EFFICIENT ASIC ARCHITECTURE OF RSA CRYPTOSYSTEM

Varun Nehru[1] and H.S. Jattana[2]

VLSI Design Division, Semi-Conductor Laboratory,
Dept. of Space, S.A.S. Nagar.
[1]nehruvarun@gmail.com, [2]hsj@scl.gov.in

ABSTRACT

This paper presents a unified architecture design of the RSA cryptosystem i.e. RSA crypto-accelerator along with key-pair generation. A structural design methodology for the same is proposed and implemented. The purpose is to design a complete cryptosystem efficiently with reduced hardware redundancy. Individual modular architectures of RSA, Miller-Rabin Test and Extended Binary GCD algorithm are presented and then they are integrated. Standard algorithm for RSA has been used. The RSA datapath has further been transformed into DPA resistant design. The simulation and implementation results using 180nm technology are shown and prove the validity of the architecture.

KEYWORDS

RSA, cryptosystem, crypto-accelerator, public key, private key, Extended Binary GCD, Stein, Miller-Rabin, modular inverse, DPA resistance

1. INTRODUCTION

The RSA algorithm [1] is a public key algorithm and is extensively in security and authentication applications. Being computationally intensive, use of separate crypto-accelerator hardware to accelerate the computations is common. The communication between the main processor (32-64 bit) and the RSA crypto-accelerator (1024-2048 bit) requires a protocol for data exchange and a FIFO register bank can implemented for the same. This paper describes an architecture design for the RSA cryptosystem useful for both the Encryption/Decryption and for the Key-Pair Generation which may be required due to security. The number to be tested as prime is fed as input to the system and the random numbers for Miller-Rabin test are generated using Pseudo-Random Number Generator (PRNG).

The paper is organized as follows: Section 2 introduces the basics of RSA algorithm. Section 3 describes fundamental algorithms, with modular architecture around which the top level system was developed. Section 4, discusses top-level implementation. Section 5 briefs about power analysis attacks. In Section 6, implementation results have been shown. In Section 7, conclusion is drawn.

2. Basics of RSA

RSA involves the use of a public key-pair {e, n} and a private key {d, n} for encryption and decryption respectively. Messages encrypted with the public key can only be decrypted using the private key. For digital signatures private key is used. The proof of the algorithm can be found in [1]. The steps for Key Generation and Encryption/Decryption are reproduced below:

2.1. Key-Pair Generation

1. Choose primes, p and q.
2. Compute modulus $n = p*q$. Its length is called the key length.
3. Compute Euler's totient function, $\varphi(n) = (p - 1)(q - 1)$.
4. Choose a public key, e, such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Determine d as $d-1 \equiv e \pmod{\varphi(n)}$.

2.2. Encryption and Decryption

Cipher text(C) is obtained as a number theory equivalent to the public key (e) exponentiation of message (M) in modulus n

$$C = M^e \bmod \{n\}.$$

Similarly, message can be recovered from cipher text by using private key exponent (d) via computing

$$M = C^d \bmod \{n\}.$$

3. Modular Design Architectures

This section describes the architectures developed for various modules used in the design of RSA cryptosystem.

3.1. Modular Multiplication

The binary interleaving multiplication and reduction algorithm is the simplest algorithm used to implement the modular multiplication [2]. The algorithm can be obtained from the expansion,

```
$P = 2 ( . . . 2 ( 2 ( 0 + A*B_k ) + A*B_{k-1} ) + . . . ) + A*B_1$, as :
Input: A, B
$R \leftarrow 0$
for {i = 0 to k-1} {
        $R \leftarrow 2R + A*B_{k-1-i}$
        $R' \leftarrow R-n$
        if {[R'] >= 0} {$R \leftarrow R'$}
                $R' \leftarrow R-n$
        If {[R']  >= 0} {$R \leftarrow R'$} }.
```

The hardware implementation of the datapath core is shown as in the Fig. 1. Signed subtractors have been used. The word-length of the subtractors and adders used is one and two bits more respectively.

3.2. Modular Exponentiation

The binary method for computing M^e (mod n) has been implemented using Left-to-Right (LR) algorithm. [2]

```
Input: M; e; n
if {$e_{h-1} = 1$} {$C \leftarrow M$} else {$C \leftarrow 1$}
```

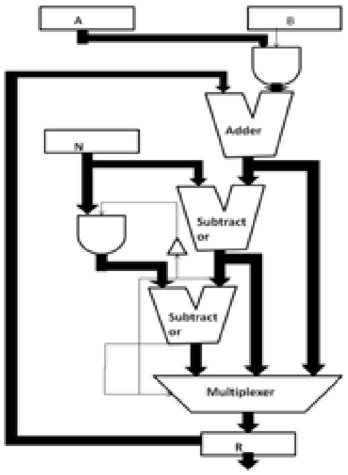


Figure 1. Architecture of RSA Datapath

```
for {i = h-2 to 0} {
    $C \leftarrow C*C$ (mod n)
    if {$e_i = 1$} {$C \leftarrow C*M$ (mod n)} }
```

The above algorithm is specific to the design of control unit for the RSA module. For the purpose of hardware optimization, it has been assumed that the MSB of exponent bit-word is always 1 i.e. the exponent always starts with the MSB.

The datapath core of RSA, as depicted in Fig. 1, is combined with some additional digital design blocks for complete RSA module. The state diagram for the same is given in Fig. 2. The states s0, s1, s2 are used for initialization and directing the primary input into the registers.

The states s4, s5 perform the binary multiplication; s5a checks the LSB of the exponent bit and if the LSB is HIGH it directs controller to another binary multiplication with changed inputs. The second binary multiplication is performed in state s9. If the LSB was LOW, the controller loops back to state s3. The state machine essentially performs binary modular multiplication. When the signal for completion of exponentiation is received, the state s11 is jumped to.

3.3. Miller-Rabin Primality test

Miller-Rabin Primality test is the most widely used primality testing algorithm [3][4]. The design for Miller-Rabin algorithm, shown in Fig. 3, is built around the RSA module described above with some additional control signals. The same RSA module has been used for exponentiation and squaring purposes.

This test provides advantages over other primality tests given by the Fermat and Euler [5]. The algorithm is reproduced below from [4][5] in an implementation friendly, Register Transfer Language (RTL), format.

```
Input: K, N
Output: P_Cb
For {i = 0 to K-1} {
    D ← N-1
    S ← 0
    While {[D_0] = 0} {
```

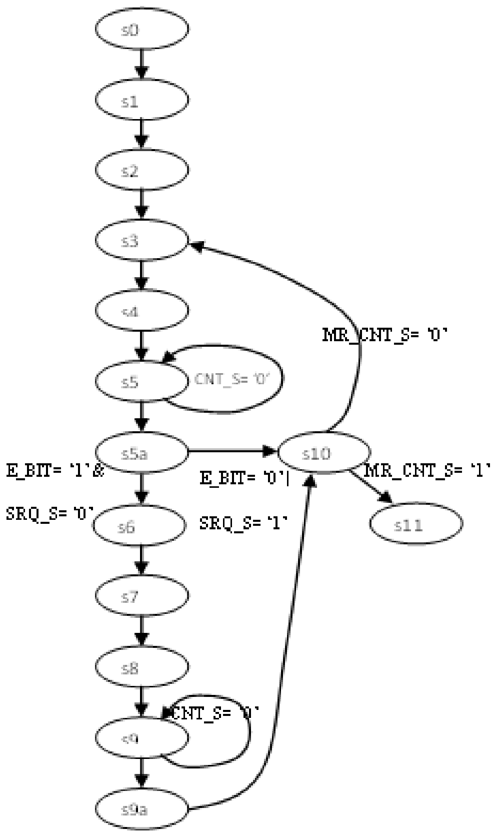


Figure 2. State Diagram for Modular Exponentiation

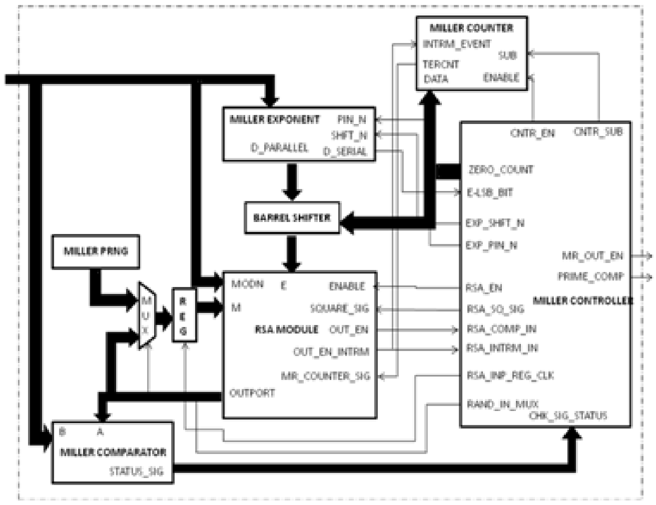


Figure 3. Architecture for Miller-Rabin Test Algorithm

```
D ← shr (D, 1)
S ← S + 1 }
        A ← RB (Random Base) {  RB ϵ [2, N-2]}
        X ← $A^D$ mod (N)
        if {[X] = 1 || [X] = N-1} {continue}
        for {r = 1 to S - 1} {
                X ← $X^2$ mod (N)
                if {[X] = 1} {P_Cb ← 0}
                if {[X] = N-1} {continue} }
        P_Cb ← 0 }
P_Cb ← 1
```

K is selected as per target accuracy and is sufficed at 7 for 512 bit primes and at 4 for 1024 bit primes [6].

The Miller exponent block, which is a modification over PI-P/SO shift register is used to calculate the ‘S' and 'D' values in the algorithm. The Miller controller detects the zeros in the exponent using shifting. A PRNG has been used to feed the random seed value to the RSA module for random base number. The counter counts a RSA intermediate event as clock. Miller controller serves as the master control unit of the system. The signal from the Miller controller further controls the events/states controlled by a separate RSA module controller which acts as a slave control unit.

The state diagram for Miller-Rabin primality test is given in Fig. 4. States s0, s1, s2 are used for initialization purposes. State s0 enables the exponent register to take input exponent, N, which is the number to be tested for primality. State s1 and s2 are used to count the number of trailing zeros in the exponent. It is to be ascertained that the exponent bit-string must begin with the MSB.

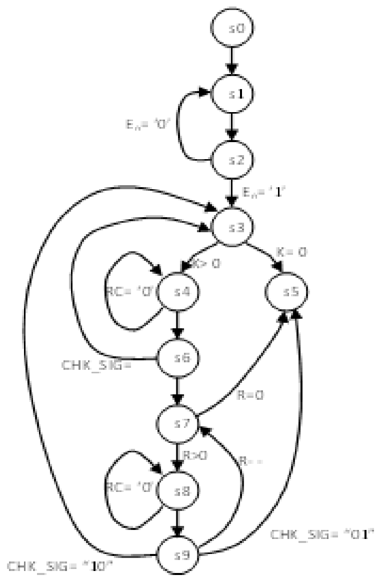


Figure 4. State diagram for Miller-Rabin Primality test

After all the trailing zeros have been counted, state s3 takes a random number from instantiated PRNG and while the number of iterations, K, for which the Miller-Rabin test is to be run is not equal to zero, it calls the state s4, which performs exponentiation.

When the exponentiation is complete state s6 checks the status in the miller comparator. If the status signal from miller comparator is "10" or "01", the controller goes back to state s3. Status "10" denotes that the result from the exponentiation is equal to N-1 and status "01" denotes the result to be unity.

For other status signals, the state s6 jumps to s7 which send a square signal to RSA module and performs the squaring operation in state s8. State s9 again checks the status and jumps of the consequent state.

3.4. Extended Binary GCD Algorithm

The binary GCD algorithm, also known as Stein's algorithm, computes the GCD of non-negative numbers using shifts, subtraction and comparisons rather than division used in Extended Euclidean algorithm. The binary GCD algorithm given in [7] can be implemented as shown in Fig. 5. The extended version of the same algorithm for calculating modular inverse has been presented below, for implementation, in RTL as

```
Inputs: A, B
Outputs: GCD, INV_OUT
Initialize: U ← 1; V ← 0; S ← 0; T ← 1; P ← A;
Q ← B
While {[B] ~= 0} {
        If {[B] = [A]} {
                        GCD ← shl (A,[R])
                INV_OUT ← S }
        Else if {[B] < [A]} {
                $A \leftrightarrow B$
                $U \leftrightarrow S$
                $V \leftrightarrow T$ }
        Else if {$[A_0] = 0$ & $[B_0] = 0$} {
                $A \leftarrow$ shr (A, 1)
                $A \leftarrow$ shr (B, 1)
                $R \leftarrow R + 1$ }
        Else if {$[A_0] = 0$ & $[B_0] = 1$} {
                $A \leftarrow$ shr (A, 1)
                If {$[U_0] = 0$ & $[V_0] = 0$} {
                        $U \leftarrow$ shr (U, 1)
                        $V \leftarrow$ shr (V, 1) }
                Else {
                        $U \leftarrow$ shr (U + Q)
                        $V \leftarrow$ shr (V – P) } }
         Else if {$[A_0] = 1$ & $[B_0] = 0$} {
                $B \leftarrow$ shr (B, 1)
                If {$[S_0] = 0$ & $[T_0] = 0$} {
                        $S \leftarrow$ shr (S, 1)
                        $T \leftarrow$ shr (T, 1) }
                Else {
                        $S \leftarrow$ shr (S + Q)
                        $T \leftarrow$ shr (T – P) } } }
GCD $\leftarrow$ shl (A, [R])
INV_OUT $\leftarrow$ S
```

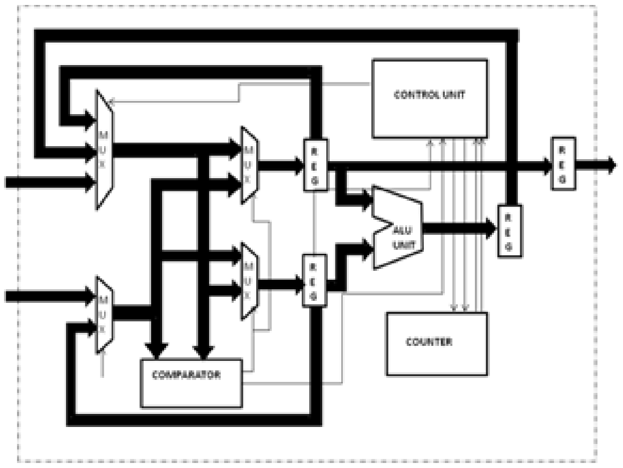


Figure 5. Architecture for BCD Algorithm

The above extended algorithm can be implemented by augmenting the architecture given in Fig. 5 with addition of few multiplexers, registers, subtraction units and control signals, as in Fig. 6.

The state diagram for Extended Binary Greatest Common Divisor (EBGCD) is given in Fig. 7. State s0 is the initialization state in which the inputs A & B are read in the various registers. In

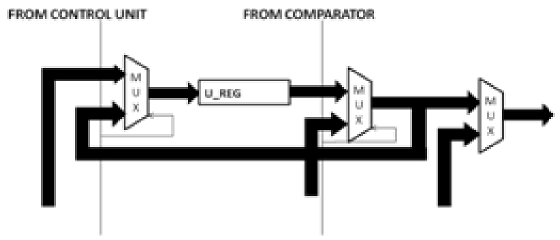


Figure 6. Additional structures required for Extended Binary GCD algorithm

state s1, the values and LSBs of both the inputs are compared. When LSBs of both A and B are LOW, the state s1 jumps to s3. The registers of both the inputs are right shifted and a counter is incremented.

When LSB of only either of the input is LOW, the state s4 or s5 are traversed to. The states s4, s4a, s4b, s4c and s5, s5a, s5b, s5c are used to perform the required computations. The states s6 through s6d operate when LSBs of both the inputs are HIGH. When both the inputs are equal, the state s1 jumps to s2 or s2b depending on whether the count for bit-shifts is zero or not. The state s2a and s2 are used to left-shift the output required number of times.

When value of B is less than A, the signal from the comparator to various MUXs goes HIGH and the interchange between various register is performed within that clock cycle.

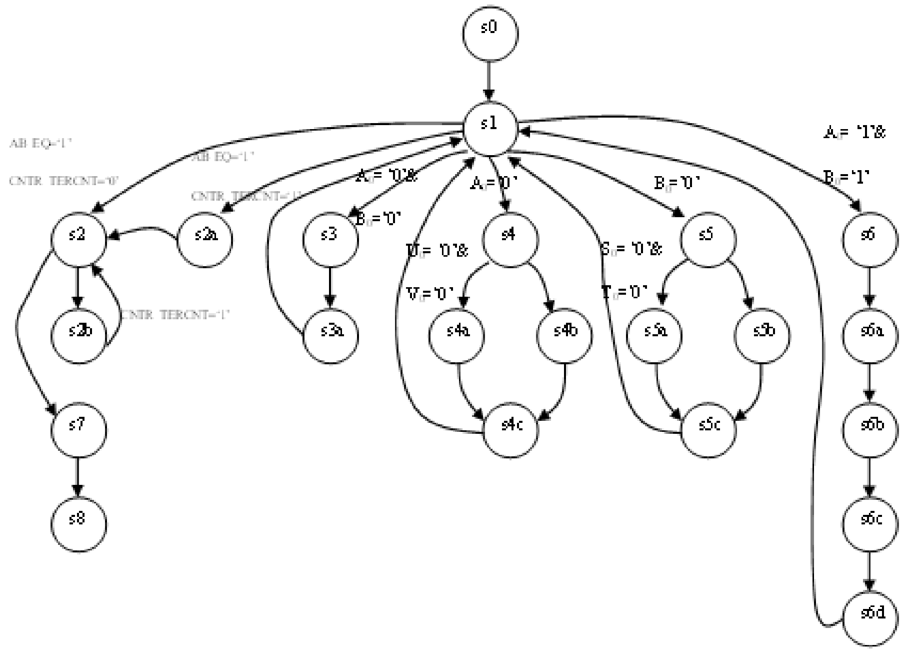


Figure 7. State diagram of Extended Binary GCD Algorithm

The Fig. 8 gives the complete architecture of the Extended Binary GCD algorithm. The signals from the comparator and EBGCD controller are used to control the data flow inside the register loops.

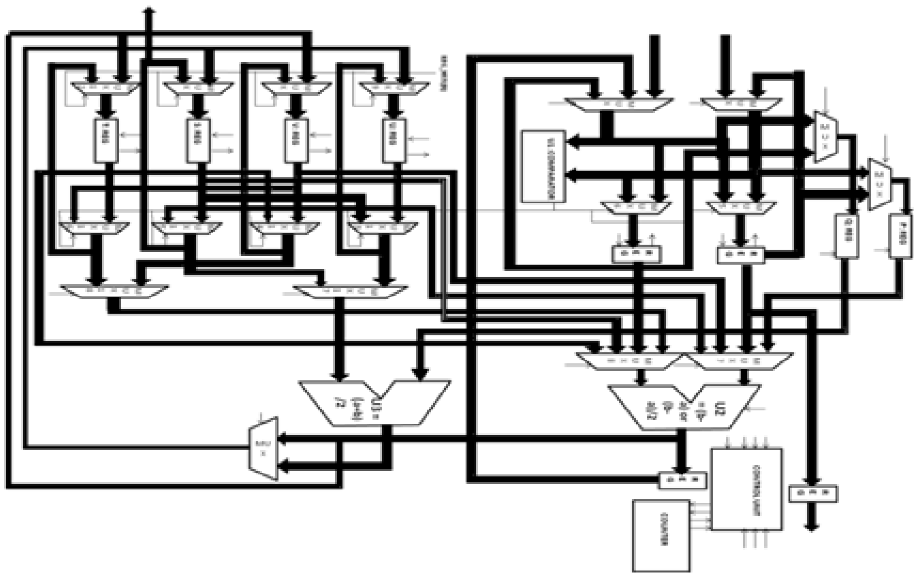


Figure 8. Detailed Architecture of Extended Binary GCD Algorithm

4. Top-Level Design

After the individual design is completed for various modules, these are integrated in top-level design of RSA cryptosystem.

The cryptosystem can be run in either of the two modes:
(i) RSA encryption/ decryption (RSA mode) and,
(ii) Key-Pair Generation (GKP mode).

The design of the complete cryptosystem as implemented is shown in Fig. 9. The modes are controlled by GKP_RSAb control input. The system has an EXPONENT_BIT_CNTR counter which counts the intermediate RSA event and sends the signal for RSA completion. The input to the counter is number of bits of exponent bit-word that are to be used for exponentiation. The number for primality test may be supplied from memory or True-RNG as input.

During RSA computation, the controller after enabling the RSA module and directing the input MUXs to feed from Primary inputs waits for a signal from RSA module for completion. A signal from the exponent bit counter is sent to RSA module to indicate last bit the exponentiation.

During generation, the top system controller runs the Miller-Rabin controller twice to obtain two primes. In case the test fails and the random number is composite, the system keeps on taking the random numbers as input till both the prime numbers are determined. The product of primes and their Euler totient function are computed in two cycles using single combinational multiplier. The values computed are fed in to the EBGCD module the output of which is compared to the unity. If the output is not unity, another random number is taken as input. If the result is unity, the random number taken as input serves as the public key and the modular

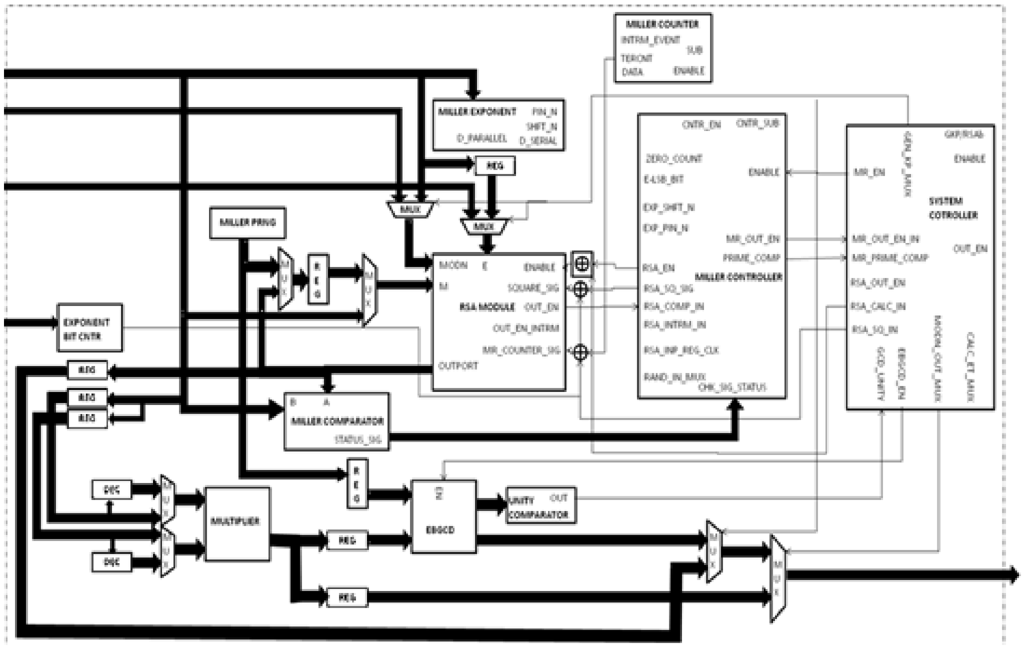


Figure 9. Top-level Architecture of RSA Cryptosystem

inverse output from the EBGCD module serves as the private key with modulus being the product of the primes.

The Miller PRNG has been used to generate a public key exponent; however, desired key may be provided externally with use of an additional multiplexer. The unity comparator block is implemented by a using a series of the OR gates.

5. Power Analysis Resistance

Power analysis attacks exploit the fact that the instantaneous power consumption of a cryptographic device depends on the data it processes and the operations it performs.

Simple power analysis (SPA) involves directly interpreting power consumption measurements collected during cryptographic operation. Differential power analysis (DPA) attacks, which require large number of power traces for analysis, are used due to the fact that these do not require detailed knowledge about the attacked device.

In CMOS technology, it is a fact that transitions are affiliated and determined by statistics of gate inputs and previous outputs, to the differing way energy is consumed between a 0→VDD and VDD→0 transitions.

To counter DPA, the device needs to be built in such a way that every operation requires approximately the same amount of energy, or it can be built in such a way that the power consumption is more or less random. To the effect of first technique a custom EDA flow was developed for transforming the synthesized design into a design compliant to Differential Power Balancing DPA resistant technique called Delay Insensitive Minterm Synthesis-3 (DIMS-3) [8]. Fig. 10 shows the typical transformation methodology used for improving the DPA resistance of the RSA datapath.

6. IMPLEMENTATION

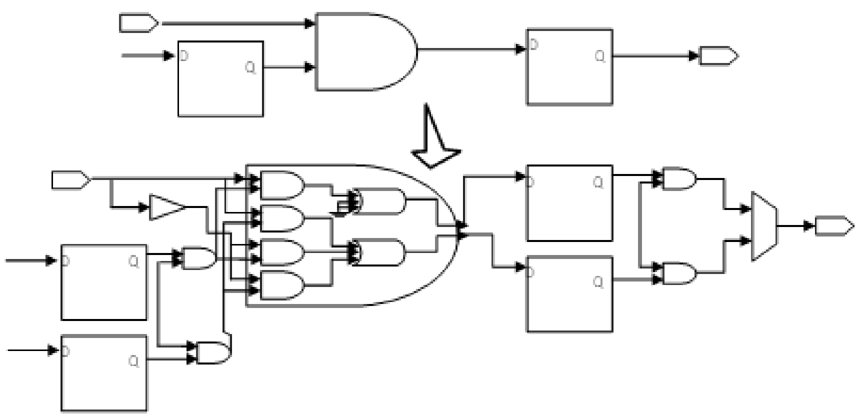


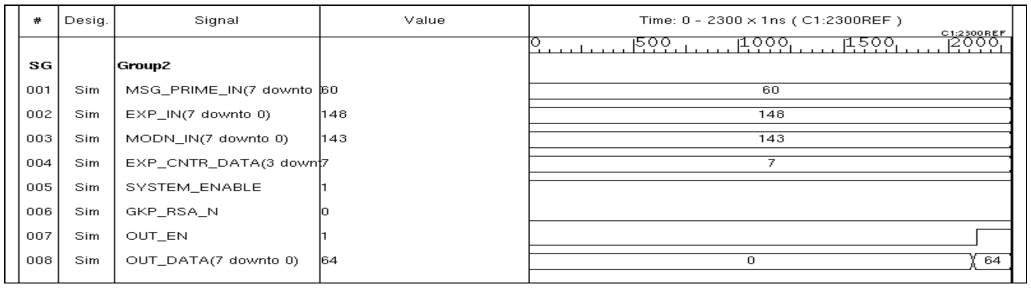
Figure 10. Delay Insensitive Minterm Synthesis-3 compliant transformation

This work describes the architecture of RSA cryptosystem built with the individual modules in the beginning to the top-level system in the end. The code of the described architecture was written in VHDL. The code for 8-bit system was synthesized and simulated using Tower 180nm digital library in Synopsys tools.

6.1. Simulation Results

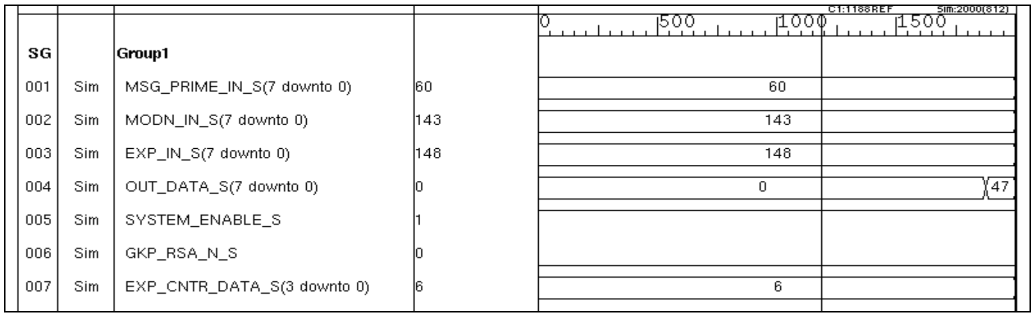
Fig. 11 and Fig. 12 show the simulation result of the above said architecture for RSA encryption/decryption. Though both of figures use the same input bit-strings, their EXP_CNTR_DATA_S input to EXPONENT_BIT_CNTR is different. Thus, in Fig. 11, effective exponent is 74("1001010") and in Fig. 12 effective exponent is 37("100101").
Fig. 13 shows the output sequencing of private key and modulus, when the system is used for key pair generation with primes 11 and 13.

Fig. 14 and Fig. 15 show the power signatures for a computation of Differential Power Balancing DIMS-3 compliant RSA datapath transformed using custom EDA flow at positive and negative clock edges respectively.



#	Desig.	Signal	Value	Time: 0 - 2300 x 1ns (C1:2300REF)
SG		Group2		
001	Sim	MSG_PRIME_IN(7 downto	60	60
002	Sim	EXP_IN(7 downto 0)	148	148
003	Sim	MODN_IN(7 downto 0)	143	143
004	Sim	EXP_CNTR_DATA(3 down	7	7
005	Sim	SYSTEM_ENABLE	1	
006	Sim	GKP_RSA_N	0	
007	Sim	OUT_EN	1	
008	Sim	OUT_DATA(7 downto 0)	64	0 64

Figure 11. Simulation of RSA Cryptosystem for RSA Encryption/Decryption with Exponent bits count = 7

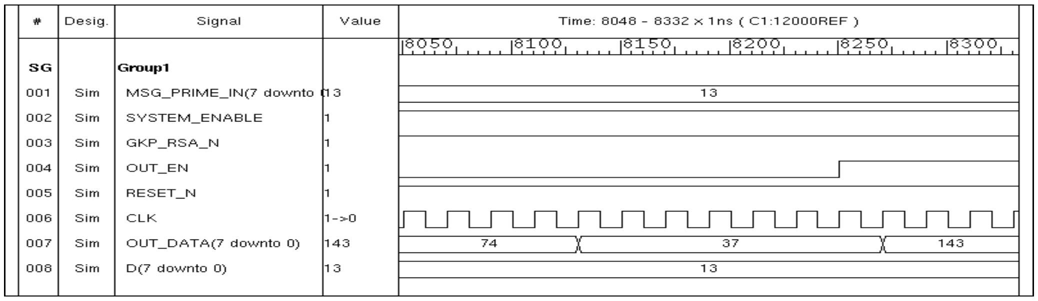


C1:1188REF Sim:2000(812)

0 500 1000 1500

SG		Group1		
001	Sim	MSG_PRIME_IN_S(7 downto 0)	60	60
002	Sim	MODN_IN_S(7 downto 0)	143	143
003	Sim	EXP_IN_S(7 downto 0)	148	148
004	Sim	OUT_DATA_S(7 downto 0)	0	0 47
005	Sim	SYSTEM_ENABLE_S	1	
006	Sim	GKP_RSA_N_S	0	
007	Sim	EXP_CNTR_DATA_S(3 downto 0)	6	6

Figure 12. Simulation of RSA Cryptosystem for RSA Encryption/Decryption with Exponent bits count = 6



#	Desig.	Signal	Value	Time: 8048 - 8332 x 1ns (C1:12000REF)
				8050 8100 8150 8200 8250 8300
SG		Group1		
001	Sim	MSG_PRIME_IN(7 downto 0)	13	13
002	Sim	SYSTEM_ENABLE	1	
003	Sim	GKP_RSA_N	1	
004	Sim	OUT_EN	1	
005	Sim	RESET_N	1	
006	Sim	CLK	1->0	
007	Sim	OUT_DATA(7 downto 0)	143	74 37 143
008	Sim	D(7 downto 0)	13	13

Figure 13. Output sequence of private key and modulus during Key-Pair generation

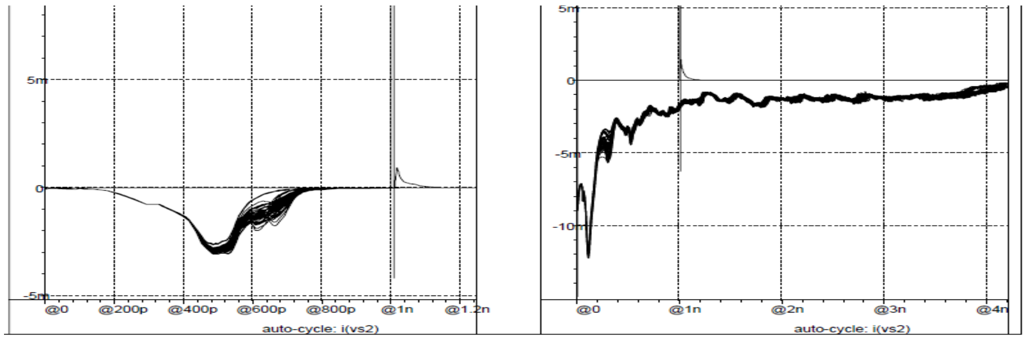


Figure 14. Power signature comparison between pre-transformed (left) and post-transformed (right) RSA datapath for various input

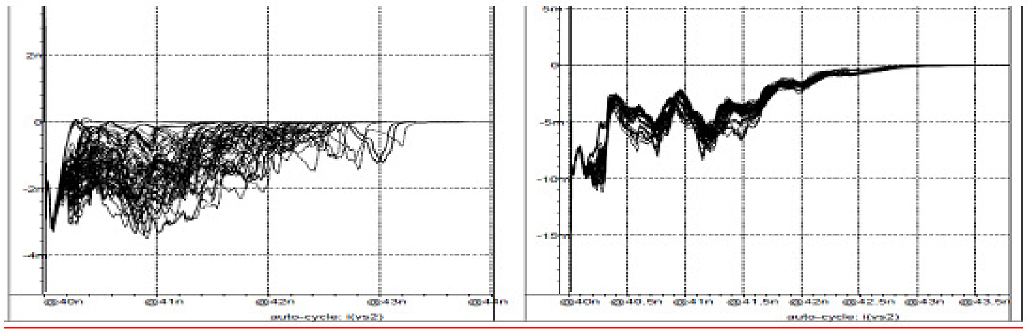


Figure 15. Power signature comparison between pre-transformed (left) and post-transformed (right) RSA datapath for various input

6.2. Implementation Results

Table I, II & III present the implementation results of synthesis of the RSA cryptosystem architecture in the 180nm Static digital CMOS library. Table I gives the count of the combinational and non-combinational cells implemented in the system. Table II enlists the area requirements of various design units the system. Table III gives the timing requirements of the core RSA module. E0 and E1 represent the number of 0's and 1's in exponent bit-word and N is the key length of the RSA. Table IV compares the area and cells required for optimized design to that for DIMS-3 compliant DPA resistant RSA datapath. Further, this work presents the results of the RSA datapath transformed into Differential Power Balanced DIMS-3 DPA resistance compliant design. The results of both the pre-transformed and post-transformed designs are presented for comparison.

Table I. 8-bit RSA cryptosystem cell count

CELL TYPE	CELL COUNT
combinational cells	1191
non-combinational	316

Table II. Area report of modules for 8-bit RSA

DESIGN UNIT	AREA	AREA %
Rsa_System	4113	100.0
Sys_Controller	250	6.1
Sys_Datapath	3863	93.9
Sys_Datapath/Comparator_Unit	22.25	0.5
Sys_Datapath/Controller_Unit	616	15
Sys_Datapath/Counter_Unit	47	1.1
Sys_Datapath/Enc_Data_Reg	56	1.4
Sys_Datapath/Exponent_Unit	86.5	2.1
Sys_Datapath/Exp_Cntr_Unit	56	1.4
Sys_Datapath/Gcd_Inv_Unit	1488.25	36.2
Sys_Datapath/Multiplier_Unit	68	1.7
Sys_Datapath/Prng_Unit	106	2.6
Sys_Datapath/Rsa_Unit	856.5	20.8
Sys_Datapath/Unity_Unit	1.75	0.0

Table III. Timing requirements

MODULE	CLKs
RSA Module	$3+E_0(4+N)+E_1(8+2N)$

Table IV. Area Reports for pre-transformed and post-transformed RSA module designs

******************************	*********************************
Report : area	Report : area
Design : RSA_DATAPATH (PRE-TRANSFORM)	Design : RSA_DATAPATH (POST-TRANSFORM)
******************************	*********************************
Number of ports: 37	Number of ports: 40
Number of nets: 191	Number of nets: 1520
Number of cells: 142	Number of cells: 1497
Number of combinational cells: 118	Number of combinational cells: 1449
Number of sequential cells: 24	Number of sequential cells: 48
Number of macros: 0	Number of macros: 0
Number of buf/inv: 28	Number of buf/inv: 95
Number of references: 14	Number of references: 13
Combinational area: 204.500000	Combinational area: 2316.500000
Buf/Inv area: 14.000000	Buf/Inv area: 57.500000
Non-combinational area: 126.000000	Non-combinational area: 306.000000
Net Interconnect area: 69.138399	Net Interconnect area: 1374.814115
Total cell area: 330.500000	Total cell area: 2622.500000
Total area: 399.638399	Total area: 3997.314115
******************************	***********************************

REFERENCES

[1] Rivest R. L., Shamir A., and Adleman L., “A method for obtaining digital signatures and public-key cryptosystems,” Communications of the ACM, 1978.

[2] Koc C. K., RSA Hardware Implementation, RSA Laboratories, Technical Report.

[3] Miller, Gray L., “Riemann’s Hypothesis and tools for Primality”, Journal of Computer and System Sciences, 300-317, 1976.

[4] Rabin, Micheal O., “Probabilistic algorithm for testing primality”, Journal of Number Theory, 128-138, 1980.

[5] Hoffoss D., Notes on "The Rabin-Miller Primality Test", University of San Diego.

[6] Kleinberg B., Notes on "The Miller-Rabin Randomized Primality Test", Cornell University.

[7] Knuth D. E. , Seminumerical Algorithms, The Art of Computer Programming Vol-2, Addison-Wesley.

[8] Murphy J., “Standard Cell and Full Custom Power-balancing logic: ASIC implementation”, Technical Report Series, New Castle University.

[9] Rahman M., Rokon I. R., Rahman M., “Efficient Hardware Implementation of RSA Cryptography”, 3rd International Conference on Anti-Counterfeiting, Security, and Identification in Communications, 2009.

[10] Shams R., Khan F. H., Umair M., “Cryptosystem an Implementation of RSA using Verilog”, International Journal of Computer Networks and Communications Security, Vol.-1, No. 3, August 2013, p102-109.

[11] Vishak M, Shankaraiah N., “Implementation of RSA key generation based on RNS using Verilog”, International Journal of Communication Network Security, Vol.-1, Issue-4, 2012.

[12] Garg V., Arunachalam V., "Architectural Analysis of RSA cryptosystem on FPGA", International Journal of Computer Applications, Vol-26, No.-8, July 2011.

AUTHORS

Nehru Varun received B.E. (Hons.) from Panjab University in 2010 and joined Semi-Conductor Laboratory. Since joining he has been working in VLSI design division and has been involved in digital designs.

Jattana H.S. received his engineering education from BITS Pilani and joined SCL as ATE engineer. He worked on test programs development /characterization for pulse dialler/tone ringer, Audio codec, IIR filter, signal processor for sonar applications and many ASICs. For the last over ten years he has been involved in design of VLSI products, and have contributed in many design projects like range of transceivers (400Mbps to 1.2 Gbps), power management chips, converters (12-bit pipeline ADC, 12-bit current steering DAC, 16-bit sigma-delta), CMOS Imaging sensor, cold sparing pads, read-hard tolerant digital cells and memory cell, and many ASICs.

He has worked at Rockwell Semiconductor, Newport Beach, USA for characterization of R65 series of devices and at AMS Austria for porting of 2 um and 1.2 um processes and ATE testing/characterization of products fabricated in these processes.