Mobile Netw Appl (2007) 12:281–291
DOI 10.1007/s11036-007-0019-z

New and Improved Architectures for Montgomery Modular Multiplication

M. Sudhakar · R. V. Kamala · M. B. Srinivas

Received: 13 August 2007 / Accepted: 15 August 2007 / Published online: 5 October 2007

Abstract In this paper an improved Montgomery multiplier, based on modified four-to-two carry-save adders (CSAs) to reduce critical path delay, is presented. Instead of implementing four-to-two CSA using two levels of carry-save logic, authors propose a modified four-to-two CSA using only one level of carry-save logic taking advantage of pre-computed input values. Also, a new bit-sliced, unified and scalable Montgomery multiplier architecture, applicable for both RSA and ECC (Elliptic Curve Cryptography), is proposed. In the existing word-based scalable multiplier architectures, some processing elements (PEs) do not perform useful computation during the last pipeline cycle when the precision is not equal to an exact multiple of the word size, like in ECC. This intrinsic limitation requires a few extra clock cycles to operate on operand lengths which are not powers of 2. The proposed architecture eliminates the need for extra clock cycles by reconfiguring the design at bit-level and hence can operate on any operand length, limited only by memory and control constraints. It requires 2~15% fewer clock cycles than the existing architectures for key lengths of interest in RSA and 11~18% for binary fields and 10~14% for prime fields in case of ECC. An FPGA implementation of the proposed architecture shows that it can perform 1,024-bit modular exponentiation in about 15 ms which is better than that by the existing multiplier architectures.

Keywords Montgomery modular multiplication · RSA · ECC · carry save adders · reconfigurable multiplier · scalable multiplier

1 Introduction

Public key cryptography is used in several applications such as e-commerce, mobile communications, etc. for providing confidentiality, authentication, key establishment, non repudiation and data integrity [1, 2]. The different public key cryptographic algorithms include widely deployed RSA [3], Diffie Hellman key exchange [4], Digital Signature Standard [5] and more recent ECC [6–8]. While RSA is popular in most of the cryptographic applications, ECC is increasingly becoming an attractive solution because it offers same level of security with smaller key sizes. This is possible due to the absence of a sub-exponential time algorithm that could solve the discrete logarithm problem (DLP) [9]. For example, it has been proved that a 163-bit ECC key size provides same level of security as an equivalent 1024-bit RSA key [10]. The result is potential bandwidth savings and faster implementations, features which are especially attractive for security applications where computational power and integrated circuit space are limited, such as smart cards and wireless devices. While these public-key encryption schemes can be implemented in software, it's a challenge to implement them in hardware while maintaining high data throughput with optimal area and performance constraints.

The performance of the public-key cryptosystems depends heavily on the implementation of the underlying

M. Sudhakar (✉) · R. V. Kamala · M. B. Srinivas
Center for VLSI and Embedded System Technologies,
International Institute of Information Technology,
Hyderabad, Andhra Pradesh, India 500032
e-mail: sudhakarmaddi@research.iiit.ac.in

R. V. Kamala
e-mail: rvkamala@students.iiit.ac.in

M. B. Srinivas
e-mail: srinivas@iiit.ac.in

modular arithmetic operations. Modular multiplication is the basic operation for both RSA and ECC crypto systems but employs expensive division operations. To avoid this, Montgomery [11] proposed an algorithm which replaces division operations with simple shift operations that are comparatively easy to perform in hardware. Montgomery multiplication is defined both in prime GF(p) and binary extension fields GF(2^n). Multiplication in GF(p) is performed modulo some prime p while multiplication in GF (2^n) is performed modulo some irreducible polynomial $f(x)$ of degree of n.

Major bottleneck to perform modular arithmetic in GF (p) is the carry propagation during addition/subtraction operations. In GF(p), use of carry propagate adders increases the critical path delay thereby deteriorating the maximum frequency of operation. To circumvent this problem, Kim et al. [12] used carry save adders (CSA) consisting of two levels of carry save logic (CSL). Bunimov et al. [13] improved this by replacing one level of CSL with a look-up table. An important limitation of these designs, however, is the conversion between input and output formats which is expensive for applications where repeated multiplications are required, for example, RSA exponentiation. To overcome this problem, McIvor et al. [14] proposed two algorithms based on using five-to-two CSA (three levels of CSL) and a four-to-two CSA (two levels of CSL), respectively. These are the first RSA units which perform Montgomery multiplication with less number of clock cycles and therefore result in high data rates. Also, the critical path delay is word-length independent. In Sudhakar et al. [15], the authors presented an efficient reconfigurable, *bit-sliced* Montgomery multiplier that can operate in both fields, GF(p) and GF(2^n) using four-to-two CSA.

In this paper, the authors first modify their earlier architecture [15] by eliminating one dual field adder (DFA) in four-to-two CSA while pre-computing some constant values resulting in reduced critical path delay that is also independent of the input operand precision. Secondly, a bit-sliced and scalable architecture is proposed that can handle operands of any precision. Hardware designs are said to be scalable if it is possible to reuse the same design in both space and time until the desired result is obtained. Practical implementations of scalable architectures exploit the inherent concurrency of Montgomery multiplication and use an array of processing elements (PEs) organized in a pipeline fashion. Normally these PEs compute modular multiplication on fixed size operand values (8, 16 or 32-bits) at a time. The existing word-based scalable architectures [16–20] can compute Montgomery multiplication efficiently for RSA because key-lengths in RSA are exact integer multiple of word-sizes. But according to certicom [21] and NIST DSS standards [22], the recommended key lengths in ECC are 163, 233, 283, 409, 571 for GF(2^n) and 192, 224, 256, 384, 521 for GF(p), most of which are not exact multiples of the word size. As discussed in [19], some PEs do not perform useful computation in the last pipeline cycle when key-lengths are not an exact multiple of the word-size. This intrinsic limitation requires a few extra clock cycles for each key-length in word-based architectures that also increase as the word size increases. Thus, there is a need to design efficient multiplier architectures which can operate on any key length while providing scalability and reconfigurability at the same time.

2 Montgomery multiplication algorithm

2.1 Background theory

Given an integer $a<p$, where p is the n-bit modulus, the p-residue of a with respect to r is defined as $A = a \times r(\bmod p)$ where $r=2^n$. Similarly, given an integer $b<p$, the image or p-residue of b with respect to r is defined as $B = b \times r(\bmod p)$. In GF(p), given two integers A, B and modulus p of length n bits, Montgomery multiplication is defined as MonPro $(A, B) = A \times B \times r^{-1}(\bmod p)$, where $r=2^n$ and p is an integer in the range $2^{n-1}<p<2^n$ such that gcd $(r, p)=1$. Since $r=2^n$, it is sufficient that the modulus p be an odd integer. For cryptographic applications, p is usually a prime number or a product of primes, thus this condition is easily satisfied. It is easy to show that the Montgomery multiplication over the p-residues A and B computes the p-residue C = MonPro (A, B), which corresponds to the integer $c = a \times b(\bmod p)$. The transformation to and from p-residues is accomplished using Montgomery multiplication:

$$A = \mathrm{MonPro}(a, r^2) = a \times r^2 \times r^{-1}(\bmod p) = a \times r(\bmod p)$$
$$a = \mathrm{MonPro}(A, 1) = A \times 1 \times r^{-1}(\bmod p) = a \times r \times 1 \times r^{-1}(\bmod p) = a$$

Usually r^2 (mod p) is pre-computed and saved; thus, only a single Montgomery multiplication is needed to perform either one of these transformations. A long sequence of multiplications, like those required in exponentiation, can be performed by converting the operands to p-residues, performing Montgomery multiplication, and converting the result back to an integer. The simple radix-2 Montgomery multiplication algorithm is presented in algorithm 1. The major bottleneck in it, however, is the carry propagation resulting from the very large operand additions. To overcome this, carry save adders (CSAs) have been used to carry out the multiplication. Four-to-two CSA is a well known method to calculate Montgomery multiplication efficiently, especially when

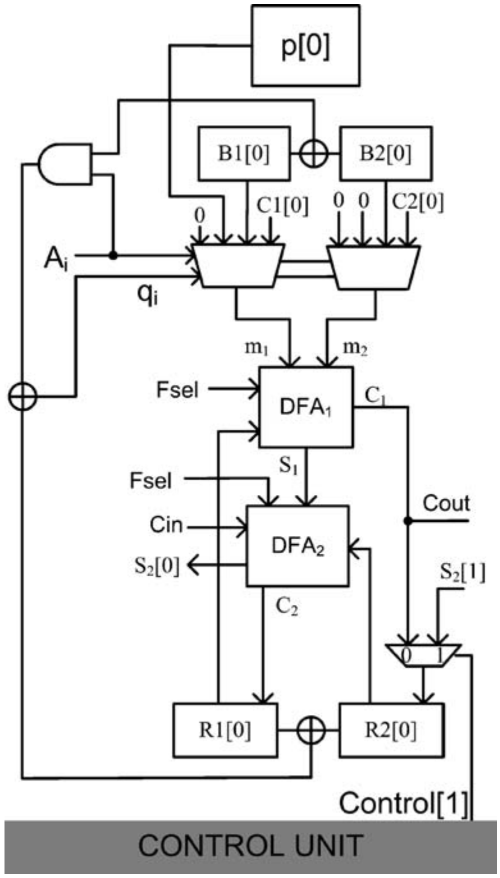

Figure 1 Data path unit of the design presented in Sudhakar et al. [15]

repeated multiplications are needed to be performed. McIvor et al. [14] discuss the superiority of four-to-two CSA in performing repeated multiplications to achieve high data rates and a critical path delay that is independent of word length. Their method calculates n-bit multiplication in just ‘n+2’ clock cycles.

Algorithm 1 Montgomery Multiplication Algorithm

$S[0] = 0;$
for i in 0 to n − 1 loop
$q_i = \left(S[i]_0 + A_i \times B_0\right) \bmod 2;$
$S[i+1] = (S[i] + A_i \times B + q_i \times p) \bmod 2;$
end loop;
return S[n];

In GF(2^n), given the input polynomials $A(x)$, $B(x)$ and the modulus $p(x)$, Montgomery multiplication is defined as MonPro $(A, B) = A(x) \times B(x) \times x^{-n}[\bmod p(x)]$. In a unified design, one can use a modified carry save adder that forces the carry to zero when operating in GF(2^n). In Sudhakar et al. [15], the authors proposed a unified Montgomery multiplication algorithm and carried out a hardware implementation based on four-to-two CSA that can be configured at bit-level. Also it computes n-bit Montgomery multiplication in ‘n+2’ clock cycles with a critical path delay comparable to that in [14]. The algorithm proposed in [15] is presented below (algorithm 2) for convenience. Given two operands ($A1$, $A2$) and ($B1$, $B2$) and the modulus p, it computes the Montgomery multiplication for either field. Note that ($A1$, $A2$) and ($B1$, $B2$) are the carry-save representation of A and B. For multiplication in GF(2^n), $B1$, $A1$, $C1$ and $R1$ are always kept to zero. Since prime field uses carry-save arithmetic and binary extension field is free from carry, a modified carry-save adder, named as Dual Field Adder (DFA) is used to force the carry to zero [15]. Figure 1 depicts data path unit for a single bit-slice design of the architecture presented in Sudhakar et al. [15] that determines the critical path delay.

Algorithm 2 Unified Montgomery Multiplication Algorithm

Inputs	A1, A2, B1, B2, p, FSEL
Outputs	R1[n], R2[n]

C1, C2 = DCSR (B1 + B2 + p + 0)
R1[0] = 0
R2[0] = 0
for i = 0 to n − 1
  $q_i = (R1[i]_0 + R2[i]_0) + (A_i \times (B1_0 + B2_0)) \bmod 2$
  if $A_i = 0$ and $q_i = 0$ then
    R1[i + 1], R2[i + 1] = DCSR (R1[i] + R2[i] + 0 + 0) >> 1
  elseif $A_i = 1$ and $q_i = 0$ then
    R1[i + 1], R2[i + 1] = DCSR (R1[i] + R2[i] + B1 + B2) >> 1
  elseif $A_i = 0$ and $q_i = 1$ then
    R1[i + 1], R2[i + 1] = DCSR (R1[i] + R2[i] + p + 0) >> 1
  else
    R1[i + 1], R2[i + 1] = DCSR (R1[i] + R2[i] + C1 + C2) >> 1
end for
return R1[n], R2[n]

2.2 Modified bit-sliced Montgomery multiplier architecture

In this section the modified bit-sliced multiplier architecture, that can reduce area and the critical path delay, is discussed. Figure 2 depicts the data path unit of the modified bit-sliced architecture. In this unit, one dual field adder (DFA_1) is eliminated by taking the advantage of pre-computed values, as explained below:

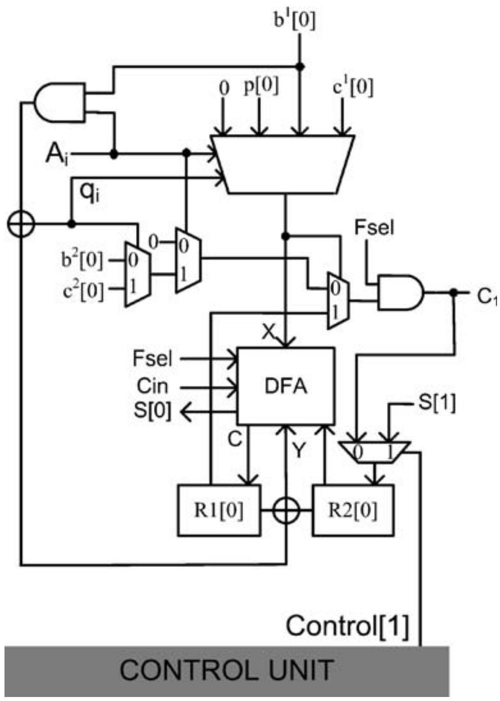

Figure 2 Modified data path unit

Since DFA is a conventional full adder with extra *Fsel* signal [15], from Fig. 1, the sum (S_1) and carry (C_1) outputs of first DFA (DFA_1) can be written as

$$\begin{aligned} S_1 &= m_1 \oplus m_2 \oplus R1[0] \\ &= X \oplus R1[0] \end{aligned} \tag{1}$$

$$\begin{aligned} C_1 &= m_1 m_2 + m_2 R1[0] + R1[0] m_1 \\ &= [(m_1 \oplus m_2) R1[0] + m_1 m_2] \times \text{Fsel} \\ &= \left[(m_1 \oplus m_2) R1[0] + \overline{(m_1 \oplus m_2)}.m_2\right] \times \text{Fsel} \\ &= \left[X.R1[0] + \overline{X}.m_2\right] \times \text{Fsel} \end{aligned} \tag{2}$$

where X represents $(m_1 \oplus m_2)$, m_1 and m_2 being the two 4:1 multiplexer outputs, and $R1[0]$ is the value of 0th bit output register $R1$. In general, ith slice m_1 and m_2 values are selected from a group of values (0, $p[i]$, $B1[i]$, $C1[i]$) and (0, 0, $B2[i]$, $C2[i]$) respectively based on the selected lines A_i and q_i as shown in Table 1. Hence the possible values for x will be either 0 or p or $(B1[i] \oplus B2[i])$ or $(C1[i] \oplus C2[i])$. Note that in Fig. 2, $b^1[0]$, $c^1[0]$, $b^2[0]$ and $c^2[0]$ represent the values $(B1[0] \oplus B2[0])$, $(C1[0] \oplus C2[0])$, ($B1[0]$ and $B2[0]$) and ($C1[0]$ and $C2[0]$), respectively. Since all these values remain constant throughout the multiplication, they can be pre-computed.

Similarly, the sum (S_2) and carry (C_2) outputs of second DFA (DFA_2) can be formulated as

$$\begin{aligned} S_2 &= S_1 \oplus Cin \oplus R2[i] \\ &= (m_1 \oplus m_2 \oplus R1[i]) \oplus Cin \oplus R2[i] \\ &= (m_1 \oplus m_2) \oplus (R1[i] \oplus R2[i]) \oplus Cin \\ &= X \oplus Y \oplus Cin \end{aligned} \tag{3}$$

$$\begin{aligned} C_2 &= \left\{(S_1 \oplus R2[j])Cin + \overline{(S_1 \oplus R2[j])}\, \text{R2}[j]\right\} \times \text{Fsel} \\ &= \left\{(X \oplus Y)Cin + \overline{(X \oplus Y)}\, R2[i]\right\} \times \text{Fsel} \end{aligned} \tag{4}$$

where Y represents $(R1[i] \oplus R2[i])$. Thus one dual field adder (DFA) can be eliminated from the design and as a consequence, a two EX-OR gate delay reduction is obtained in the critical path compared to the earlier design [15]. This significantly increases the maximum clock frequency of operation and also results in reduced area. Table 2 provides a comparison of the modified data path unit with previous CSA-based approaches [12–15].

An N-bit bit-sliced Montgomery multiplier architecture, incorporating the modified four-to-two CSA, is shown in Fig. 3. It can operate in either prime field GF(p) or binary extension field GF(2^n). If *Fsel*=1, it works in GF(p) mode else in GF(2^n) mode. In addition, it can be configured for any operand length 'n' ($n < N$) by generating control signals from the control unit. The working principle of this multiplier is similar to that proposed in [15] and hence is not elaborated further here.

3 A new scalable, unified and reconfigurable Montgomery multiplication algorithm

In the previous section, a modified, bit-sliced but non-scalable Montgomery multiplier architecture has been presented. Since area is critical for multiplication on large operands, a new bit-sliced, scalable and unified multiplier architecture is presented in this section. Algorithm 3 describes the proposed scalable and unified Montgomery multiplication algorithm.

Table 1 Different possible X values

Selection lines		m_1	m_2	$X = (m_1 \oplus m_2)$
A_i	q_i			
0	0	0	0	0
0	1	$P[i]$	0	$p(i)$
1	0	$B1[i]$	$B2[i]$	$b^1[i] = B1[i] \oplus B2[i]$
1	1	$C1[i]$	$C2[i]$	$c^1[i] = C1[i] \oplus C2[i]$

Table 2 A comparison of different CSA based multiplier architectures

Reference	Critical path delay	Conversion delay	Clock cycles	Unified/ reconfigurable
[12]	32 full adders	n/32 iterations of 32-bit CPA	n+34 (for n=1,024)	No/no
[13]	Carry propagation of n conventional adders	n-bit conventional adder	Not documented	No/no
[14][a]	2 full adders + 4:1 multiplexer + 2 XORs + 1 AND	None	n+2	No/no
[15]	2 full adders + 4:1 multiplexer + 2:1 Multiplexer + 2 XORs + 1 AND	None	n+2	Yes/yes
Proposed	1 full adder + 4:1 multiplexer + 2 2:1 multiplexer + 2 XORs + 1 AND	none	n+2	Yes/yes

[a] Considered four-to-two CSA algorithm

Algorithm 3 Scalable and Unified Montgomery Multiplication Algorithm

Inputs	$A1, A2, B1, B2, p, FSEL$
Outputs	$R1^{(n)}, R2^{(n)}$

$C1, C2 = DCSR(B1 + B2 + p + 0)$
$R1_0^{(0)} = 0,\ R2_0^{(0)} = 0,$
$e = \lceil n/N \rceil$
for $i = 1$ to n
  $q_i = (R1_0^{(i-1)} + R2_0^{(i-1)}) + (A_i \times (B1_0 + B2_0)) \bmod 2$
  for $j = 1$ to e
    $m = \begin{cases} N, & j \neq e \\ n - ((e-1) \times N), & j = e \end{cases}$
    for $k = (1 + (j-1) \times N)$ to $(m + (j-1) \times N)$
      if $A_i = 0$ and $q_i = 0$ then
        $R1_k^{(i)}, R2_k^{(i)} = DCSR(R1_k^{(i-1)} + R2_k^{(i-1)} + 0 + 0) >> 1$
      elseif $A_i = 1$ and $q_i = 0$ then
        $R1_k^{(i)}, R2_k^{(i)} = DCSR(R1_k^{(i-1)} + R2_k^{(i-1)} + B1_k + B2_k) >> 1$
      elseif $A_i = 0$ and $q_i = 1$ then
        $R1_k^{(i)}, R2_k^{(i)} = DCSR(R1_k^{(i-1)} + R2_k^{(i-1)} + p_k + 0) >> 1$
      else
        $R1_k^{(i)}, R2_k^{(i)} = DCSR(R1_k^{(i-1)} + R2_k^{(i-1)} + C1_k + C2_k) >> 1$
    end for
  end for
end for
return $R1^{(n)}, R2^{(n)}$

This algorithm computes the partial results of $R1$ and $R2$ for one bit of A, scanning the present bit-values of $B1$, $B2$, p, $C1$ and $C2$ from corresponding registers. Let's represent the time to execute one bit of A as one *process cycle*. The next process cycle starts by taking another bit of A from the barrel shifter [15]. For $e = \lceil n/N \rceil$, the hardware is reused e times (e being the number of clock cycles) for each process cycle. In the representation $X_Z^{(Y)}$, superscript (Y) represents the process cycle whereas subscript (Z) represents the bit number. In this algorithm, outer-most loop represents one process cycle which operates for single bit of A. The second outer-loop corresponds to the scalability feature and the inner loop computes modular multiplication on m bits for every clock cycle, which is determined by the control unit.

For $n \leq N$, e equals 1 which means that each process cycle consists of one clock cycle. The remaining (N–n) bit-slices are inactive as discussed in [15]. For $n > N$, e is not equal to 1 meaning that for each bit of A (one process cycle), the hardware is used e times by calculating m value for every clock cycle. The dependency graphs for different values of N and n are shown in Fig. 4, where each circle represents a single bit-slice. The ability to deactivate the unused bit-slices distinguishes the proposed design from the existing word-based multipliers.

3.1 Architecture for bit-sliced, scalable Montgomery multiplication

Figure 5 shows the block diagram of the proposed bit-sliced, scalable Montgomery multiplier. At architecture level, it resembles the existing scalable word-based architectures [19, 20] in some aspects like control section, RAM, shift register (barrel shifter) and queue. The processing unit (PU) contains N bit-slices and register controller. Functionally, it can be assumed that a w-bit processing element (PE) of [19, 20] is equivalent to w bit-slices. The RAM modules are used to store $B1$, $B2$, p, $C1$ and $C2$ while the A_i bits come from an n-bit barrel shifter as presented in [14]. Since $R1^{(j)}$ and $R2^{(j)}$ values are computed based on $R1^{(j-1)}$, $R2^{(j-1)}$ and current input values, the partial results of $R1$ and $R2$ must be queued until the next processing cycle starts for $n > N$. The final and intermediate results are stored in a queue whose maximum size depends on values of e and N. The control block function can be inferred from the description of the algorithm provided.

3.2 Functional description of Processing Unit (PU)

Figure 6 depicts the internal design of the N-bit PU. It consists of N bit-slices, two rows (not shown in the figure

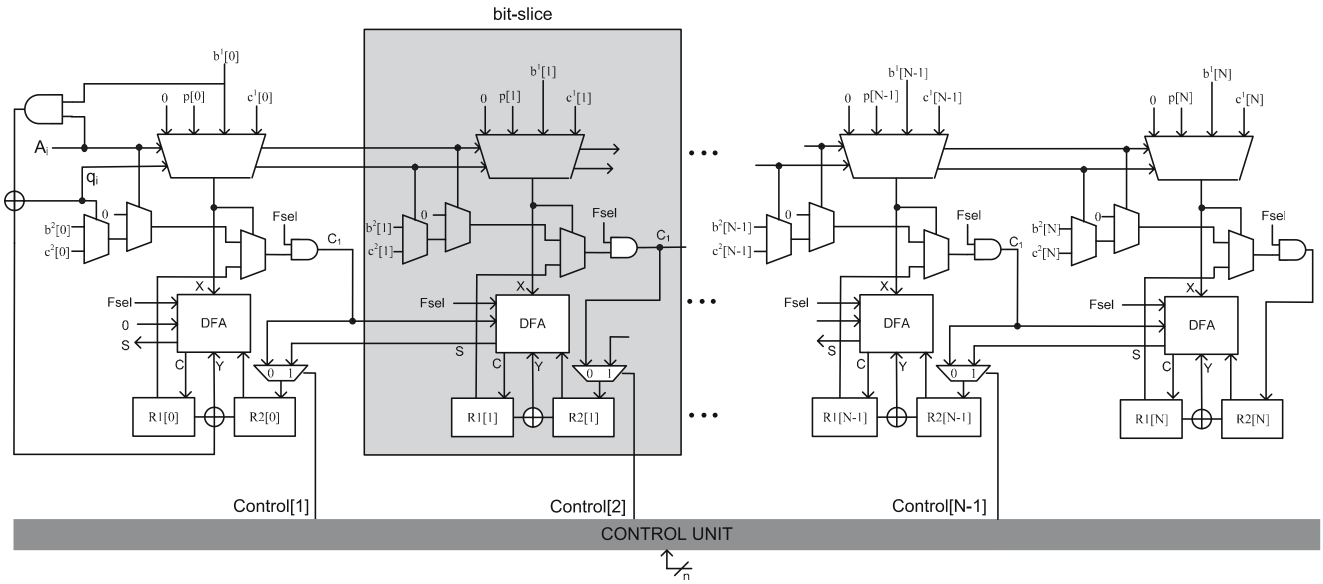

Figure 3 Modified reconfigurable and unified Montgomery multiplier architecture

for simplicity) of $R1$ and $R2$ registers and some control logic. The bit-slices are numbered from 1 to N while the inputs and outputs are in carry-save representation. Each slice consists of three 2:1 multiplexer, one 4:1 multiplexer, one dual field adders (DFA) and a few one-bit registers. When *Fsel* is active high, the multiplier acts as a scalable multiplier in GF(p) and when active low, it acts as a scalable multiplier in GF(2^n).

Upon reset, $R1$ and $R2$ registers are initialized with zeros and register p is loaded with the modulus while the control unit generates the control signals for the slices by taking the operand length m as input. In the same clock cycle, the registers B and C are loaded with the corresponding pre-computed values of b^1 and c^1. The bit A_i is computed from $A1$ and $A2$ using the barrel shifter for every process cycle. In GF(2^n) mode, $A1$ is set to zero and the multiplication can be performed with $A2$ shifted right by one bit for every process cycle. The value of control signal for ith slice generated by the control unit is defined as:

$$\text{control}[i] = \begin{cases} 1 & \text{if } n < N \\ 0 & \text{otherwise} \end{cases}$$

where control[i]=1 indicates that the slice is active whilst control[i]=0 indicates the inactive state. When selection line is zero, ith slice carry is stored in $R2$ register else (i+1)th slice DFA sum is stored.

While working in scalable mode ($n > N$), a few tasks need to be considered to get the correct solution. The bit q_i should be latched until next process cycle starts and the signal En is used for this purpose. Since there is interdependency between two successive bit-slices, care should be taken while dealing with the last slice. The Nth slice carry output (C^*) should be passed as an input to first slice for each clock cycle until next process cycle starts.

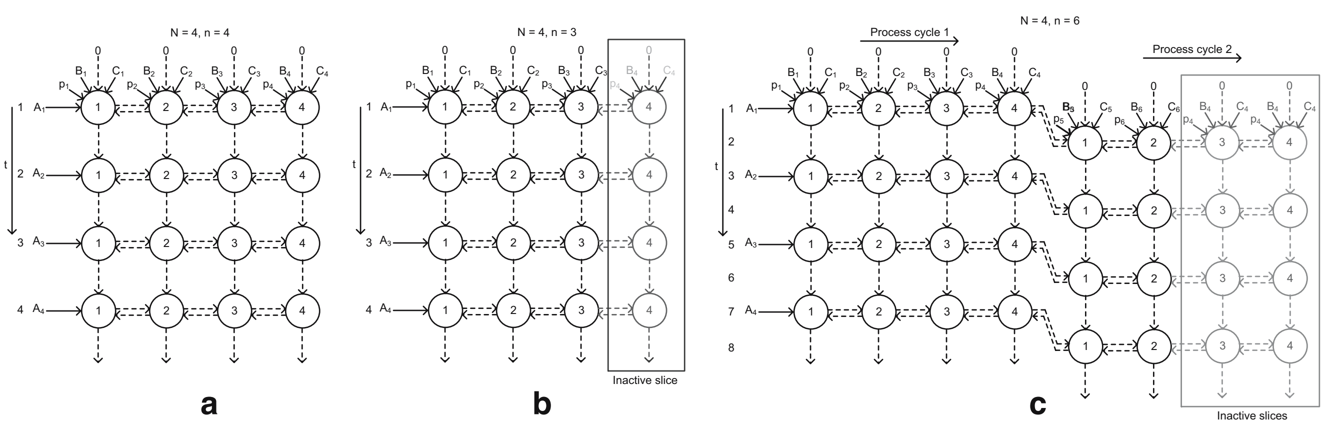

Figure 4 Dependency graph for **a** N=4, n=4; **b** N=4, n=3; and **c** N=4, n=6

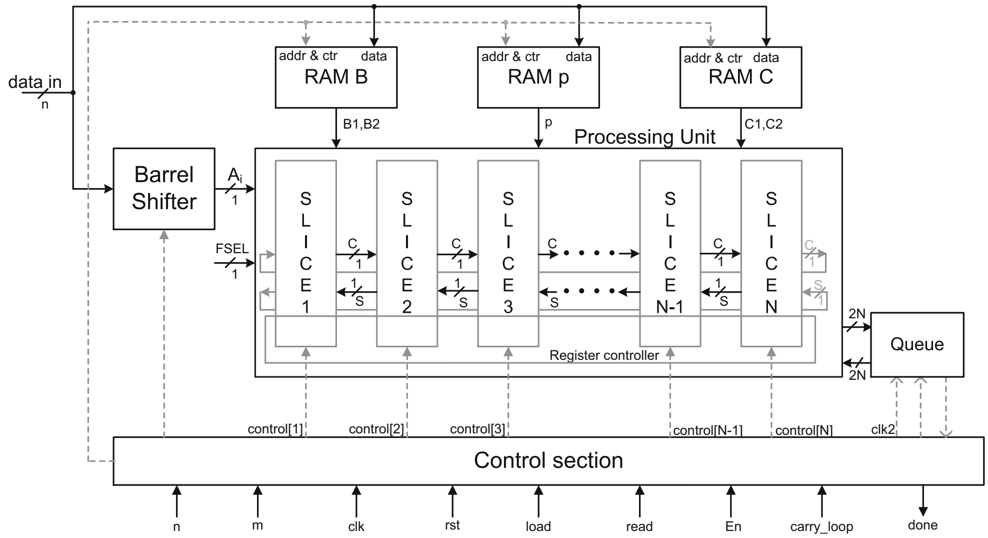

Figure 5 Proposed unified, scalable and reconfigurable Montgomery Multiplier Architecture

Once this process cycle is over, zero should be assigned for next bit of A. For this, a 2:1 multiplexer is used with *carry_loop* as a select signal which is generated by the control unit. As ith slice $R2$ register is loaded with $(i+1)$th slice sum, it takes one clock cycle delay for Nth slice $R2$ register to get next slice (0th slice) sum, shown as S^*. To store these values, an e-bit shift register is used internally.

A two-cycle latency is required to read/write $R1$ and $R2$ values from/into FIFO and routing. In order to eliminate this delay two rows of $R1$ and $R2$ registers are used which can be inter-operable by using a 2×2 crossbar switch as shown in Fig. 7 and another clock signal (clk2) for FIFO which is double the main clock frequency. Note that Digital Delay-Locked Loops (DLLs) or Digital Clock Managers (DCMs) in all Virtex series FPGA devices provide frequency doubling internally [23, 24]. While developing the verilog code, clk2 frequency is considered as double the main clock frequency. For every main clock signal these two rows of R1 and R2 registers attach or detach alternatively by toggling the control signal *sel* for every clock cycle. The detached row's partial results are loaded into FIFO on first clock cycle of clk2 and reloaded with previous values which are stored in FIFO on the next clock cycle of clk2 so that it does not effect the normal flow of execution of the multiplier. Note that the Nth slice R2 register is loaded with the e-bit shift register values. For $n < N$, all these tasks are not required and multiplication result can be achieved by assigning En=1 and *carry_loop*=1 throughout the execution. The total computation time T in clock cycles when N bit-slices are used to compute Montgomery multiplication with n bits of precision is

$$T = \begin{cases} n+2 & n \le N \\ \lceil n/N \rceil \times N + 2 & n > N \end{cases}$$

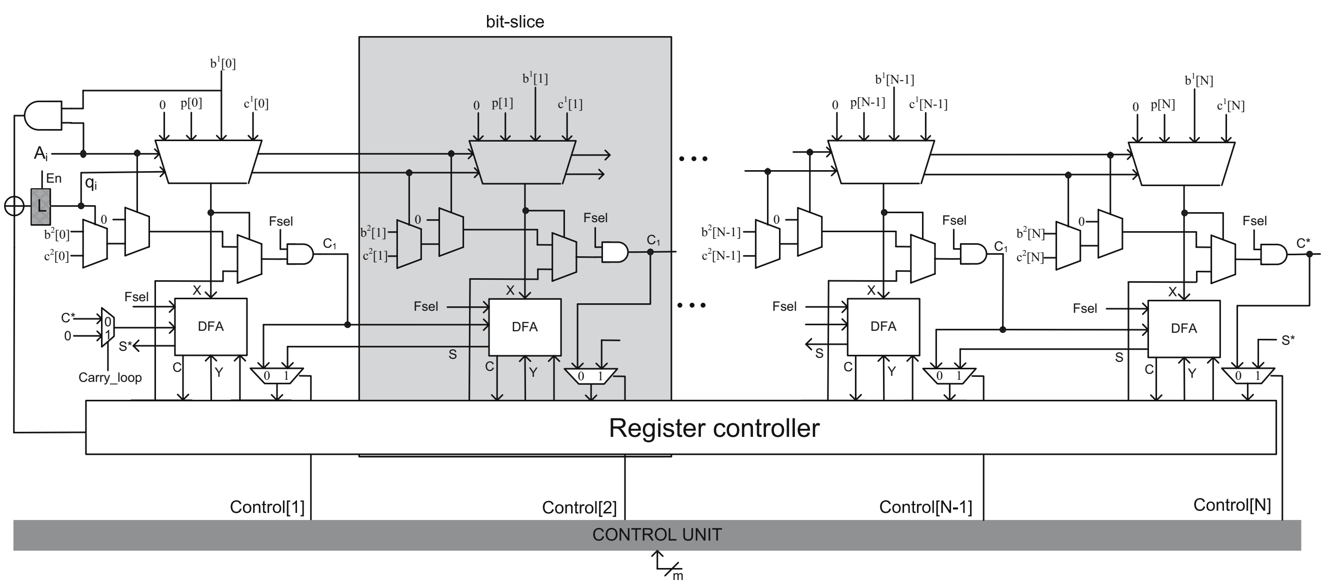

Figure 6 Internal design of an N-bit Processing Unit (PU)

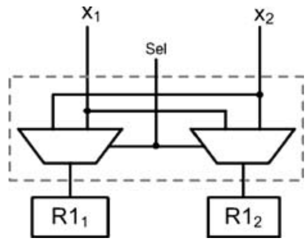

Figure 7 2×2 Crossbar switch

First case corresponds to a non-scalable mode while second one corresponds to scalable mode.

4 Results and comparison

The proposed unified, bit-sliced and scalable architecture has been coded and verified using Verilog HDL for different values of n and N. The design has been synthesized using Leonardo spectrum targeting Xilinx Virtex-E v50ecs144 (speed grade-8) FPGA [23] which achieved a maximum clock frequency of 140 MHz for 1,024-bit modular multiplication that is comparable to the existing architectures. The results however have not been verified on an ASIC implementation. The area results are presented at the end of this section for N=256 and 1,024 bit-slices. RAMs for the input and output operands and queue depend on the operand precision and are not considered here. But 5n bits of storage for $B1$, $B2$, p, $C1$ and $C2$ and $e \times (2N-1)$ bits for queue are required, approximately. In the next two subsections, the authors provide a comparison of the proposed scalable architecture with existing ones in terms of number of clock cycles for both ECC and RSA key-lengths.

4.1 A comparison for different ECC key-lengths

Table 3 lists the total computational time in terms of number of clock cycles for recommended ECC key-lengths. Großschädl [17] proposed a unified and bit-serial multiplier architecture in which modular multiplication was carried out based on MSB-first shift-and-add method. While it computes n-bit multiplication in ‘n’ clock cycles in GF(p), it requires approximately 1.5n clock cycles in GF(2^n) which is high compared to that achieved in present work. Since the architectures presented in [18–20] are word-based architectures, the authors consider word-size as 32 in this paper for the sake of comparison with best values. Note that the computational time decreases as the word-size increases. Also, since the maximum key-length in ECC is 571 bits, it is considered that the number of bit-slices present in the hardware as 576 (wp=576 for word-based architectures and N=576 for the proposed one).

Savas et al. [18] proposed a dual-radix multiplier architecture which operates in radix-2 in GF(p) mode and in radix-4 in GF(2^n) mode. Thus, they achieved an almost 50% reduction in number of clock cycles in GF(2^n) mode compared to GF(p) mode. Tenca et al. [19] described in detail the design trade offs in terms of area and time for different word sizes. Recently, Harris et al. [20] improved Tenca-Koc multiplication algorithm [19] to eliminate the two-clock cycle latency from one PE to the next which results in fewer clock cycles. As described in Tenca and Koc [19], for n=5, w=1 and e=6, two processing elements (p=2) require 23 clock cycles to compute the modular multiplication while the proposed architecture requires only 17 clock cycles with N=2, because of the ability to configure the architecture at bit-level rather than at word-level. For example, the percentage of improvement in number of clock cycles for the proposed architecture over that presented in [20] is shown Table 3 which shows the inefficiency of word-based architecture for ECC key-lengths. Referring to Table 3, it is clear that the proposed design requires fewer clock cycles compared to [17–20] for all practical ECC key-lengths.

Table 3 Scalable Montgomery multiplier latencies (clock cycles) for recommended ECC key-lengths

ECC key Lengths		Großschädl [17]	18 PEs×32-bit=576			Proposed N=576	% improvement over [20]
			Savas et al. [18]	Tenca and Koc [19]	Harris et al. [20]		
Binary field	163	163	175	374	201	165	18
	233	233	249	487	262	235	11
	283	283	300	599	321	285	11
	409	409	434	862	462	411	11
	571	571	599	1,200	643	573	11
Prime field	192	288	391	411	220	194	12
	224	336	456	486	260	226	13
	256	384	529	561	300	258	14
	384	576	783	824	441	386	13
	521	782	1,062	1,088	584	523	10

Table 4 Scalable Montgomery multiplier latencies (clock cycles) for RSA key-lengths

Bit cells	N	Tenca and Koc [19] w=32	Harris et al. [20] w=32	Percent improvement over Tenca and Koc [19] (%)	Proposed	Percent improvement over Harris et al. [20] (%)
256	256	550	303	45	258	15
	512	1,102	1,111	−1	1,026	8
	1,024	4,238	4,263	−1	4,098	4
	2,048	16,654	16,711	0	16,386	2
512	256	534	287	46	258	10
	512	1,070	575	46	514	10
	1,024	2,142	2,159	−1	2,050	5
	2,048	8,350	8,399	−1	8,194	2
1,024	256	526	279	47	258	8
	512	1,054	559	47	514	8
	1,024	2,110	1,119	47	1,026	8
	2,048	4,222	4,255	−1	4,098	4

4.2 A comparison for different RSA key-lengths

Since RSA is the most widely used public-key cryptosystem, authors also discuss the performance of the proposed scalable architecture for RSA key-lengths. Even though the architectures proposed in [16–18] are scalable, they are not flexible in selecting the number of processing elements according to the input operand precision and the number of bit-cells present in the hardware. This is important when operating on larger precisions to get better time and area trade-offs. Table 4 provides a comparison of the proposed scalable multiplier architecture with those presented in [19, 20]. The architecture presented in Harris et al. [20] was shown, for operand precisions 'n' up to the number of bit cells wp, to be about twice as fast as that in Tenca and Koc [19]. For larger operand lengths, the performance of the two designs is comparable. However, being a bit-sliced architecture with parallel execution (Fig. 4), the proposed architecture requires fewer number of clock cycles (15~2% less) than that in Harris et al. [20] to compute Montgomery multiplication.

From Tables 3 and 4, it is clear that the proposed bit-sliced architecture requires fewer clock cycles for both RSA and ECC key lengths. In Table 5, a comparison of proposed scalable architecture with existing ones [18–20] in terms of area, maximum frequency of operation and the time to execute 256 and 1,024-bit modular exponentiation is presented. Here it is assumed that n-bit exponentiation requires at most $(2n+2)$ modular multiplications including the conversion to and from p-residues [20].

Even though redundant carry-save representation of the operands results in a critical path delay that is independent of the operand precision, the cost of CS representation comes from more registers and buses. This cost is even higher in a bit-sliced architecture due to bit-level computation. But in the proposed architecture, authors have used a modified four-to-two CSA which has only one level of CSL resulting in reduced area compared to Harris et al. [20] as shown in Table 5. From the results obtained, it is obvious that the improved multiplier architecture can be used for implementations where both area and performance are of concern.

5 Conclusion

In this paper, the authors proposed an improved bit-sliced, radix-2 Montgomery multiplier architecture based on modified four-to-two carry save adders (CSAs) that results

Table 5 A performance comparison of different scalable multiplier architectures

Reference	Bit cells	Technology	Hardware	Clock speed (MHz)	Scalable/unified	256-bit time (ms)	1,024-bit time (ms)
Proposed	N=256	Xilinx Virtex-E	1,417 LUTs	140	Yes/Yes	0.9	59.2
	N=1,024		5,412 LUTs			0.9	15
[20]	16 PEs×16 bits	Xilinx Virtex Pro	1,514 LUTs	144	Yes/Yes	1.1	59
	64 PEs×16 bits		5,598 LUTs			1.0	16
[19]	40 PEs×8 bits	0.5 μm CMOS	28 Kgates (kernel only)	80	Yes/Yes	3.8	88.2
[18]	16 PEs×16 bits	0.5 μm CMOS	28 Kgates	64	Yes/No	1.6	46

in reduced critical path delay and area. Also the authors proposed, possibly for the first time, a unified, bit-sliced and scalable Montgomery multiplier architecture useful for both RSA and ECC cryptosystems. It has been shown that the proposed architecture computes Montgomery multiplication in fewer clock cycles, for RSA as well as ECC key lengths, than the existing word-based multiplier architectures. Also a high degree of regularity and repetition of bit-slices at logic, circuit and silicon levels are the added advantages of the design. The proposed architecture has been implemented on FPGA only for the purpose of comparing with the existing architectures. Since it is a technology-independent architecture it is expected that an ASIC implementation may lead to a better performance.

References

1. Stallings W (1995) Network and internetwork security principles and practise. Prentice Hall, New York, USA
2. Menezes A, van Oorschot P, Vanstone S (1997) Handbook of applied cryptography. CRC Press, Florida, USA
3. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Comm ACM 21(2):120–126
4. Diffie W, Hellman ME (1976) New directions in cryptography. IEEE Trans Inf Theory 22:644–654
5. National Institute for Standards and Technology (2000) Digital Signature Standard (DSS). FIPS PUB 186–2, January
6. Menezes AJ (1993) Elliptic curve public key cryptosystems. Kluwer Academic Publishers, Boston, MA
7. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48(177):203–209, January
8. Miller VS (1986) Uses of elliptic curves in cryptography. In: Williams HC (ed) Advances in cryptology—CRYPTO '85, vol. 218 of Lecture Notes in Computer Science. Springer, Berlin, Germany, pp 417–426
9. Lenstra AK, Verheul ER (2000) Selecting cryptographic key sizes. In: Imai H, Zheng Y (eds), Public Key Cryptography—PKC 2000, vol. 1751 of Lecture Notes in Computer Science. Springer , Berlin, Germany, pp 446–465
10. US National Institute of Standards and Technology (NIST) (2002) Cryptographic toolkit Jun. 2002, pp 83–84 [online]. Available: http://csrc.nist.gov/CryptoToolkit/kms/guideline-1.pdf
11. Montgomery PL (1985) Modular multiplication without trail division. Math Comput 44(70):519–521
12. Kim YS, Kang WS, Choi JR. Implementation of 1024-bit modular processor for RSA cryptosystem'. http://www.ap-asic.org/2000/proceedings/10-4.pdf
13. Bunimov V, Schimmler M, Tolg B (2002) A complexity-effective version of Montgomery's algorithm. Presented at the Workshop on Complexity Effective Designs (WECD02), May 2002
14. McIvor C, McLoone M, McCanny JV (2004) Modified Montgomery modular multiplication and RSA exponentiation techniques. IEE Proc Comput Digit Tech 151(6):402–408
15. Sudhakar M, Kamala RV, Srinivas MB (2007) An efficient, reconfigurable and unified Montgomery multiplier architecture. Proceedings. IEEE/ACM VLSI Design 2007, Bangalore, India, January 6–10, pp 750–755
16. Savas E, Tenca AF, Koc CK (2000) A scalable and unified multiplier architecture for finite fields GF(p) and GF(2^m). Proceedings of CHES, pp 281–296
17. Großschädl J (2001) A bit-serial unified multiplier architecture for finite fields GF(p) and GF(2^m). Proc. CHES 2001, August, pp 202–218
18. Savas E, Tenca AF, Ciftcibasi ME, Koc CK (2004) Multiplier architectures for GF(p) and GF(2^n). IEE Proc Comput Digit Tech 151(2):147–160, March
19. Tenca AF, Koc CK (2003) A scalable architecture for modular multiplication based on Montgomery's algorithm. IEEE Trans Comput 52(9):1215–1221, Sept
20. Harris D, Krishnamurthy R, Anders M, Mathew S, Hsu S (2005) An improved unified scalable radix-2 Montgomery multiplier. IEEE (ARITH-17), pp 172–178
21. Certicom Research, Recommended Elliptic Curve Domain Parameters, http://www.secg.org/download/aid-386/sec2_final.pdf
22. NIST DSS Standard, Basicrypt—Elliptic Curve Cryptography (ECC) Benchmark Suite benchmark, http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf
23. Virtex™-E 1.8 V Field Programmable Gate Arrays, http://www.xilinx.com/bvdocs/publications/ds022.pdf
24. Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet http://www.xilinx.xom/bvdocs/publications/ds083.pdf

M. Sudhakar received B.Tech (2005) in Electronics and Communication Engineering from Jawaharlal Nehru Tec.hnological University, Hyderabad, India. He submitted master thesis for MS by research degree in VLSI and Embedded systems at International Institute of Information Technology, Hyderabad, India. Currently he is working as a design engineer in ARM Embedded Technologies Pvt. Ltd., Bangalore, India. His areas of research include hardware cryptography and reconfigurable hardware design.

Kamala Ramachandruni Venkata received B.Tech (2004) in Electronics and Communication Engineering from Jawaharlal Nehru Technological University, Hyderabad, India. She received master degree (MS by research) in VLSI and Embedded systems from International Institute of Information Technology, Hyderabad, India. She was a recipient of six gold medals for her excellence in academics. Currently she is working as a ASIC engineer in NVIDIA, Hyderabad, India. Her areas of interest include hardware cryptography, testing and verification methods, low power VLSI, digital circuit design and systolic arrays.

Dr. M. B. Srinivas is an associate professor and founding-chair of the Center for VLSI and Embedded System Technologies at the International Institute of Information Technologies (a university), Hyderabad, India. His research interests include high speed logic and arithmetic circuit design, nano-CMOS VLSI design, VLSI interconnects and high performance architectures for signal processing, communications and cryptography. He has published more than 100 research papers in various international journals and conferences. His other interests include developing low-cost information and communication technologies for providing health care in developing countries. He was a recipient of the Microsoft digital inclusion award for the year 2006. Dr. Srinivas is also the chairman of IEEE, Hyderabad section.