Design and Implementation of Robust Embedded Processor for Cryptographic Applications

Kazim Yumbul
Gebze Institute of Technology
Gebze, Kocaeli, 41400 Turkey
kyumbul@gyte.edu.tr

Serdar Süer Erdem
Gebze Institute of Technology
Gebze, Kocaeli, 41400 Turkey
serdem@gyte.edu.tr

Erkay Savaş
Sabanci University
Tuzla Istanbul, 34956 Turkey
erkays@sabanciuniv.edu

ABSTRACT

Practical implementations of cryptographic algorithms are vulnerable to side-channel analysis and fault attacks. Thus, some masking and fault detection algorithms must be incorporated into these implementations. These additions further increase the complexity of the cryptographic devices which already need to perform computationally-intensive operations. Therefore, the general-purpose processors are usually supported by coprocessors/hardware accelerators to protect as well as to accelerate cryptographic applications. Using a configurable processor is just another solution. This work designs and implements *robust* execution units as an extension to a configurable processor, which detect the data faults (adversarial or otherwise) while performing the arithmetic operations. Assuming a capable adversary who can injects faults to the cryptographic computation with high precision, a nonlinear error detection code with high error detection capability is used. The designed units are tightly integrated to the datapath of the configurable processor using its tool chain. For different configurations, we report the increase in the space and time complexities of the configurable processor. Also, we present performance evaluations of the software implementations using the robust execution units. Implementation results show that it is feasible to implement robust arithmetic units with relatively low overhead in an embedded processor.

Categories and Subject Descriptors

C [**Computer System Organization**]: Special-Purpose and Application-Based Systems; C.3 [**Special-Purpose and Application-Based Systems**]: Smartcards; E [**Data**]: Data Encryption; E.3 [**Data Encryption**]: Public Key Crypto Systems

General Terms

Security

Keywords

Security, Cryptographic Algorithms, Montgomery, Robust Arithmetic Operations, Instruction Set Extensions, Computer Architecture

1. INTRODUCTION

Fault attacks are active attacks in which the attacker forces a cryptographic device to perform faulty operations in order to extract the secret information. Faults can be injected by manipulating the working conditions of the device (supply voltage, working temperature, clock rate, etc.) or exposing it to some radiation source (white light, laser, X-ray, ion beam, etc.) [16, 7]. Fault attacks are more effective than passive attacks since the attacker is capable of doing much more than just observing the side channel information of a properly working device. Furthermore, tamper resistance is not enough to stop this kind of attacks. There is a large number of works in the literature dealing with fault attacks on both private key and public key systems [3, 4, 1, 13].

Cryptographic devices must have tamper- and fault-detection capabilities against fault attacks. When a cryptographic device detects a fault attack, it takes the appropriate action (reseting the device, erasing the secret key, etc.). Sensors embedded into the device can provide tamper detection. Also, error detection codes can be used to detect the data errors which are either maliciously introduced by adversary or naturally occurred. For example, the works [2, 11, 10] propose simple error detection schemes using parity bits for the symmetric cipher implementations. Also, the works [14, 15] add simple error detection circuits into finite field multipliers to detect faulty computations. However, all these works are effective against simple error patterns which can be encountered naturally during computations. Thus, they may fail to detect more complex error patterns generated by an intelligent attacker.

Non-linear error detecting codes have certain advantages over linear error detecting codes for the attack models in which the attacker cannot have a priori information about the data. This is because undetected error patterns are completely data dependent for nonlinear codes. Some systematic non-linear error detecting codes have been proposed in [9]. The use of these codes in cryptographic applications has been studied in [8, 12]. Later, the work [6] presented a new class of nonlinear systematic codes which shows similar error detection properties with the codes in [9]. These codes are termed 'Robust codes' since they show a uniform error detection capability for different error patterns.

In this work, we design a variety of execution units for a configurable processor to perform integer addition and multiplication operations, which are secured against the fault analysis. While these execution units perform the integer operations, they also calculate parity information from the input operands and the results, using the non-linear error detecting code given in [6]. The execution units (i.e. multiplier and adder) are hence referred as robust arithmetic units. Also, they issue error signals, if the redundancy checks fail. In other words, any fault (i.e. an unintended bit flipping in a register or an execution unit) can be detected with an overwhelming probability.

We choose the configurable processor, Xtensa's LX2 [17], as the underlying platform. We implement the execution units using Tensilica Instruction Extension Language (TIE), which is a Verilog-like language. Then, using the Tensilica tools, we obtain the RTL description of the core processor with the robust execution units being added to its data path. The resulting RTL code can be mapped to ASIC or FPGA implementations easily.

Using configurable processors has the following advantages over using coprocessors or hardware accelerators:

- Additional execution units are more tightly coupled to the processor core. Thus, the communication overhead with the CPU is less and compromising the sensitive data is more difficult.
- The operations in the additional execution units can share the same pipeline with other operations.

We add custom instructions utilizing the designed robust execution units to the Xtensa LX2 processor instruction set so that the software implementations can perform robust addition and multiplication operations. We adopt three different approaches in the design of the additional execution units performing the error detection calculations. As a result, we generate three different configurations for the Xtensa LX2 processor with hardware error detection support. Using Tensilica tools, we obtain the space and time complexities for the ASIC and the FPGA implementations of all the three configurations and the base LX2 core. The results show that we can add the robust addition and multiplication operations to the LX2 instruction set without a significant increase in space and time complexities.

Also, using robust addition and multiplication instructions, we developed C implementations of the Montgomery modular multiplication algorithm, which is commonly used in cryptographic applications [5]. Then, we evaluate the performance of these implementations for all configurations to further test our design.

This paper is organized as follows. Section 2 discusses the nonlinear error detection codes (robust codes). Section 3 describes the general architecture for the hardware extensions as well as presents various designs for the robust addition and multiplication execution units. Section 4 introduces several configurations of the Xtensa LX2 processor which provide robust arithmetic instructions for software implementations. Section 5 gives the implementation results.

2. ROBUST ARITHMETIC CODES

The work [6] presents the robust quadratic codes

$$C = \{(x, w) \mid x \in \mathbb{Z}_{2^k}, w = x^2 \bmod p \in \mathbb{F}_p\}$$

where x is the k-bit data part and w is the r-bit redundancy part of the code, respectively, where $r = \lceil \log_2(p) \rceil$. Let e_x denote the error in the data part x and e_w denote the error in the parity part w. Then, an error (e_x, e_w) in this code cannot be detected, if

$$\begin{aligned}(x + e_x \bmod 2^k)^2 \bmod p &= w + e_w \bmod 2^r \ , \\ &= (x)^2 \bmod p + e_w \bmod 2^r \ .\end{aligned}$$

Assume that an attacker has no a priori information for the data x and its computed redundancy w. If the attacker generates the error $e = (e_x, e_w)$, the probability that this error remains undetected (error masking probability) can be given by [6]

$$Q(e) = Q(e_x, e_w) = \frac{|\{x|(x + e_x, w + e_w) \in C\}|}{|C|} \ .$$

If any given code C minimizes $Q(e)$ value for all non-zero values of x, then this code is called robust code. Overall robustness is achieved for $max_{e \neq 0}(Q(e)) = 2^{-r}$. If an upper limit as $max_{e \neq 0}(Q(e)) \leq \epsilon 2^{-r}$ for a code C is given, then this code is called ϵ-robust. ϵ is a constant number much smaller then 2^r. In [6], it is shown that

$$\epsilon = \max(4 \ , \ 2^k - p + 1)$$

where $k = r$.

In this study, we chose $k = r = 32$, since the word length in a general-purpose microprocessor is 32. Then, the closest prime number to $2^k = 2^{32}$ is $2^{32} - 5$. Therefore, $Q(e)$ is calculated as follows:

$$Q(e) = (2^k - p + 1) \times 2^{-k} = 3 \times 2^{-31}$$

As we can see from the above equation, the probability of forming an undetected error by an attacker is approximately one in a billion. On the other hand, forming an undetected error for linear codes is quite easy. For instance, in the case of having a parity length equal to the data length, generating the same error in both data and parity will yield an undetected error, $e_x = e_w$. Robust code can be defined for all values of k. However, choosing the prime number p as close as possible to 2^k not only facilitates the calculations but also minimizes the number of undetected errors.

3. GENERAL ARCHITECTURE

We propose to extend the configurable Xtensa 2 processor with cryptographic unit to perform robust addition and multiplication operation whereby errors, malicious or random, are detected with a very high probability. Using the TIE language developed for this processor, one can design various hardware units to add in parallel to the ALU of the processor.

The proposed cryptographic unit (specified as robust execution zone) is illustrated in Figure 1. The unit has two parts:

1. The cryptographic register file (CRF) which consists of sixteen 32-bit registers and some special-purpose registers used during computation as temporary storage.
2. The robust execution unit (REU).

The CRF is used to store the temporary results of the arithmetic operations. The REU includes the circuits performing robust multiplication and robust addition operations.

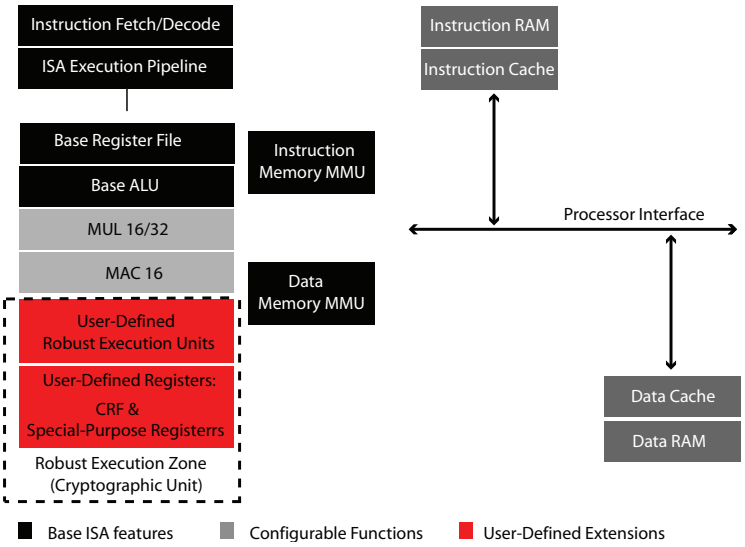

Figure 1: General Architecture

As seen in Figure 1, the REU is tightly integrated into the processor core. It uses the pipeline structure and interconnection infrastructure. Thus, the instructions added by the cryptographic unit are executed in the same fashion as the native instructions of the processor core. Figure 2 shows the functional units inside the REU where data transfer and arithmetic/logic operations are performed on 32-bit operands. These units perform 32-bit integer additions and 32-bit integer multiplications with 64-bit results. While these arithmetic operations are performed, non-linear residue codes are also calculated from the inputs and the outputs for fault detection. By this way, the REU enhances Xtensa 2 processor with robust arithmetic operation capability, protecting the execution against fault attacks.

We use the error detection capabilities of the robust codes for $k = 32$ bit data word in the REU. Let x be an input or an output operand. Its non-linear residue is calculated as $|x^2|_p = x^2 \bmod p$. We implemented the robust arithmetic only for the multiplication and addition operations. Nevertheless, these operations are widely used in cryptographic algorithms and thus, the algorithms using them will become robust for faults too.

As can be seen from the Figure 2, the data path and the interconnection structure of robust instruction execution is the same as those of the existing integer data path of RISC processor. Therefore, the new instructions are executed in the same manner as any other native RISC instruction.

Figure 3 shows the pipeline stages for robust addition and multiplication instructions. Similarly, Figure 4 illustrates some examples that show pipeline stalls due to data dependences among the robust instructions. For example, two back-to-back robust multiplication instructions (the first two instructions in Figure 4) do not cause pipeline stall as long as they are independent. However, the second and third instructions in Figure 4 creates two-cycle stall since the latter needs the parity value from the former, which will be available two clock cycles later than usual.

3.1 Robust Adder

We add the RADDC (robust addition with carry) instruction to Xtensa processor to perform the robust addition operation. This instruction works in a similar way to conventional integer addition with carry instruction which is found in almost every processor except that it checks its output for errors using non-linear residue codes. The operands of the RADDC instruction are a, b, and c_{in} while its outputs are the sum c_L and the carry c_H. When the RADDC instruction is executed, two different parity values $\left|c^2\right|_p$ and $\left|c^2\right|_p^*$ are also calculated. While $\left|c^2\right|_p$ is calculated from the input operands a, b, and c_{in}, $\left|c^2\right|_p^*$ is calculated from the outputs c_L and c_H. If there is no error, these two parity values must be equal to each other.

$\left|c^2\right|_p$ can be calculated as follows.

$$\begin{aligned}\left|c^2\right|_p &= \left|(a+b+c_{in})^2\right|_p \\ &= \left|\left|a^2\right|_p + \left|b^2\right|_p + 2(ab + c_{in}(a+b)) + c_{in}\right|_p\end{aligned}$$

$\left|c^2\right|_p^*$ can be calculated as follows.

$$\begin{aligned}\left|c^2\right|_p^* &= \left|\ (c_H.2^k + c_L)^2\ \right|_p \\ &= \left|\ c_H.\left|2^{2k}\right|_p + c_H.\left|c_L\right|_p.\left|2^{k+1}\right|_p + \left|c_L^2\right|_p\ \right|_p \\ &= \left|\ c_H.\left|2^{2k} + c_L.2^{k+1}\right|_p + \left|c_L^2\right|_p\ \right|_p\end{aligned}$$

As a result, we check the following equality in the hardware to ensure that the parity values are equal, i.e. $\left|c^2\right|_p = \left|c^2\right|_p^*$.

$$\left|\ c_H.\left|2^{2k} + c_L.2^{k+1}\right|_p + \left|c_L^2\right|_p\ \right|_p = \left|\ \left|a^2\right|_p + \left|b^2\right|_p + 2(ab + c_{in}(a+b)) + c_{in}\ \right|_p$$

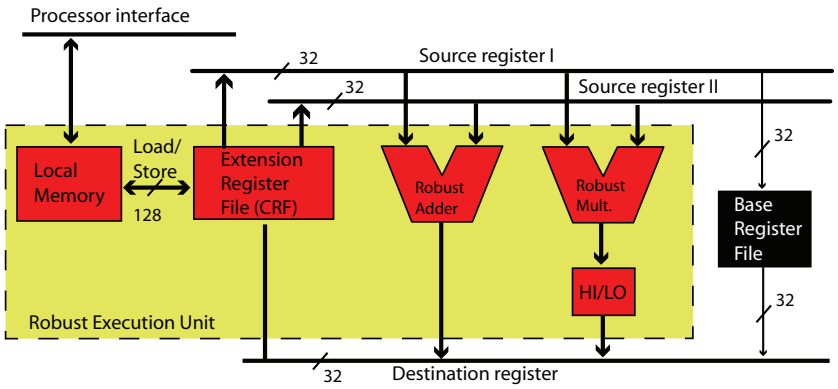

Figure 2: Detailed Architecture of the REU

If the above equality holds, it can be said that the obtained result does not contain any error. Figure 5 shows the block diagram of the robust addition operation circuit, which is implemented and integrated to the datapath of Xtensa 2 processor.

3.2 Robust Multiplier

We also use the non-linear residue codes to detect the errors in the multiplication operation. The multiplication operation takes two $k = 32$ bit inputs, a and b, and produces the $2k$ bit output $c = ab$. Also, the lower and higher k bit parts of the output are denoted by c_L and c_H, respectively. When the multiplication operation is executed, two different parity values $\left|c^2\right|_p$ and $\left|c^2\right|_p^*$ are calculated as in the addition. While $\left|c^2\right|_p$ is calculated from the input operands a and b, $\left|c^2\right|_p^*$ is calculated from the outputs c_L and c_H. If there is no error, these two parity values must be equal to each other.

The calculations of $\left|c^2\right|_p$ and $\left|c^2\right|_p^*$ are as follows.

$$\left|c^2\right|_p = \left| \left|a^2\right|_p . \left|b^2\right|_p \right|_p$$

$$\begin{aligned}\left|c^2\right|_p^* &= \left| \left(c_H.2^k + c_L\right)^2 \right|_p \\ &= \left| c_H^2.2^{2k} + c_H.c_L.2^{k+1} + c_L^2 \right|_p \\ &= \left| \left|c_H^2\right|_p . \left|2^{2k}\right|_p + |c_H|_p . |c_L|_p . \left|2^{k+1}\right|_p + \left|c_L^2\right|_p \right|_p\end{aligned}$$

As a result, we check the following equality in the hardware to ensure that the parity values are equal, i.e. $\left|c^2\right|_p = \left|c^2\right|_p^*$.

$$\left| \left|a^2\right|_p . \left|b^2\right|_p \right|_p = \left| \left|c_H^2\right|_p . \left|2^{2k}\right|_p + |c_H|_p . |c_L|_p . \left|2^{k+1}\right|_p + \left|c_L^2\right|_p \right|_p$$

If the above equality holds, it can be said that the obtained result does not contain any error. The block diagram of the robust multiplication operation circuit is shown in Figure 6.

4. PROCESSOR CONFIGURATIONS FOR SECURE SOFTWARE

The Montgomery modular multiplication algorithm is implemented in C language by employing the robust addition and multiplication instructions mentioned previously. In this implementation, we choose the SOS (separate operand scanning) method proposed in the literature for the Montgomery modular multiplication algorithm [5]. The C implementation of the robust Montgomery multiplication is simulated on Xtensa LX2 processor whose instruction set is extended by the robust multiplication and addition instructions. Since these instructions are integrated to the data path of the processor, the software can use them freely like any native instruction.

We study four different hardware configurations and name them as Configuration 0, Configuration 1, Configuration 2, and Configuration 3. These configurations differ from each other with their robust arithmetic capabilities. The following steps are carried out in this work:

- A 32 bit simple processor compilation is implemented on Tensilica.
- A robust *adder* is designed and implemented on Tensilica TIE language.
- A robust *multiplier* is designed and implemented on Tensilica TIE language.

4.1 Configuration 0

Configuration 0 means a simple 32-bit Xtensa LX2 microprocessor which does not contain any protection against fault injection attacks. In this configuration, the embedded processor has the following hardware resources.

- 16-bit MAC with 40 bit Accumulator
- Multiplication instructions MUL16, MUL32, accumulated MUL32, MULUH, and MULSH
- Pipeline length 5
- Widths of Cache and Memory Interface
  - Width of Instruction Fetch Interface 32 bit
  - Width of Data Memory/Cache interface 32 bit

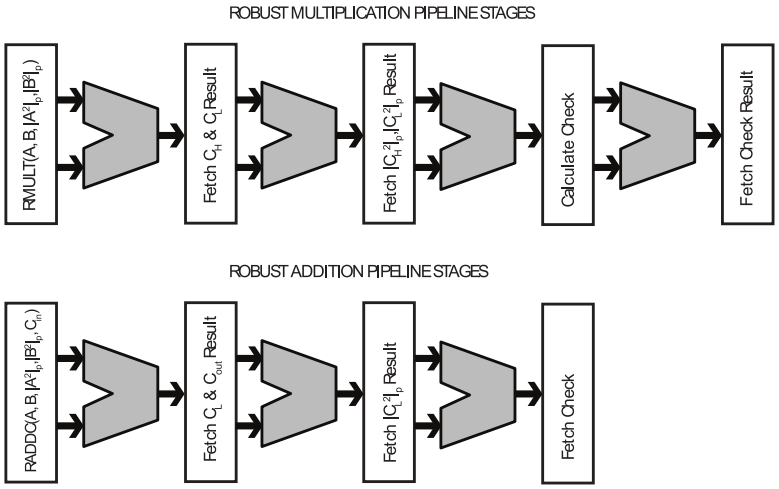

Figure 3: Robust Addition And Multiplication Pipeline Structure

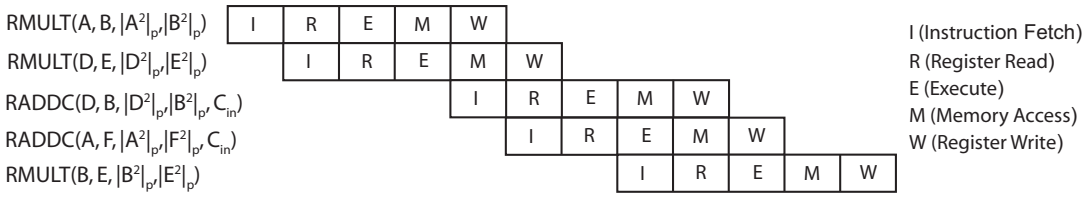

Figure 4: Robust Instruction Behavior at CPU Pipeline

- Width of PIF interface 32 bit
- Width of Interface to instruction cache 32 bit

- Instruction Cache size / Line size (Bytes) 4096 /16
- Data Cache / Line size (Bytes) 4096 / 16
- System RAM size 128k
- System Rom size 16k

4.2 Configuration 1

In this configuration, the robust addition and multiplication instructions are added to Xtensa LX2 microprocessor. The hardware circuits performing these instructions are depicted in Figure 5 and Figure 6. These circuits are described in TIE language which is developed for Xtensa LX2 microprocessors and integrated to the datapath of the processor core.

The robust addition and multiplication are called as follows.

$$(c_L, c_{\text{out}}) = \text{RADDC}\,(a, b, c_{\text{in}})$$
$$(c_L, c_H) = \text{RMUL}\,(a, b)$$

The Configuration-1 protects the cryptographic computation against faults occurred during the execution of integer multiplication and addition operations. It does not protect temporary values while they are in the registers. Therefore, this configuration provides only limited protection against fault attacks.

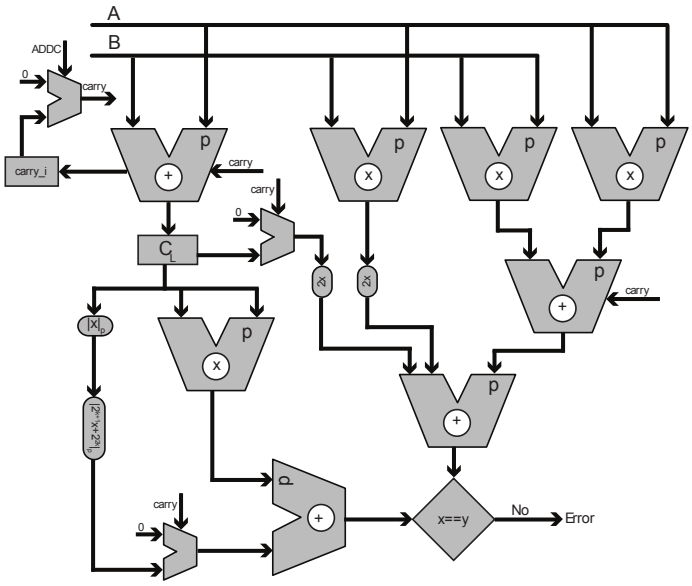

Figure 5: Robust Addition for Configuration-1

4.3 Configuration 2

As mentioned previously, configuration 1 can detect only the errors that occur during the execution of multiplication

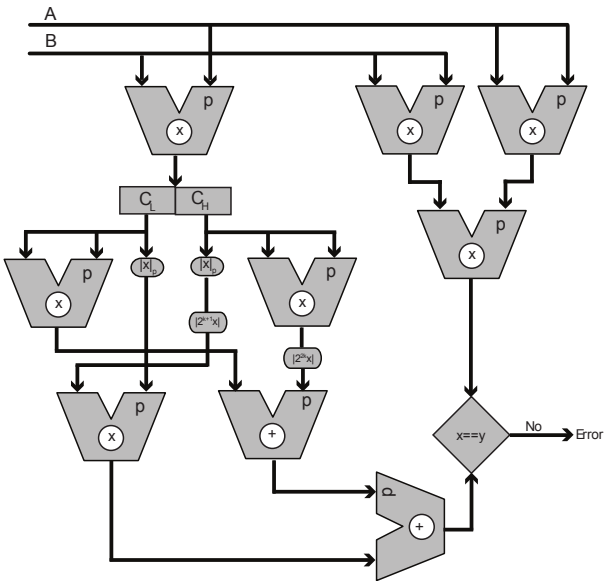

Figure 6: Robust Multiplication for Configuration-1

and addition operations. It, however, cannot detect the errors which the attackers may insert before or after the execution of these operations. Nevertheless, the inputs and outputs of these operations are vulnerable to fault injection attacks when they are held in the registers or in the memory.

As a remedy to this problem, the robust arithmetic instructions of Configuration 2 take the previous parity values calculated for their input operands as additional inputs and return their outputs together with the updated parity values. The robust addition and multiplication operation are implemented as follows.

$$(c, \left|c^2\right|_p, c_{\text{out}}) = \text{RADDC}\left(a, \left|a^2\right|_p, b, \left|b^2\right|_p, c_{in}\right)$$
$$(c_L, \left|c_L^2\right|_p, c_H, \left|c_H^2\right|_p) = \text{RMUL}\left(a, \left|a^2\right|_p, b, \left|b^2\right|_p\right)$$

As seen above, these robust instructions not only take a and b but also the parity values as input. Similarly, they return the updated parity values together with the operation results. Thus, Configuration 2 can keep track of the parity values throughout the entire execution of a cryptographic algorithm. The hardware circuits performing the robust instructions are depicted in Figure 7 and Figure 8. These circuits are described in TIE language which is developed to Xtensa LX2 microprocessors and integrated to the datapath of the processor core.

Contrary to Configuration 1, Configuration 2 protects the data not only during the arithmetic operations but also when stored in the register file or memory. Therefore, it provides better protection. On the other hand, Configuration 2 has a larger complexity than Configuration 1.

4.4 Configuration 3

In Configuration 3, only the instructions whose hardware organization is depicted in Figure 9 are used. Robust multiplication and addition operations are implemented in software by using these instructions. The a, $|a^2|_p$, b and $|b^2|_p$ variables are used as arguments to the functions implemented in software. The results of multiplication and addition op-

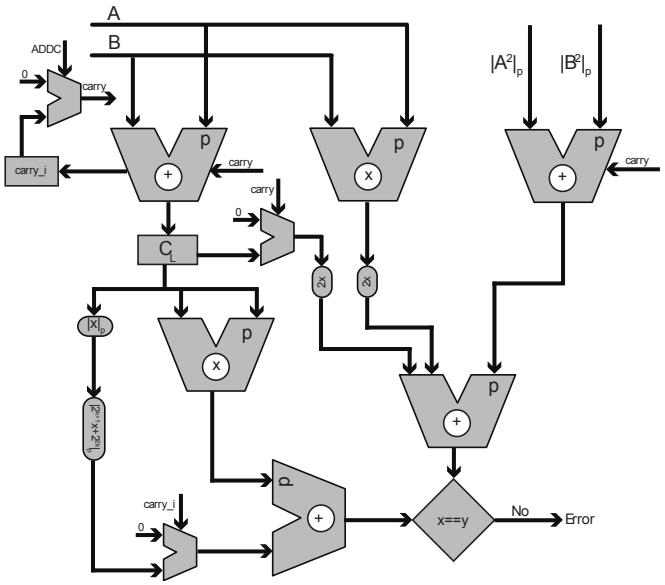

Figure 7: Robust Addition for Configuration-2

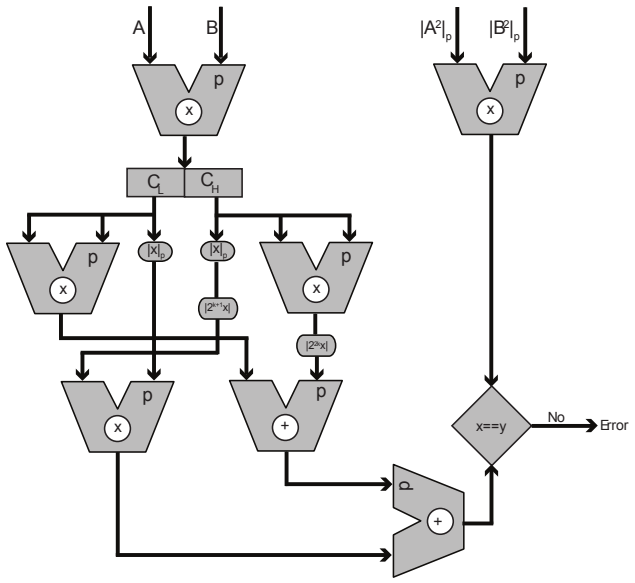

Figure 8: Robust Multiplication for Configuration-2

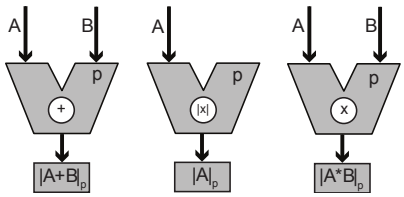

Figure 9: Robust Operations for Configuration-3

Table 1: Clock Cycle Comparison for Montgomery Implementation

Configurations	2048-bit	1024-bit	512-bit
Configuration 0	350,858	97,004	30,260
Configuration 1	154,076	42,022	13,675
Configuration 2	190,687	51,511	16,263
Configuration 3	1.942.406	490,029	125,464

erations and error conditions in performed operations are controlled. In addition to arithmetic results $\left|c_H^2\right|_p$ and parity values are calculated and given as output. Multiplication and addition functions are defined as follows:

$C = RADDC\left(a, \left|a^2\right|_p, b, \left|b^2\right|_p, c_{in}\right)$

$C = RMUL\left(a, \left|a^2\right|_p, b, \left|b^2\right|_p\right)$

5. IMPLEMENTATION RESULTS

This section outlines the results of hardware and software implementations. Table 1 enumerates the execution times of the C implementations of the Montgomery multiplication algorithm for the four hardware configurations mentioned previously. Montgomery implementation running on Configuration 0 does not perform error detection calculations neither in software nor in hardware. However, it is not the fastest in Table 1. This is because LX2 base core does not have a long multiplication instruction which can generate 64 bit results. Its multiplication instruction just takes 32 bit input operands and generates 32 bit multiplication result. Nevertheless, other configurations have robust multiplication units which can generate 64 bit results. Configuration 2 is slower than Configuration 1 since it is more complicated. Configuration 2 robust operations must take the parity data of the previous operations as additional inputs, and give the updated parity data as additional outputs. Configuration 3 is the slowest since it has to perform error detection by the simplest robust operation units.

Table 2 summarizes speed and area information obtained for ASIC implementation. This table shows that the instruction extensions do not affect the processor speed [1]. However, it causes some increase in the area. The configuration with the highest space requirement is Configuration 2. Then, Configuration 3 and 1 follows it, in this order.

Table 3 summarizes speed and space information obtained for FPGA implementation. As seen on the table maximum execution periods exhibit variations as a result of additional instructions. The fastest configuration is Configuration 0, which is the default configuration. Then, Configuration 3, 2 and 1 follow it in this order. Moreover, the number of gates used differs in each configuration. Increase in the number of gates is in synchronization with area figures in ASIC implementation. In this case Configuration 2 has the maximum number of gates. Configuration 1 and 3 follows it, in this order.

When we observe the values presented in Table 1 and Table 2 it can be seen that Configuration 1 has better results in comparison to the default and other configurations. Although there is an increase in the area, there is also a two-fold increase in speed because of reduced number of instructions. Other configurations do not have this. However, when we look into the maximum clock periods obtained from FPGA compilations, Configuration 1 is almost two times slower than the default configuration. Although Configuration 3 appears as the fastest in all of the robust configurations, it takes too many clock cycles to produce a result with all the necessary instructions. Optimal area and speed performances have been achieved in Configuration 2, which also provides the best protection.

[1]The tool chain does not provide a reliable estimate for critical path delay in ASIC realization. For this, FPGA realizations provide a more realistic estimates

6. CONCLUSION AND FUTURE WORK

We designed and implemented a robust execution unit in an embedded processor that allows efficient and secure execution of cryptographic algorithms. We estimated the area overhead of the robust zone for the ASIC implementation. We also provided area usage on an FPGA device after placement-and-routing for the full design including an embedded base processor and robust execution unit. Since the number and organization of the subsystems in the robust execution unit are carefully designed, we observed only a moderate area overhead while no deterioration in maximum applicable clock frequency is reported in ASIC. Although FPGA realizations incur some penalty in maximum allowable clock frequency, overall execution time is not affected in some robust configurations. Robust execution units also provide faster execution of modular multiplication which is the most time-consuming operations in many cryptographic applications. Our results show that it is possible to incorporate a powerful robust error detection circuitry into an embedded processor core transparently with an acceptable cost in area. The proposed error detection capability benefits many cryptographic applications that uses basic arithmetic operations.

Acknowledgment

This work is supported by the Scientific and Technological Research Council of Turkey (TUBITAK) under project number 105E089 (TUBITAK Career Award).

Table 2: Speed and Area Information for ASIC Implementation

Configurations	CPU Speed	Base CPU Area	TIE Area	Total Area
Configuration 0	$350MHz$	66,000	0	66,000
Configuration 1	$350MHz$	66,000	33,071	99,071
Configuration 2	$350MHz$	66,000	36,604	102.604
Configuration 3	$350MHz$	66,000	22,564	88,564

Table 3: Speed and Gate Information for FPGA Implementation

Configurations	Max Period	Slice	Lut	RAM16b	DSP48s
Configuration 0	$21.206ns$	6.949	18,176	80	0
Configuration 1	$37.189ns$	9,468	26,452	80	6
Configuration 2	$31.011ns$	9,494	27,492	80	6
Configuration 3	$29.674ns$	7,741	22,775	80	1

7. REFERENCES

[1] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. Fault attacks on rsa with crt: Concrete results and practical countermeasures. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 260–275, 2002.

[2] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. *IEEE Transactions on Computers*, 52(4):492–505, 2003.

[3] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO 97*, volume 1294 of *LNCS*, pages 513–525, 1997.

[4] Dan Boneh, Richard Allan DeMillo, and Richard Jay Lipton. On the importance of eliminating errors in cryptographic computations. *Journal of Cryptology*, 14(2):101–119, 2001.

[5] Çetin Kaya Koç, Tolga Acar, and Burton S. Kaliski Jr. Analyzing and comparing montgomery multiplication algorithms. *IEEE MICRO*, 16(3):26–33, 1996.

[6] Gunnar Gaubatz, Berk Sunar, and Mark G. Karpovsky. Non-linear residue codes for robust public-key arithmetic. In *FDTC*, pages 173–184, 2006.

[7] Hagai Bar-El Hamid, Hamid Choukri, David Naccache Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, 2006.

[8] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In *DSN'04*, pages 93–101, 2004.

[9] Mark G. Karpovsky and Alexander Taubin. New class of nonlinear systematic error detecting codes. *IEEE Transactions on Information Theory*, 50(8):1818–1820, 2004.

[10] Ramesh Karri, Grigori Kuznetsov, and Michael Goessel. Parity-based concurrent error detection of substitution permutation network block ciphers. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 113–124, 2003.

[11] Ramesh Karri, Kaijie Wu, Piyush Mishra, and Yongkook Kim. Concurrent error detection of fault based side channel cryptanalysis of 128-bit symmetric block ciphers. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.

[12] Alexander Taubin Konrad J. Kulikowski, Mark G. Karpovsky. Robust codes for fault attack resistant cryptographic hardware. In *FDTC*, pages 1–12, 2005.

[13] Jean-Jacques Quisquater and Gilles Piret. A differential fault attack technique against SPN structures, with application to the AES and KHAZAD. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 77–88, 2003.

[14] Arash Reyhani-Masoleh and M. Anwar Hasan. Error detection in polynomial basis multipliers over binary extension fields. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 515–528, 2002.

[15] Arash Reyhani-Masoleh and M. Anwar Hasan. Towards fault-tolerant cryptographic computations over finite fields. *ACM Transactions on Embedded Computing Systems*, 3(3):593–613, 2004.

[16] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In B. S. Kaliski, Ç. K. Koç, and C. Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 2–12, 2003.

[17] Tensilica. Xtensa LX2 Embedded Processor Core. Website. `http://www.tensilica.com/products/xtensa_LX2.htm`.