Crypto Embedded System for Electronic Document

Illiasaak Ahmad[1], Norashikin M. Thamrin[1], Mohamed Khalil Hani[1]

[1]VLSI-ECAD Research Laboratory,Microelectronic and Computer Engineering Department (MiCE)
Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
Tel: +60-7-5535223, Fax: +60-7-5566272, E-mail: ilyasak_ahmad@yahoo.com,
norashikin.mthamrin@gmail.com, khalil@fke.utm.my

Abstract

In this paper, a development of low-cost RSA-based Crypto Embedded System targeted for electronic document security is presented. The RSA algorithm is implemented in a re-configurable hardware, in this case Field Programmable Gate Array (FPGA). The 32-bit soft cores of Altera's Nios RISC processor is used as the basic building blocks of the proposed complete embedded solutions. Altera's SOPC Builder is used to facilitate the development of crypto embedded system, particularly in hardware/software integration stage. The use of Cryptographic Application Programming Interface (CAPI) to bridge the application and the hardware, and the associated communication layer in the embedded system is also discussed. The result obtained shows that the crypto embedded system provides a suitable compromise between the constraints of speed, space and required security level based on the specific demands of targeted applications.

Keywords:

Embedded System, Public Key Cryptography, FPGA, HW/SW Co-design

1. Introduction

Cryptography has gained an important role in today's information security problems. Security of the system can be enhanced if it is embedded in a re-configurable hardware. Such implementation is harder to tap, decompose, and attack, in general.

The protocols in public key cryptography like RSA, El-Gamal, etc, are excellent examples for implementing HW/SW co-design concept. Public key cryptography is based on the difficulty of factoring large numbers. To increase the operation speed, the algorithm is most often realized as a hardware component based on a parallel array of processing elements [1][2]. The hardware structures are generally fast enough, but not suitable for algorithm sequencing and they cannot be adapted to algorithm changes. Software, on the other hand, adapts itself easily but it is much slower and less secure. Thus implementing cryptographic algorithms in re-configurable hardware/SOC offers the best solution: it can consist of an embedded processor, one or more coprocessors, and software.

In this paper 1024-bit RSA algorithm is implemented. Due to hardware resource constraint, the encryption and verification modules are implemented as embedded code, while decryption and signing operations are performed in hardware. Chinese Remainder Theorem (CRT) is deployed not only to speed up decryption and signing operations, but also to utilize the 512-bit RSA co-processor designed in [3].

This paper is organized as follows: Section 2 covers the fundamental concept of RSA algorithm. The design of crypto embedded system is discussed in Section 3. Section 4 looks at the verification of the crypto embedded system and its performance. We discuss, in brief, the use of Cryptographic Application Programming Interface (CAPI) as high-level interface to the crypto embedded system in Section 5. Finally, concluding remarks is presented in the final section.

2. Overview of RSA Algorithm

As reported in [4], the most widely used public-key algorithm is RSA algorithm. This is due to the fact that, RSA can provide both confidentiality and digital signatures using the key-pair and under the same mathematical operation. Figure 1(a), 1(b), and 1(c) summarize RSA algorithm.

1. Generate 2 primes, p and q randomly, where $p \neq q$
2. Calculate M, where $M = p * q$
3. Calculate $\phi(M)$, where $\phi(M) = (p - 1)(q - 1)$
4. Generate E (public exponent) that fulfills $1 < E < \phi(M)$ and GCD $(\phi(M), E) = 1$
5. Calculate D (private exponent), where $D = E^{-1}$ Mod $\phi(M)$

Figure 1(a). RSA Key-Pair Generation

Plaintext (P) < M
Ciphertext (C) = P^E Mod M

Figure 1(b). RSA Encryption/Verification

Ciphertext (C)
Plaintext (P) = C^E Mod M

Figure 1(c). RSA Decryption/Signing

3. Design of Crypto Embedded System

An important aspect in embedded system design is partitioning the overall system into hardware and software components. This involves the physical partitioning of functionality into hardware or software, and it is influenced by system requirement, availability of device resources, IP core, and execution time. Figure 2 depicts the generic embedded HW/SW design flow.

The architecture of the crypto embedded system is shown in Figure 3. It has a processor core, on-chip memory, a co-processor, UART communication, and internal system bus. Nios [5], a 32-bit soft core from Altera, is used as the processor, while Avalon Bus [6] is used to enable communication between processor and RSA co-processor. The RSA co-processor is designed to perform the intensive part of computation of RSA algorithm. Nios processor is used to implement the more control intensive, parameterizable portions of RSA algorithm like embedded encryption module, and some parts of decryption module. SHA-1 has also been implemented in Nios (to be used with digital signature operation).

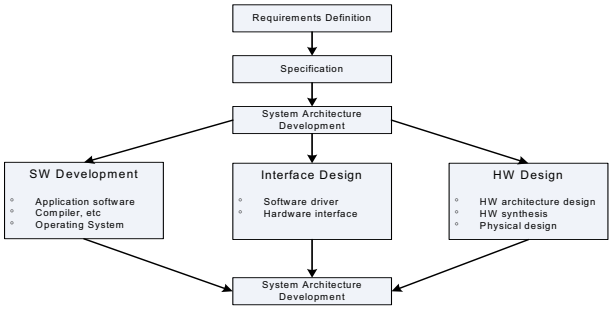

Figure 2. Generic Embedded HW/SW Design Flow

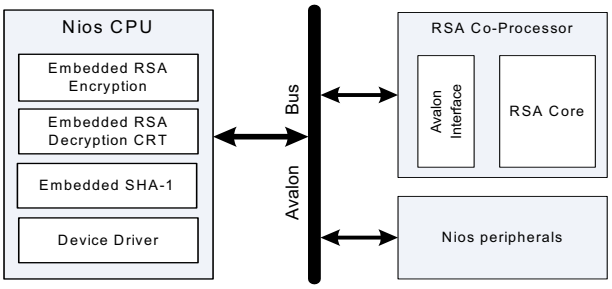

Figure 3. System Architecture of Crypto Embedded System

3.1 Interfacing RSA Co-processor with Embedded Processor

In this work, 512-bit RSA co-processor designed in [3] to perform 1024-bit RSA operation is used. This co-processor is designed to handle intensive modular arithmetic computations. It is, however, beyond the scope of this paper to discuss the implementation of this co-processor in detail.

To interface this co-processor with Nios processor, Avalon Bus is used as the bus system. Avalon Bus is a simple bus architecture designed for connecting on-chip processor and peripherals together into a system on a programmable chip. It is an interface that specifies the port connections between master and slave components, and also specifies the timing by which the components communicate [6]. Avalon Bus transactions transfer a single byte, half-word, or word (8, 16, or 32-bits) between a master and slave peripheral.

The RSA co-processor used in this work uses Avalon slave transfer mode that accept Avalon bus transfer from master port, which is Nios processor.

3.2 HW/SW Integration

Tools are available to integrate the processor, co-processor, and peripherals to become a complete embedded system. In this work, we used Altera SOPC Builder [7]. See Figure 4 for an illustration of SOPC Builder HW/SW integration flow. It is the interest of this paper to explore in greater details the Software Development portion of the flow.

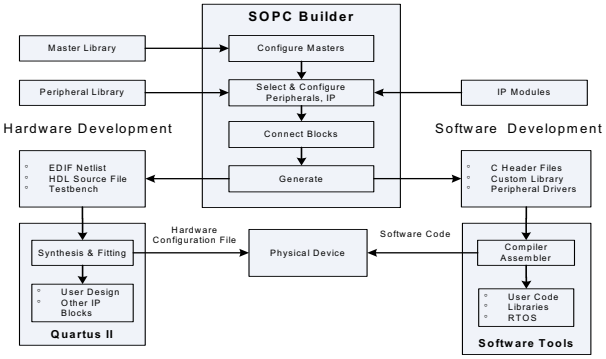

Figure 4. SOPC Builder HW/SW integration flow [8]

To generate the software development kit (SDK) for the embedded system, all the peripherals and IP core used must be added and configured, see Figure 5. The RSA Co-processor is added here as user-defined interface. The base address of this co-processor is set to *0x0900900*. Since Altera Apex EP20K20EFC484-2X development board is used, the clock frequency is set to 33.33 MHz. Other configuration parameters include assignment of interrupt priorities, and the setup, hold and wait requirements of each peripheral.

On the completion of generation process, SOPC Builder generates the hardware and software driver file called excalibur.h. Excalibur.h includes all the software interfaces for all the blocks in the embedded system. Apart from that, excalibur.h also includes the address for all registers and memories inside the SOPC Builder as well as associated software application programming interfaces (APIs) for IP blocks that include APIs. Figure 6(a) and 6(b) show the excerpts taken from excalibur.h.

3.3 Embedded Software Development

Based on the previous section, the SOPC builder generates a custom SDK. The SDK contains the memory map and data structures for accessing hardware components in the system.

It also provides routines for accessing the peripherals like UART. So this SDK can be used to communicate easily with fundamental system components and custom IP cores.

Application programming interfaces are developed using C language to control the RSA Co-processor operations. The interfaces are:

- RSA_OperandM()
- RSA_OperandE()
- RSA_OperandR()
- RSA_OperandX()
- RSA_Ouput()
- RSA_MonMult_vec()
- Compute_R()

Besides that, embedded code for encryption and decryption with CRT are also coded. Figure 7 illustrates the APIs and the embedded software.

A device driver for the crypto embedded system is also developed to enable communication between crypto embedded system and external world like personal computer. The flow of the device driver can be described as state diagram. See Figure 8.

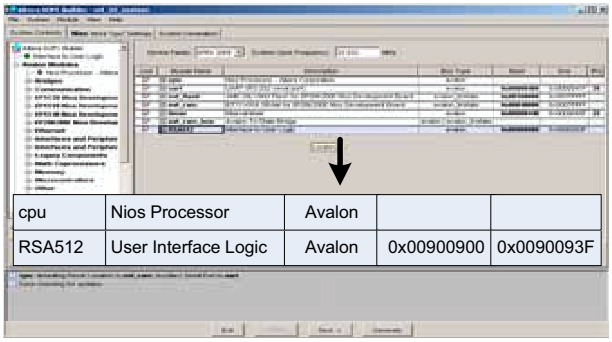

Figure 5. Configuring Crypto Embedded System Using SOPC Builder

```
// The Memory Map
#define na_cpu                  ((void *) 0x00000000) // altera_nios
#define na_cpu_base             0x00000000
#define na_uart                 ((np_uart *) 0x00000400) // altera_avalon_uart
#define na_uart_base            0x00000400
#define na_uart_irq             26
#define na_timer                ((np_timer *) 0x00000440) // altera_avalon_timer
#define na_timer_base           0x00000440
#define na_timer_irq            25
#define na_ext_ram              ((void *) 0x00040000) // altera_nios_dev_board_sram32
#define na_ext_ram_base         0x00040000
#define na_ext_ram_end          ((void *) 0x00080000)
#define na_ext_ram_size         0x00040000
#define na_ext_flash            ((void *) 0x00100000) // altera_nios_dev_board_flash
#define na_ext_flash_base       0x00100000
#define na_ext_flash_end        ((void *) 0x00200000)
#define na_ext_flash_size       0x00100000
#define na_RSA512               ((np_usersocket *) 0x00900900) // altera_avalon_user_defined_interface
#define na_RSA512_base          0x00900900
```

Figure 6(a). Excerpt from excalibur.h: Memory Map

```
// Structures and Routines For Each Peripheral

// Nios CPU Routines
void nr_installcwpmanager(void);    // called automatically at by nr_setup.s
void nr_delay(int milliseconds);    // approximate timing based on clock speed
void nr_zerorange(char *rangeStart,int rangeByteCount);
void nr_jumptoreset(void);

// Default UART routines
void nr_txchar(int c);
void nr_txchar2(int c, int channel);
void nr_txstring(char *s);
int nr_rxchar(void);
```

Figure 6(b). Excerpt from excalibur.h: APIs

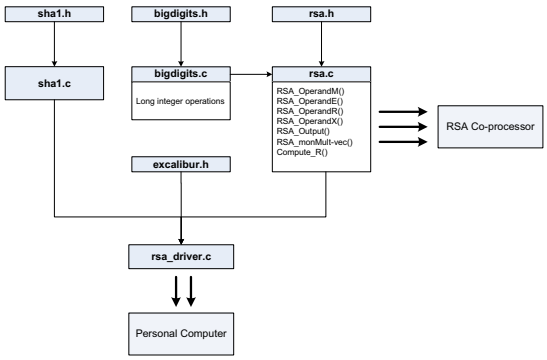

Figure 7. APIs and Embedded Code

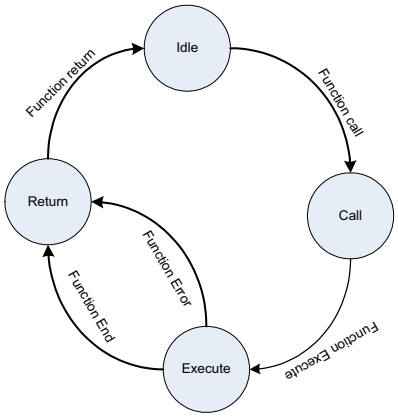

Figure 8. Device driver state diagram

4. Result

To evaluate and verify software/hardware modules in this solution and its impact on the speed of the RSA operation, test application is developed. The test application is run on Nios processor.

The test vector is obtained from National Institute of Standards and Technology (NIST) [9] and hard-coded in the test program. For this test we have predefined the value of public key to be '17', and this value can be easily changed according to the user's requirement later. Table 1 shows the timing of the RSA operations (CRT is deployed in RSA decryption).

Table 1. Execution times of RSA on Altera APEX EP20K200EFC484-2X clocked at 33.33 MHz

Operation	Time (ms)
Encryption (software)	10
Decryption (software with CRT)	18338
Decryption (hardware with CRT)	111

The result obtained here is also compared with ARM SecurCore [11], one of the commercial products available in the market. Table 2 below summarizes the comparisons that are made. Based on that table, we can see that the result obtained is quite competitive in terms of performance.

Table 2. Comparison with commercial product

Specifications	Our design	ARM SecurCore
KeySize	1024	1024
Clock	33.33 Mhz	20 MHz
Decryption (with CRT)	111 ms	330 ms
Encryption (software)	10 ms	-

5. Cryptographic Application Programming Interface (CAPI)

Modern embedded systems are built using various techniques that provide flexibility and reliability. One of the most important techniques centres on the use of applications programming interface.

An application-programming interface (API) is basically a well-defined boundary between two system components that isolates a specific process or a set of services. For example, it is quite common now for an application to interact with e-mail service using e-mail API like MAPI (Microsoft), VIM (Lotus), and others. In such cases, the API defines a set of services that allow an application to retrieve or submit mail messages from or to the mail server.

A cryptographic application programming interface (CAPI), like other APIs, is an API specifically designed to support cryptographic functions. Technically, a CAPI would provide and interface to a set of cryptographic services such as encryption/decryption, digital signatures/verification, key generation, etc. Figure 9 depicts the relation between CAPI and crypto services.

A simple and easy to use CAPI called *myCAPI* is developed for the crypto embedded system presented in this paper. *myCAPI* has a set of well-defined APIs that enable application developers integrate crypto embedded system services into their application. The list of available APIs is listed in Table 3.

In addition to *myCAPI,* the crypto embedded system services can also be called-up through Microsoft CryptoAPI interface [10]. This integration would benefit the application developers. Since Microsoft CryptoAPI has been defined as one of the standard CAPI, application that utilizes Microsoft CAPI can access multiple cryptographic implementations through a single interface, see Figure 9.

6. Conclusion

In this paper, the design of crypto embedded system targeted for electronic document security has been presented. The crypto embedded system is implemented in re-configurable hardware, which is FPGA. Altera CAD tool, SOPC Builder is used to facilitate and demonstrate hw/sw design and development flow. The result obtained shows that the crypto embedded system provides a suitable compromise between the constraint of speed, space and required security level based on the specific demands of targeted applications.

Table 3. *myCAPI* APIs

API	Description
utmGenKeyPair()	Key-pair generation
utmRSASigning()	RSA digital signatures
utmRSAVerification()	RSA digital signatures verification
utmRSAEncryption()	RSA encryption
utmRSADecryption()	RSA decryption

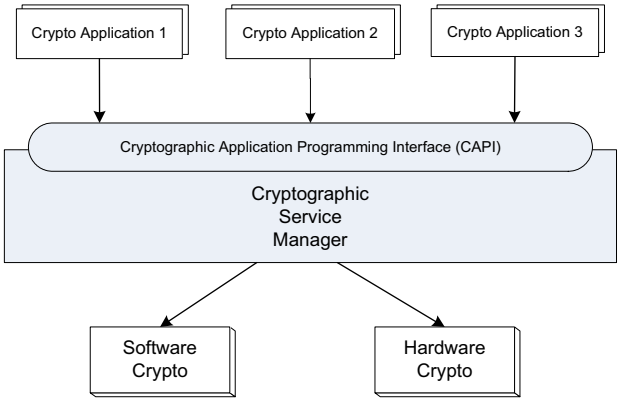

Figure 9. CAPI Architecture

References

[1] A. F. Tenca and C. K. Koc. A scalable architecture for Montgomery multiplication. In C.K. Koc and C. Paar, editors, Cryptographic Hardware and Embedded Systems, number 1717 in Computer Science, pages 94-108, Berlin, Germany, 1999. Springer Verlag.

[2] M. Drutarovskỳ, V. Fischer, and M. Śimka. Two Implementation Methods of Scalable Montgomery Coprocessor Embedded in Reconfigurable Hardware. Cryptographic Hardware and Embedded Systems 2003.

[3] Paniandi. A, 2005. A Hardware Implementation of RSA Co-processor for Resource Constrained Embedded Systems. Master Dissertation, Faculty of Electrical Engineering, Uni. Teknologi Malaysia.

[4] B. Schneier. Applied Cryptography: Protocol, Algorithm and Source Code in C. 2nd Edition, John Wiley & Sons Inc, NY. 1996.

[5] Nios Soft Core Embedded Processor, http://www.altera.com/nios

[6] Avalon Bus Specification, http://www.altera.com/literature/manual/

[7] Altera SOPC Builder, http://www.**altera**.com/products/software/products/**sopc**/

[8] Ekas, P; Jentz B, Fall 2003. Developing and integrating FPGA coprocessors. Embedded Computing Design.

[9] Keller. S. S, 2004. The RSA Validation System (RSAVS). National Institute of Standards and Technology (NIST). USA.

[10] Microsoft Developer Network, http://msdn.microsoft.com/library/

[11] ARM Products and Solutions – Core Type, http://www.arm.com/products/CPUs/securcore.html