A TTA-like Processor for Fast RSA Key Generation Using RNS

Wei Guo, Jingwei Hu, Jizeng Wei
School of Computer Science and Technology
Tianjin University, Tianjin, China 300072
Email: { weiguo, jingweihu, weijizeng}@tju.edu.cn

***Abstract*— RSA key generation is of great concern for implementation of RSA cryptosystem on embedded system due to its long processing latency. In this paper, a novel architecture is presented to provide high processing speed to RSA key generation for embedded platform with limited processing capacity. In order to exploit more data level parallelism, Residue Number System (RNS) is introduced to accelerate RSA key pair generation, in which these independent elements can be processed simultaneously. A cipher processor based on Transport Triggered Architecture (TTA) is proposed to realized the parallelism at the architecture level. In the meantime, division is avoided in the proposed architecture, which reduces the expense of hardware implementation remarkably. The proposed design is implemented by Verilog HDL and synthesized in a 0.18μm CMOS process. A rate of 3 pairs per second can be achieved for 1024-bit RSA key generation at the frequency of 100 MHz.**

***Index Terms*— RSA, key generation, RNS, Montgomery multiplication, TTA-like processor**

I. INTRODUCTION

Public key cryptography has gained extreme popularity in many applications such as smart cards, digital certificate and so on. One of the most widely used public key algorithm is RSA [1]. However, the process of RSA key generation is very complex and time-consuming. Some implementations [2] try to generate the RSA key pairs on a desktop and upload the pair, or only the private key, into a smart card. Take communication security and high-performance processing into consideration, the entire procedure of RSA key generation is preferably performed totally inside cipher chip in order to guarantee the efficiency and the security of the applications. Nevertheless, the limited computational power of embedded systems can not afford high speed RSA key generation. In this paper, the issue about how to provide high processing speed to RSA key generation for embedded platform with limited processing capacity is discussed.

An on-card implementation is presented in [3] which takes up to 6.82 seconds to create a key pair. A scalable hardware architecture is proposed in [4], they have presented the architecture applied for the multiple word Radix-2 Montgomery multiplication algorithm and a processing element (PE) is designed. A new algorithm [5] which correctly identifies every positive integer tested as being either prime or composite is considered. [6] rather shows a simple way to substantially reduce the value of hidden constants to provide much more efficient prime generation algorithms. They apply our techniques to various contexts (DSA primes, safe primes, ANSI X9.31-compliant primes, strong primes, etc.) and show how to build fast implementations on appropriately equipped smart-cards, thus allowing on-board key generation. A very efficient recursive algorithm [7] for generating nearly random provable primes is presented. The expected time for generating a prime is only slightly greater than the expected time required for generating a pseudo-prime of the same size that passes the Miller-Rabin test for only one base. possibilities of realization of Miller- Rabin big number primality test on assembler of Texas Instruments digital signal processors of TMS320C54x family are considered [8]. Applying these modules, it could be achieved considerably higher level of the system security regarding to the software-only security systems. [9] moves on to a very simple new deterministic test, then they discuss various ways of constructing so-called strong primes. However, these new algorithms generally focus on the improvement of the computational complexity and how to apply them on chip remains to be solved.

We propose an efficient solution to accelerate RSA key pair generation from both data and instruction level parallelism, in which RNS and TTA are combined closely. The advantage of RNS Montgomery multiplication algorithm [10] is that large number multiplications can be divided into small elements and each independent elements can be processed simultaneously. The advantages of Transport Triggered Architecture (TTA) [11] is that it can be used as an application specific processor, especially as a coprocessor for different DSP applications in SoC. Comparing with the traditional ASIC, TTA is more flexible in application and consumes less silicon area. And comparing with traditional DSP processor, it has more efficiency and better performance. In this paper, function unit (FU) "MMAC" is elaborately designed, which meets the highly parallelism of RNS Montgomery multiplication and reduces redundant data transmission. So an optimized architecture, called TTA-like, is proposed to meet the desire for high-efficient performance. In order to reduce the expense of hardware implementation,

This work is supported by the Key Project Foundation of Fundamental and Frontier Technology Research of Tianjin under Grant No.11JCZDJC1580 and the Special Financing Project of Internet of Things by the National Development and Reform Commission.

doi:10.4304/jcp.8.1.33-40

division is discarded in the process due to an appropriate selection of algorithms. This improvement makes it possible to fulfil RSA key pair generation with pure hardware design, which is quite different from prior methods that use software programming to do the division.

The rest of paper is organized as follows. Section 2 describes the RSA key pair generation and details how to reduce timings related to prime finding algorithm which is the kernel of the whole process. Section 3 presents our architecture for it. The implementation results in this paper are compared with what is given in [3][4] in this section. Finally, in section 4, the conclusions and future work are discussed.

II. Design Flow and Algorithm Analysis

Common RSA key pair generator generally include a stage of trial division in the sieve function procedure [2] [3]. We investigate in this section a way of how to avoid this by utilizing invertible numbers which is quite suitable for the hardware implementation. Some algebraic techniques are introduced to speed up modular exponentiation [10] [12], so the primality test, which is the most time-consuming section practically, can be optimized greatly to achieve a satisfactory implementation; And the private key generation algorithm [13] is improved as well for the sake of limited computation resources in our embedded processor. Figure 1 gives out a common procedure of the RSA key pair generation. First, a random number [14] is generated and then it is thrown into the "sieve function" procedure. After that we get so-called "prime candidate" which is not a prime for sure but very likely [14][16]. The primality of these candidates are confirmed via primality test(here, Miller-Rabin test is adopted). The rest of the procedure is simple: public key and private key is generated in successive order.

A. Sieve Function

The purpose of the sieve function is to reduce the times of the primality test which is the most time-consuming part of RSA key pair generation. Unlike ordinary sieve functions which use small primes to divide the candidate number, trial divisions is replaced by one-time modular exponentiation [6] as shown in Algorithm 1.This algorithm starts with the following consideration [7]:

- construct $= \prod_i p_i (p_i \neq 2$ and p_i is a small prime).
- Then we can calculate $k (k <$) to make k is co-prime with .
- A large integer L (usually $L = \nu$, ν is a small integer).
- $q = k + L$, which q is co-prime with definitely or in another word, q is also co-prime with small primes p_i. So the function of sieve function is reached.

Here, in order to explain details of our algorithm , two questions are presented to help:

1) How to find such k which is co-prime with ?
2) How to get the large integer L to ensure that $q = k + L$ is co-prime with which is equivalent to the question: how to make sure that q is co-prime with small prime p_i ?

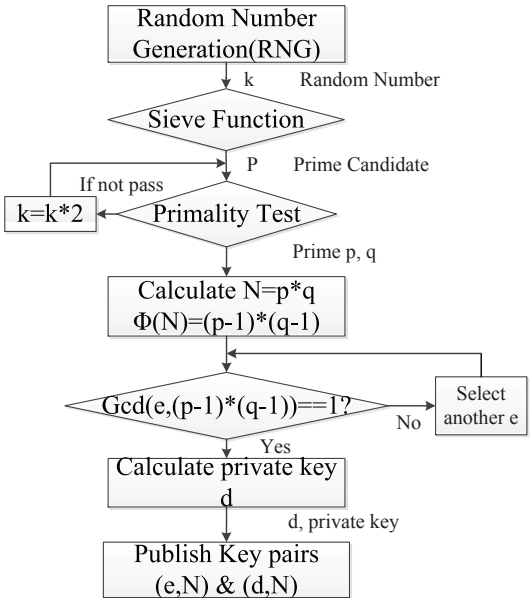

Figure 1. A common flow of RSA key pair generation

For question 1, we introduce a concept in number theory: co-prime congruences mod n form a multiplication group.

For example,

mod 4 has 2 co-prime congruences: 1 and 3;

mod 8 has 4 co-prime congruences: 1, 3, 5 and 7

mod 16 has 6 co-prime congruences: 1, 3, 5, 7, 9, 11, 13 and 15

For the convenience of narrative, co-prime congruences mod is denoted as Z_{Π}^{*} and the greatest common divisor is denoted as gcd.

So,**theorem 2.1.1.**

$$\forall k \in Z_{\Pi}^{*}, \text{iff } gcd(k, \ ) = 1$$

It means we just have to find $k \in Z_{\Pi}^{*}$, and here comes a easier way to get it called "Carmichael Theorem":

we denote Carmichael function as $\lambda(n)$ and $\forall a \in Z, a^{\lambda(n)} = 1 (\text{mod } n)$

So,**theorem 2.1.2.**

$$\forall k \in Z_{\Pi}^{*}, k^{\lambda(\Pi)} = 1 \ (\text{mod } n)$$

Take notice that [13],

$\lambda(\ ) = lcm(\prod p_i) = lcm(\lambda(p_1), \lambda(p_2), ..., \lambda(p_t))$,

($\lambda(p) = p - 1$,p is a prime; least common multiplier is denoted as lcm)

figure 2 provides a method to generate such k.

For question 2,L should be set as an integral multiple of

Precomputations: $\Pi = \prod_{i=1}^{k} p_i^{\delta_i}$ and $\lambda(\Pi)$
Output: an invertible number c modulo Π
1. pick an n-bit random number $c < \Pi$
2. while $c^{\lambda(\Pi)} \bmod \Pi \neq 1$ do $c \leftarrow c + 1$
3. output c

Figure 2. Generator g' of Invertible Number of modulo Π

Derivation:
we have $gcd(a, b)$=$gcd(b, a \bmod b)(a > b$,$a \bmod b \neq 0)$;
so, $gcd(k+L, \quad)$=$gcd(k+m \quad , \quad)$=$gcd(k, \quad)$=1 which means that such L meets the requirement [15].

According to the architecture presented in this paper, modular exponentiation can be done very fast and no extra hardware consuming is needed. So the processing time of the sieve function can be trivial and circuit area will be saved as well.

Algorithm 1 Invertible Number Generation

Input: randomly selected $k \quad = \prod_{i=1}^{74} p_i, \lambda(\ ) = lcm(p_1 - 1, p_2 - 1, ..., p_{74} - 1), p_i = 3, 5, 7, ..., 379$
$L = 8 \times \quad , M = 3 \times$
Output: q co-prime with the smallest 74 odd primes
if $k^{\lambda(\Pi)} \bmod \quad = 1$ **then**
  $q = k + L$
else
  $k = k + 1$, go to 1
end if
if q is even **then**
  $q = q + M$
end if

B. Primality Test

The candidate must be tested for primality in order to be useful for the generation of a RSA key pair [12][16]. In this section, Miller-Rabin's method [10][13] is used for our primality test. The algorithm is described in Algorithm 2 below.

We consider another property of arithmetic modulo p for a prime number p that can be used as a certificate for compositeness [14][17].

Definition 2.2.1.
Let $1 \leq a < n$. Then a is called a **square root of** 1 modulo n if a^2 mod n = 1.

In the situation of this definition, the numbers 1 and n-1 are always square roots of 1 modulo n (indeed, $(n-1)^2 \equiv (-1)^2 \equiv 1 \pmod{n}$); they are called the trivial square roots of 1 modulo n. If n is a prime number, there are no other square roots of 1 modulo n.

Algorithm 2 Miller-Rabin Test

Input: Odd integer $n \geq 3$
Output: Confirm the primality of n
Find u and k so that $n - 1 = u * 2^k$
Let a be randomly chosen from $2, 3, ..., n - 1$
$b \leftarrow a^u \bmod n$
if $b \in 1, n - 1$ **then**
  print "n is prime"
end if
for $i = 1$ **to** $k - 1$ **do**
  $b \leftarrow b^2 \bmod n$
  if $b = n - 1$ **then**
    print "n is prime"
  end if
  if $b = 1$ **then**
    print "n is composite"
  end if
end for
print "n is composite"

a	$b_0 = a^{81}$	$b_1 = a^{162}$	$b_2 = a^{324}$
2	252	129	66
7	307	324	1
32	57	324	1
49	324	1	1
65	0	0	0
126	1	1	1
201	226	51	1
224	274	1	1

Figure 3. Powers a^{n-1} mod n calculated with intermediate steps, n = 325

Lemma 2.2.1.
If p is a prime number and $1 \leq a < p$ and a^2 mod p = 1, then a = 1 or $a = p - 1$.
Proof. We have $(a^2 - 1)$ mod p = $(a + 1)(a - 1)$ mod p = 0; that means, p divides $(a + 1)(a - 1)$. Since p is prime, p divides $a + 1$ or $a - 1$, hence a = $p - 1$ or a = 1

Thus, if we find some nontrivial square root of 1 modulo n, then n is certainly composite. Figure 3 gives us an example to show this principle.
In figure 3, calculating 201^{324} mod 325 with two

b_0	b_1	$\cdots$				$\cdots$	b_{k-1}	b_k	Case
1	1	$\cdots$	1	1	1	$\cdots$	1	1	1a
$n-1$	1	$\cdots$	1	1	1	$\cdots$	1	1	1b
$*$	$*$	$\cdots$	$*$	$n-1$	1	$\cdots$	1	1	1b
$*$	$*$	$\cdots$	$*$	$*$	$*$	$\cdots$	$*$	$n-1$	2
$*$	$*$	$\cdots$	$*$	$*$	$*$	$\cdots$	$*$	$*$	2
$*$	$*$	$\cdots$	$*$	1	1	$\cdots$	1	1	3
$*$	$*$	$\cdots$	$*$	$*$	$*$	$\cdots$	$*$	1	3

Figure 4. Powers a^{n-1} mod n calculated with intermediate steps, possible cases

intermediate steps leads us to detect that 51 is a nontrivial square root of 1, which proves that 325 is not prime.

What can the sequence $b_0, ..., b_k$ look like in general? We first note that if b_i = 1 or $b_i = n-1$, then the remaining elements $b_{i+1}, ..., b_k$ must all equal 1, since $1^2 = 1$ and $(n-1)^2$ mod n = 1. Thus in general the sequence starts with zero or more elements $\notin 1, n-1$, and ends with a sequence of zero or more 1s. The two parts may or may not be separated by an entry $n-1$. All possible patterns are depicted in Figure 4, where "*" represents an arbitrary element $\notin 1, n-1$. We distinguish four cases:

- Case 1_a: b_0 = 1.
- Case 1_b: $b_0 \neq 1$, and there is some $i \leq k-1$ such that $b_i = n-1$.
  ł In Cases 1_a and 1_b we certainly have that b_k = 1; no information about n being prime or not is gained.
- Case 2: $b_k \neq 1$. ł Then n is composite.
- Case 3: $b_0 \neq 1$, but b_k = 1, and $n-1$ does not occur in the sequence $b_0, ..., b_{k-1}$. ł Consider the minimal $i \geq 1$ with b_i = 1. By the assumption, $b_{i-1} \notin 1, n-1$, hence b_{i-1} is a nontrivial square root of 1 modulo n. Thus n is composite in this case.

Definition 2.2.2.

Let $n \geq 3$ be odd, and write $n - 1 = \mu \cdot 2^k$, μ odd, $k \geq 1$. A number a, $1 \leq a < n$, is called an **A-witness** for n if a^μ mod $n \neq 1$ and $a^{\mu \cdot 2^i}$ mod $n = n-1$ for all i, $0 \leq i < k$. If n is composite and a is not an A-witness for n, then a is called an **A-liar for** n.

Lemma 2.2.2.

If a is an A-witness for n, then n is composite.

Proof. If a is an A-witness for n, then to the sequence$b_i = a^u \cdot 2^i$ mod n, $0 \leq i \leq k$, Case 2 or Case 3 of the preceding discussion applies, hence n is composite.

Finally, We combine this observation with the idea of choosing some a from $2, ..., n-2$ at random into a strengthening of the Fermat test, called the Miller-Rabin test.The detail of this algorithm has been described in Algorithm 2.

Since Miller-Rabin test is dominant in the processing time of RSA key pair generation, it is important to improve this part for the sake of high performance. Take notice of $b \leftarrow a^u$mod n which is the most timing-cost computation in Algorithm 2, we focus on this part and make use of RNS Montgomery multiplication to accelerate Miller-Rabin test. The technical details shall be discussed in section 2.4.

C. Calculation of Private Key

The private key is the modular inverse of the public key as described in algorithm 3 [13].

Algorithm 3 Improved Stein's method for modular inverse

Input: e(public key), (n)($(n) = (p-1)\times(q-1)$,p, q are two primes)
Output: $d = e^{-1}$ mod (n)
$(X_1, X_2, X_3) = (1, 0, e)$, $(Y_1, Y_2, Y_3) = (0, 1, (n))$
while $X_3 \neq 1$ **or** $Y_3 \neq 1$ **do**
  if X_3 is even **then**
    if X_1, X_2 are even **then**
      $(X_1, X_2, X_3) = (X_1/2, X_2/2, X_3/2)$
    else
      $X_1 = X_1 + (n)$,$X_2 = X_2 - e$
    end if
  else if Y_3 is even **then**
    if Y_1, Y_2 are even **then**
      $(Y_1, Y_2, Y_3) = (Y_1/2, Y_2/2, Y_3/2)$
    else
      $Y_1 = Y_1 + (n)$,$Y_2 = Y_2 - e$
    end if
  else if $X_3 > Y_3$ **then**
    $(X_1, X_2, X_3) = (X_1 - Y_1, X_2 - Y_2, X_3 - Y_3)$
  else
    $(Y_1, Y_2, Y_3) = (Y_1 - X_1, Y_2 - X_2, Y_3 - X_3)$
  end if
  if $X_3 = 1$ **then**
    return $X_1 = e^{-1}$ mod (n)
  else
    return $Y_1 = e^{-1}$ mod (n)
  end if
end while

The particular method chosen for computing modular inverse avoids trial division to make it easier to be implemented in hardware.

D. RNS Montgomery Modular Multiplication

Modular multiplication is the kernel operation of RSA key pair generation which is called in quantity and takes up the most time of the whole procedure, it is of great

importance to analyze this part and bring out the optimal algorithm for the hardware we present in this paper.

The RNS Montgomery modular multiplication algorithm proposed in [10] is a fast parallel algorithm for modular multiplication with large operands, which is the basic operation of public key cryptosystems. The algorithm is rewritten in algorithm 4.

Algorithm 4 RNS Montgomery Modular Multiplication

Input: $|X|_{a\cup b}, |Y|_{a\cup b}, (X, Y < 2N)$
Output: $[r]_{a\cup b}, (r = XYA^{-1})\text{mod}N, r < 2N$
for i=1 **to** k **do**
  step1: $z_i = (x_i \times y_i) \text{ mod} a_i$
  step2: $q_i = (z_i \times |-N^{-1}|_{a_i})\text{mod}a_i$
end for
step3: $q_j = BT(q_i, 0)$
for j=1 **to** k **do**
  step4: $z_j = (x_j \times y_j) \text{ mod } b_j$
  step5: $w_j = (z_j + q_j \times N_j) \text{ mod } b_j$
  step6: $r_j = (w_i \times |A_i^{-1}|_{b_j})\ (i = j)$
end for
step7: $r_i = BT(r_j, 0x80000000)$

Two bases a and b are introduced, and subscript i and j are used to indicate the elements related to base a and b respectively. For example, a_i represents the ith element of base a, and x_j represents the jth element of X which is represented by base b. In this paper, $[X]_{a\cup b}$ means X is represented in RNS by base a and b, and $|A_i^{-1}|_{b_j}$ represents the inverse of A_i modulo b_j. This algorithm chooses $A = \prod_{i=1}^{k} a_i$ as Montgomery constant. In Algorithm 4, step3 and step7 are base transformation (BT) between different base representation, which is shown in Algorithm 5. In this case, step3 transforms q_i which is represented by base a to base b representation into q_j. Derived from base transformation from [10], the BT algorithm is reformulated to satisfy architecture designed in this paper. $|X|_{m_i}$ is used to represent X modulo m_i in the following algorithm.

Algorithm 5 Base Extension $q_j = BT(q_i, \alpha)$

Input: $[q]_a$
Output: $[q]_b, q_r$
for i=1 **to** k **do**
  $l_i = q_i \times |A_i^{-1}|_{m_i}$
end for
$w = (\alpha + \sum_{i=1}^{k} l_i) >> 32$
for j=1 **to** k **do**
  $q_j = |\sum_{n=1}^{k} |A_n|_{a_i} \times l_i|_{m_i}$
end for
return $[q]_b$

Another concern about RNS Montgomery Multiplication is about the proper selection of RNS base. In this paper, an efficient RNS base is chosen in the form of $2^n - c_i$, where n is the length of RNS base which decides the data width of hardware implementation. In this form, modular addition and modular multiplication are efficiently realized, which are shown in Algorithm 6 and Algorithm 7. Because of the decrease of the complexity of multiplication, the cost for mod operation is considerably reduced in the design.

Algorithm 6 Modular Addition Algorithm addmod(a, b, m_i)

Input: $a, b, m_i, 0 < a, b < m, m_i = 2^n - c_i$ and $c_i < 2^t, and 1 < t < (n-1)/2$
Output: $r = a + b \text{ mod } m_i$
step1: $P = a + b$
step2: $Q = P + c_i$
step3:
if $Q \geq 2^n$ **then**
  $r \leftarrow Q\text{mod } 2^n$
else
  $r \leftarrow P$
end if

Algorithm 7 Modular Addition Algorithm mulmod(a, b, m_i)

Input: $a, b, m_i, 0 < a, b < m, m_i = 2^n - c_i$ and $c_i < 2^t, and 1 < t < (n-1)/2$
Output: $r = a \times b \text{ mod } m_i$
step1: $y = a \times b$
step2: $y_1 = y \div 2^n$; $y_0 = y \text{ mod } 2^n$
step3: $y' = c_i y_1 + y_0$
step4: $y'_1 = y' \div 2^n$; $y'_0 = y' \text{ mod } 2^n$
step5: $y'' = ciy'_1 + y'_0$
step6: $y''' = y'' + c_i$
step7:
if $y''' \geq 2^n$ **then**
  $r \leftarrow y''' \text{ mod } 2^n$
else
  $r \leftarrow y''$
end if

III. Implementation For RSA Key Pair Generation

In this section, a TTA-like architecture is presented to illustrate the fast RSA key pair generation in our way. The implementation results are compared with [3] [4] in section 3.

A. Transport Triggered Architecture

Transport Triggered Architecture (TTA) [11] is statically programmed ILP modular architecture which is similar to VLIW architecture. Instead of specifying operation typing and controlling the FUs directly, TTAs specify the required data transports. These transports may trigger operations as side effect implicitly. TTA modular template is illustrated in Figure 5. TTA is organized as a set of functional units (FUs) and register files (RFs) including general- purpose registers. The input and output ports of these resources are con- nected together with an interconnection network composed of move buses and input/output sockets. The data transports encoded in each instruction slot are carried out on each move bus, the number of which thus determines the maximum parallel data moves that can be performed in each cycle. Input sockets contain multiplexers which feed data from the buses into the desti- nation registers in FUs. Output sockets contain de-multiplexers that put FU results from source registers on the buses. The connectivity between RFs and FUs is more complex in VLIW architecture as the number of input/output ports increase, leading to larger area and critical path delay overhead. However, interconnections of TTAs are simpler. The interconnection network of TTAs can be fully connected, in which case all the ports in all the FUs are connected to all the buses. But usually they are partially connected and optimized for specific application according to the traffic on the move buses in practice. The functional units follow the triggering strategy. Each FU triggers the operation when a certain operand is transported to the triggered operand register in the FU, Therefore each FU must hold one trigger register at least to perform the corresponding operations. Additional operand registers may be required for multi-operands operations. The FUs can hold more than one result registers for occasional multiple results requirement. Every FU can be pipelined to reduce the critical path delay and improve the processing throughput.

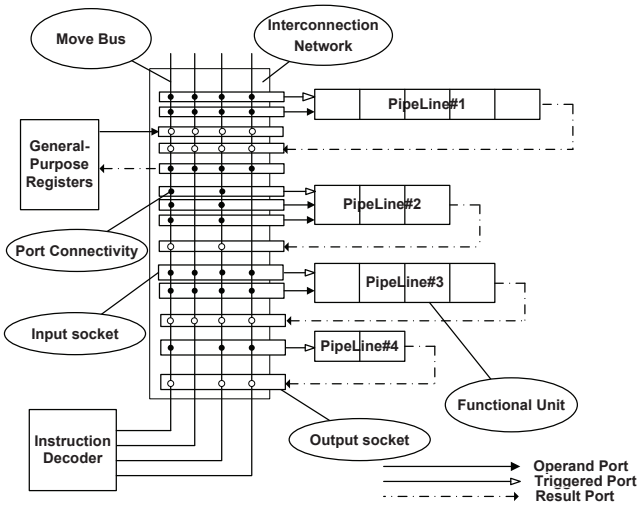

Figure 5. Template of Transport Triggered Architecture

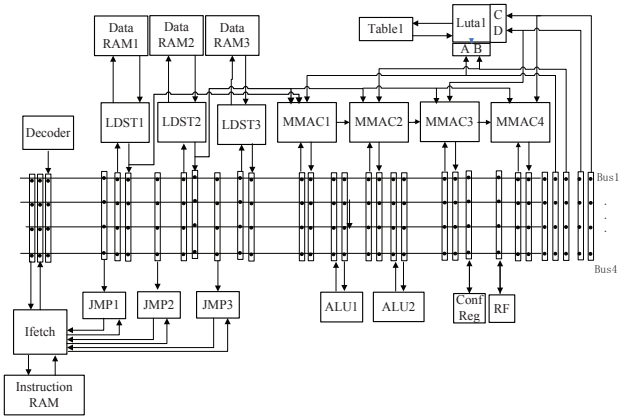

Figure 6. The proposed architecture in this design

B. Architecture Design

The proposed architecture in this design is shown in Figure 6. It mainly consists of five parts: FUs, RFs, control logic, transport network and on-chip RAM/ROM. Similar to common processors, the control logic composes of instruction fetch, instruction decoder and PC control units. In this design, JMP unit can affect the PC value to realize jump, branch and loop operations.

The FUs are the key factors which decide the performance of this processor. According to different applications, various of FUs can be designed and attached to the transport network. To implement RNS Montgomery multiplication algorithm, modular multiplication and modular multiplication-and-accumulation are the key operations in n-bit level, where n is the base size. So, the MMAC units are designed to accelerate the execution speed of these key operations. MMAC units can only do 32-bit$\times$ 32-bit modular multiplication, But combing them together with RNS Montgomery multiplication algorithm, long bit modular multiplication can be done very fast. In this work, four MMAC units are designed according to the maximum number of function units used in the processing of executing the operations referring to step 1 step 3 in algorithm 4; The remaining operations after TIME E also depend on these four MMAC units in order to increase the reusability of the architecture and save the valuable hardware resources.

Two ALU units are included to implement the modular addition and modular subtraction operations. And some controlling operations are needed such as logical right-shift, arithmetic right-shift and case-select. They are included in ALU units to help finish the extra operation in the process of RSA key pair generation as well.

There are three Load-store units (LDST), one Look-up Table unit (LUT). LDSTs are connected with the independent Data RAM. LUT are used to store the pre-computed data. There are direct data paths between LDSTs and MMAC units and also between LUT and the group of MMAC, which are used to reduce redundant data transmission.

The transport network is used to transport data from

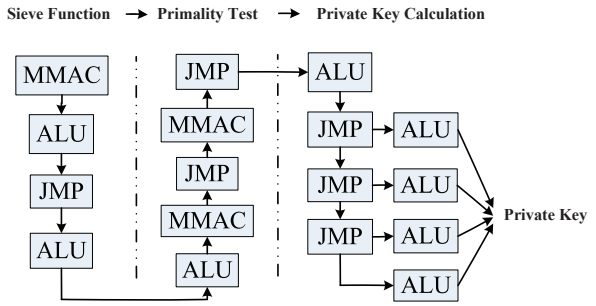

Figure 7. The operational process of RSA key pair generation from function unit level

source register to destination register. Four buses are adopted in the transport network to fully exploit the computation capacity of four MMAC units in parallel. The width of each bus is 32-bit which is decided by the selected base size.

Figure 7 illustrates the operational process of RSA key pair generation from function unit level. The whole process has three stages:

1) Sieve Function
2) Primality Test
3) Private Key Calculation

The arrows in this figure show the data flow of the procedure: In the Sieve Function, MMAC is used to do modular exponentiation specially, ALU helps to do some ordinary arithmetic operations such as mod-add or add, JMP then controls the programming mode; Similar to Sieve Function, we use the same units to fulfil calculation (mainly modular exponentiation) in Primality Test procedure; For the last stage, It is simple to generate private key with ALU while jump instructions occupy a large proportion of it, so JMP is set up here to solve this problem.

According to the algorithm analysis in section 2, these function units are arranged to fulfil the RSA key pair generation: MMAC units do the modular exponentiation; ALU units complete the operation of addition and substraction and some other logical operations; And JMP units serve as judge and jump instruction.

C. Implementation Result

The design was implemented by Verilog HDL and synthesizes with 0.18 μm CMOS technology. At 100 MHz, the processing time of 1024-bit RSA key pair generation is 306 ms in average, and the logic area is 131k gates. To compare the performance of our implementation with other works [3] [4], the processing time consumed in primality test, prime finding and RSA key pair generation is analyzed. Because of the architecture specified for modular multiplication, the consuming time in primality test is reduced greatly and numbers of times in prime finding are 36 which is a satisfactory compromise in both circuit area and processing time. As shown in Table 1, the proposed work requires less clock cycles than the other works.

TABLE I.
RSA KEY PAIR GENERATION COMPARISON

	Ref [3]	Ref [4]	This work
Frequency (MHz)	32	10.52	100
Primality test (ms)	84.2	25.29	4.1
prime finding (ms)	3100	/	151
RSA key pair generation (ms)	6820	/	306

IV. Conclusion and Future Work

This paper presents a novel RSA key pair generation hardware implementation based on TTA, and Montgomery modular multiplication based on RNS is adopted in both sieve function and primality test to improve the performance significantly. FUs suiting the algorithms is designed on TTA, and direct data paths are used to reduce redundant data transmission. Above all, pipeline and parallel technology to improve the computing speed are introduced. At the frequency of 100 MHz, 1024-bit RSA key pair generation needs 306 ms in average, the logic area of the proposed architecture consists of 131k gates. This result shows that our proposed work can achieve high performance and small area for RSA key pair generation.

On-going and future developments include: (1) Preparation for some pre-computed data especially in RNS Montgomery multiplication can be optimized which affect the rate of RSA key pair generation significantly. (2) The concept of scalable and reconfigurable architecture is introduced, in which not only 1024-bit RSA key pair but also 2048, 4096-bit can be implemented in this platform.

V. Acknowledgment

This work is supported by the Key Project Foundation of Fundamental and Frontier Technology Research of Tianjin under Grant No.11JCZDJC1580 and the Special Financing Project of Internet of Things by the National Development and Reform Commission.

References

[1] R. L. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," Communications of the ACM, vol. 21, pp. 120-126, 1978

[2] Chenghuai Lu, Andre L. M. dos Santos, Francisco R. Pimentel, "Implementation of Fast RSA Key Generation on Smart Cards," Proceedings of the ACM Symposium on Applied Computing, pp. 214-220, 2002

[3] Bahadori M, Mali M. R, Sarbishei O, Atarodi M, Sharifkhani M, "A Novel Approach for Secure and Fast Generation of RSA Public and Private Keys on SmartCard," 2010 8th IEEE International NEWCAS Conference (NEWCAS 2010), pp. 265-268, 2010

[4] Cheung Ray C. C, Brown Ashley, Luk Wayne, Cheung Peter Y K, "A Scalable Hardware Architecture for Prime Number Validation," Proceedings - 2004 IEEE International Conference on Field-Programmable Technology, FPT '04, pp. 177-184, 2004

[5] Da Silva, Joao Carlos Leandro, "Carmichael numbers and a new primality test," IEEE 26th Convention of Electrical and Electronics Engineers in Israel, pp. 631-633, 2010

[6] Joye M, Paillier P, Vaudenay S, "Efficient generation of prime numbers," Cryptographic Hardware and Embedded Systems -CHES 2000. Second International Workshop. Proceedings (Lecture Notes in Computer Science Vol. 1965), pp. 340-54, 2000

[7] Maurer U. M, "Fast generation of prime numbers and secure public-key cryptographic parameters," Journal of Cryptology, pp. 123-55, 1995

[8] Bordevic G, "On optimization of Miller-Rabin primality test on TI TMS320C54x signal processors," 14th International Workship in Systems, Signals and Image Processing and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services - EC-SIPMCS 2007, pp. 229-32, 2007

[9] Landrock P, "Primality tests and use of primes in public key systems," Lectures on Data Security. Modern Cryptology in Theory and Practice, pp. 127-33, 1999

[10] Laurent Imbert, Jean-Claude Bajard, "A Full RNS Implementation of RSA," IEEE Transactions on Computers, vol. 53, 2004, pp. 769-774.

[11] Wei Guo, Jizeng Wei, Yongbin Yao et al, "Design of a configurable and extensible Tcore processor based on Transport Triggered Architecture," World Congress on Computer Science and Information Engineering, pp. 536-540, 2009

[12] Hanae Nozaki, Masahiko Motoyama, Atsushi Shimbo et al, "Implementation of RSA Algorithm Based on RNS Montgomery Multiplication," Cryptographic Hardware and Embedded Systems(CHES), pp. 364-376, 2001

[13] Kenneth R, Castleman, "Digital image processing[M]," New Jersey:Prentice-Hall, 1996

[14] Igumnov V.S, "Generation of the large random prime numbers," J International Siberian Workshop on Electron Devices and Materials. Proceedings. 5th Annual (IEEE Cat. No.04EX813), pp. 117-18, 2004

[15] Adleman L. M, "On distinguishing prime numbers from composite numbers," 21st Annual Symposium on Foundations of Computer Science, pp. 387-406, 1980

[16] Yee L. P, "Digital signatures and public key cryptosystems with multilayer perceptrons," CONIP '02. Proceedings of the 9th International Conference on Neural Information Processing. Computational Intelligence for the E-Age (IEEE Cat. No.02EX575), pp. 2308-11 vol.5, 2002

[17] Duran Diaz R, "Safe primes density and cryptographic applications," Proceedings IEEE 33rd Annual 1999 International Carnahan Conference on Security Technology (Cat. No.99CH36303), pp. 363-7, 1999

Wei Guo received the M.S. degree from Louisiana State University in 1991. After then, she joined Motorola Semiconductor Ltd. and worked in integrated circuit design for 12 years. Now she is a professor and director of VLSI Research Lab at School of Computer Science and Technology, Tianjin University, Tianjin, China. Her research interests are SoC design technology, computer architecture, and cryptographic processor.

Jingwei Hu received the B.E. degree in Electronic Engineering from Maritime University, Dalian, China, in 2011. Now he is a master candidate at School of Computer Science and Technology, Tianjin University. His research interests are applied cryptography and cryptographic processor.

Jizeng Wei received the B.E. degree in computer science from Harbin Institute of Technology, Harbin, China, in 2004, the M.S. and the Ph.D. degree in computer science and technology from Tianjin University, Tianjin, China, in 2007 and in 2010, respectively. Now he is an assistant professor at School of Computer Science and Technology, Tianjin University. His research interests include application-specific instruction-set processor (ASIP), cryptographic processor, and electronic system level (ESL) modeling.