve Cryptography (ECC), Hardware Acceleration, Programmable, V2X**

I. Introduction

Although post-quantum cryptography has achieved extensive development [1], [2], classical public-key cryptography algorithms still remain the mainstream security approach currently. Elliptic Curve Cryptography (ECC) is a widely used public-key cryptography system, offering various security functions such as data integrity and authentication, among others. Currently, ECC has been deployed in various cyber-physical system applications, such as Intelligent Transportation Systems (ITS) [3]. In ITS, vehicles need to communicate with other members through V2X (vehicle-to-everything) protocols. Specifically, vehicles can communicate with other vehicles through V2V (vehicle-to-vehicle) protocol, which may involve messages containing private information like position [4]. Digital signature algorithms based on ECC have been widely used to guarantee authentication and data integrity.

For V2V, different standards are adopted across the globe. China [5], the United States [6], and Europe [7] recommend different curves (SM2, NIST P256r1, and Brainpool P256r1) and algorithms (SM2 and ECDSA). As the standards are still evolving, Ghosal and Conti [8] suggest that the communication and security architecture should be compatible with upcoming vehicular technologies. For example, Secp256k1 has been used in decentralized V2X [9]. Therefore, a critical requirement is supporting both mainstream and potential algorithms as well as curves while providing compatibility for future vehicular technologies. Another crucial requirement is low latency for rapid response and high throughput. In the worst-case scenario, each vehicle should be able to verify 4000 messages per second, with each verification requiring a latency of less than 20 ms [10].

TABLE I: ECDSA verification schemes in previous work.

Scheme	Throughput	Power	Latency	Programmability
CPU [11]	✗	✗	✓	✓
GPU [12]	✓	✗	✗	✓
FPGA [7]	✓	✓	✓	✗
ASIC [13]	✓	✓	✓	✗
This work	✓	✓	✓	✓

There have been several attempts at elliptic curve cryptography accelerator (ECCA) for V2V on different platforms, as shown in TABLE I. Regarding the programmability discussed in this article, we consider it reflects the extent to which the system supports various curves and algorithms. However, none of them can concurrently meet all the requirements for V2V. Although CPU [11] and GPU [12] can offer good programmability, the throughput or latency limits their deployment on automotive platforms. Some customized designs [7], [10], [13]–[15] based on FPGA or ASIC achieve promising performance for specific curves or algorithms. However, these designs generally lack programmability for future security algorithms.

Regarding the above key issues, this paper proposes a low-latency, programmable, and multi-curve-supported ECC accelerator. Our main contributions can be summarized as follows:

- We present an Application Specific Instruction Set Processor (ASIP) for flexible ECC acceleration. The pipeline and pseudo-dual-issue architecture is designed to further improve the parallelism of computation.
- A hybrid control mechanism for operator scheduling in

*Corresponding author is Xueyan Wang, e-mail: wangxueyan@buaa.edu.cn.
Xueyan Wang's work was supported in part by the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001), Beihang Frontier Cross-Fund Project, and State Key Laboratory of Integrated Chips and Systems.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.

ASIP to achieve efficient programmability.

- A novel Barrett modular multiplier is designed based on a truncated multiplier and multiple reduction units, which enables efficient pipelined modular multiplication.
- The proposed strategies are implemented and verified on FPGA platform, which has demonstrated highly promising results.

The remainder of this paper is structured as follows: Section II presents essential background information. Section III details the architecture of the ECC accelerator and the program approach. Section IV introduces the proposed finite field multiplier. Section V presents the experimental results. Finally, Section VI concludes the article.

II. Preliminary

A. Modular Operation

A finite field, or Galois field (GF), is a mathematical construct containing a finite set of elements. Take the prime number 7 as an example, the elements in GF(7) are $\{0, 1, \cdots, 6\}$. All operations within GF(7) are performed $mod\ 7$, encompassing multiplication (MM), addition (MA), subtraction (MS), and inversion (MI). This modular arithmetic ensures that the results remain within the defined set of elements.

MM is the main modular operation in ECC. MM contains a standard multiplication and a reduction. Both of these operations are timing-consume operations. The classical algorithms for MM are Montgomery [16] and Barrett (BMM) [17]. These algorithms aim to accelerate MM by incorporating hardware-friendly $Mod/Div\ 2^n$ operation, as Algorithm 1 shows. They have no modulus restrictions but require at least three multiplications. Moreover, Montgomery reduction leads to the consumption for the conversion between the Montgomery domain and normal number presentation. The Barrett reduction needs pre-calculation of variable u. Another method is to optimize the reduction algorithm for specific modules like pseudo-Mensen prime [18], which is suitable for a fast reduction algorithm. For example, NIST P256r [16] and SM2 suit these methods. Unfortunately, Brainpool P256r1 does not possess such properties.

Algorithm 1: Barrett reduction Algorithm

Input: $p, b \geq 3, k = \lfloor log_b p \rfloor + 1, 0 \leq z < b^{2k},\ \mu = \lfloor b^{2k}/p \rfloor$.
Output: $z\ mod\ p$.

1 $q = \lfloor \lfloor z/b^{k-1} \rfloor * \mu / b^{k+1} \rfloor$.
2 $r = (z\ mod\ b^{k+1}) - (q * p\ mod\ b^{k+1})$.
3 If $r < 0$ then $r = r + b^{k+1}$.
4 While $r \geq p$ do: $r = r - q$.
5 Return r.

B. ECC Backgroud

V2V protocols typically utilize the Weierstrass Curve form. The elliptic curve E over GF(p_{256}) is defined by Weierstrasβ form [10], as Eq. (1) shows, $a, b \in$ GF(p_{256}).

$$E : y^2 = x^3 + ax + b\ mod\ p_{256} \tag{1}$$

The most time-consuming operations are point multiplication (PM) and double point multiplication (DPM) for signature generation/verification, respectively. Let P_1 and P_2 be two points in E, and let k and l be random scalar numbers. The PM result is another point Q_1 in E that $Q_1 = kP_1$, and the DPM result is also a point Q_2 in E that $Q_2 = kP_1 + lP_2$. In [16], Shamir's trick and comb methods are introduced to boost PM and DPM, as shown in Fig. 1. It is feasible to reuse the precomputed points if the curve parameters are not changed.

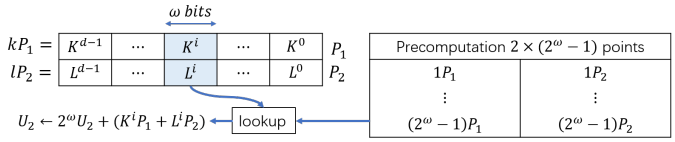

Fig. 1: Shamir'trick and Comb Methods for PM/DPM

$$(x, y) \rightarrow \{(X, Y, Z) | X = x, Y = y, Z = 1\} \tag{2}$$

$$\{(X, Y, Z) | Z \neq 0\} \rightarrow (X/Z^{-2}, Y/Z^{-3}) \tag{3}$$

$$\begin{cases} T_{pa} = X_2 Z_1^2 - X_1 Z_2^2, \\ X_{pa} = (Y_2 Z_1^3 - Y_1 Z_2^3)^2 - T_{pa}^2 (2X_1 Z_2^2 + T_{pa}), \\ Y_{pa} = (Y_2 X_1^3 - Y_1 Z_2^3)(X_1 (Z_2 T_{pa})^2 - X_{pa}) - Y_1 Z_2^3 T_{pa}^3, \\ Z_{pa} = Z_1 Z_2 T_{pa}. \end{cases} \tag{4}$$

Both PM and DPM heavily rely on Point Addition (PA) and Point Double (PD) operations. Generally, PM and DPM results are in Affine coordinates represented as Eq. (2). It's worth using Jacobian coordinate, represented as Eq. (3), to reduce time-consuming MI operations, and we only need one-time MI computation when Jacobian converts to Affine by Eq. (3). In the Jacobian coordinate representation, the PA operation is described by Eq. (4), and the PD operation is described by Eq. (5). To adapt to Comb PM/DPM, we employ the repeated point doubling algorithm (MPD) described in [16] with $\omega = 4$.

$$\begin{cases} X_{pd} = (3X_1^2 + aZ_1^4)^2 - 8X_1 Y_1^2, \\ Y_{pd} = (3X_1^2 + aZ_1^4)(4X_1 Y_1^2 - X_3) - 8Y_1^4, \\ Z_{pd} = 2Y_1 Z_1. \end{cases} \tag{5}$$

C. Digital Signature Scheme

Generally, digital signature schemes consist of signature generation and signature verification. They can be used to provide data origin authentication, data integrity, and non-repudiation proofs [16]. In the USA and Europe, ECDSA [19] is considered for V2V communication. There are many other signature schemes used in V2X, such as SM2 [20] in China.

III. ECCA Architecture

A. Design Concept

As emphasized in Section I, programmability stands as a critical feature in the evolution of future vehicle technologies. Finite State Machines (FSMs) offer a simple way to design systems without complex control logic, making them suitable for smaller designs with fixed functions. For instance, in top-level Elliptic Curve Cryptography (ECC) applications like Signature generation, which involve specific operations with straightforward control logic, FSM-designed modules work well. Complementing the simplicity of FSMs, the concept of an Application Specific Instruction Set Processor (ASIP) stands out for its capacity to accommodate complex functionalities through diverse program execution. It facilitates using classic techniques like pipelining for enhanced performance and dual-issuing for parallelism. Furthermore, ASIPs afford the programmability of incorporating user-defined instructions, thereby facilitating compatibility with various protocols.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.

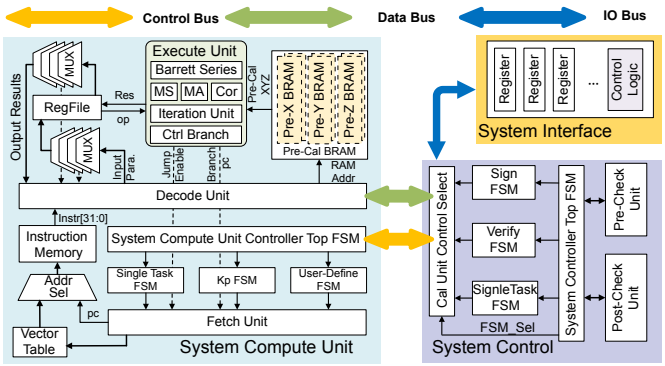

Fig. 2: Proposed overall ECC accelerator architecture.

rs1[3:0]	rs2[3:0]	opcode[3:0]	rd[3:0]

Fig. 3: Instruction format.

We develop a hybrid framework that combines the strengths of both FSMs and ASIPs. The true innovation lies in a pseudo-dual-issue pipeline ASIP, optimized for low-level operations, alongside an FSM tailored for high-level applications. The proposed framework achieves a good balance between performance, programmability, and ease of use, which are all of great importance for modern vehicle communication systems.

B. The ASIP Specification

All instructions are in the same 16-bit format, and each instruction consists of four parts: rs1, rs2, rd, and opcode, each of which is 4 bits long, as shown in Fig. 3. The rs and rd are the indices of registers in the Register file (RF). The opcode determines the function of an instruction. TABLE II demonstrates that our architecture supports three kinds of instructions: arithmetic, branch and system. For programmability, we reserve some instruction space for user implementation.

TABLE III details the names and functions of registers in RF. The T_0 register supplies constant numbers $(a, 0, 2^{-1}\ mod\ p)$ controlled by sys instructions. The RF is designed by a 16×258-bit register array with four read ports. Thus, it supports four simultaneous read requests generated by two instructions. Our design effectively supports simultaneous writing to the RF, leveraging the varying pipeline depths of the BMM and MA/MS to enable writing to all registers in 1 cycle, thus providing significant parallelism and meeting the high concurrency write requirements of the Datapath.

C. Datapath

1) Fetch Unit and Firmware Memory: All the program information is stored in the firmware memory (FM) and the Fetch Unit chooses the appropriate segment of code for execution. For firmware memory, it's combined with vector table (VT) and instruction memory (IM). The VT, implemented by a 32×9 bit register file, provides the program start address (Entry) in IM. The IM, which a 32×1024bit SRAM implements, stores the code for basic function programs like PM, PD, etc. The user-defined program follows the basic function program. By changing VT and user-defined programs, users can add their functions to their application. The fetch unit reads instructions from the instruction memory (IM) each cycle, incrementing the memory address (program counter,

TABLE II: Proposed instruction set for ECCA.

Type	Name	Opcode	Explanation
Arith	add	0xA	$rd = rs1 + rs2\ Mod\ p$
	sub	0xB	$rd = rs1 - rs2\ Mod\ p$
	bmm-a	0x4	$rd = rs1 \times rs2\ Div\ 2^{254}$
	bmm-b	0x6	$rd = rs1 \times p\ Mod\ 2^{258}$
	bmm-c	0x7	$rd = rs1 \times u\ Div\ 2^{258}$
	bmm-d	0x2	$rd = rs1 \times rs2\ Mod\ 2^{254}$
	cor	0x3	$rd = BMMCorrection(rs1, rs2)$
Branch	branch	0x9	Branch,Jump condition depend on rd
System	sys	0x0	Program success/fail, change system parameter
Reserved	Reserved	others	leave for user-define

TABLE III: Register name and function in assembly

Index	Name	Function
0	T_0	Constant Nums can be configed
1-6	$X_1, Y_1, Z_1, X_2, Y_2, Z_2$	Input parameter; Save temp data
7-9,14,15	X_3, Y_3, Z_3, T_5, T_6	Output result; Save temp data
10-13	T_1, T_2, T_3, T_4	Save temp data

PC) one by one during normal program execution. When encountering a branch instruction, the PC may jump to a new section of the program. Finally, when a section of the program is completed, followed by a sys instruction, the PC will be held and await the start of a new program.

2) Decode Unit: Decode Unit parses the instruction and allocates the data and specific operations to the corresponding compute unit. It is worth noting that as the ASIP adopts a pseudo-dual-issue architecture, the Decode Unit receives two instructions per cycle, assuming that they are named $Instr_1$ and $Instr_2$. Among them, $Instr_2$ is a bmm-series instruction (bmm-a/b/c/d), while $Instr_1$ must not be a bmm-series instruction, such as $bmm - c, t5, p, t5; add, z1, t1, t3$.

To facilitate data input and output, we have designed a data matrix, as depicted in Fig. 4. During program initialization, input parameters are loaded into input functional registers in RF via the matrix. After the program finishes, results are retrieved from the output functional registers through the matrix. Encoding mux selection as a binary sequence efficiently meets specific data input requirements. Moreover, the Decode Unit controls the pre-computed memory, which saves pre-computed points in Fig. 1. It contains two sets of pre-computed points (P_1 and P_2), each comprising three blocks for (X, Y, Z) in Jacobian coordinates. Thus, a total of six 15×258-bit SRAM memories are required.

3) Execute Unit: As Fig. 4 shows, our proposed Execute Unit (EU) consists of several Compute Units (CUs), like MA. The execution of the instructions relies on the corresponding CUs, as TABLE IV demonstrates. Almost all the CUs operate in pipeline mode, except for the Iteration Unit used for MI. Given the varying pipeline depths of the CUs, we have chosen to optimize the code to prevent data conflicts, rather than developing complex hardware pipeline control logic. However,

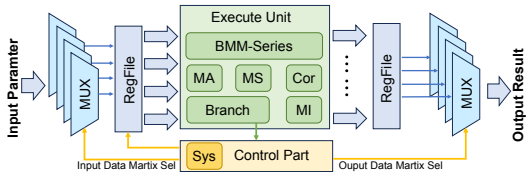

Fig. 4: Datapath architecture

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.

TABLE IV: Matching for instructions and Compute Units

Compute Unit	Latency	Instruction	Writeback
Mod Add	1	add	yes
Mod Sub	1	sub	yes
BMM-Cor	1	cor	yes
BMM	2	bmm-(a/b/c/d)	yes
Ctrl Branch	2	branch	no
Iteration Unit	~500	/	yes

MI is controlled by a FSM, since it is not a frequent operation. The sys instruction executing in the control part controls the program and sets parameters in the RF, which bridges the gap between the control and calculation parts.

To leverage the parallelism of low-level operators, the proposed architecture employs a pseudo-dual-issue architecture, which supports launching two instructions per cycle with almost the same area, provided that the two instructions are executed on different compute units in TABLE IV. For instance, the $bmm - a$ cannot be issued concurrently with $bmm - b$, but it can be issued concurrently with add.

D. Control Mechanism

Our proposed control mechanism is divided into FSM and ASIP two parts. Fig. 5 (a) illustrates our understanding of the hierarchy of compute abstract level. The hierarchy is divided into three levels: application level ($layer_{\text{app}}$), point operation level ($layer_{\text{point}}$) and modular operation level ($layer_{\text{mod}}$).

In our architecture, the $layer_{\text{app}}$ is implemented by the System Control Unit (Sysctrl), whereas the $layer_{\text{point}}$ and $layer_{\text{mod}}$ are implemented by the System Compute Unit (Syscomp), as Fig. 2 illustrates. From a control perspective, PM/DPM in the $layer_{\text{point}}$ are both scheduling tasks that do not participate in computing; however, they are also sub-level operators relative to $layer_{\text{app}}$. Therefore, we implement PM/DPM using an FSM model, and the remaining components in Syscomp are implemented using an ASIP model.

For low-level and highly reusable operators, the compute flow is that Fetch→IM→Decode→RF→CU→RF. Until the program finishes, the compute flow continues. It is worth noting that the compute flow is fixed and there is no branch requirement in these functions except for the input check. Therefore, it is feasible to optimize performance by considering the pipeline depths of various operators in the accelerator architecture during code compilation, whether it is done automatically by software with constraints or manually.

For high-level operators, we adopt a design model based on FSM. The order of control flow is as follows: FSM in Sysctrl→FSM in Syscomp→EU, as demonstrated in Fig. 5 (b).

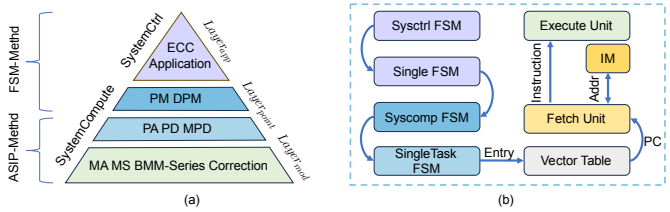

Fig. 5: Computation hierarchy for ECC acceleration. (a) control mechanism decomposition and (b) control flow example

E. An Illustrative Example

Our idea to support user-defined programs is to make user-defined programs a specific function in our compute abstract architecture. In detail, we leave some entries for the user's program, corresponding to different program startup addresses in the VT and users can launch it from $layer_{\text{app}}$. Next, we use a user-defined function example to show how the controller works with FSM and ASIP control mechanism as Fig. 5 (b). We set the function ($x_3 = x_1 \times y_1 + x_2 \times y_2 + z_1 \ mod \ p$) as a single task and change VT's entry for this function. The Sysctrl schedules data and launches sub-level FSM, transmitting the command until the lowest-level FSM in Syscomp. The lowest-level FSM looks up the VT for a specific program entry (single task), sets the PC as the entry value, launches the Fetch Unit, and waits for the program to finish.

As mentioned in Section III-A, we have constructed a pseudo-dual-issue architecture intended to take advantage of parallel computing. The code presented in TABLE V serves as the demo of the user-defined function. Steps 4/6 demonstrate the pipeline advantage, where $x_2 \times y_2$ begins execution before $x_1 \times y_1$ completes. And steps 7/8 highlights the pseudo-dual-issue advantage, where two instructions are executed simultaneously.

TABLE V: Example for pipeline and pseudo-dual-issue

(Step.)Code	Assembly	(Step.)Code	Assembly
(1).124e000a	bmm-a,x1,y1,t5;nop	(7).456ffe3a	bmm-d,x2,y2,t6; cor,t6,t5,t1
(2).126f000a	bmm-d,x1,y1,t6; nop	(8).e02e3aac	bmm-c,t5,p,t5; add,z1,t1,t3
(3).e07e000a	bmm-b,t5,u,t5;nop	(9).000a000a	nop;nop
(4).454d000a	bmm-a,x2,y2,t4;nop	(10).000afe3b	nop; cor,t6,t5,t2
(5).e02e000a	bmm-c,t5,p,t5;nop	(11).000acba7	nop;add,t3,t2,x3
(6).d07e000a	bmm-b,t4,u,t5;nop	(12).000a0080	nop;FINISH

IV. Barrett Modular Multiplier

To support a wide range of curves, we choose to implement the Modular Multiplication based on the Barrett reduction algorithm. However, directly implementing the algorithm can lead to high latency and large area overhead. While decomposing the multiplier in BMM with smaller multipliers is feasible, it complicates the reduction operation. The key problem lies in designing a suitable decomposition method with an efficient reduction strategy. Our idea involves using small multipliers to directly generate partial products (PR) and employing Carry-Save Adder (CSA) technology [21] to produce the reduction result. This entails extracting carry-out information from the PR without accumulating them. This section will introduce the BMM in our design, encompassing both the multiplier and BMM reduction units.

A. Proposed Multiplier

Our proposed BMM is based on the Algorithm 1 with parameter $b = 4, k = 128$. Therefore, the bit length of u is 257 bits. Let the input of BMM as $A, B < p$. With $z = A * B < p^2 < 2^{512}$, the bit length of $z \ mod \ b^{k+1}$ is 258 bits. To reduce the accelerator area, our design reuses one multiplier for all multiplication in the Barrett Algorithm. The multiplier is a $258 \ bit \times 257 \ bit$ multiplier ($Mul_{258\times257}$) to accommodate the maximum bit width of operands in BMM. The input of $Mul_{258\times257}$ is In_a (258 bits) and In_b (257 bits). Fig. 7 demonstrated our decomposition strategy. The compute task for this stage is shown in Eq. 6.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.

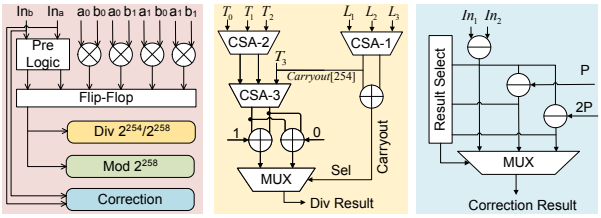

Fig. 6: Hardware architecture for BMM. (a) BMM architecture (b)$Div\ 2^{254}$ unit and (c) correction unit.

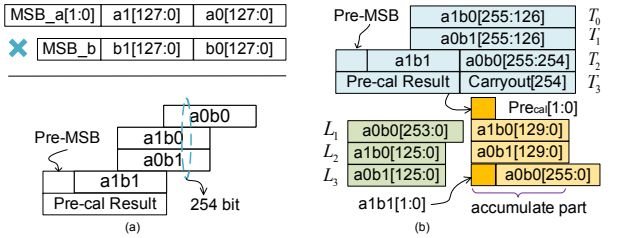

Fig. 7: Decomposition strategy for BMM. (a) multiplication decomposition and (b)decomposition for $Mod\ \backslash Div$.

To balance the multiplier area and avoid the occurrence of the large multiplier, we design our $Mul_{258\times257}$ based on 128-bit multipliers. In this stage, we pre-calculate Pre_{msb} and Pre_{cal} and save the multiplier result such as a_1b_1 without accumulating PR_{ab}. All the PRs in this stage are registered for the next reduction unit.

$$\begin{cases} In_a = msb_a 2^{256} + a = msb_a 2^{256} + a_1 2^{128} + a_0, \\ In_b = msb_b 2^{256} + b = msb_b 2^{256} + b_1 2^{128} + b_0, \\ PR_{ab} = a_1b_1 2^{256} + a_1b_0 2^{128} + a_0b_1 2^{128} + a_0b_0, \\ Pre_{msb} = msb_a \times msb_b, \\ Pre_{cal} = msb_a B 2^{256} + msb_b A 2^{256}. \end{cases} \tag{6}$$

B. Proposed Reduction Unit

The Barrett reduction contain $Mod\ 2^{258}$, $Div\ 2^{258}$, $Div\ 2^{254}$ and correction operation. Since it's easy to implement $Div\ 2^{258}$ operation based on $Div\ 2^{254}$ operation, the key problem is to handle $Mod\ 2^{258}$ and $Div\ 2^{254}$ operation. The partial product for $Mod\ 2^{258}$ involves four parts, highlighted in yellow in Fig. 7 (b). Since $a_0b_0[127:0]$ is equal to $Mod\ 2^{256}$ result[127:0], we use CSA (also named as 3:2 compressor) to compress high 130 bit, the accumulate part in Fig. 7 (b), to two output and use only a 130bit adder to get the $Mod\ 2^{256}$ result[257:128]. In this method, the timing path for $Mod\ 2^{258}$ is equal to $T_{mod\ 2^{258}} = 2T_{CSA} + T_{Adder-130bit}$.

$Mod\ 2^{258}$ operation does not need the carry-out value, while $Div\ 2^{254}$ operation needs the carry-out value generated by the low 254 bit. In Fig. 7 (b), the green part is the low 254 bits and the blue part is the high 261 bits. The Fig. 6 (b) demonstrates the $Div\ 2^{254}$ architecture. We use CSA1 to compress the low 254 bit and add the CSA1 output to get carry-out info (co). The Carryout[254] of CSA1 can be absorbed in the compression of high 261 bits by CSA3. The summation of the high 261 bits is Tmp_{div254}. To reduce the critical path delay (CPD), we calculate $Tmp_{div254}+1$ parallel and select the right $Div\ 2^{254}$ result by CSA1's co bit. In this method, the timing path is $T_{mod\ 2^{258}} = 2T_{CSA} + T_{Adder-261bit} + T_{MUX}$.

The reduction unit depicted in Fig. 6 (c) corresponds to steps 2-4 outlined in Algorithm 1. Let $In_1 = z\ mod\ b^{k+1}$ and $In_2 = q * p\ mod\ b^{k+1}$ be the input of reduction unit. Considering the steps 2 and 3 in Algorithm 1, the equivalent operation is $r = ((z\ mod\ b^{k+1}) - (q*p\ mod\ b^{k+1}))\ mod\ b^{k+1}$. Hankerson et al. [16] have proved that when $z < b^{2k}, b^k > p \geq b^{k-1}$ satisfied, at most two subtractions at step 4 are required to make sure $0 \leq r < q$. Therefore, to unroll the loop in step 4, we calculate $r-q, r-2q$ in parallel. Finally, according to the priority order of $r, r-q, r-2q$, select the number that is positive itself but the subsequent number is negative as the output.

V. Experimental Results

A. Experimental Setup

We implement our design on Xilinx Virtex-7 and Kintex UltraScale+ (Kintex US+) platforms using Vivado Design Suite 2023.2. Since PM/DPM operations are the main performance overhead in the signature schemes, we focus on comparing their latency with previous works. To ensure a fair comparison, we use two PMs as a substitute for those works that did not incorporate DPM. As elucidated in Section II-B, our PM/DPM method necessitates precomputed points. We define the best-case performance scenario as when precomputation is prepared (without pre-cal), whereas the worst-case scenario entails the need to calculate precomputation points before PM/DPM (with pre-cal). Furthermore, we relied on data from [22] for the hash function, which entails 80 cycles for reference purposes.

B. Effect of Pseudo-Dual-Issue Techniques

Since the PM and DPM are influenced by the underlying PM/DPM algorithm, we compare the basic point operations PA, PD and MPD over Brainpool P256r1 to evaluate the optimal outcome of the pseudo-dual-issue design. TABLE VI shows that in comparison with the only pipeline optimization, PA decreases cycle counts by 22, achieving a performance surge of +24.4%. Likewise, PD cuts down 22 cycles, leading to a performance enhancement of +30.5%. Furthermore, MPD reduces cycles by 68, boosting performance by +30.3%.

TABLE VI: Pseduo-dual-issue performance improvement

Operation	Pipe.	Pipe.+pseudo-dual-issue	Improve.
Point Addition	90	68	+24.4%
Point Double	72	50	+30.5%
Repeated Point Doubling (ω=4)	224	156	+30.3%

Fig. 8 demonstrates that SystemIO, SystemCtrl and SystemComp account for ~7%, ~7% and ~86% of this implementation's total LUT resource on the Xilinx Virtex-7 and Kintex US+ platforms, respectively.

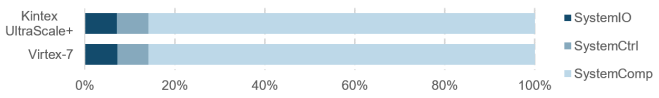

Fig. 8: The LUT resource utilization of submodules on FPGA.

C. Comparison With Previous Work

TABLE VII represents the utilization and maximum clock frequency of our proposed design in different platforms. TABLE VIII demonstrates the performance of our implementation on the Kintex US+ platform operating at 120.2 MHz. For optimal performance, our design can sign 7610 messages/s and verify 5530 messages/s. Additionally, the power analysis

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.

TABLE VII: Comparing PM and DPM performance on FPGA platform

Work	Curve	Platform	Slice	Lut(k)	FF(k)	DSP	Freq.(MHz)	PM Perf.	DPM Perf.	Programmable
TOIT'22 [10]	NIST P256r1	Virtex-7	-	29.3k	10.2	384	162	458.8μs	917.6μs	✗
	Brainpool P256r1	Virtex-7	-	29.4k	10.7	384	162	504.8μs	1009.6μs	✗
TC'20 [23]	Secp256k1	Virtex-7	13.26k	46.96	29	280	125	324μs	648μs	✗
Integration'18 [24]	Any	Virtex-7	4611	18.8	-	1036	86.6	730μs	1460μs	✗
TCAS-I'24 [7]	NIST P256r	Virtex-7	4227	16.9	10.7	48	120	140.3μs	174.7μs	✗
This work (After Place&Route)	Any (w/ pre-cal)	Virtex-7	14264	57.1	17.1	256	66.7	226.3μs	305.8μs	✓
	Any (w/o pre-cal)	Virtex-7	14264	57.1	17.1	256	66.7	211.3μs	275.8μs	
	Any (w/ pre-cal)	Kintex US+	13818	55.3	16.0	200	120.2	125.5μs	169.6μs	
	Any (w/o pre-cal)	Kintex US+	13818	55.3	16.0	200	120.2	117.2μs	153μs	

TABLE VIII: ECDSA and SM2 performance on Kintex US+

Algorithm	Operations	w/ pre-cal (μs)	w/o pre-cal (μs)
ECDSA	Sign	131.4	123.3
	Verify	180.8	164.5
SM2	Sign	131.4	123.3
	Verify	176.3	159.9

report from Vivado indicates that the total on-chip power is $1.013W$, which is acceptable for the vehicular environment.

As mentioned in Section I, the vehicles in V2V must have the ability to verify at least 4000 messages/s in extreme situations. The work in [10] designs a PM accelerator for V2V based on RNS, compatible with NIST P256r and Brainpool P256r1. On Kintex UltraScale+ FPGA, it computes PM in 458μs for NIST P256r and 504μs for Brainpool P256r1 but verifies only 1087 messages/s. Our implementation outperforms theirs by 5× in verification. Our proposed architecture can process PM/DPM over the secp256k1 curve with at least 30% and 52% performance improvement, respectively, while consuming almost the same resources on Virtex-7 as compared to the work presented in [23]. The work presented in [7] can achieve ~40% performance improvement with fewer resources compared to ours. However, it only supports the ECDSA algorithm over the NIST P256r1 curve. In contrast, our design (on the Kintex US+ platform) can meet the performance requirements for extreme V2V scenarios and support generic Weierstrass curves with programmability for additional functions, such as the SM2 algorithm. [24] proposed a generic PM acceleration architecture implemented on the Virtex-7 platform, which executes a PM/DPM in 730μs/1460μs, consuming 1036 DSPs. However, our implementation uses only ~25% of DSP resources while achieving at least 3.2×/4.7× performance improvement.

VI. Conclusion

We propose a low-latency framework for ECC acceleration utilizing ASIP for operator scheduling and program protocol, complemented by FSM for ECC application implementation. Meanwhile, the proposed pipeline Barrett modular multiplier reduces latency through truncated multiplication, while our pseudo-dual-issue architecture enhances performance by at least 24%. Implemented on Xilinx Kintex UltraScale+ platforms, for the ECDSA algorithm, our design can sign 7610 messages/s and can verify 5530 messages/s, satisfying the requirement of V2V. Finally, compared with previous work, our design exhibits superior programmability.

References

[1] Z. Guan *et al.*, "Esc-ntt: An elastic, seamless and compact architecture for multi-parameter ntt acceleration," in *DATE*, 2024, pp. 1–6.

[2] L. Ding *et al.*, "Pima-lpn: Processing-in-memory acceleration for efficient lpn-based post-quantum cryptography," in *DAC*, 2023, pp. 1–6.

[3] Z. Lu *et al.*, "Leap: A lightweight encryption and authentication protocol for in-vehicle communications," in *ITSC*, 2019, pp. 1158–1164.

[4] Z. Lu *et al.*, "A survey on recent advances in vehicular network security, trust, and privacy," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 2, pp. 760–776, 2019.

[5] R. Tao *et al.*, "Security certificate management system for v2v communication in china," *SAE Int. J. Transp. Cybersecur. Privacy*, vol. 2, no. 11-02-02-0015, pp. 169–183, 2019.

[6] C. Hicks *et al.*, "A vehicular daa scheme for unlinkable ecdsa pseudonyms in v2x," in *Proc. - IEEE Eur. Symp. Secur. Priv., Euro S P*, 2020, pp. 460–473.

[7] H. Mosanaei-Boorani *et al.*, "A digital signature architecture suitable for v2v applications," *TCAS-I*, vol. 71, no. 2, pp. 731–739, 2024.

[8] A. Ghosal *et al.*, "Security issues and challenges in v2x: A survey," *Computer Networks*, vol. 169, p. 107093, 2020.

[9] I. Agudo *et al.*, "A blockchain approach for decentralized v2x (d-v2x)," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 4001–4010, 2021.

[10] M. A. Mehrabi *et al.*, "Efficient cryptographic hardware for safety message verification in internet of connected vehicles," *ACM Trans. Internet Technol.*, vol. 22, no. 4, nov 2022.

[11] S. Lee *et al.*, "Hybrid approach of parallel implementation on cpu–gpu for high-speed ecdsa verification," *The Journal of Supercomputing*, vol. 75, pp. 4329–4349, 2019.

[12] W. Pan *et al.*, "An efficient elliptic curve cryptography signature server with gpu acceleration," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 1, pp. 111–122, 2017.

[13] J. Ding *et al.*, "High-speed ecc processor over nist prime fields applied with toom–cook multiplication," *TCAS-I*, vol. 66, no. 3, pp. 1003–1016, 2019.

[14] M. Knežević *et al.*, "Low-latency ecdsa signature verification—a road toward safer traffic," *IEEE Trans. Very Large Scale Integr.*, vol. 24, no. 11, pp. 3257–3267, 2016.

[15] P. Choi *et al.*, "Ecc coprocessor over a nist prime field using fast partial montgomery reduction," *TCAS-I*, vol. 68, no. 3, pp. 1206–1216, 2021.

[16] D. Hankerson *et al.*, *Guide to Elliptic Curve Cryptography*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[17] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Lect. Notes Comput. Sci.* Springer, 1986, pp. 311–323.

[18] C. Costello *et al.*, "A brief discussion on selecting new elliptic curves," *Microsoft Research. Microsoft*, vol. 8, 2015.

[19] D. Johnson *et al.*, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, pp. 36–63, 2001.

[20] D. Liu *et al.*, "An efficient batch verification scheme for sm2 signatures," in *2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, 2021, pp. 208–212.

[21] B. Parhami, *Computer arithmetic*. Oxford university press New York, NY, 2010, vol. 20, no. 00.

[22] L. Baldanzi *et al.*, "Digital random number generator hardware accelerator ip-core for security applications," in *Applications in Electronics Pervading Industry, Environment and Society*, S. Saponara *et al.*, Eds. Cham: Springer International Publishing, 2020, pp. 117–123.

[23] M. A. Mehrabi *et al.*, "Elliptic curve cryptography point multiplication core for hardware security module," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1707–1718, 2020.

[24] S. Asif *et al.*, "A fully rns based ecc processor," *Integration*, vol. 61, pp. 138–149, 2018.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on December 12,2024 at 11:14:21 UTC from IEEE Xplore. Restrictions apply.