

1-Mar-05 (1)

# CEG5010

## Implementations of RSA

1-Mar-05 (2)

## FPGAs for Cryptography

- Particularly suitable for cryptographic accelerators
- State of the art VLSI performance
  - Faster products with zero design effort
- Multiple algorithms on the same hardware
  - Change functionality by downloading different designs to the same device
- Faster design times
  - more sophisticated algorithms, lower design cost, shorter time to market
- No NRE costs
  - Cheaper for low volumes
  - Conversion to ASIC for high performance, high volume
- Only recently have become cost effective for cryptography

1-Mar-05 (3)

## RSA Cryptography

- Invented in 1977 by Rivest, Shamir and Adleman
- Credit card machines, bank ATMs, e-mail applications, web browsers and mobile phones
- Encryption, key exchange, authentication

$$\begin{array}{l}
 P \ \& \ Q \ \text{PRIME} \\
 N = PQ \\
 ED \equiv 1 \pmod{(P-1)(Q-1)} \\
 C = M^E \pmod{N} \\
 M = C^D \pmod{N} \\
 \text{RSA Algorithm}
 \end{array}$$

1-Mar-05 (4)

## Licensing History

- US Patent # 4,405,829, "Cryptographic Communications System And Method" issued to MIT September 20, 1983
  - licensed exclusively to RSA Security and expires on September 20, 2000.
- Released to public domain September 6, 2000

1-Mar-05 (5)

## RSA Cryptography

- Generate 2 secret prime numbers P and Q
- Generate public modulus  $M=P \times Q$
- E is a public fixed odd number
  - usually  $E=2^{16}+1$  (CCITT standard)
  - E can be used iff it is not divisible by 2,3 or 7
- Find secret exponent D
  - find any D s.t.  $E \times D = 1 \pmod{(P-1) \times (Q-1)}$

1-Mar-05 (6)

## RSA Cryptography

- An  $n=km$  block bit message divided into m blocks of k bits
  - To encrypt a block A
    - $P(A) = A^E \pmod M$
  - To decrypt a block A
    - $S(A) = A^D \pmod M$
- M,E are public, p,q,D are secret  
 N.B.  $S(P(A)) = P(S(A)) = A \quad (0 \leq A < M)$

1-Mar-05 (7)

## Security

- The security of RSA hinges on the difficulty of finding  $p, q$  from  $M$ 
  - if we know  $p, q$  we can compute  $D$
  - for reasonable security,  $M \approx 512$  bits
  - In Aug 1999, a 512 bit number factored in 3.7 months, 35.7 CPU years, approx 300 computers
- Best hardware implementations
  - DES 1Gbits/s
  - 512 bit RSA 600Kbits/s (1500 times slower)

1-Mar-05 (8)

## Example (not very secure)

- $p=11, q=7$  (secret)
- $M=pq=77$  &  $E=13$  (public)
- Anyone can encrypt a message e.g.  $A=4$ 
  - $4^{13} \bmod 77 = 53$
- Decryption
  - $(p-1)(q-1)=60, 13 \times 37 \bmod 60 = 1$  ( $D=37$ )
  - $53^{37} \bmod 77 =$   
628358038363668332248635694548393830  
4940731973668146791149026213  $\bmod 77 =$   
4

1-Mar-05 (9)

## Hardware complexity

- RSA relies on computing a modular exponentiation
  - time complexity  $hk^2$
  - $h$  is a constant
  - $k$  is number of bits

1-Mar-05 (10)

## Chinese Remainder Theorem (to speed up decryption)

- If we use  $E=2^{16}+1$ , 512b, decryption 32 times slower than encryption (why?)
- CRT
  - $A_p=A \bmod P$ ,  $A_q=A \bmod Q$ ,
  - $B_p=A_p^{D \bmod (P-1)} \bmod P$ ,  $B_q=A_q^{D \bmod (Q-1)} \bmod P$
  - $S_p=A_p \times (Q^{P-1} \bmod M) \bmod M$
  - $S_q=B_q \times (P^{Q-1} \bmod M) \bmod M$
  - $A^E \bmod M = \text{if } S \geq M \text{ then } S_p + S_q - M \text{ else } S_p + S_q$

1-Mar-05 (11)

## CRT

- Problem reduced to two exponentiations of half the size
- Cycles
  - 1  $k/2$  bit multiplier
    - $2 h(k/2)^2$
  - 2  $k/2$  bit multipliers
    - $h(k/2)^2$
- Overall 4 times speedup (minus a small overhead)

1-Mar-05 (12)

## Exponentiation - binary lsb first

- $B = A^E \pmod M$

$B[0] = 1, P[0] = A;$

for  $i = 0$  to  $k-2$

{

$P[i+1] = P[i] * P[i] \pmod M$

$B[i+1] = \text{if } e_{i+1} = 1 \text{ then } P[i] * B[i] \pmod M \text{ else } B[i]$

}

$B = P[k-1] * B[k-1] \pmod M$

1-Mar-05 (13)

## Exponentiation - binary msb first

- $B = A^E \pmod M$

$B[0] = A,$

for  $i = 0$  to  $k-2$

{

$P[i] = B[i] * B[i] \pmod M$

$B[i+1] = \text{if } e_{\{k-i-2\}}=1 \text{ then } P[i] * A \pmod M \text{ else } P[i]$

}

$B = B[k-1]$

1-Mar-05 (14)

## Exponentiation

- Comparison
  - both  $\log_2(E)-1$  squares
  - both  $v_2(E)$  multiplications ( $v_2()$  number of 1's)
  - LSB method needs 2 registers, MSB only 1
  - LSB can use 2 k bit multipliers  $hk^2$  (1.5× faster but double the area - a bad deal)
  - MSB takes  $hk(k + v_2(E))$  with 1 mult

1-Mar-05 (15)

## Addition chains

- Exponentiation depends on addition
  - if  $A^X = A^{X_1} A^{X_2} \dots A^{X_k}$ 
    - $X = X_1 + X_2 + \dots + X_k$
- An addition chain for  $x$  is a seq of integers
  - $a_0 = 1, a_1, a_2, \dots, a_r = x$
  - finding minimal addition chain is an NP complete problem
    - instead try to calculate “good” addition chains (e.g. window method)

1-Mar-05 (16)

## Square and add (before 200BC)

- Square and add technique
  - write number in binary (e.g.  $23 = 10111$ )
  - if we see a “1” write “SX”, if “0” write “S”
  - cross out leading “SX” SSXSXSX
  - e.g. to compute  $x^{23}$  we compute  $x^2, x^4, x^5, x^{10}, x^{11}, x^{22}, x^{23}$  (1,2,4,5,10,11,22,23)
  - requires  $k - 1 + v_2(E)$  multiplications

1-Mar-05 (17)

## Square and add (addition chain form)

- Generates the chain
  - $(1, 2, \dots, a_i, \dots, x)$
  - $a_{i-1} = (a_i - 1)$  if  $a_i$  odd else  $a_i/2$

$1920004740_{10} = 111\ 00\ 100\ 1110\ 000\ 111\ 100101\ 0000\ 100_2$   
 binary method:  $k-1+v_2(E)=31-1+14=44$  muls  
 no temporary storage apart from partial  
 result

1-Mar-05 (18)

## Addition chain for $x^{23}$

- Square and add
  - $(1, 2, 4, 5, 10, 11, 22, 23)$
- Minimal addition chain
  - $(1, 2, 3, 5, 10, 20, 23)$
  - i.e.  $x^1, x^2, x^3, x^5, x^{10}, x^{20}, x^{23}$

1-Mar-05 (19)

## Window method

```
111 00 100 1110 000 111 100101 0000 100
  7      4   14      7      37      4
```

- Compute addition chain

1,2,4,5,7,14,16,30,37 (8 muls)

(2\*(2\*(2\*(2\*7))+4)+14 ...)

ops:

length(addition chain-1) (generate chain)

k-(#bits in MSB window) (shift to place)

+(#nonzero windows-1) (adds required)

= 31-3+5+8=41

1-Mar-05 (20)

## Star chains

- A star chain has additional restriction that the addition be with the previous number
  - special case of addition chains, they are not much longer and much easier to implement in hardware
  - multiplier input is the output of previous mul
- Difficulty lies in computing the star chain (maybe not a problem if we decrypt a lot)

1-Mar-05 (21)

## Star chains

- 512b exponentiation
  - normal method average = 768 multiplications
  - improved methods based on larger radix “windows” approx 1.22k = 620 multiplications

1-Mar-05 (22)

## Computing $N \bmod M$ (Euclid's algorithm)

- $N$  a  $k+p$  bit integer,  $M$  a  $k$  bit integer

```

R[0]=[nk+p-1 ... np+1]
for i=0 to p-1
{
  S[i] = 2R[i]+n{p-i};
  q{i} = if S[i]<M 0 else 1;
  R[i+1]=S[i]-q{i}M;
}
R=R[p+1]

```

1-Mar-05 (23)

## Hensel's Odd Division (1900)

- Computes  $N2^{-p} \bmod M$

$R[0] = [n_{k+p-1} \dots n_0]$

for  $i=0$  to  $p-1$

{

$q\{i\} = R[i] \bmod 2;$

$R[i+1] = (R[i] + q\{i\}M) \text{ div } 2;$

}

$R = R[p];$

1-Mar-05 (24)

## Euclid vs Hensel

- Euclid requires a  $k$  bit comparison between  $M$  and  $S[i]$
- Hensel's algorithm allows for quotient pipelining

1-Mar-05 (25)

## Quotient pipelining

- Reorganize Hensel's odd division equations s.t. they can be computed in a 4 stage pipeline
  - speedup of 4 (40MHz vs 10MHz for non-pipelined version)

1-Mar-05 (26)

## Modular product (Montgomery multiplication)

- All hardware and software implementations use this technique
- Computes  $\beta^{-n}A \times B \bmod M$

$P[0]=0$

for  $t=0$  to  $n-1$

{

$q_t = \mu(a_0 b_t + p_0(t)); \quad \# p_0(t) = \text{lsb of } P$

$P[t+1] = \beta^{-1}(Ab_t + P[t] + q_t M);$

}

1-Mar-05 (27)

## Example

- $\beta=2$ ,  $n=5$  (#digits in A, B and M),  
A=26=11010, B=11=01011, M=19=10011,
- $m_0\mu+1 \bmod \beta=0$ , i.e.  $\mu=1$
- $\beta^{-n}A \times B \bmod M$
- $P=[13, 29, 24, 25, 22]$
- $q=[0, 1, 1, 0, 1]$
- multiply by  $\beta^n \bmod M$  to get  $A \times B \bmod M$
- $26 \times 11 = 22 \times 2^5 \pmod{19}$

1-Mar-05 (28)

## Montgomery Multiplication Radix

- $\beta$  does not need to be 2
- $\beta=2^2$  is two times faster for only slightly more hardware

1-Mar-05 (29)

## Adders

- Carry save adder
  - compute sum and carry separately
  - $O(1)$  instead of  $O(k)$  time
  - convert back to binary form after each modular product
  - use an asynchronous carry completion adder
    - worse case carry is  $k$  bits
    - average case is  $\log_2 k$  bits
    - add hardware to check when all carries are 0, keep doing additions until no more carries occur

1-Mar-05 (30)

## Example

```

00100111+
00110101=
00010010 SUM 1
01001010 CARRY 1
01011000 SUM 2
00000100 CARRY 2
01011100 SUM 3
00000000 CARRY 3

```

1-Mar-05 (31)

## Summary of speedups

- CRT 4×
- Star chain 1.25×
- Hensel's odd division 1.5×
- Carry completion adder  $(2 - \log_2 k)/k$
- Quotient pipelining 4×
- TOTAL=60

1-Mar-05 (32)

## Performance

- The work described was presented in 1993 using XC3000 parts
- RSA decryption
  - 600Kbits/s (512b keys)
  - 165Kbits/s (1024b keys)

1-Mar-05 (33)

## Use of reconfigurability

- Different modular multiplier for different modulus
  - coefficients hardwired into design
- Different circuit used for encryption and decryption
- Not possible in an ASIC, we would tend to use same hardware for encryption and decryption and use a general modulus

1-Mar-05 (34)

## Performance

- Although this work was done in 1993 using XC3000 Xilinx parts, for almost 10 years it was the fastest RSA implementation in any technology
  - order of magnitude faster than any implementation in 1992
- Success due to
  - specialization via reconfigurability
  - more sophisticated algorithms (helped by reconfigurable hardware)

1-Mar-05 (35)

# Modular Exponentiation using Parallel Multipliers

S.H. Tang, K.S. Tsui and P.H.W. Leong  
*Custom Computing Laboratory*  
*The Chinese University of Hong Kong*  
<http://www.cse.cuhk.edu.hk/~phwl>

1-Mar-05 (36)

## This work

- High radix implementation of modular exponentiation
  - Utilizes dedicated multipliers of Virtex II devices
  - Systolic design
  - High radix

1-Mar-05 (37)

## Modular Exponentiation Algorithm ( $P=C^E$ )

- L-R binary exponentiation method
- On average takes  $1.5h$  multiplications
- Multiplications are modulo  $M$

```
P = 1;
for i = h-1 .. 0 do
{
  P = P x P;
  if  $e_i = 1$ 
    P = P x C;
}
```

1-Mar-05 (38)

## The Montgomery Algorithm

- Used in most software and hardware implementations of modular exponentiation
- Performed in an  $R$ -residue where  $R$  is chosen as a power of 2
  - Divisions are shift operations
  - Reduction is interleaved with the multiplication

1-Mar-05 (39)

## Montgomery Multiplication

- Input: A, B and  $M'$
- Output:  $S = ABR^{-1} \pmod{M}$
- Precompute  $M'$  s.t.  $(-M m') \pmod{R} = 1$
- Note that  $R^{-1}$  term is unwanted

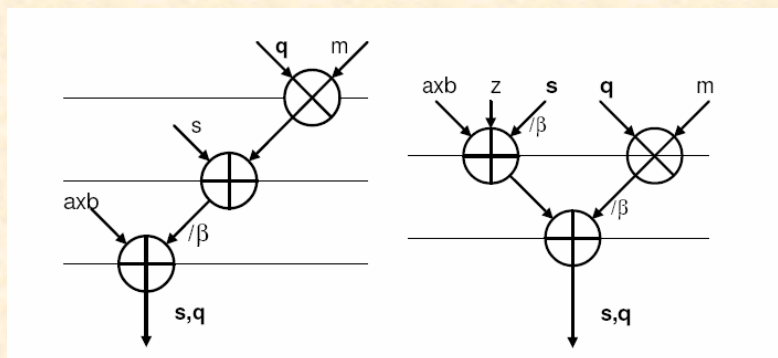
```

for i = 0 ... n do
{
    q = ((s mod β) × (M' mod β)) mod β
    s = (s + qM) / β + a_i B
}

```

1-Mar-05 (40)

Orup's optimization simplifies calc of q and reduces logic levels



$$S = (S + qM) / \beta + a_i B$$

$$S = S / \beta + qM / \beta + z + a_i B$$

1-Mar-05 (41)

## Optimized Montgomery Multiplication (Orup)

- Input: A, B and  $M'$
- Output:  $S = AB R^{-1} \pmod{M}$

```

 $\mathcal{M} = M (M' \bmod \beta)$ 
for i = 0 ... n do
{
    q = s mod  $\beta$ 
    z = 1 if q  $\neq$  0 else 0
    s =  $S/\beta + (q\mathcal{M})/\beta + z + a_i B$ 
}

```

1-Mar-05 (42)

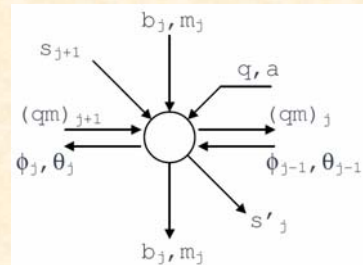
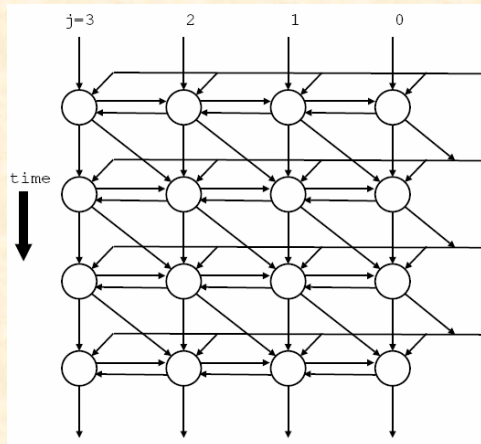
## Exponentiation using Montgomery Multiplication

- Extra pre and post processing steps are needed to remove unwanted  $R^{-1}$  term.

|  |  |
|--|--|
| $C^2 \leq (C)(C)$ $C^4 \leq (C^2)(C^2)$ $C^5 \leq (C^4)(C)$ $C^{10} \leq (C^5)(C^5)$ | $CR \leq (C)(R^2)R^{-1}$ $C^2R \leq (CR)(CR)R^{-1}$ $C^4R \leq (C^2R)(C^2R)R^{-1}$ $C^5R \leq (C^4R)(CR)R^{-1}$ $C^{10}R \leq (C^5R)(C^5R)R^{-1}$ $C^{10} \leq (C^{10}R)(1)R^{-1}$ |
|--|--|

1-Mar-05 (43)

## Systolic Design



- Multiplier is too large for  $qM$  and  $a_iB$
- Use a systolic array structure
  - Project vertically (row)

1-Mar-05 (44)

## Processing element

$$\theta_j = q \times m_j / \beta$$

$$\phi_j = a \times b_j / \beta$$

$$(qm)_j = \langle q \times m_j \rangle_\beta + \theta_{j-1}$$

$$(ab)_j = \langle a \times b_j \rangle_\beta + \phi_{j-1}$$

$$s_j = s_{j+1} + (qm)_{j+1} + (ab)_j$$

1-Mar-05 (45)

## Carry Save Adder

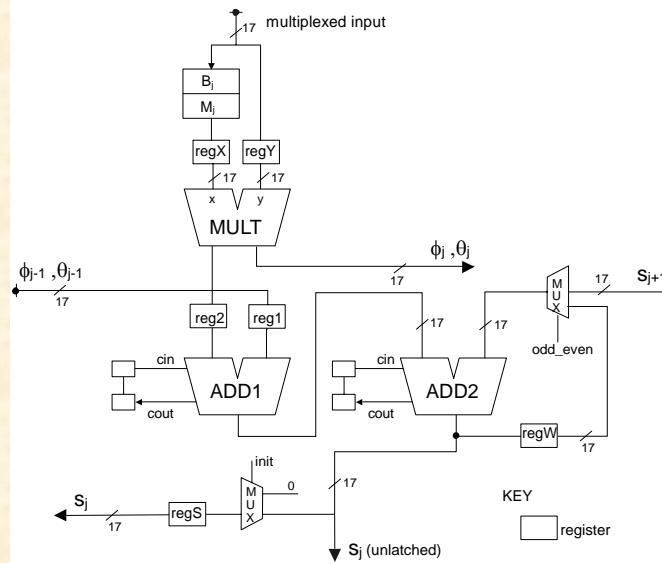
- The carry propagates through all PEs
  - 1024b addition takes 70 ns
  - Fast carry broken into 4 segments
- Use carry save architecture
  - The carry is saved in individual PE and used in later cycles
  - Addition is 17 bits

1-Mar-05 (46)

## PE Design

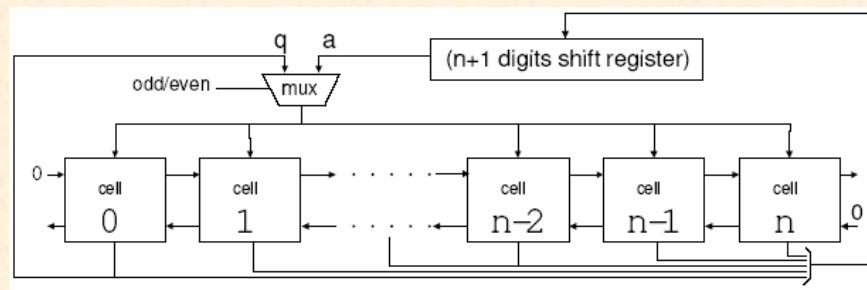
- Single multiplier per cell (time multiplexed)
- Two clock cycles for each PE to compute S (pipelined)
  - ab on even cycles
  - qm on odd cycles
- Multiplex inputs to reduce wiring

# PE Block Diagram



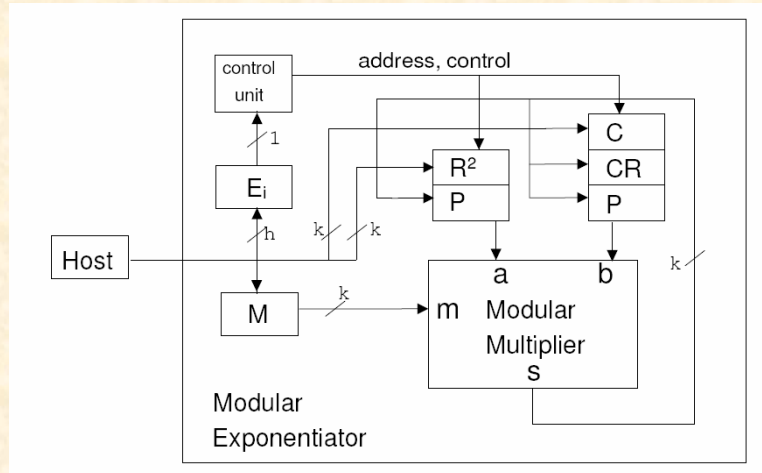
# Semi-Systolic Array

- Has broadcast signals but reduces latency



1-Mar-05 (49)

# Exponentiation



1-Mar-05 (50)

# Results

- Platform: XC2V3000-6, XC2V4000-6
- Design description: VHDL
- Can work w/ or w/o CRT for RSA
- Designs of 512-bit and 1024-bit key are implemented
- All designs use a radix of  $2^{17}$

1-Mar-05 (51)

## Clock Cycles

- Multiplication
  - $2n$  cycles ( $n$  = number of digits)
- Exponentiation
  - $3h/2$  cycles ( $h$  = number of exponent bits)
- Total
  - Approximately  $3nh$  cycles (i.e.  $O(h^2)$ )

1-Mar-05 (52)

## Clock Cycles

| Operation                          | Generalized <sup>1</sup>                 | 512b  | 1024b  |
|------------------------------------|--|-------|--------|
| <b>Modular Multiplication</b>      |  |       |        |
| MP × MP                            | $2(n + 5)$                               | 74    | 134    |
| <b>Modular Exponentiation</b>      |  |       |        |
| Montgomery pre/post-processing     | $2(n + 5) \times 3$                      | 222   | 402    |
| Exponentiation                     | $2(n + 5) \times (b - h + \frac{3h}{2})$ | 59200 | 209844 |
| <b>Input / Output</b>              |  |       |        |
| Input of $R^2, M, A, E$            | $4 \times (b/64) + 1$                    | 37    | 69     |
| Output of $A^E \bmod M$            | $(b/64) + 1$                             | 10    | 18     |
| <b>Measured total clock cycles</b> |  | 59468 | 210333 |

<sup>1</sup> $b$  – number of gross bits, 544 and 1054 for  $r=17$  and  $h=512, 1024$  respectively

1-Mar-05 (53)

## Results

- Operating frequency: 90MHz
- Resources
  - 14334 of 14336 slices
  - 62 of 96 multipliers
- Using CRT, 1024-bit RSA can be computed in 0.66ms
  - i.e. 1.5 Mbps throughput
  - 10 times faster than an Intel P4 1.7GHz (OpenSSL with all optimizations)
- Software verification using OpenSSL library routines and random data

1-Mar-05 (54)

## Performance Comparison

| Author      | Year | Technology      | Speed (1024b decryption) |
|-------------|------|-----------------|--------------------------|
| Orup et al  | 1991 | ASIC 0.6u       | (512b) 5.5 ms            |
| Shand et al | 1993 | 16x XC3090      | 6.1 ms                   |
| Itoh et al  | 1999 | TMS320C6201 DSP | 11.7 ms                  |
| Blum et al  | 2001 | XC40250XV-09    | 3.1 ms                   |
| OpenSSL     | 2003 | P4 1.7 GHz      | 6.9 ms                   |
| THIS WORK   | 2003 | XC2V4000-6      | 0.7 ms                   |

## Conclusion

- Systolic array for modular exponentiation
- High clock frequency
  - Aggressive pipelining (systolic design)
  - Carry save adder
- Low number of clock cycles (approx 3nh)
  - Orup's improved Montgomery Multiplication algorithm
  - 17x17 multipliers used to achieve high radix ( $\beta=2^{17}$ )
  - Reduced latency
- Parallelism
  - Multipliers and adders are used every clock cycle
- Performance of 1.5 Mb/s achieved on an XC2V4000-6 device

## References

- M. Shand and J. Vuillemin, "Fast Implementations of RSA Cryptography", Proc. 11th IEEE Symposium on Computer Arithmetic, Canada, July 1993.
- D. Knuth, "The Art of Computer Programming", Volume 2 Seminumerical Algorithms, 3rd Edition, Addison-Wesley, 1997
- P. Kornerup, "A Systolic, Linear Array Multiplier for a Class of Right-Shift Algorithms", IEEE Transactions on Computers, Vol. 43(8), August 1994, pp 892-898.
- S.H. Tang, K.S. Tsui and P.H.W. Leong, "Modular Exponentiation using Parallel Multipliers," Proceedings of the 2003 IEEE International Conference on Field Programmable Technology (FPT), Tokyo, pp. 52-59, 2003