SYSTEMC-BASED HARDWARE/SOFTWARE CO-DESIGN OF ELLIPTIC CURVE CRYPTOGRAPHIC SYSTEM FOR NETWORK MUTUAL AUTHENTICATION

Y. W. Hau[1], Mohamed Khalil-Hani[2], Muhammad N. Marsono[3]
VLSI-eCAD Research Laboratory (VeCAD)
Department of Microelectronic & Computer Engineering,
Universiti Teknologi Malaysia,
81310 Skudai, Johor, Malaysia.
Tel: 07-5535223 Fax: 07-5566272
hauyuanwen@gmail.com[1], khalil@ fke.utm.my[2], nadzir@fke.utm.my[3]

ABSTRACT

This paper presents the hardware-software co-design of an elliptic curve cryptographic (ECC) system-on-chip (SoC) implementation of a mutual authentication protocol for network/data communication systems. Designing such computationally intensive cryptosystems, particularly for resource-constrained embedded applications using the conventional register transfer level (RTL) methodology leads to extended design cycles, inefficient design-space exploration, very long simulation cycles, tedious verification procedure, and sub-optimal final realization. The solution to this problem is to apply hardware-software co-simulation methods abstracted at the Electronic System Level (ESL). In the ESL modelling framework proposed in this paper, the Unified Modelling Language (UML) is used to create the design documents that describe the system static architecture and functional behaviour. SystemC is used in generating the cycle-accurate executable simulation models. A technique for design space exploration at the system level is also proposed to obtain the best hardware-software partitioning of the ECC SoC well before the final prototype is available. Experimental works with the proposed ESL co-design platform show that early system verification can be performed efficiently, and simulation speed of an ESL model is shown to be about 1000 times faster than the simulation of the equivalent RTL model. System execution time is estimated to be within 95% accuracy of its equivalent RTL.

Keywords: Electronic System Level, Elliptic Curve Cryptography, mutual authentication protocol, System-on-Chip, SystemC, Unified Modelling Language

1.0 INTRODUCTION

It is widely accepted that elliptic curve cryptography (ECC) [1] can provide better security strength with fewer bits of key length than other public key cryptographic algorithms. For example, a 160-bit elliptic curve cryptosystem has security strength comparable to a 1024-bit Rivest-Shamir-Adleman cryptosystem [2]. The shorter security key results in lower communication bandwidth, reduced memory storage requirements, reduced processing times as well as lower power consumption. These advantages are especially crucial in resource-constrained embedded system designs such as in handheld communication devices. With the state-of-the-art very large scale integrated circuit (VLSI) and field programmable gate array (FPGA) technologies, complex embedded systems or substantial parts of the systems can be integrated on a single microchip to perform most or all the functionalities of an application.

Previous works on FPGA-based implementation of ECC at the register transfer level (RTL) abstraction level are reported in [3–6]. References [7, 8] presented the HW/SW co-design of ECC coprocessor designs using GEZEL, a non-standardized system-level design language. Reference [9] reported the SystemC modelling of the ECC point multiplier without considering the HW/SW co-design at system-level abstraction. References [10, 11] developed functional, cycle-accurate SystemC model of a MIPS32 processor for their proposed elliptic curve computation algorithm.

An embedded system implemented in the single microchip is referred to as an embedded System-on-Chip (SoC) or SoC-based embedded system. Typically, an SoC comprises at least one processing element (e.g. microprocessor) that executes the embedded programs (software), and a number of hardware modules that typically accelerate time-consuming functions. It may also include other modules that perform communications with the outside world. The development of an SoC-based embedded system, such as an elliptic curve cryptographic system-on-chip (ECC-SoC), has become increasingly complex, as more severe demands are imposed. These demanding requirements include among others, lower cost, higher performance, lower power, better product quality, higher security, and

shorter time-to-market. What makes it even more challenging is that these requirements typically conflict with one other. One approach to handle these design challenges is to perform an optimal hardware/software (HW/SW) partitioning of the system functionality. The goal of this system partitioning is to achieve a design that exhibits high system performance, while meeting design constraints such as power, area cost and operating frequency. Hence, in the post-partitioned design, certain functionalities once executed in software are now implemented in hardware, and these hardware accelerators are integrated with higher-level software application programming interfaces (APIs). Clearly, SoC designers must develop both hardware and software (programs executed on the embedded processor) concurrently. This requires an efficient HW/SW co-design and design-space exploration environment.

The current established methodology for designing complex digital hardware design is the RTL design approach. When a priori definition of the partition is made, separate hardware and software specifications are created. Changes to the HW/SW partitioning necessitate extensive redesigns which usually result in sub-optimal designs. Hence, the RTL-centric SoC design becomes impractical as it is too labour-intensive and has very long simulation times [12]. It is desirable that the design of a complex embedded SoC, such as the ECC-SoC presented in this paper, begins at a level of abstraction higher than RTL. This higher design abstraction is referred to as the Electronic System Level (ESL), where cycle-accurate behavioural models are created. The ESL models enable early system verification, design-space exploration, and facilitate IP reuse at high abstraction level [13].

This paper proposes an ECC-SoC design based on the HW/SW co-design methodology at the ESL abstraction level. The Unified Modelling Language (UML) is used to create the design documents to describe the system static architecture and functional behaviour, while SystemC modelling creates a cycle-accurate executable model. We demonstrate the ESL modelling of the ECC-SoC over *GF(p)* that performs a mutual authentication protocol for network/data communication systems. It will be shown that the system simulation of the ESL model is about 1000 times faster than the RTL model.

The rest of the paper is organized as follows. Section 2 presents the background of ECC mutual authentication protocol and its associated algorithms. Section 3 presents the definitions and modelling framework proposed in the paper. Sections 4 and 5 present the system modelling and verification of the ECC-SoC, respectively. Section 6 explores potential design-space for HW/SW partitioning for the ESL model of the ECC-SoC and comparison with equivalent RTL model. Conclusion and future work are in Section 7.

2.0 THE ECC-BASED MUTUAL AUTHENTICATION PROTOCOL

The ECC-based mutual authentication protocol was proposed by Aydos *et al.* [14, 15] for network/data communication applications. The protocol, which is based on Elliptic Curve Digital Signature Algorithm (ECDSA) scheme [16], is also suitable for embedded systems in hand-held devices and smartcards. Fig. 1 illustrates the architecture of a network ECC-based mutual authentication system.

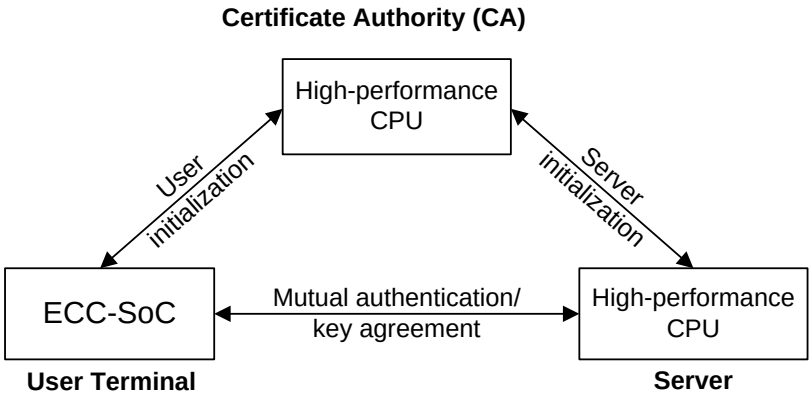
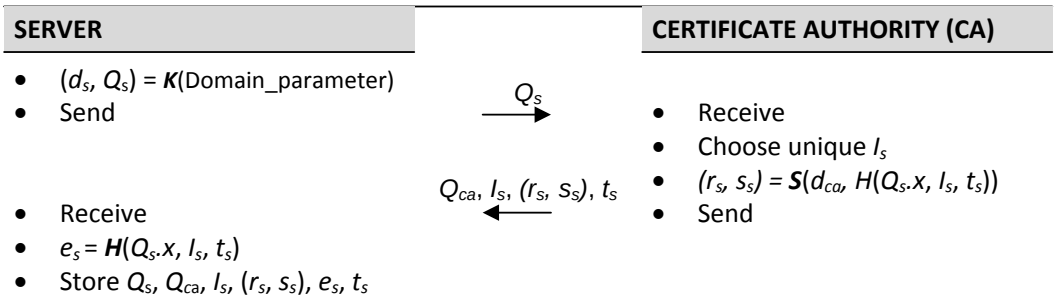
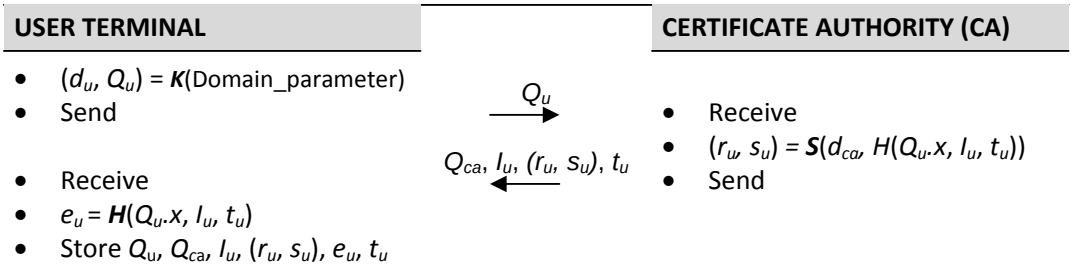


Fig. 1: Network Mutual Authentication System Architecture

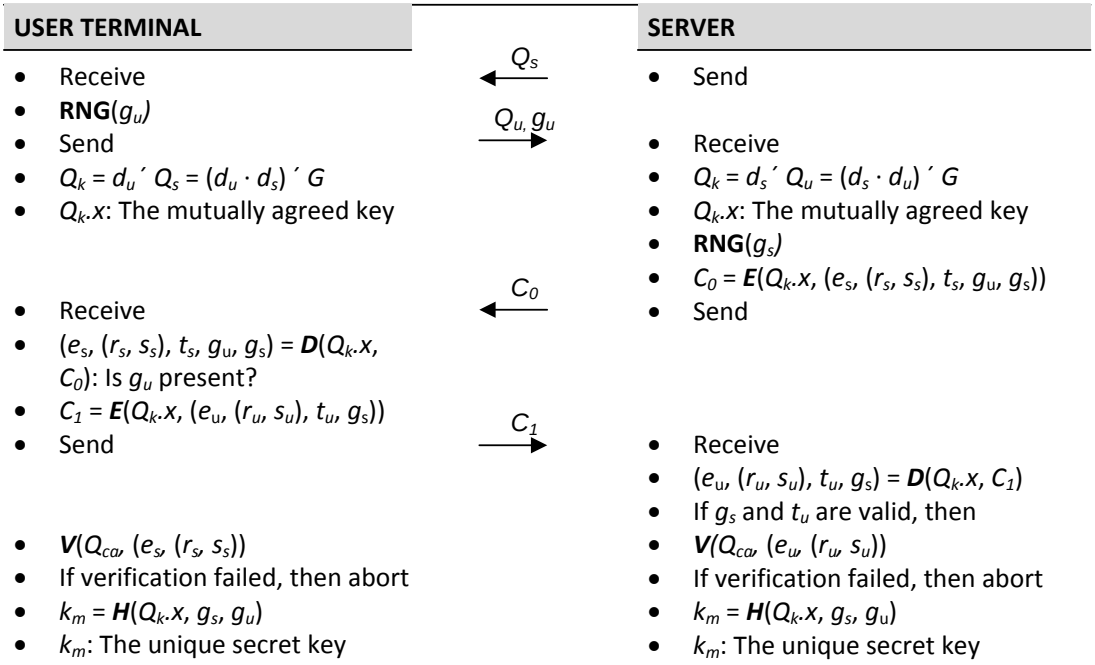
Aydos's mutual authentication protocol consists of two main phases. As shown in Fig. 2, the first phase of the protocol is initializations of the terminal and server (Fig. 2a and 2b), and the other is the terminal-server mutual authentication/key agreement phase (Fig. 2c). The initialization phases involve off-line certificate generations signed by a Certificate Authority (CA) through a secure and authenticated channel. The contents to be signed by the CA's



(a) Network Server Initialization



(b) User Terminal Initialization



(c) Authentication / Key Agreement

Legends:
K = ECC Key Deployment
E = Symmetric Encryption
H = Secure Message Hashing
D = Symmetric Decryption
S = ECDSA Signature Signing
V = ECDSA Signature Verification
RNG = Random Number Generation

d_{ca}, d_s, d_u: Private key of CA, server and user terminal
Q_{ca}, Q_s, Q_u: Public key of CA, server and user terminal
I_s, I_u: Temporary identity of server and user terminal
t_s, t_u: Certificate expiration date (time stamp) of server and user terminal
$(r_s, s_s), (r_u, s_u)$: Digital signature of server and user terminal
e_s, e_u: message digest of server's and user's information
g_s, g_u: random challenge generated by server and user terminal
C_0, C_1: Ciphertext generated by server and user terminal
Q_k: Mutually agreed key
k_m: Session key for data communication

Fig. 2: Mutual Authentication Protocol [14, 15]

private key are the public key (*Q*), the temporary identity (*I*), and the certificate expiration date (*t*) of the server or terminal. The CA generates a digital signature, which is a pair of integers denoted as a tuple (*r*, *s*). The mutual authentication/key agreement phase between terminal and server involves the signature verification of both parties, each signed by the CA during the initialization stage. This phase of the protocol should be completed before the data communication between them can take place. Seven cryptographic computations are applied in this mutual authentication protocol, and they include ECDSA key deployment (*K*), signature signing (*S*) and verification (*V*), message hashing (*H*), symmetric data encryption (*E*) and decryption (*D*), and random number generation (*RNG*). The derivation of the mutually agreed key Q_k in the mutual authentication and key agreement is based on the Elliptic Curve Diffie-Hellman (ECDH) scheme [16].

An ECC-based data security application typically consists of several cryptographic operations and multi-level elliptic curve arithmetic computations over finite field. The algorithm hierarchy of the ECC-based mutual authentication system is depicted in Fig. 3. The lowest level of the hierarchy is the finite field arithmetic, *GF(q)*. There are two types of finite fields, namely (1) prime finite field, *GF(q=p)*, and (2) binary finite field, $GF(q=2^m)$. Each has different underlying field arithmetic computation. Field arithmetic over prime field mainly involves big integer modular arithmetic, whereas field arithmetic over binary field is based on polynomial arithmetic.

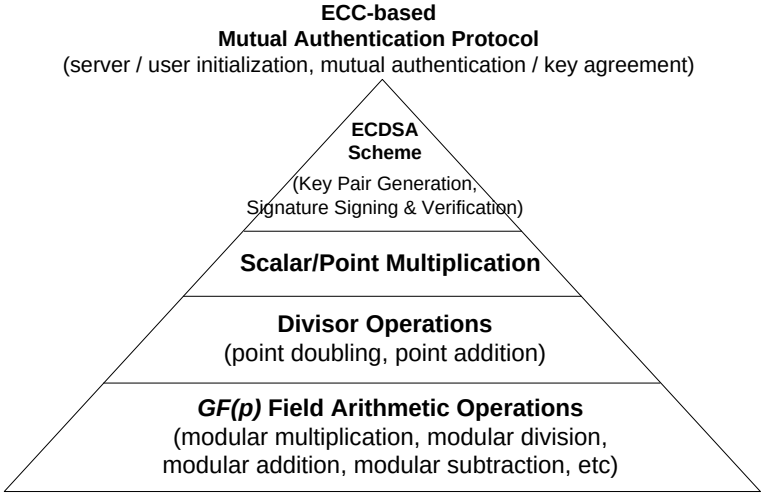


Fig. 3: Algorithm Hierarchy of ECC-based Mutual Authentication Protocol

The main security scheme that contributes to Aydos's mutual authentication is the ECDSA scheme which involves key deployment, message signing, and signature verification. These crypto schemes execute a number of compute-intensive functions which include random number generation, message hashing, elliptic curve arithmetic over *GF(p)*, and large integer modular arithmetic. The most crucial and time-consuming operations are the latter two functions. The elliptic curve arithmetic computation applies point multiplication and point addition, which are formed by big integer modular arithmetic functions. Modular arithmetic operations consist of modular division, modular multiplication, modular addition, and modular subtraction. In the mutual authentication/key agreement operation of the protocol, a secure hash algorithm is used to hash the information on server or terminal, creating a message digest. Encryption and decryption is based on symmetric-key cryptography for high performance. Block cipher encryption algorithm, such as AES [17], or stream cipher, such as the XOR-based encryption algorithm can be applied. The unique secret key k_m derived is used as the session key for symmetric encryption. The RNG is used to generate a random number during server-user challenge.

3.0 ESL MODELING FRAMEWORK AND DEFINITIONS

The ESL modelling framework proposed in this work consists of four levels of abstraction descriptions, namely (1) system specification description, (2) untimed functional system-level model (UTF), (3) cycle-accurate time-functional system-level model (CTF), and (4) the RTL abstraction. In this paper, due to space limitations, we demonstrate only the application of the system-level CTF modelling and its corresponding RTL design. A discussion of the complete design framework can be found in reference [28]. Fig. 4 shows the proposed co-design and co-verification framework involving the CTF and RTL abstraction levels. In this paper, a design model at CTF abstraction level is represented by (a) UML modelling document, and (b) an SystemC executable model. The

modelling document is a set of graphical diagrams that represent the structure and behaviour of the system. The executable model is an extended computer program that exhibits the same execution behaviour as the targeted system. The UML document serves as the design documentation to the SystemC executable model. Other design capture models may be required to generate these UML documents, and these include control flow graphs (CFG) and data flow graphs (DFG).

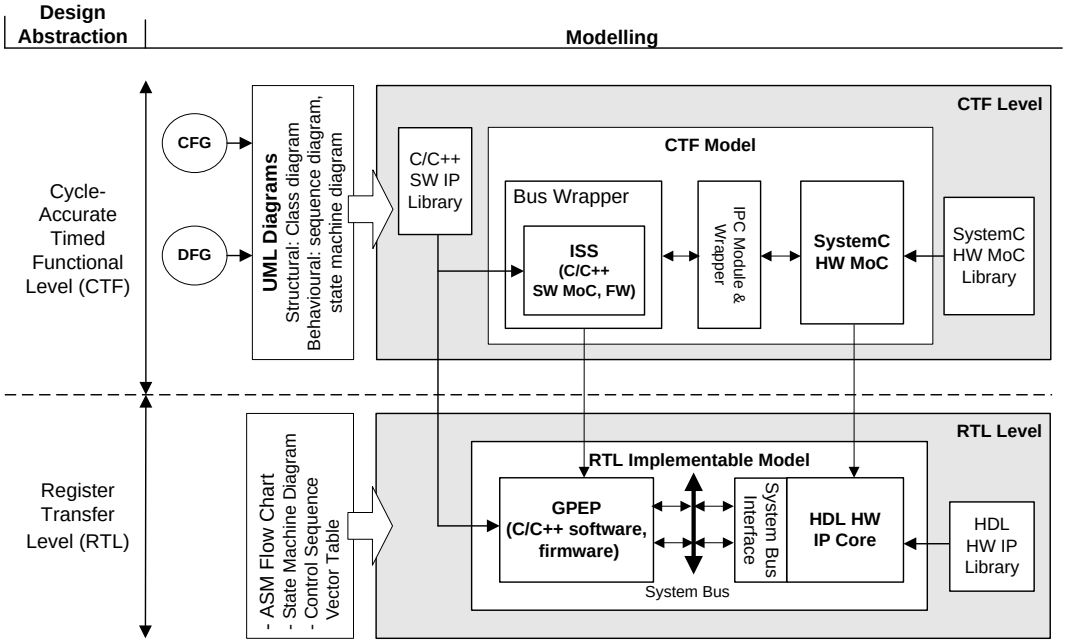


Fig. 4: Co-Design Frameworks: CTF vs. RTL

Our ESL modelling framework facilitates system partitioning and design-space exploration of system architectures at the CTF level. The following terminologies, used extensively in this paper, are defined. The term, ***Model-of-Computation, MoC*** is defined as a modelling entity representing an element that executes a specific operation or computation functionality. A MoC can be a function or a processing element (PE), which may be implemented in hardware or software. A ***hardware MoC (HW MoC)*** is a behavioural cycle-accurate model, which represents a piece of logic circuit as a hardware accelerator in the final deployment model. It is used to offload the time-consuming operations from a software-running general-purpose embedded processor (GPEP). A ***software MoC (SW MoC)*** is a piece of programming, which is described in high-level programming language. It is run on GPEP to execute certain operations and application functionalities. A ***firmware (FW)*** is a special type of software programming, which is described in a high level programming language and is executed on a GPEP. FW usually acts as the device driver or API of the HW MoC. It instructs or controls the hardware accelerators, coprocessors, or cores to perform specific computations.

The system partitioning results in the system functionality being divided into HW and SW MoCs, taking into account the resulting cost-performance design trade-offs and design constraints. It is clear that an MoC of a CTF system model has three possible architectures, namely, (a) All-hardware, (b) All-software, and (c) Mixed hardware-software. The MoCs in our CTF model are all cycle-accurate. The HW MoCs are pin- and cycle-accurate models, which are modelled in SystemC and simulated using a SystemC kernel. The SW MoCs and FW are written in C/C++, and simulated on a cycle-accurate instruction set simulator (ISS) [18]. The RTL implementable model is obtained from the CTF model via the conventional approach. Algorithmic State Machine (ASM) flow chart, state machine diagram, and control sequence vector table are used as a design documentation to derive the RTL code. It should be noted here that the modelling framework proposed in this paper only considers a single-master, multi-slave SoC architecture. This means that there is only one master processor which acts to initiate an operation and execute the SW MoCs. All the HW MoCs are configured as slave devices that receive instructions from the master to commence a computation. The computations are executed sequentially as there is only one embedded processor. Consequently, bus arbitration is not needed as would be required in multi-processor SoC architecture.

4.0 ESL MODELLING OF THE ECC-BASED MUTUAL AUTHENTICATION NETWORK SYSTEM

A real complex application or system design typically requires decomposition into multiple MoCs. Each MoC itself may be further decomposed to multi-level sub-modules, which is modelled as an *SC_MODULE* object. The ECC-based mutual authentication protocol is implemented on a network/data communication application powered by multiple ECC-SoCs. From the UML class diagram given in Fig. 5, the application architecture consists of three entities, CA, user terminal, and server. The user terminal is implemented as an SoC embedded system, whereas the CA and server are ECC-based system software targeted for implementations on high-performance computers. The design specifications of ECC-SoC implementing the test application described above are given in Table 1.

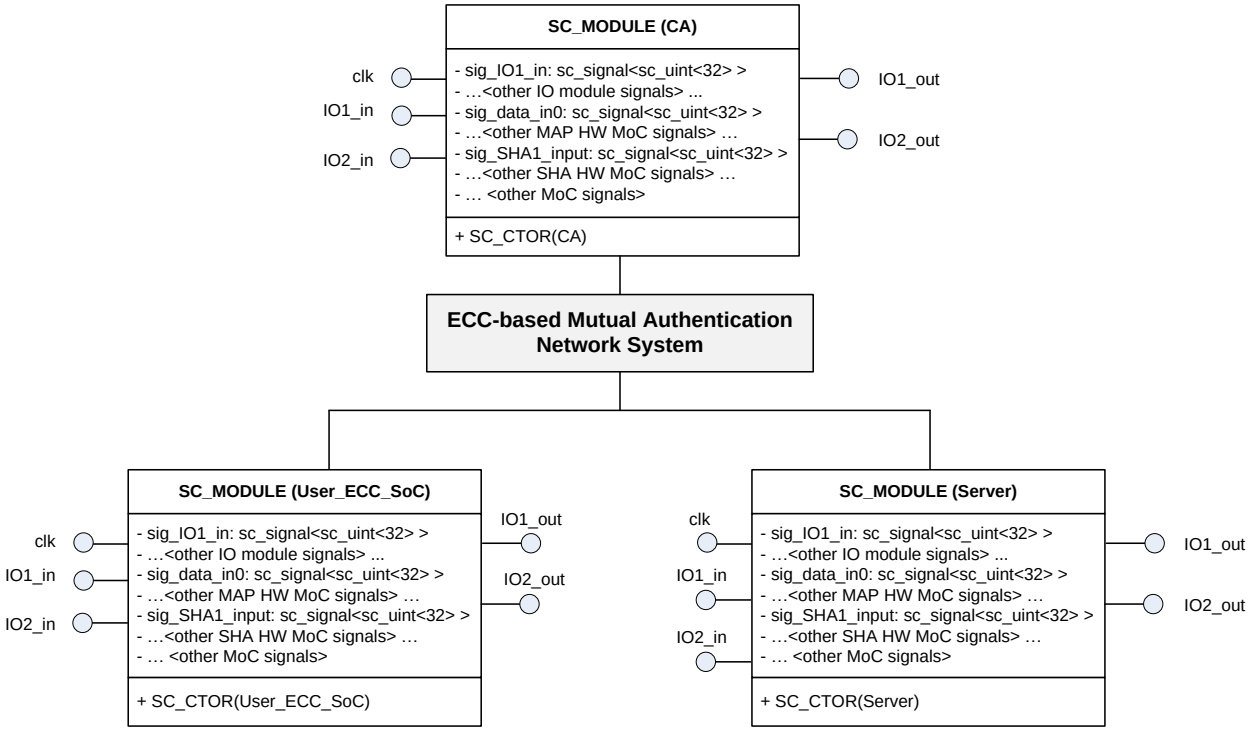


Fig. 5: UML Class diagram of ECC-based Mutual Authentication System

Table 1: Design Specifications of ECC-SoC

Feature	Description
Application	ECC-based Mutual Authentication Protocol [14]
Finite Field	Prime finite field, *GF(p)*
Elliptic Curve (EC)	Koblitz curve
EC Point Representation	Affine coordinate system
EC Field Size	160-bit
EC Domain parameter	*secp160r1* according to Certicom [19]
EC Point Multiplication	LSB-first algorithm [1]
Symmetric Encryption	XOR-based stream cipher
Message Hashing	Secure Hashing Algorithm [20]
Random Number Generation	Pseudo RNG [2]

From the UML class diagram given in Fig. 5, a SystemC executable file, *main.cpp* is obtained as the top-level implementation file. This SystemC implementation file instantiates the CA, user terminal, and server to form the complete mutual authentication network system. It also declares the first-in-first-out (FIFO) channels *(sc_fifo)* for data communication among these modules.

Since the functionality of the data communication application has been decomposed to CA, user terminal, and server, respectively, a UML sequence diagram is required to describe the data flow of the system application. Fig. 6 shows the UML sequence diagram example of CTF model of ECC-SoC for the user terminal initialization process shown in Fig. 2(a). Each system in the data communication application is represented as a sequence element. The system application is carried out by a series of function calls executed sequentially in each ISS of CA, user terminal, and server.

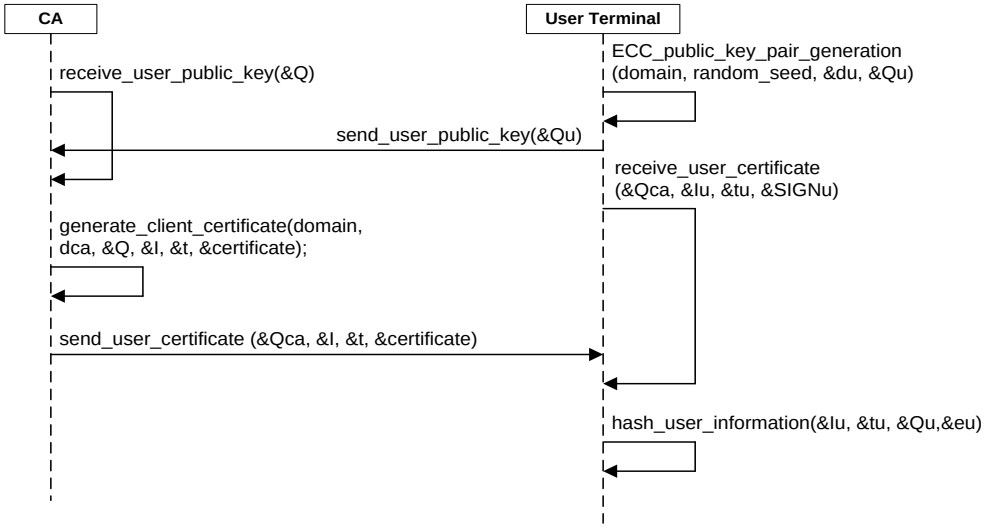


Fig. 6: UML sequence diagram of User Terminal Initialization between CA and User

4.1 ECC-SoC Top-Level Module

We now discuss the SystemC design of the CTF executable model of the ECC-SoC. Analysis of the algorithm hierarchy of ECC-based system suggests a cost-effective system partitioning as shown in Fig. 7. At the lowest level of the pyramid, the prime finite field modular arithmetic operations are implemented in a Modular Arithmetic Processor (MAP) modelled as mixed hardware/software architecture. The HW MoC is modelled as a cycle-accurate behavioural model, which is based on add-and-shift algorithms [22, 23]. The SW MoC implements multi-precision prime field arithmetic algorithm with a C source code from [21]. The layers above the prime field arithmetic, from EC divisor operations up to point multiplications, are all realized as SW MoCs.

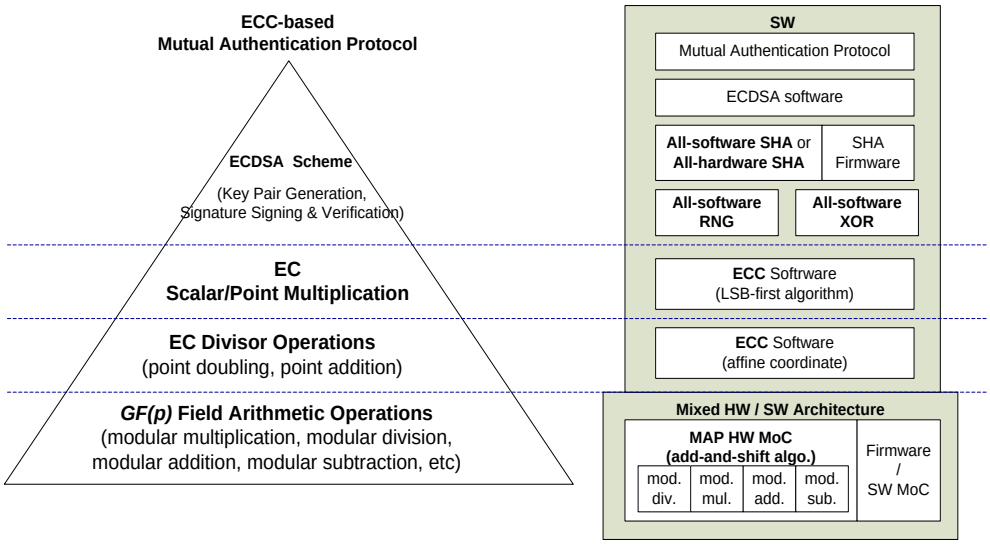


Fig. 7: System Partitioning of ECC-SoC at CTF Abstraction Level

The SHA MoC could be modelled either as an all-hardware SHA MoC or an all-software SHA MoC. However, the RNG and the XOR encryption stream cipher modules are modelled as all-software MoCs. This is because the RNG computation is performed infrequently and the bit-wise XOR instruction is available in GPEP instruction set architecture. Therefore, they can be efficiently implemented in software. From the application programmer's point of view, the function call to execute a specific computation is similar, either as software subroutine or firmware device driver. At this level, several special modules are inserted and refined. For example, a timer module is added to collect the number of computation cycles. In addition, two abstract I/Os are added to represent the UART or USB I/O, which allow an ECC-SoC to transfer data to off-chip systems. The ISS models the GPEP, which acts as the main controller of the entire system that controls the execution of the IP cores through the device drivers (FW) and executes the application program.

The UML class diagram that depicts the top-level system architecture of the ECC-SoC is given in Fig. 8, showing the MoCs and its associated functions. In Fig. 8, the private methods such as *MAP_write()* and *MAP_read()* in the ECC-SoC are the firmware device drivers that send/receive data to/from the MAP HW MoC. These private methods are transparent to the application programmer. From the application programmer's point of view, they only can access the public functions sequentially to execute the operations.

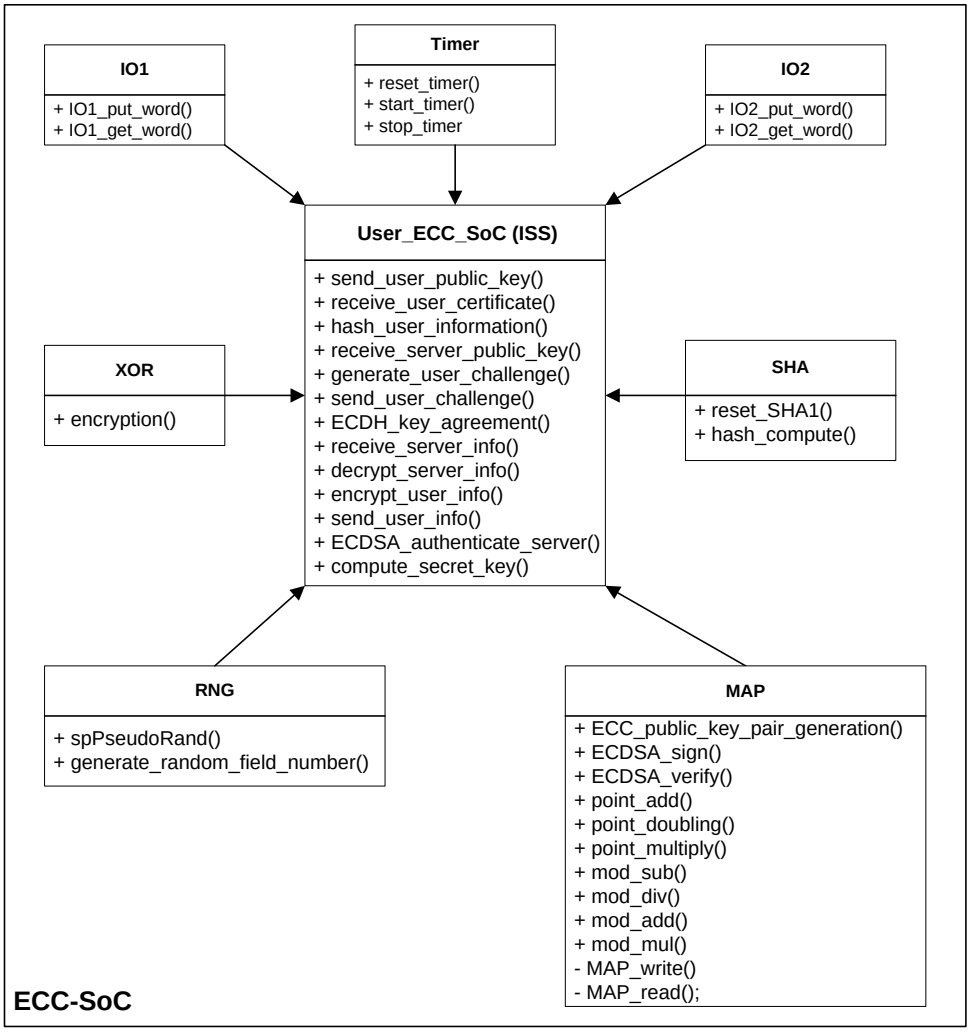


Fig. 8: UML Class Diagram of ECC-SoC CTF Model

The functionality of an ECC-SoC is partitioned into HW and SW MoCs. As a result, the UML sequence diagram is also required to model the data exchanges among the MoCs via function call by the ISS within a ECC-SoC. Fig. 9 shows the simplified sequence diagram example of the *ECC_public_key_pair_generation()* of *User_ECC_SoC* as shown in Fig. 6. It is based on ECDSA key pair generation operation, which consists of multiple lower-level functions. In this sequence diagram, only the ISS and HW MoCs are represented as sequence elements. The function

calls can either be a SW MoC simulated by GPEP, or as a FW that sends instructions to a HW MoC. When a higher level HW MoC is trying to pass a data to its sub-module, the data is passed as local hardware signals.

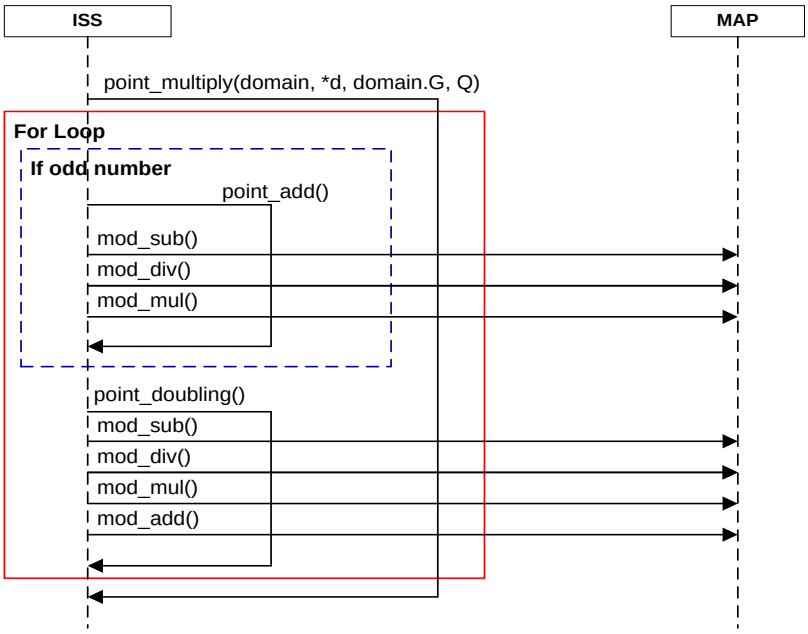


Fig. 9: UML sequence diagram of *ECC_public_key_pair_generation()*

4.2 ECC-SoC Hardware MoC Modelling

The simplified UML class diagram of MAP HW MoC is given in Fig. 10. It shows that the MAP consists of *bus_interface* module and *MAP_core*. The *bus_interface* is an interface module to exchange data between ISS and *MAP_core*, which is the main processing module. It consists of four sub-modules, which are *mod_div*, *mod_mul*, *mod_sub*, and *mod_add*, each performing the modular division, multiplication, subtraction, and addition, respectively. The top-level HW MoCs communicate with their sub-level HW MoCs via internal hardware signals, which are declared as *sc_signal* properties. All of the computations in HW MoCs are described in SystemC *SC_THREAD* or *SC_METHOD* processes. All of these structural information of a HW MoC, such as I/O port and interconnect signal declaration are provided in a header file *(.h)*. The behaviour of a HW MoC such as sub-modules instantiation and computation processes execution are defined in an implementation file *(.cpp)*.

The design flow of HW MoC behaviour modelling, from algorithm to SystemC model is shown in Fig. 11. The associated algorithm of every HW MoC, including its sub-module, is first mapped to a corresponding CFG and DFG. From the CFG/DFG combination, the UML state machine diagram is derived and hence generate the SystemC *.cpp* implementation file.

To further describe the design flow, we use *mod_mul* as an illustration example, where its algorithm is shown in Listing 1. Its associated CFG and DFG are shown in Fig. 12.

Input : $y, x \in [1, p\text{-}1]$ and p
Output : u, where $u = (y*x) \bmod p$

1. $U = 0, V = x, A = y, P = M$;
2. **while** $A \neq 0$ do
3. **if** $A_0 = 1$ then $U = U + V$;
4. **if** $U \geq P$ then $U = U - P$;
5. $A = A / 2$;
6. $V = 2V$;
7. **if** $V \geq P$, then $V = V - P$;
8. Return U.

Listing 1: Interleave Modular Multiplication Algorithm

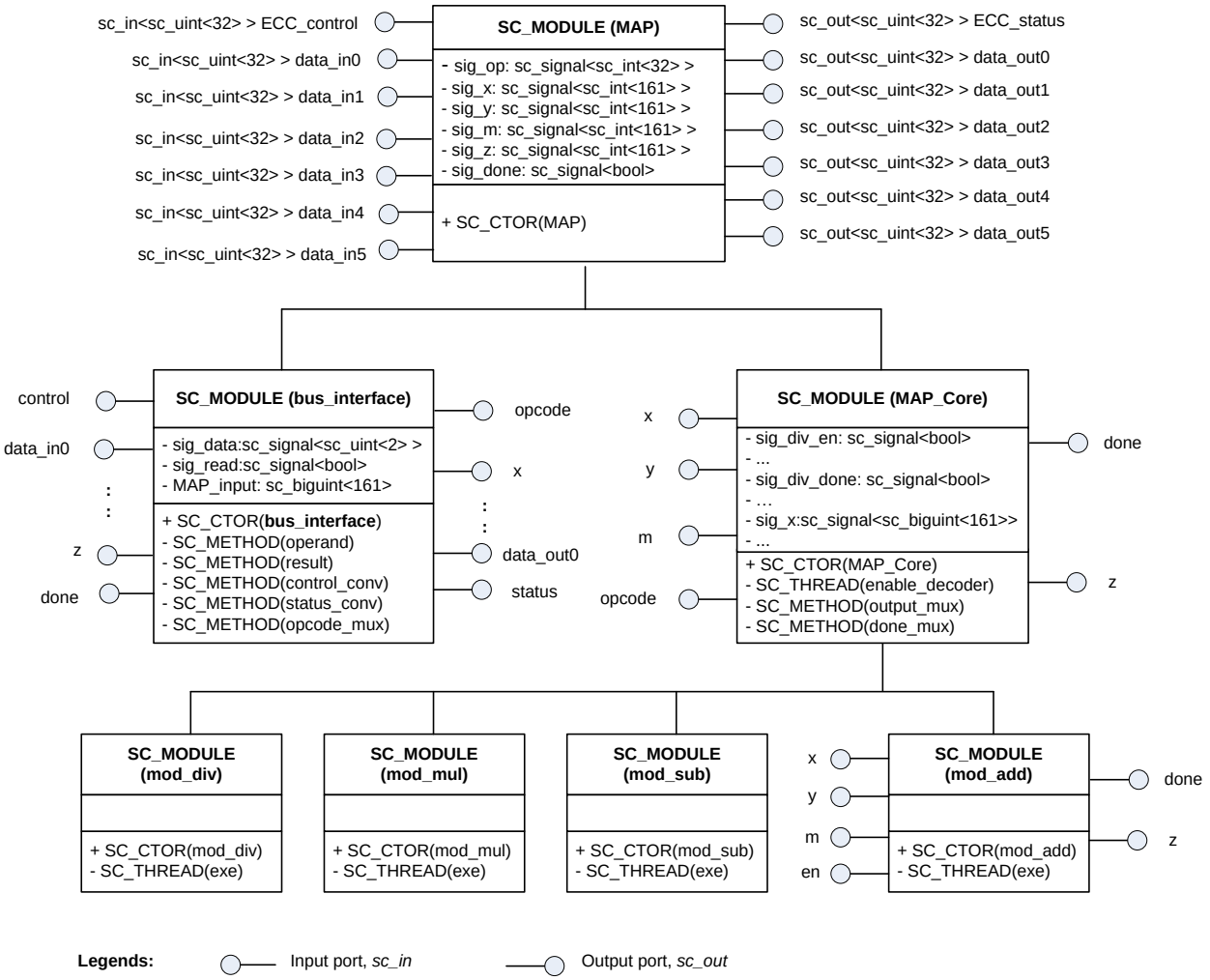


Fig. 10: UML class diagram of MAP HW MoC

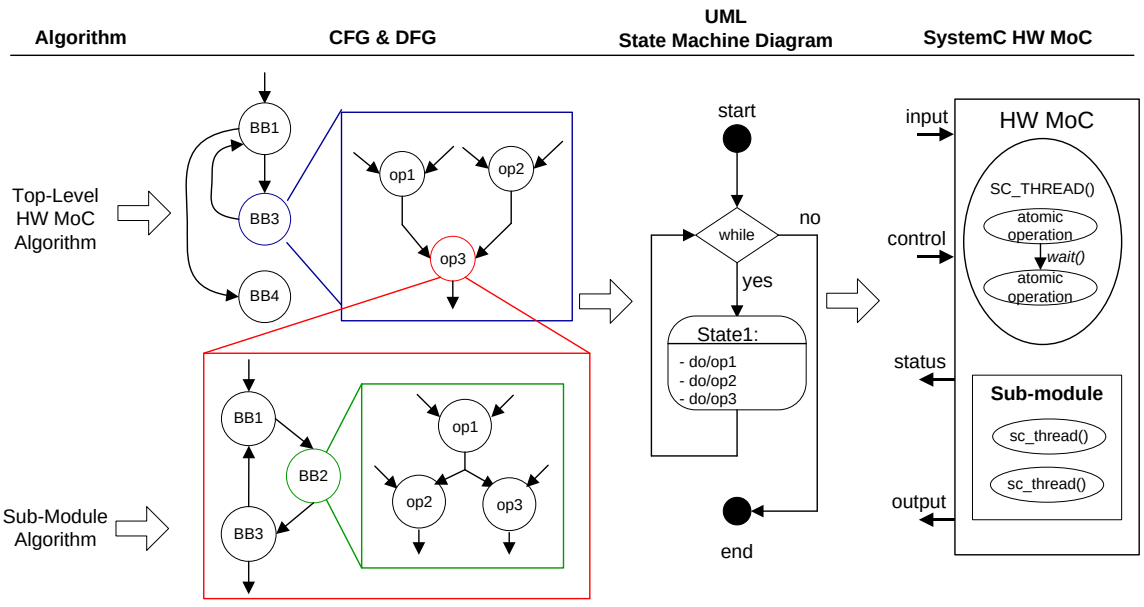
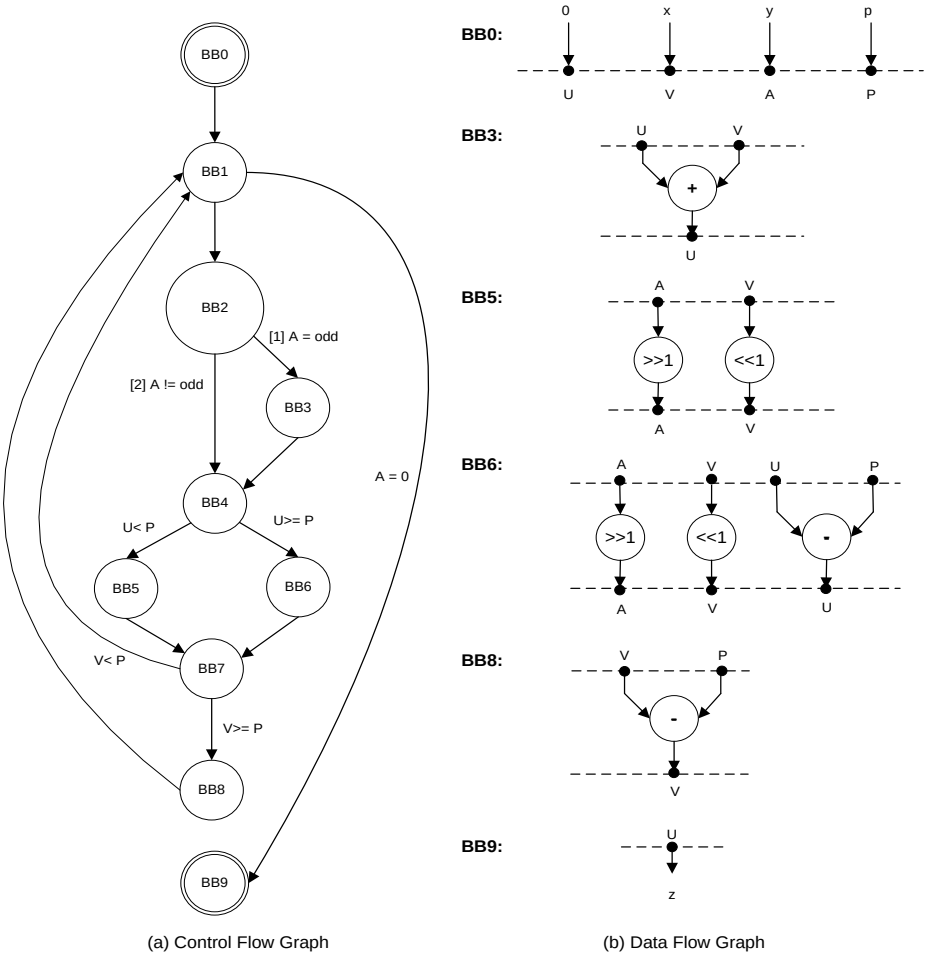


Fig. 11: Hardware MoC Modelling Flow at CTF



(a) Control Flow Graph (b) Data Flow Graph

Fig. 12: *mod_mul*: CFG and DFG

The CFG of the *mod_mul* algorithm consists of ten basic blocks (BBs). BB0 and BB9 are root and exit vertices, respectively. BB1 represents the *while* loop, whereas BB2, BB4, and BB7 collectively represent the *if* loop. For example, in BB2, if *A* is odd number, the path to BB3 is taken, or BB4 otherwise. Hence, each BB produces a corresponding DFG. The DFGs provide the atomic operations that can be executed concurrently in one state, and also show the data dependencies between the operators. For instance, in BB0, the values of 0, *x*, *y*, and *p* are stored into register *U*, *V*, *A* and *P*, respectively. In BB2, BB4, and BB7, they require one comparator to compare values of registers for condition checking to determine which next path should be taken. In RTL implementable model, the comparator is described as a combinational logic. It generates status signals to the control unit to determine the next state in final RTL model. However, in SystemC behavioural model, the status signals are implied and not explicitly described. BB3, BB5, BB6, and BB8 check for data dependency, compute the desired operations, and store the result in dedicated registers. In BB9, the data from register *U* is fetched out to output value, *z*.

Based on CFG and DFG of *mod_mul* algorithm, the UML state machine diagram of *mod_mul* is obtained as shown in Fig. 13(a). The corresponding SystemC code, i.e. *SC_THREAD(exe)* of *mod_mul* sub-module is derived, as shown in Fig. 13(b). Based on the UML state machine diagram, concurrent atomic operations are grouped together and separated from other atomic operations using *wait()* statement to define different states.

4.3 ECC-SoC Firmware Modelling

FW is created in association with a HW MoC to allow the system master controller to control the operations of the MoCs. In the FW code, each I/O port is mapped to a unique address (the address written in SystemC-instantiated MoC). As an example, Listing 2 shows C/C++ code of *mod_mul()*, which is the FW associated with *mod_mul* HW MoC sub-module in MAP.

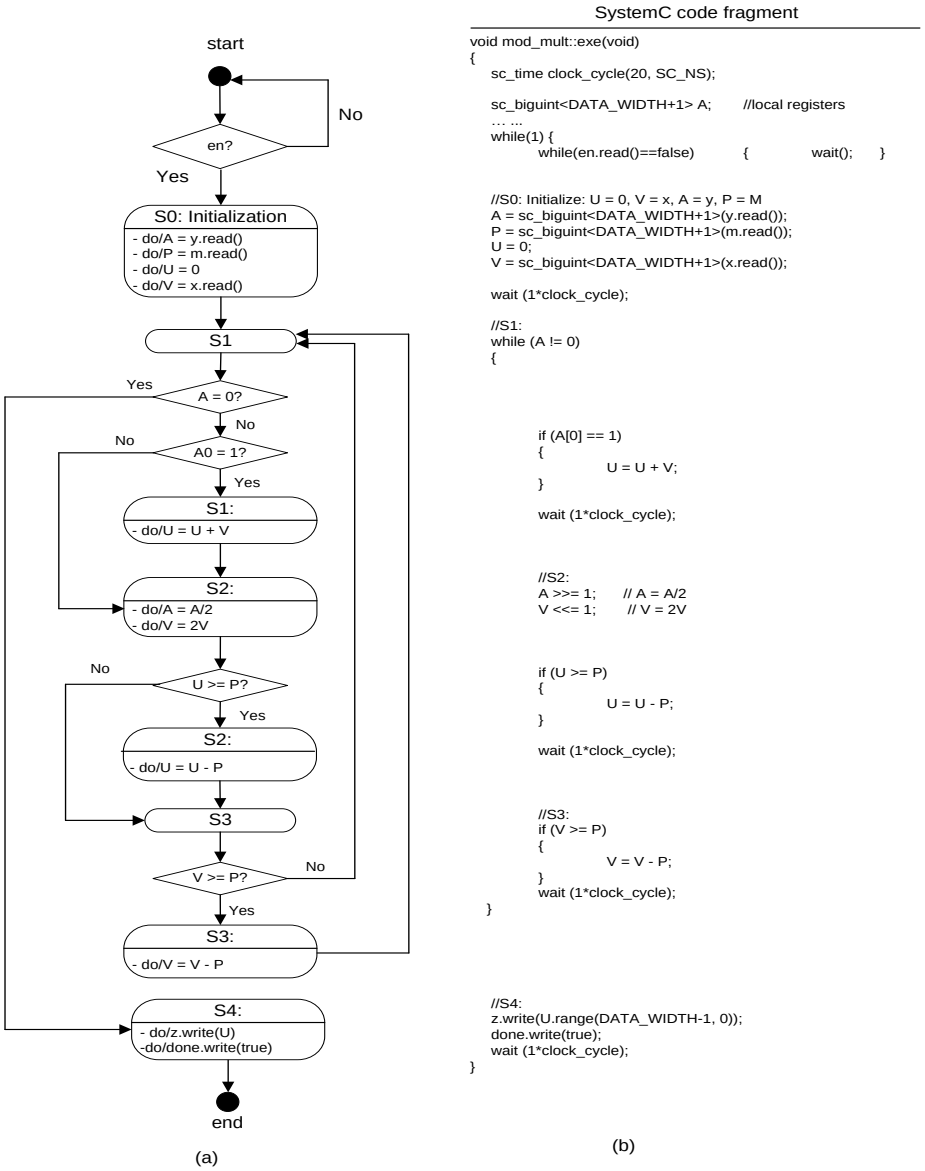


Fig. 13: *mod_mul*: (a)UML state machine diagram (b) SystemC code fragment

```
#include "MAP. h"
void mod_mul (FIELDP input_y , FIELDP input_x , FIELDP input_m , FIELDP* output_z )
{       //Send the input to MAP HW MoC
        map_write (operand_x, input_x);
        map_write (operand_y , input_y) ;
        map_write ( operand_m, input_m) ;

        // set opcode=1 to start compute z = ( y/x ) mod m
        *MAP_control = map_control_mod_mul_mask ;

        //Wait until the MAP done == 1
        while ( ( *MAP_status & map_status_done_mask ) != map_status_done_mask ) ;

        //Read the output from MAP HW MoC
        *MAP_control = map_control_mod_mul_mask + map_control_read_mask;
        map_read ( output z ) ;
        *MAP control = map clear mask ;
}
```

Listing 2: *mod_mul()*: FW

4.4 ECC-SoC Software MoC Modelling

The SW MoC is created using conventional approach of procedural programming in C/C++ language. Listing 3 shows the C/C++ code of the *point_doubling* SW MoC example in ECC-SoC. Note that in the *point_doubling* function body, the *mod_mul*, which is the FW driver of HW MoC is called as a normal software function.

```
void point_doubling (PRIME_DOMAIN_PARAMETER domain , POINT Q, POINT* R)
{
        int i = 0 ;
        FIELDP two = {0 x00000000 , . . . , 0x00000002 } ;
        FIELDP three = {0 x00000000 , . . . , 0x00000003 } ;
        FIELDP s={0} , temp1={0} , temp2 = {0} , temp3 = {0} , temp4 = {0};

        // s = (3Qxˆ2 + a ) / (2Qy) mod p
        mod_mul (Q.x , Q.x , domain.p , &temp1 ) ;
        mod_mul (three , temp1 , domain.p , &temp2 ) ;
        mod_add (temp2 , domain.a , domain.p , &temp3 ) ;
        mod_mul (two, Q.y, domain.p, &temp4);
        mod_div (temp3, temp4, domain.p, &s);

        //Rx = s ˆ2 − 2Qx mod p
        mod_mul (s, s, domain.p, &temp1);
        mod_mul (two, Q.x, domain.p, &temp2);
        mod_sub (temp1, temp2, domain.p, &R−>x);

        //Ry = −Qy + s (Qx − Rx) mod p
        mod_sub (Q. x, R−>x, domain.p, &temp1);
        mod_mul (s, temp1, domain.p, &temp2);
        mod_sub (temp2, Q. y, domain.p, &R−>y);
}
```

Listing 3: *point_doubling* SW MoC

5.0 SYSTEM VERIFICATION OF ECC-SoC

This section presents the early system verification of the ECC-SoC using the bottom-up approach as illustrated in Fig. 14. The functional verification is divided into three levels, which are the IP core, system, and application levels. The functionalities of each HW/SW MoC of the ECC-SoC (IP core level) are first verified. It is followed by ECDSA security scheme verification (system-level), and then the mutual authentication protocol (application level). At the application level, multiple elliptic curve cryptosystems (ECCs) are possible.

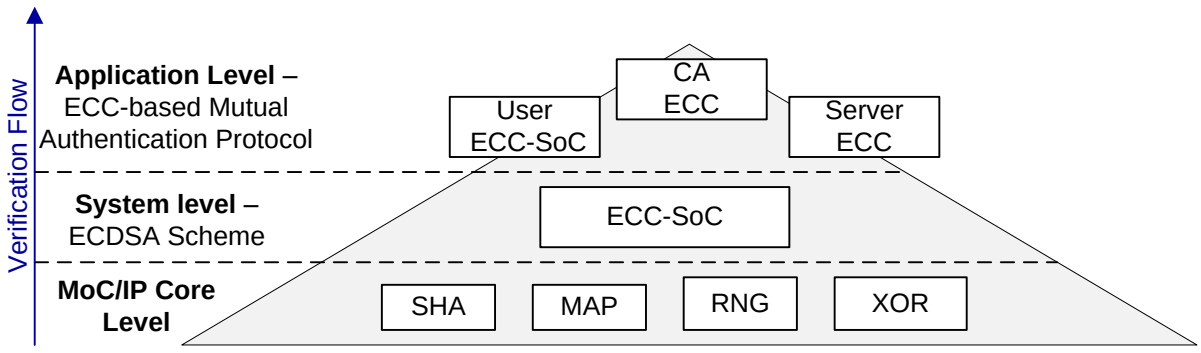


Fig. 14: Early Functionality Verification Flow

5.1 Verification of the MoCs at the IP Core Level

The verification of the MAP MoC starts with the prime field arithmetic operations. The verification is based on the result comparison between the HW partition of MAP MoC (HW MAP) with the equivalent SW partition (SW MAP)

obtained from [21]. Although the SW MAP may be not the optimum one, it is used as the reference model in the MAP MoC functionality verification. The test vectors are randomly generated by RNG and compared with the SW MAP computation result. If the computation outputs match, both of the HW and SW partitions of MAP MoCs functionalities are considered verified. For the elliptic curve arithmetic computations at higher system function level, mixed HW/SW MAP MoC architectures are explored. All of the prime field arithmetic computations are executed by HW partitions, while the arithmetic computations above the finite field arithmetic level are computed by SW partitions. For verification, portions of the *secp160r1* benchmark test vectors from [24] are used. For the RNG MoC, since the random number is pseudo generated and the source code is taken from [21], the authors assume it has been functionally verified. For SHA MoC verification, we use the benchmark test vector provided by [20] to compare the computation simulated output. The XOR stream cipher encryption block is verified using the round-trip-test method, where a same secret key is used to encrypt or decrypt a data. The decryption of a ciphertext using the same secret key should reveal the plaintext.

5.2 Verification of the ECDSA scheme at the System Level

We use two methods to verify the ECDSA scheme. The first verification is done by using the benchmark test vector provided by [24]. For this, the RNG MoC is disabled and all of the input data are fixed by hard-coding the value in the system application software according to [24]. This is to allow the authors to compare the final simulation output with the Certicom benchmark value [24], as well as the intermediate computation result. The second verification method is the round-trip-test verification, as performed by the XOR stream cipher encryption module. If a digital signature is signed and verified by using the public and private key of the same entity, the signature verification should return a valid signature verification result. In this verification method, all of the crypto MoCs are enabled, except the XOR stream cipher encryption module, which is not required in ECDSA scheme. The ECC public key pair is randomly generated to sign and verify the signature of a same message.

5.3 Verification of the ECC-based Mutual Authentication Protocol at the Application Level

In the functional verification of the highest application level, it consists of three ECCs, which are CA, server, and user terminals, running in parallel to perform the certificate signing (server/user initialization) and verification (mutual authentication/key agreement). In user ECC-SoC, all of the MoCs are enabled, including timer and abstract I/Os, to allow all three ECCs communicate with each other. For the verification purposes, the application software running at user terminal and server will ask the permission to corrupt the certificate signed by the CA, respectively. This is to model an intruder attack of the communication application, particularly in the wireless data communication.

6.0 EXPERIMENTAL WORK AND RESULTS

The system partitioning has been conducted to estimate the system performance metrics of alternative ECC-SoC system architectures, based on which the most effective system architectural partitioning is determined. The exploration is only applied to the MAP MoC. It is because the MAP MoC is the main component that directly contributes to the system performance of the ECC-based mutual authentication protocol.

Table 2 shows the alternative architectures based on different HW/SW partitioning of a MAP MoC. For example, in architecture Arch. 7 of the MAP MoC, the Mod-Add, and Mod-Sub functions are implemented in SW MoCs running on the ISS, while Mod-Mult and Mod-Div are implemented as HW MoCs. The rest of this section discusses the performances for each partitioning option measured in terms of number of clock cycles, communication overhead, and performance gain. Note that the ECC-SoC domain parameters are based on *secp160r1* from [19] and the ECDSA scheme test vector is taken from [24].

Table 2: HW/SW partitioning of the MAP MoC Architecture

Architecture	MAP MoC			
	Mod. Add	Mod. Sub.	Mod. Mult.	Mod. Div.
Arch. 1	All-Software Architecture			
Arch. 2	SW	SW	SW	HW
Arch. 3	SW	SW	HW	SW
Arch. 4	SW	HW	SW	SW
Arch. 5	HW	SW	SW	SW
Arch. 6	HW	HW	SW	SW
Arch. 7	SW	SW	HW	HW
Arch. 8	All-Hardware Architecture			

6.1 Total Execution Time

The system timing performance is measured in the unit of clock cycles based on the dynamic simulation approach. Hence, the CTF simulation model must include a 32-bit timer to measure the number of MoC computation cycles. In addition, the total times of data access to each HW MoC are collected to estimate the number of communication cycles. The total execution time of an SoC model, *D*, in number of clock cycles, *T*, at higher level abstraction, is derived as $T(D) = \sigma + \mu$, where σ is the number of MoC computation cycles, and μ is the number of communication cycles for data exchanged between HW MoCs with the ISS. The number of MoC computation cycles, σ, is calculated as $\sigma = \alpha_{SW} + \alpha_{HW}$, where α_{SW} is the number of SW MoC computation cycles, and α_{HW} is the number of SystemC HW MoC computation cycles.

From the total times of data access to HW MoCs, a formula for the number of communication cycle estimation of data exchange between HW MoC and the ISS can be derived. The number of communication cycles, μ, of a complete SoC is given by the equation:

$$\mu = \sum_{j=1}^{n} (\Delta I_{cj} + \Delta I_{sj} + \Delta I_{ij} + \Delta I_{oj}) * c \qquad (1)$$

where
Δ = total access count
c = average clock cycles of each data transmission
I_{cj} = Control register of HW MoC$_j$
I_{sj} = Status register of HW MoC$_j$
I_{ij} = Input register(s) of HW MoC$_j$
I_{oj} = Output register of HW MoC$_j$

After the total number of MoC computation and communication cycles are calculated, the communication overhead of different HW/SW partitioning is calculated using the equation $\gamma = \frac{\mu}{T(D)}$. This information helps the SoC designer to study the performance bottleneck of an SoC architecture.

Table 3 shows the number of clock cycles in computation (σ), communication (μ) and total execution time ($T(D)$), together with the communication overhead (γ) and performance gain (w_t). The all-software MAP MoC (Arch. 1) shows the highest number of total execution cycles. It is because the complex ECDSA operations of all-software MAP MoC are completely executed by the GPEP (modelled by ISS). The all-hardware MAP MoC (Arch. 8) takes least number of total execution cycles, due to all of the prime field arithmetic operations are completely offloaded and accelerated to the hardware partition architecture. For other HW/SW partitioning options, there are still differences in the actual number of clock cycles to compute the ECDSA signing operation.

Table 3: System Timing Performance of different architecture sets

Arch.	Computation Cycle Count, σ	Communication Cycle Count, μ	Total Cycle Count, *T(D)*	Communication Overhead, γ (%)	Performance Gain, w_t
Arch. 1	1,522,294,844	0	1,522,294,844	0.00	1.00
Arch. 2	338,098,057	342,874	338,440,931	0.10	4.50
Arch. 3	1,197,024,523	1,233,252	1,198,257,775	0.10	1.27
Arch. 4	1,513,411,597	554,838	1,513,966,435	0.04	1.01
Arch. 5	1,520,046,892	93,651	1,520,140,543	0.01	1.00
Arch. 6	1,511,162,521	648,489	1,511,811,010	0.04	1.01
Arch. 7	12,826,380	1,576,354	14,402,734	10.94	105.69
Arch. 8	1,688,626	2,224,843	3,913,469	56.85	388.99

6.2 Performance Gain

A system performance gain is important to an SoC designer to estimate how much an SoC's performance speedup can be attained for different system architectures during the design-space exploration. The performance gain of a modelled SoC, w_t, is based on Equation 2. The all-software architecture of the SoC CTF model is always used as a reference model compared with the other system architectures, which is based on various mixed HW/SW architecture.

$$W_t = \frac{T(D) \text{ of all-software architecture}}{T(D) \text{ of mixed HW/SW architecture}} \qquad (2)$$

The performance gain is the reference when exploring the other metrics, such as communication overhead and area cost. Table 3 suggests that the most cost effective HW/SW partitioning is Arch 7 in which Mod-Add and Mod-Sub is in SW while Mod-Div and Mod-Mult are computed in HW. The hardware acceleration of these operations could boost the system performance up to about 100 times faster than the all-SW architecture. Intuitively, this is as expected, as modular multiplication and division are computation-intensive and hence, should be offloaded as HW accelerators to enhance the overall system performance. Moreover, if all of the field arithmetic operations are accelerated in hardware, the system performance increases about 389 times faster. In contrast, if the timing performance is not a critical issue, accelerating the modular division alone (Arch. 2) could achieve a reasonable improvement, which is about 4 times faster compared to all-SW architecture (Arch.1). The HW acceleration of the other field arithmetic operation alone would not give any significant improvement to the ECC-SoC system performance.

6.3 Communication Overhead vs. Performance Gain

In any SoC design, it is important to analyze the computation and communication overheads, which greatly affect the overall system performance. In certain cases, accelerating certain operations using the hardware accelerator does not guarantee significant improvement in system performance. This is due to the fact that the performance bottleneck is caused by the communication overhead for the data exchange between GPEP and the hardware accelerators. Hence, based on the analysis of the computation and communication overhead, the system designer could decide which operations to be hardware-accelerated in final deployment model using the performance gain as the reference metric.

In the ECC-SoC design, the targeted final RTL implementable model is prototyped on Altera Nios II Stratix development board. As a result, the number of communication cycles of each data transmission is measured to be 19 clock cycles. This metric is obtained based on empirical study from previous design experience. Table 3 shows that in most cases (from Arch.1 to Arch. 6), most of the system timing performance is dominated by the computation, since the communication overhead is less than 0.10%. However, special attention is given to Arch. 2, where the modular division is hardware accelerated. With the almost same computation and communication overhead, this architecture is 4 times faster than Arch. 1. In addition, although Arch. 8 suffers from a serious communication overhead, which is more than 50% of overall system performance, it boosts up the system performance with about 389 times faster compare to all-software MAP MoC (Arch. 1).

6.4 Simulation Speed between ESL and RTL Models

In the simulation speed comparison between CTF and RTL models of the ECC-SoC, the fixed test vector which was discussed in the Section 5 using the bottom-up verification approach has been reused. To reduce the simulation time of the final RTL deployment model, all of the computations such as large integer modular arithmetic computation and message hashing are done by the hardware accelerator (HW MoC), except the random number generation since fixed test vectors are used. SystemC CTF model simulation is performed in terminal environment in Ubuntu Linux open source environment. The RTL simulation is performed on the VHDL design synthesized for implementation in an Altera Stratix Nios II-based FPGA development board using ModelSim 6.4. Both SystemC CTF and the VHDL RTL models use the same test vectors. The simulation is running on Intel Core2 CPU T5500 running at 1.66 GHz with 1GB RAM. The simulation speed gain, $\Delta G = t_{RTL}/t_{CTF}$ is computed for each ECC computations.

Table 4 shows that for simple computations such as field arithmetic computation like modular division and modular multiplication, the simulation speed of the SystemC CTF model is far more efficient than the HDL RTL deployment model. The SystemC simulation speed factor is at least 500 times faster than the RTL simulation. With the increases in algorithm complexity, such as ECDSA signature signing and signature verification (where the main operation is the point multiplication), the simulation speed gain increases to almost 1200 times.

Table 4: Simulation Speed Comparison between CTF and RTL model

Algorithm Complexity	Simulation Time of CTF, t_{CTF} (seconds)	Simulation Time of RTL, t_{RTL} (seconds)	Simulation Speed Gain, ΔG
Modular Division	0.18	121	673
Modular Multiplication	0.19	121	638
Modular Addition	0.17	117	690
Modular Subtraction	0.18	119	659
Point Addition	0.19	176	928
Point Doubling	0.20	181	907
Point Multiplication	3.82	4780	1251
ECDSA Key Deployment	3.84	4793	1248
ECDSA Signature Signing	3.84	4478	1166
ECDSA Signature Verification	7.23	9140	1264

6.5 Execution Time Difference Between CTF and RTL Models

Table 5 shows the difference of total execution time of the SystemC CTF model with the equivalent RTL deployment model, where the all-hardware MAP MoC is connected as coprocessor. The difference of total execution time, Δt is computed as

$$\Delta t = \frac{|T(D)_{CTF} - T(D)_{RTL}|}{T(D)_{RTL}} \qquad (3)$$

Table 5: Cycle count difference ratio of MAP MoC, Δt

Algorithm Complexity	CTF Cycle Count, $T(D)_{CTF}$	RTL Cycle Count, $T(D)_{RTL}$	Δt (%)
Modular Division	3,818	1,920	99
Modular Multiplication	3,876	1,935	100
Modular Addition	3,050	1,458	109
Modular Subtraction	3,050	1,452	110
Point Addition	19,610	13,285	48
Point Doubling	23,841	17,629	35
Point Multiplication	3,977,957	3,815,297	4
ECDSA Key Deployment	3,979,877	3,984,273	0
ECDSA Signature Signing	3,913,469	3,751,878	4
ECDSA Signature Verification	7,742,205	7,480,874	3

Table 5 shows that for all-hardware MAP MoC, simple operations such as the large integer modular arithmetic show an extremely high difference in terms of the total execution time. However, the execution time difference is reduced as the system complexity increases to less than 5% of complex arithmetic operations, such as point multiplication and ECDSA operations. This result is expected as the estimation of number of clock cycles is based on dynamic simulation approach. Hence, it has the unavoidable estimation overhead, and this overhead would be magnified when performing a simple operation. With complex operation such as ECDSA digital signature signing and verification, the estimation result provided by the co-simulation framework is similar to the one obtained in RTL.

7.0 CONCLUSION AND FUTURE WORK

This paper has proposed a system-level modelling methodology and design-space exploration technique for complex ECC-SoC design at Electronic System Level, using a combination of SystemC and UML. The test application is a network/data communication authentication system based on elliptic curve cryptography. The design covers the abstraction level from cycle-accurate timed functional model and the final implementable model, which is prototyped on Altera Stratix FPGA development board based on Nios II technology. Results show that the designs and architectures can be derived systematically. The system-level modelling methodology is capable of performing early system verification and design-space exploration. Moreoever, the design-space exploration performance estimation of SystemC CTF model to equivalent RTL model is 95% cycle count accuracy and up to 1000 times faster simulation speed for complex system modelling. The modelling of the HW MoC in this paper is limited to the basic RTL architecture that only supports certain hardware optimization techniques, such as resource sharing or tree reduction. Further work could involve the pipelining, array-based architecture, and other advance techniques. In addition, our design methodology only caters for FSM-based MoC, due to the reference SystemC discrete event simulation kernel. For other MoC architectures, such as Synchronous Data Flow (for DSP application) and Communicating Sequential Process (CSP), we need to enhance the approach towards a heterogeneous system. In addition, the design-space exploration could be further enhanced by exploring the other system architecture, such as the impact of sizes of instruction or data cache to the application performance, bus arbitration technique, on-chip communication architecture such as network-on-chip (NoC), and hardware or software architecture analysis.

REFERENCES

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

[2] Certicom, “The elliptic curve cryptosystem: Current public-key cryptographic systems”, White paper, Certicom Corporation, 2000.

[3] R. Laue, S. A. Huss, “Parallel Memory Architecture for Elliptic Curve Cryptography over *GF(P)* aimed at efficient FPGA Implementation”, *Journal of Signal Processing System*, Vol. 51, No. 1, April 2008, pp. 39–55.

[4] K. W. Lim, *An FPGA Implementation of an Elliptic Curve Processor for an Embedded Public-Key Cryptosystem*, M.Eng. Thesis, Universiti Teknologi Malaysia, 2005.

[5] M. Bednara, M. Daldrup, J. Teich, J. Gathen, J. Shokrollahi, “Tradeoff analysis of FPGA based Elliptic Curve Cryptography”, in *IEEE International Symposium on Circuits and Systems*, Scottsdale, Arizona, 26-29 May 2002, Vol. 5, pp. 797–800.

[6] S. Janssens, J. Thomas, W. Borremans, P. Gilsels, Verbauwhere, F. Vercauteren, B. Preneel, J. Candewalle, “Hardware/Software Co-design of an Elliptic Curve Public-key cryptosystem”, in *IEEE Workshop on Signal Processing Systems*, Antwerp, Belgium, 26-28 September 2001, pp. 209– 216.

[7] A.Hodjat, L.Batina, D.Hwang, I.Verbauwhede, “A Hyperelliptic Curve Crypto Coprocessor for an 8051 Microcontroller”, *in IEEE Workshop on Signal Processing Systems Design and Implementation*, Athens, Greece, 2-4 November 2005, pp. 93-98.

[8] K. Sakiyama, L. Batina, B. Preneel, I. Verbauwhede, "Superscalar Coprocessor for High-Speed Curve-based Cryptography", *in Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science*, Vol. 4249, 2006, pp. 415–429.

[9] Y. Hao, S. Ma, G. Chen, X. Zhang, H. Chen, W. Zeng, "Optimization Algorithm for Scalar Multiplication in the Elliptic Curve Cryptography over Prime Field", in *International Conference on Intelligent Computing, Lecture Notes in Computer Science*, Vol. 5226, 2008, pp. 904– 911.

[10] J. Grobschadl, S. S. Kumar, C. Paar, "Architectural Support for Arithmetic in Optimal Extension Fields", in *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, Galveston, Texas, 27-29 September 2004, pp. 111-124.

[11] J. Grobschadl, E. Saves, "Instruction Set Extensions for Fast Arithmetic in Finite Fields *GF(P)* and *GF(2^m)*", *in Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science*, Vol. 3156, 2004, pp. 133– 147.

[12] B. Bowyer, "The 'what and why' of transaction-level modeling (TLM)", March 2006, URL http://masters.donntu.edu.ua/2007/fvti/smeshkov/library/source04.htm

[13] P. Schaumont, I. Verbauwhede, "A Component-based Design Environment for ESL design", in *IEEE Design and Test of Computers*, Vol. 23, Issue: 5, May 2006, pp. 338– 347.

[14] M. Aydos, T. Yantk, C. K. Koc, "An high-speed ECC-based Wireless Authentication protocol on an ARM Microprocessor", in *Proceedings of the 16th Annual Conference on Computer Security Applications*, New Orleans, LA, USA, 2000, pp. 401–409.

[15] M. Aydos, B. Sunar, C. K. Koc, "An Elliptic Curve Cryptography based Authentication and Key Agreement Protocol for Wireless Communication", *in Proceedings of the 2nd International Workshop on Discrete Algorithms and Method for Mobile Computing and Communication Symposium on Information Theory*, Dallas, Texas, USA, 1998, pp 1-12.

[16] D. Johnson, A. Menezes, S. Vanstone, "The Elliptic Curve Digital Signature Algorithm (ECDSA)", White paper, Certicom Corporation, 2001.

[17] Federal Information Processing Standards, "Advanced Encryption Standard, Standard U.S. FIPS PUB 197", Federal Information Processing Standards Publication, 2001.

[18] SimIt-ARM. URL http://sourceforge.net/projects/simit-arm

[19] Certicom, "GEC1. Recommended Elliptic Curve Domain Parameters, Standards for Efficient Cryptography", Certicom Research, 1999.

[20] Federal Information Processing Standards, "Secure Hash Standard, Standard U.S. FIPS PUB 180-1", Federal Information Processing Standards Publication, 1995.

[21] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publications, Greenwich, CT, USA, 1999.

[22] J. Hlavac, *ALU for computing SLE's in Modular Arithmetic*, Diploma thesis, Czech Technical University Prague, 2003.

[23] S. C. Shantz, "From Euclid's GCD to Montgomery Multiplication to the Great Divide", *Technical Report TR-2001-95*, Sun Microsystems Laboratories, 2001.

[24] Certicom, "GEC2: Test vector for SEC1, Standards for Efficient Cryptography", Certicom Research, 1999.

[25] Y. W. Hau, "The RTL design of Modular Arithmetic Processor (MAP) for Elliptic Curve Cryptosystem over *GF(p)*", *Technical Report VeCAD-CRYPTO-IP-TR2008006*, VLSI-eCAD Research Laboratory (VeCAD), December 2008.

[26] Altera, *Stratix Device Handbook: Vol. 1*, User Manual, Altera Corporation, 2003.

[27] Altera, *Avalon Interface Specification*, User Manual, Altera Corporation, 2008.

[28] Y.W. Hau, *SystemC-based Design Framework for an Embedded System implemented as System-on-Chip*, Ph.D thesis, Universiti Teknologi Malaysia, 2009.

BIOGRAPHY

Yuan Wen, Hau obtained her Bachelors degree in Computer Engineering, and in 2005, obtained her Masters in Electrical Engineering degree, and eventually PhD degree with Merit in 2009, all from Universiti Teknologi Malaysia. Her specialization is in the field of advanced digital system design, embedded system and System-on-Chip (SoC), crypto hardware, and Electronic System Level (ESL) modelling. She is currently a post-doctoral researcher of the VeCAD Research Laboratory in the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, under the Post-Doctoral Fellowship Scheme (PDF) awarded by Ministry of Science, Technology & Innovation of Malaysia (MOSTI).

Mohamed Khalil-Hani obtained his PhD in Electrical and Computer Engineering from Washington State University, Pullman in 1992. He received his M.Eng in Electrical Engineering from Florida Atlantic, Boca Raton in 1985, and his B.Eng in Electrical Engineering (Communications) from University of Tasmania, Australia in 1978. He is a Senior Member of IEEE. Currently, he is a Professor of Microelectronic Embedded Systems and Computer Engineering at VeCAD Research Laboratory in the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Skudai, Malaysia. His areas of specialization are Advanced Digital Computing Architectures, Microelectronic embedded systems, System-on-Chip (SoC), FPGA and VLSI designs. His current fields of research include VLSI routing algorithms, Electronic System Level (ESL) modeling, SoC and hardware-software codesign techniques, high-performance encryption and cryptosystems for data security, neuro hardware for pattern recognition, image processing hardware architectures and real-time biometrics in embedded systems.

Muhammad N. Marsono received the Ph.D. degree in computer engineering from the University of Victoria, Victoria, BC, Canada in 2007, and the B.Eng and the M.Eng degrees in computer engineering and electrical engineering from Universiti Teknologi Malaysia, Malaysia, in 1999 and 2001, respectively. He is a member of IEEE. He is currently a Senior Lecturer with the Faculty of Electrical Engineering, Universiti Teknologi Malaysia, Malaysia. His research interests are in system-level design, digital VLSI design for communication and signal processing, and computer communication networks.