An 830mW, 586kbps 1024-bit RSA Chip Design

*Chingwei Yeh**
Nat'l Chung-Cheng University
ieecwy@ccu.edu.tw

En-Feng Hsu
Nat'l Chung-Cheng University
Charle28@vlsi.ee.ccu.edu.tw

Kai-Wen Cheng
Nat'l Chung-Cheng University
91kevin@vlsi.ee.ccu.edu.tw

Jinn-Shyan Wang
Nat'l Chung-Cheng University
ieegsw@ccu.edu.tw

Nai-Jen Chang
Nat'l Chung-Cheng University
92asura@vlsi.ee.ccu.edu.tw

Abstract

This paper presents an RSA hardware design that simultaneously achieves high-performance and low-power. A bit-oriented, split modular multiplication algorithm and architecture are proposed to fully exert the radix-4 computational capability. Further, we identify the switching profile of RSA data and accordingly propose power-optimized designs for the storage elements and key computational components. The complete RSA modular exponentiation hardware has been implemented using cell-based 0.18um CMOS technology. Post-layout simulation shows that the design delivers an average performance of 586kbps at 460MHz, 1.8V while consuming only 830mW.

1. Introduction

With the ever increasing popularity of networked computing devices, it is widely recognized that security will become a major concern. In 1976, Rivest et al. proposed the RSA public-key cryptography [1, 2]. Since then, RSA has gained increasing popularity and is now the public-key cryptosystem that receives the widest deployment in real applications.

The kernel operation for RSA is modular multiplication. A pioneering work for modular multiplication was attributed to P.L. Montgomery [3], who computed a modular multiplication of two n-bit numbers via n-iterations of simple additions and shiftings. Since then, many follow-up works have been proposed to speed up the algorithm via array-type hardware accelerators [4, 5, 6, 7, 8, 9]. Specifically, [9] maintained to be the fastest ASIC realization as it successfully exploited the inherent parallelism between multiplication and modular reduction in Montgomery's algorithm. The record was recently broken by [10], where the combination radix-4 and cellular array were shown to outperform the work of [9].

However, straightforward implementation of [10] results in poor hardware utilization (33%[10]) and long critical path. As a remedy, [10] first applied pipelining to cut down the critical path, then interleaved *four* independent data inputs to raise the hardware utilization. Nevertheless, there are drawbacks in doing so. First of all, [10] requires many Flip-Flops to support systolic array-type computation. This is particularly serious for RSA since the number of bits is already large (≧1024 for universally acceptable security). Secondly, a 4-to-1 multiplexer is required at each interleaving point, which happens to be on the critical path. Lastly, the interleaving of multiplication and square operation (required in modular exponentiation, see pp. 480 of [10] for details) implies that one cycle is allocated to the square operation regardless of the real value of the exponent bits. This means the design cannot skip the 0-bits in the exponent and hence always follows the worst-case execution time.

In this regard, we resort to the algorithm of [9] and propose a radix-4 extension to cut down the number of iterations in half. We further utilize the algorithmic features to simplify the logic functions along the critical path. The resultant design incurs only a little more hardware cost than [9], yet provides more than twice the speed-up. Finally, we identify the switching profile of RSA data and accordingly propose a power-optimized design for the storage elements and key computational components. The complete 1024-bit RSA is realized via cell-based design style using 0.18um CMOS technology. Post-layout simulation shows that the design delivers an average performance of 586kbps at 460MHz, 1.8V while consuming only 830mW.

2. Algorithm Design

We adopt the H-type modular exponentiation algorithm [9] and extend the split modular multiplication scheme [9] to radix-4 as follows (proof of correction omitted for brevity). Note that when $q_{2i}=0$ and $q_{2i+1}=n_1$, we actually have the choice of adding $3N$ or subtracting N. The former requires an n-bit register to store the value of 3*N*, while the latter requires wider bit range in the operand and some hardware to handle sign extension during the addition of partial products. In this paper, we choose 3*N* for ease of implementation.

```
$Inputs:$
  $Modulus: N = (n_{n-1}n_{n-2}n_{n-3}...n_2n_1n_0)_2$
  $Multiplier: A = (a_{n-1}a_{n-2}a_{n-3}...a_2a_1a_0)_2$
  $Multiplicand: B = (b_{n-1}b_{n-2}b_{n-3}...b_2b_1b_0)_2$
  $where\ 0 \le A, B < N,\ and\ N\ is\ odd$
$Output:$
  $Result: R = A \times B \times 2^{-n} \bmod N$

$BSR4(A, B, N)$
{
$//P1: multiplication\ operation$
  $A \times B = C = C1 \times 2^n + C0;$
$// C1 = (c_{2n-1}c_{2n-2}c_{2n-3}...c_{n+2}c_{n+1}c_n)_2$
$// C2 = (c_{n-1}c_{n-2}c_{n-3}...c_2c_1c_0)_2$
  $P[0] = 0;$

$//P2: modular\ reduction$
  $for(i = 0; i < n/2; i++)$
  {
    $(q_{2i+1}, q_{2i}) = (P[2i] + c_{2i+1}c_{2i}) (\bmod\ 4);$
    $if(q_{2i} == 0)$
    {
      $if(q_{2i+1} == 0)$  $P[2i+2] = (P[2i] + c_{2i+1}c_{2i})/4;$
      $else$               $P[2i+2] = (P[2i] + c_{2i+1}c_{2i} + 2N)/4;$
    }
    $else$
    {
      $if(q_{2i+1} == n_1)$  $P[2i+2] = (P[2i] + c_{2i+1}c_{2i} + 3N)/4;$
      $else$                 $P[2i+2] = (P[2i] + c_{2i+1}c_{2i} + N)/4;$
    }
  }
  $return\ \ R = P[n] + C1;$
}
```

3. Architecture and Logic Design

Fig. 1 shows our hardware block diagram. Each bold solid line on the output of a component in oval shape represents one stage of pipeline register. For each loop iteration, the multiplication/square operation is performed via the "Multiply/Square Product Generator (*MSPG*)" module followed by the "Double-Output Carry-Save Adder (*DoCSA*)" module. Note that to support radix-4 operations, both modules are designed to deliver two bits at one clock cycle (to be elaborated in Section 5).

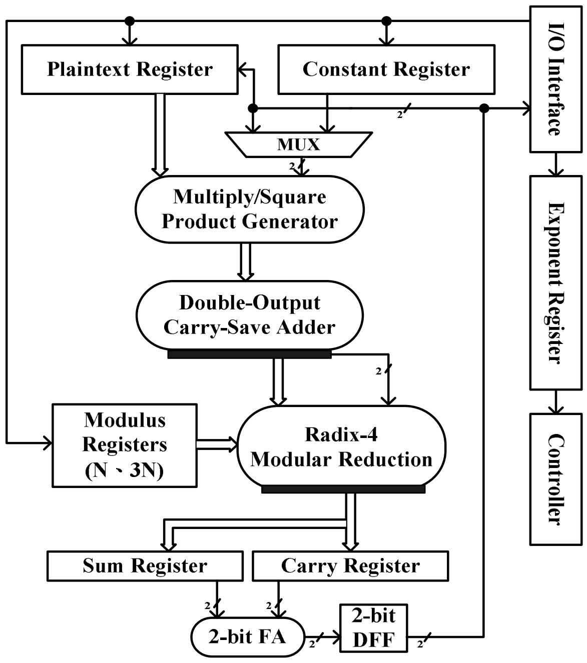


Fig. 1 Hardware Block Diagram

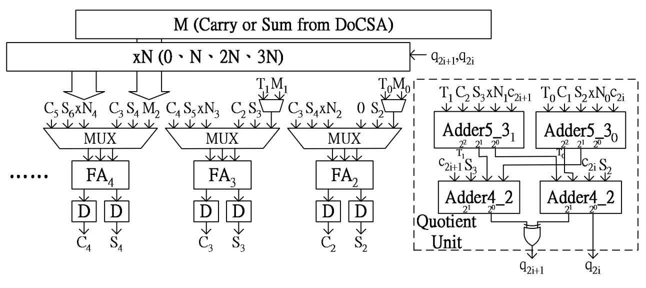
The two-bit multiplication/squaring results ($c_{2i+1}c_{2i}$) are fed into the "Radix-4 Modular Reduction (*R4MR*)" module. A straightforward implementation of the module is shown in the upper part of Fig. 2. The module first generates the quotient bits(q_{2i+1},q_{2i}) via the "Quotient Unit" (dashed box in Fig. 2), and then uses the quotient bits to compute ($P[2i]$ + $c_{2i+1}c_{2i}$ +kN) / 4, where $0<k<4$ in a carry-save fashion. The multiplexers select one of the two modes: (1) adding *kN* and shifting two bits; (2) adding the value of carry or sum from DoCSA. Notice that the index of FAs in Fig. 2 starts from 2, for the 2-bit LSB are always 0 according to the algorithm.

Except the Quotient Unit, the complete *R4MR* has *almost the same hardware component* as the corresponding module in [9]. Hence, all the rest 1022 (∵2 bits are already computed by the Quotient Unit) components in *R4MR* can operate *as fast as* the radix-2 counterpart in [9]. In other words, the whole *R4MR* will be able to operate as fast as its radix-2 counterpart, *only if* we can speed-up the Quotient Unit. Since the Quotient Unit only occupies a very small portion of *R4MR* (2/1024), it is very worthy of optimization so as to boost the system performance. We will come to this point shortly after description of the entire operation flow.

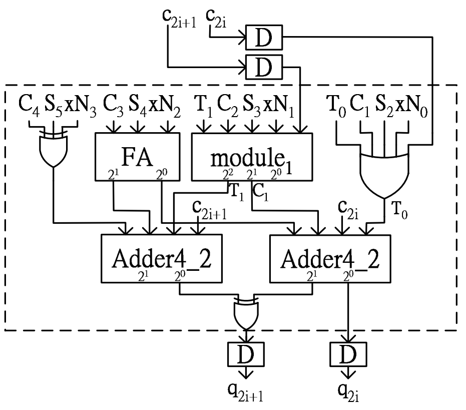
The *DoCSA* continues to send $c_{2i+1}c_{2i}$ bits in the first n/2 cycles. Once done, the *DoCSA* then contains the partial sum and partial carry of C1 (the MSB part of the multiplication/squaring, see algorithm description). Both are sent to *R4MR* to generate the modular-reduced sum and carry, which are copied to the registers outside of

R4MR. These registers save the values of the current sum and carry for the 2-bit addition, so that *R4MR* will be able to operate on the next sum and carry concurrently.

We now present the optimization of the Quotient Unit. As mentioned in the algorithm, the Quotient Unit is responsible for the computation of q_{2i+1} and q_{2i} (the two LSB in $\boldsymbol{P[2i]+c_{2i+1}c_{2i}}$). Due to the inherent complexity of the original design, typical logic synthesizers can not do much with respect to reducing delay. In lieu of this, we recognize that according to the algorithm, the last two bits in $(P[2i] + c_{2i+1}c_{2i} + kN)$, $0<k<4$, must equal to zero. Thus, the two Adder5_3 can be reduced to a 5-input OR gate together with the "$module_1$" in Fig. 2. Once done, the typical logic minimizer can be applied on $module_1$ to get further-optimized circuit. Lastly, to reduce the critical path, we put an extra stage of pipeline register at the input, leading to the modified circuit shown in the lower part of Fig. 2. Although the overall execution time is increased by one cycle, the tradeoff is very beneficial as the original cycle time is already large.



(a)



(b)

Fig. 2 *R4MR* and Quotient Unit Designs: original (a); optimized (b).

The architectural comparison of our work with the two most recent academic works [9, 10] is shown in Table I and II. Specifically, Table I shows direct area comparison, and Table II shows the area normalized by theorectical performance implied by the algorithms. We use the numbers of the major logic components, expressed in the order of RSA bit width (n), as the base area index. The base area values are then multiplied by the actual size of each logic component based on a 0.18um CMOS technology to get more accurate area results (Σ_{AREA}).

It can be seen from Table I that in terms of the most critical component of the design—the D-Flip-Flops (DFFs), our design (12.5n) is slightly larger than its radix-2 counterpart [9] (12n), and dramatically smaller than the other radix-4 design [10] (19n). Overall, the prior radix-4 work [10] paid lots of area for performance. In contrast, our design, being able to deliver twice the performance than [9], incurs only minor area increase in terms of real process technology (from 157.6n to 176.4n).

For in-depth comparison, we normalize the area result in Table I by the theoretical performance implied by the algorithms (#cycles), and derive the comparison data in the last column (Norm. Σ_{AREA} / #cycles) with n=1024. The result clearly shows that our design is 78% and 29% better than its radix-2 and radix-4 competitors, respectively.

TABLE I
ARCHITECTURAL COMPARISON OF ACADEMIC RSA DESIGNS-AREA

Cmpnt. (Area)	DFF (10)	FA (9)	mux (2)	and (1.2)	xor (2.2)	Σ_{AREA}
[9][a]	12n	2n	8n	3n	0	**157.6*n***
[10][b]	19n	2n	6n	3n	2n	**228.0*n***
Proposed[b]	12.5n	3n	11n	2n	0	**176.4*n***

[a]Radix-2; [b]Radix-4

TABLE II
ARCHITECTURAL COMPARISON OF ACADEMIC RSA DESIGNS-NORMALIZED AREA

Cmpnt. (Area)	Σ_{AREA}	Average #cycles	Σ_{AREA} / #cycles
[9][a]	157.6n	$1.5n^2+3.5n+2$	**1.78**
[10][b]	228.0n	$(2\lceil (n+3)/2 \rceil)\times(n+1)$	**1.29**
Proposed[b]	176.4n	$0.75n^2+2.5n+2$	**1.00**

[a]Radix-2; [b]Radix-4

4. Storage Strategy

The data in Table I show that DFFs, and thus the constituent registers, could consume more than 50% of the area. Since these registers serve different purposes, they may well have different switching patterns. In this section, we show how to take this opportunity to cut down the cost and power consumption of storage elements in a dramatic way. For all the descriptions of components herein, please refer to Fig. 1.

4.1. SRAM Deployment

The *Constant Register* and *Exponent Register* in Fig. 1 *never* need update in an RSA exponentiation. Further, they only need to give one or two bits output per cycle. These characteristics suggest an SRAM design for compact area and lower power consumption.

A 128x8 SRAM (7-bit address & 8-bit data width) is designed for this purpose. The design contains 7.3k transistors and consumes 3.2mW, both of which are much better than the cell-based design of a 1024-bit shift registers (37k transistors, 35mW by synthesis). Namely, by replacing the two registers with an SRAM, we save about 60k transistors and 60mW power.

4.2. Latch Deployment

Although the content of the *Modulus Register* is fixed, too, it is not possible to use SRAM since this register needs to supply 1024 bits of data per cycle to the *R4MR* module. The same happens in the *Plaintext Register*.

In this regard, we revise the design into the one with 1024-bit latches and one 32-bit shift register (Fig. 3). The 2-bit input data are stored in a 32-bit shift register. Once full, the entire 32-bit data are sent to the proper 32-bit latch block. Notice that the enable signal of 32-bit latch block is triggered by negative clock edge to prevent the problem of data racing. In adopting the latch design, the transistor count is reduced from 55k to 21k.

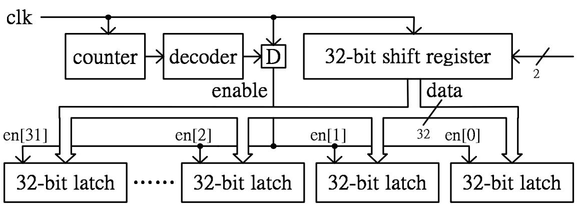


Fig. 3 The 1024-bit Latch-Based Register Design

4.3. Conditional Flip-Flop Design

After the prior two efforts, the utilization of DFFs is reduced to 35% of whole chip. Still, a power-optimization opportunity remains in taking advantage of the switching activity versus the types of flip-flops (Table III). This leads to the adoption of the Conditional Skew-Tolerant Flip-Flop (CSTFF) proposed in [12] for all DFFs in the current RSA design. For detail of CSTFF, please refer to [12].

5. Low Power Computation Components

5.1. Adders in DoCSA

To support the radix-4 algorithm, the origin full adder in *DoCSA* must be modified. We take the advantage of the regularity of the RSA design by re-designing the *adder4_2* circuit into Fig.4 and repetitively applying the new design to form the complete revision. The advantage of the revision can be readily seen from the comparison of *adder4_2* circuits shown in Table III.

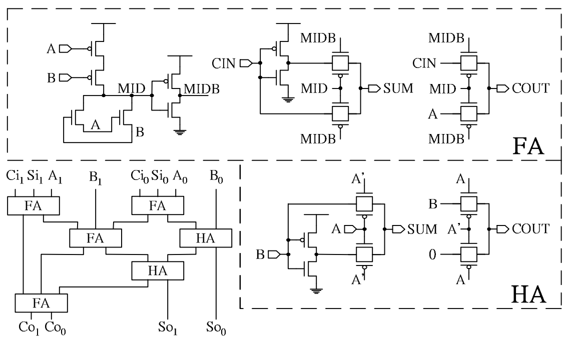


Fig.4 The *adder4_2*

Table III Comparison of Adder4_2

	#Transistors	Delay(ns)
Original	256	0.79
Modified	88	0.70

5.2. MUX-Adders in R4MR

The *R4MR* module contains the Quotient Unit and 1024 cells that support addition and multiplexing. Due to frequent operation of the module, the power consumption is large. To solve the problem, we redesign these cells by combining multiplexers and adders, forming the so-called Mux-Adder shown in Fig.5. The data in Table IV shows the design compares favorably to the one realized with separate adders and multiplexers.

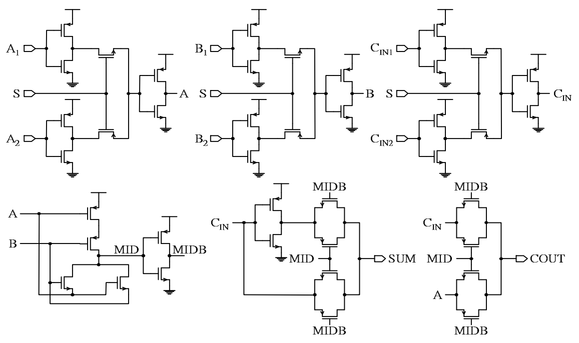


Fig.5 The *Mux-Adder*

TABLE IV
Comparison of Mux-Adder

	Power (μW)	Delay(ns)
Separate Design	312.88	0.56
Merged Design	202.1	0.42

5.3. The Carry and Sum Registers

The function of Carry/Sum registers and the associated addition is simple: storing the value of carry and sum at the proper time, shifting the data by two bits per cycle, and adding the two 2-bit output data. However, naïve implementation of shift operation costs lots of power because all 1024 DFFs shall be switching in every cycle. We propose to modify the module as shown in Fig.7. The revision incurs negligible area overhead but significantly reduces the power consumption from 63mW to 5.9mW.

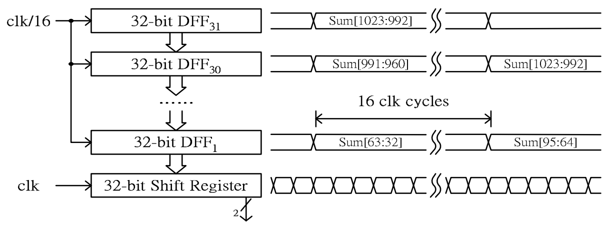


Fig.6 Architecture of Carry/Sum Register

6. Empirical Evaluation

The complete 1024-bit RSA is done in cell-based design style using 0.18um CMOS technology. Fig.7 shows the layout of the chip. The chip occupies an area of 5.76mm^2 (2400um$\times$2400um). Table V shows the comparison with the recent academic works [9, 10]. Note that both [9] and [10] were 512-bit designs. So an estimate of double the transistor count is assumed for the case of 1024-bit. Also, both [9] and [10] provided only pre-layout simulation (pre-sim) clock rate. Hence, we simply take the pre-sim clock rate and assume the 1024-bit design runs *as fast as* the 512-bit one. In doing so, the comparison has greatly biased towards [9, 10] as our data were obtained via post-layout simulation (post-sim) of the 1024-bit design. Still, Table V substantiates the advantage of our design in remarkable performance gain (586 kbps post-sim versus 146 and 79 kbps pre-sim) at very competitive transistor count and silicon area.

TABLE V
COMPARISON OF **ACADEMIC** RSA HARDWARE DESIGNS

	Tech Volt	Clk MHz	Avg. Kb/s	Tx (k)	Area (mm^2)
[9][a] Pre-layout sim.	**0.6u**	**125**	**79**	**645**	**55.9**[b]
[10][a] Pre-layout sim.	**0.6u**	**150**	**146**	**912**	**N/A**
Proposed Post-layout sim.	**0.18u**	**460**	**586**	**710**	**5.76**

[a]Original design is for 512-bit modulus, data except "**area(mm^2)**" deduced for 1024-bit based on the descriptions in [9] and [10].
[b]Area is for 512-bit modulus.

We also compare our design with the state-of-the-art commercial products that are within our reach (via public internet access). Specifically, we would like to show how the point-designs presented in Section IV and V can be integrated together to contribute to dramatic power savings, yet still maintain the performance advantage of the original algorithm and architecture. Note that some of these commercial products provide other cryptographic functions (e.g., DES, MD5, etc) on the same chip as well, though the RSA is usually the largest and the most power and time consuming among all. Hence, the listed data only serve to ascertain the quality of our design and do not reflect the actual RSA performance of these commercial products.

From Table VI, it can be seen that our design delivers the best performance in terms of RSA exponentiation operations per second, and is 21% better than the second-best design (Nitrox-II CN2560). Meanwhile, to make a fair comparison of power consumption, we have divided the power consumption with the performance figure (RSA OPS) and normalized all designs with ours. The result shows a remarkable power versus performance advantage of our design—22 times smaller than the second-best design (Nitrox-II CN2560).

TABLE VI
COMPARISON OF **INDUSTRIAL** 1024-BIT RSA HARDWARE DESIGNS

	Tech(um)/ Voltage(V)	Clk MHz	RSA OPS	Power (P) (mW)	Norm. P /OPS
IBM RICO-1	**0.5/--**	**48**	**45**	**350**	**458**
Motorola MPC180	**0.25/1.8**	**66**	**31**	**600**	**1139**
Hifn 7956	**-- /1.8**	**66**	**84**	**1000**	**700**
Cavium Nitrox-II	**0.13/1.0**	**400**	**40000**	**15000**	**22**
Proposed	**0.18/1.8**	**460**	**48500**	**830**	**1**

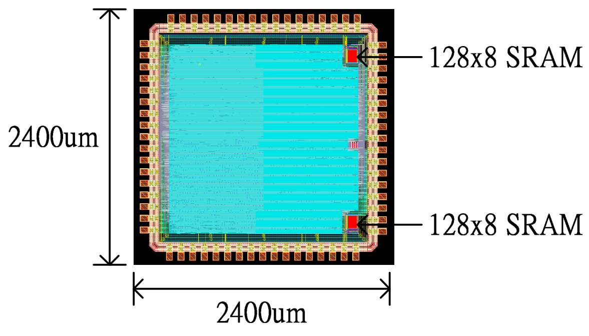


Fig.7 Chip Layout

1. CONCLUSION

We have presented a 1024-bit RSA design that delivers an average performance of 586kbps at 460MHz, 1.8V while consuming only 830mW. The successful proposition and integration of algorithm, architecture, logic and circuit designs have attributed to superior performance indices that compare very favorably to both the academic and industrial state-of-the-art designs.

References

[1] W. Diffie and M.E. Hellman, "New directions in cryptography," IEEE Trans. Inform. Theory, vol. IT-22, pp. 644-654, Nov. 1976.
[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems," Com. of the ACM, vol. 21, No. 2, pp. 120-126, Feb. 1978.
[3] P. L. Montgomery, "Modular multiplication without trial division," Math. of Computation, vol. 44, No. 7, pp. 519-512, Apr. 1985.
[4] C.D. Walter, "Systolic Modular Multiplication," IEEE Trans. on Computers, vol. 42, pp.376-378, March. 1993.
[5] M. Shand and J. Vuillemin, "Fast implementation of RSA cryptography," Proc. 11th IEEE Symp. Comput. Arith., Jun. 1993, pp. 253-259.

[6] P. Kornerup, "A systolic, linear-array multiplier for a class of right-shift algorithms," IEEE Trans. Comput., vol. 43, pp. 892-898, Aug. 1994.

[7] P.S. Chen, S.A. Hwang, and C.W. Wu, “A systolic RSA public key cryptosystem,” in Proc. IEEE Int. Symp. Circuits and Systems (ISCAS), vol. 4, Atlanta, GA, May 1996, pp. 408–411.

[8] P.A. Wang, W.C. Tsai, C.B. Shung, "New VLSI architectures of RSA public-key crypto-system," IEEE International Symposium on Circuits and System, pp.2040-2043, 1997.

[9] C.C. Yang, T.S. Chang, C.W. Jen, "A new RSA cryptosystem hardware design based on Montgomery's algorithm," IEEE Trans. Circuits Syst. II, vol. 45, pp.908-913, 1998.

[10] J.-H. Hong and C.-W. Wu, "Celluar modular multiplier for fast RSA public-key cryptosystem based on modified Booth algorithm," IEEE Trans. VLSI Syst., vol. 11., Jun. 2003, pp. 474-484.

[11] H.T. Bui, Y. Wang, and Y. Jiang, “Design and Analysis of Low-Power 10-Transistor Full Adders Using Novel XOR-XNOR Gates,“ IEEE Transaction on Circuit and System, vol.49, January 2002.

[12] J.S. Wang et al, “An Ultra Low Power, Fast Lock-in, Small Jitter, All Digital Delay Locked Loop,” Proc. IEEE ISSCC, 2005, paper 22.7.