Fast Implementation of Public-Key Cryptography on a DSP TMS320C6201

Kouichi Itoh[1], Masahiko Takenaka[1], Naoya Torii[1], Syouji Temma[2], and Yasushi Kurihara[2]

[1] FUJITSU LABORATORIES LTD. 64 Nishiwaki, Ohkubo-cho, Akashi 674-8555 Japan.
{kito, takenaka, torii}@flab.fujitsu.co.jp
[2] FUJITSU LTD. 4-1-1 Kami-kodanaka, Nakahara-ku, Kawasaki 211-8588 Japan.
{temma, kurihara}@cl.mfd.cs.fujitsu.co.jp

Abstract. We propose new fast implementation method of public-key cryptography suitable for DSP. We improved modular multiplication and elliptic doubling to increase speed. For modular multiplication, we devised a new implementation method of Montgomery multiplication, which is suitable for pipeline processing. For elliptic doubling, we devised an improved computation for the number of multiplications and additions. We implemented RSA, DSA and ECDSA on the latest DSP (TMS320C6201, Texas Instruments), and achieved a performance of 11.7 msec for 1024-bit RSA signing, 14.5 msec for 1024-bit DSA verification and 3.97 msec for 160-bit ECDSA verification.

1 Introduction

Public-key cryptography is an important encryption technique. It can be applied to many practical uses such as electronic commerce systems and WWW systems for enabling digital signatures and key agreement. The server systems for them are required to process a vast number of public key operations.

Additionally, for communicating with various kinds of clients, the server systems are required to provide various public-key cryptography functions, such as RSA [15] , Diffie-Hellman key agreement [5], DSA [16] and elliptic curve cryptography (ECC) [9][12]. These functions are under standardization in IEEE P1363 [17].

In this paper, we describe a fast implementation method using DSP as a cryptographic engine for server systems. In public-key cryptography, modular multiplications are the most time-consuming operations. A DSP can compute these operations efficiently with a fast hardware multiplier. Furthermore, a DSP can be used as the hardware engine for various algorithms since it is programmable.

In the past, fast public key cryptographic implementations on DSPs have been reported [1][2][6]. They concentrated on the implementation of RSA using the latest DSP at the time. We implemented RSA, DSA and ECDSA over prime fields based on the IEEE P1363 draft, and propose new implementation methods suitable for DSP. Our methods concern modular multiplication and elliptic doubling.

For modular multiplication, we devised a fast implementation method for Montgomery multiplication [14]. Our method is suitable for pipeline processing.

For elliptic doubling, we devised a new method which reduces the number of multiplications and additions in comparison with that specified in the IEEE P1363 draft. In general, the running time of addition is considered negligible compared with that of multiplication. But in fact, the running time of addition is not negligible on a processor such as a DSP, which has a fast hardware multiplier.

There are some reports concerning the fast implementation of ECC [3][4][13]. They used the special elliptic curve domain parameters (EC domain parameters) for speeding up. On the other hand, our implementation can use any EC domain parameters for the server systems. The server systems require high performance and communicating with client systems that use various types of EC domain parameters.

We implemented public-key cryptography functions with our method on the latest DSP TMS320C6201 (Texas Instruments). This DSP can operate eight function units in parallel and has a performance of 1600 MIPS at 200 MHz. The performance achieved in our implementation was 11.7 msec for 1024-bit RSA signing, 14.5 msec for 1024-bit DSA verification and 3.97 msec for 160-bit ECDSA verification.

We describe our improvement method for Montgomery multiplication in section 2, our elliptic doubling method in section 3 and the performance in section 4.

2 Fast Implementation Method of Montgomery Multiplication

2.1 Montgomery Multiplication

Basic algorithm. Set $N > 1$. Select a radix R co-prime to N such that $R > N$ and such that computations modulo R are inexpensive to process. Let N' be integers satisfying $0 < N' < R$ and $N' = -N^{-1} \pmod R$. For all integers A and B satisfying $0 \leq AB < RN$, we can compute $REDC(A, B) = ABR^{-1} \pmod N$ with Algorithm 1.

Algorithm 1. *Montgomery multiplication algorithm REDC.*
input : A, B, R, N.
output : $Y = ABR^{-1} \pmod N$.

```
101 $N' := -N^{-1} \pmod R$
102 $T := AB$
103 $M := (T \pmod R)N' \pmod R$
104 $T := T + MN$
105 $T := T/R$
106 if $T \geq N$ then return $T - N$ else return $T$
```

If R is a power of 2, line 105 can be computed fast with shift operations.

Modular multiplication with Montgomery method. Since $REDC(A, B) = ABR^{-1} \pmod{N}$, it can not compute modular multiplication directly. But on reviewing $REDC(AR, BR) = ABR \pmod{N}$, it can be seen that REDC can compute modular multiplication by converting $A \pmod{N}$ to $AR \pmod{N}$. After this conversion, a series of modular multiplications can be computed fast with REDC. For example, we show an m-ary exponentiation [7] with REDC in Algorithm 2, where e is a k-bit exponent and e_i is an m-bit integer which satisfies $e = \sum(2^m)^i e_i$.

Algorithm 2. *m-ary exponentiation method with REDC.*
input : A, e, N, R
output : $Y = A^e \pmod{N}$

```
201 $A' := A \times R \pmod{N}$
202 $T[0] := 1 \times R \pmod{N}$
203 for $i := 1$ to $2^m - 1$
204   $T[i] = REDC(T[i-1], A')$
205 next $i$
206 $Y := 1 \times R \pmod{N}$
207 for $i := \lceil k/m \rceil - 1$ down to $0$
208   for $j := 1$ to $m$
209     $Y := REDC(Y, Y)$
210   next $j$
211   $Y := REDC(Y, T[e_i])$
212 next $i$
213 $Y := Y \times R^{-1} \pmod{N}$
214 return $Y$
```

REDC routine with single-precision. To implement REDC on general processors, multi-precision computation must be divided into iterations of single-precision computation. In [10], many types of REDC routines are constructed with single-precision computation. Algorithm 3 shows a Finely Integrated Operand Scanning (FIOS) type of REDC routine in [10].

We will use the following notations. Capital variables such as A or B, mean a multi-precision integer. Small letter variables such as a_i, b_j or $tmp1$ mean a single-precision integer of w-bit length.

A multi-precision integer, for example A, is expressed as the series of single-precision variables $(a_{g-1}, a_{g-2}, \ldots, a_0)$. The expression such as (a, b) means the concatenation of single-precision variables a and b. We also use the expression such as (A, b), which means the concatenation of a multi-precision variable A and a single-precision variable b.

In Algorithm 3, the block-shift is executed by reading from y_i and writing to y_{i-1}. Note that the w-bit variables $tmp3$ and c_1 have 1-bit value.

Algorithm 3. *REDC routine with single-precision computation. (FIOS [10].)*
input: $A = (a_{g-1}, a_{g-2}, \ldots, a_0)$, $B = (b_{g-1}, b_{g-2}, \ldots, b_0)$, $N' = (n'_{g-1}, n'_{g-2}, \ldots n'_0)$, $R = (2^w)^g$.
output: $Y = (y_g, y_{g-1}, \ldots, y_0) = ABR^{-1} \pmod{N}$.

```
301 $Y := 0$
302 for $j := 0$ to $g-1$
303   $(tmp2, tmp1) := y_0 + a_0 \times b_j$
304   $m := tmp1 \times n'_0 \pmod{2^w}$
305   $(tmp4, tmp1) := tmp1 + m \times n_0$
306   $(c_1, c_0) := tmp2 + tmp4$
307   for $i := 1$ to $g-1$
308     $(tmp3, tmp2, tmp1) := y_i + (c_1, c_0) + a_i \times b_j$   single-precision multiplication
309     $(tmp4, y_{i-1}) := tmp1 + m \times n_i$                    single-precision reduction
310     $(c_1, c_0) := tmp4 + (tmp3, tmp2)$                         carry computation
311   next $i$
312   $(c_1, c_0) := (c_1, c_0) + y_g$
313   $y_{g-1} := c_0$
314   $y_g := c_1$
315 next $j$
316 if $Y \geq N$ then $Y := Y - N$
317 return $Y$
```

2.2 Proposed Method

To speed up Algorithm 3 on a DSP, let us consider improving the core loop in lines 308-310 suitable for pipelining. For the improvement, we considered the following problems:

(1) *Single-precision multiplication* in line 308 cannot execute until *single-precision reduction* in line 309 and *carry computation* in line 310 finish.
(2) The contents of the computation are different among *single-precision multiplication*, *single-precision reduction* and *carry computation.*
(3) The result of *carry computation*, (c_1, c_0) in line 310, has $(w+1)$-bit length value so that it must be processed as a multi-precision variable.

We reviewed the computation to solve these problems. Figure 1 shows the construction of the core loop. On reviewing the carry processing in Fig.1, carry of the *single-precision multiplication* and carry of the *single-precision reduction* are added to $C = (c_1, c_0)$, and C is input to the carry of *single-precision multiplication* in the next loop. To review this processing, we combine the computation in the core loop as follows:

$$(C, y_{i-1}) := y_i + C + a_i \times b_j + m \times n_i$$

From this equation, we can divide the carry C into the carry c_1 for the $a_i \times b_j$ and the carry c_2 for the $m \times n_i$ as follows:

$$(c_1, tmp1) := y_i + c_1 + a_i \times b_j \quad \textit{single-precision multiplication}$$
$$(c_2, y_{i-1}) := tmp1 + c_2 + m \times n_i \quad \textit{single-precision reduction}$$

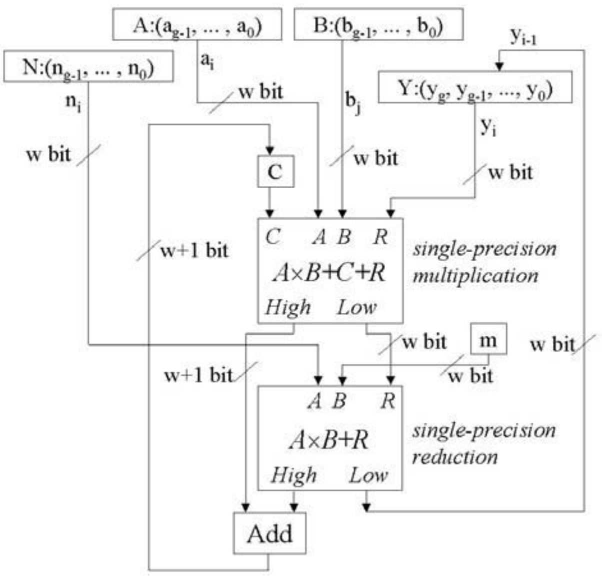

Fig. 1. Construction of core loop in Algorithm 3.

From these equations, we can see that problems (1), (2) and (3) are solved as follows:

Problem (1) is solved because both carry c_1 and c_2 feed back to themselves, which enables *single-precision multiplication* to start computing without waiting until *single-precision reduction* finishes. Problem (2) is solved because the computation between *single-precision multiplication* and *single-precision reduction* is the same. Problem (3) is solved because the right term of these equations never exceeds $2^{2w}-1$ even if all single-precision variables in the right terms are 2^w-1, so that the lengths of c_1 and c_2 do not exceed w-bit.

Algorithm 4 shows an improved routine of Algorithm 3. Figure 2 shows the construction of the core loop in Algorithm 4.

Algorithm 4. *Proposed Montgomery multiplication algorithm.*
input: $A = (a_{g-1}, a_{g-2}, \ldots, a_0), B = (b_{g-1}, b_{g-2}, \ldots, b_0),$
$N' = (n'_{g-1}, n'_{g-2}, \ldots, n'_0), R = (2^w)^g.$
output: $Y = (y_g, y_{g-1}, \ldots, y_0) = ABR^{-1} \pmod N$.

```
401 $Y := 0$
402 for $j := 0$ to $g-1$
403   $(c_1, tmp1) := y_0 + a_i \times b_j$
404   $m := tmp1 \times n'_0 \pmod{2^w}$
405   $(c_2, tmp1) := tmp1 + m \times n_0$
406   for $i := 1$ to $g-1$
407     $(c_1, tmp1) := y_i + c_1 + a_i \times b_j$      single-precision multiplication
408     $(c_2, y_{i-1}) := tmp1 + c_2 + m \times n_i$   single-precision reduction
```

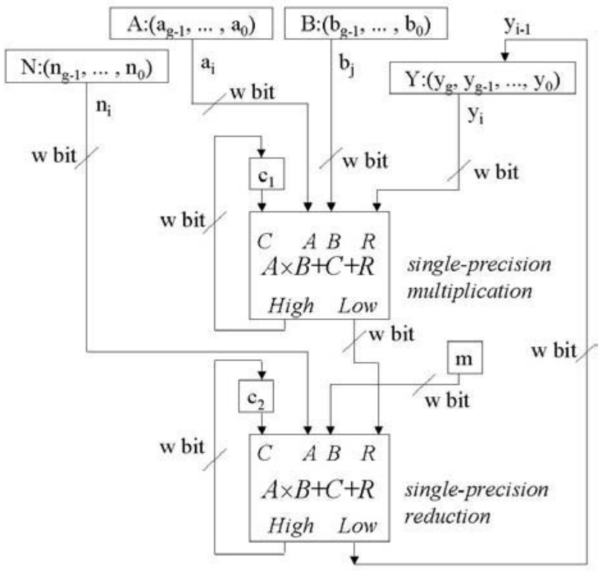

Fig. 2. Construction of the core loop in Algorithm 4.

```
409  next i
410  $(c_2, c_1) := c_1 + c_2 + y_g$
411  $y_{g-1} := c_1$
412  $y_g := c_2$
413 next j
414 if $Y \geq N$ then $Y := Y - N$
415 return $Y$
```

3 Fast Elliptic Doubling

We used a Weierstrass equation, $y^2 \equiv x^3 + ax + b \pmod{p}$ for the elliptic curve over prime fields where $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$, and projective coordinate (X, Y, Z) which satisfies $(x, y) = (X/Z^2, Y/Z^3)$.

For exponentiation, such as m-ary [7] or window method [7], m elliptic doublings and 1 elliptic addition are processed alternatively. Remarking on this point, the *m-repeated* elliptic doublings method is proposed in [8] which is concerned with the computation on affine coordinates over binary fields. Compared to m times elliptic doublings, this method reduces the number of inverses by computing $2^m P$ for $P = (x, y)$ directly without computing intermediate points $2^i P (1 \leq i \leq m - 1)$.

We also remark this m-repeated elliptic doublings method, but take another approach to decrease the number of computation in terms of projective coordinates over prime fields. Our method is based on the m times elliptic doublings specified in the IEEE P1363 draft [17] and also reduces the number of additions and multiplications.

3.1 Reducing the Number of Multiplications

In this section, we describe our m-repeated elliptic doublings method which requires smaller multiplications than the m times elliptic doublings specified in the IEEE P1363 draft. In our method, the temporary value used in the t-th elliptic doubling is reused in the $(t+1)$-th elliptic doubling, and this eliminates 2 multiplications. Therefore, our method requires 10 multiplications in the first elliptic doubling, but requires only 8 multiplications from the second doubling to the m-th. Let $(X_m, Y_m, Z_m) = 2^m(X_0, Y_0, Z_0)$, Algorithm 5 shows m times elliptic doublings specified in the IEEE P1363 draft [17].

Algorithm 5. *m times elliptic doublings specified in the IEEE P1363 draft.*
input: Elliptic curve point (X_0, Y_0, Z_0), m and EC domain parameter a.
output: Elliptic curve point $(X_m, Y_m, Z_m) = 2^m(X_0, Y_0, Z_0)$.

```
501 for $i := 0$ to $m - 1$
502   $W_i := aZ_i^4$
503   $M_i := 3X_i^2 + aZ_i^4$
504   $S_i := 4X_iY_i^2$
505   $T_i := 8Y_i^4$
506   $X_{i+1} := M_i^2 - 2S_i$
507   $Y_{i+1} := M_i(S_i - X_{i+1}) - T_i$
508   $Z_{i+1} := 2Y_iZ_i$
509 next $i$
```

If we consider $W_i = aZ_i^4$ and $Z_{i+1} = 2Y_iZ_i$ in line 502, 508, we notice that W_i can be computed from $W_i = 2T_{i-1}W_{i-1}$, which eliminates 2 multiplications. We show the improved routine of Algorithm 5 in Algorithm 6.

Algorithm 6. *Improved routine of Algorithm 5.*
input: Elliptic curve point (X_0, Y_0, Z_0), m and EC domain parameter a.
output: Elliptic curve point $(X_m, Y_m, Z_m) = 2^m(X_0, Y_0, Z_0)$.

```
601 $W_0 := aZ_0^4$
602 $M_0 := 3X_0^2 + W_0$
603 $S_0 := 4X_0Y_0^2$
604 $T_0 := 8Y_0^4$
605 $X_1 := M_0^2 - 2S_0$
606 $Y_1 := M_0(S_0 - X_1) - T_0$
607 $Z_1 := 2Y_0Z_0$
608 for $i := 1$ to $m - 1$
609   $W_i := 2T_{i-1}W_{i-1}$
610   $M_i := 3X_i^2 + W_i$
611   $S_i := 4X_iY_i^2$
```

```
612 $T_i := 8Y_i^4$
613 $X_{i+1} := M_i^2 - 2S_i$
614 $Y_{i+1} := M_i(S_i - X_{i+1}) - T_i$
615 $Z_{i+1} := 2Y_iZ_i$
616 next i
```

3.2 Reducing the Number of Additions

Generally, an addition is regarded as much faster than a multiplication, and its running time is not considered. But on a DSP, multiplication can be computed efficiently with a fast hardware multiplier, and the running time of addition is not negligible. Table 1 shows a comparison of the running time of a modular multiplication and a modular addition based on our implementation on the DSP.

Table 1. Comparison of the running time of a modular multiplication and a modular addition @ 200 MHz.

	160-bit	192-bit	239-bit
Multiplication	1.36 μsec	1.76 μsec	2.68 μsec
Addition	0.250 μsec	0.254 μsec	0.291 μsec

In projective elliptic doubling, some computations such as modular multiplication by $2, 3, 4$, and 8 can be implemented by the combination of modular addition(s) and subtraction(s). Appending modular multiplication by 1/2 to these computations, we define them "addition" in this paper. We estimate the computation amount of "addition" as follows:

- Modular addition and subtraction are "1 addition".
- Modular multiplication by 2 and 1/2 are "1 addition".
- Modular multiplication by 3 and 4 are "2 additions".
- Modular multiplication by 8 is "3 additions".

Now we consider reducing the number of additions in Algorithm 6 with this estimate. For example, computing $4Y^2$ as $(2Y)^2$ eliminates 1 addition compared with computing it as $4 \times (Y^2)$. Thus, additions in Algorithm 6 are reduced with $2Y$-based computation. With this technique, we can reduce the number of additions in Algorithm 6 by the following techniques:

(A) At the beginning, compute $Y'_0 = 2Y_0$ as a base value, and compute $Y'_i (= 2Y_i)$ without computing Y_i for $i < m$.
(B) By reason of (A), compute $T_i = 16Y_i^4$ instead of $8Y_i^4$.
(C) Compute $S_i = 4X_iY_i^2, Z_i = 2Z_{i-1}Y_{i-1}$ and $T_i = 16Y_i^4$ based on $Y'_i = 2Y_i$, viz. compute $S_i = X_i(Y'_i)^2, Z_i = Z_{i-1}(Y'_{i-1})$ and $T = (Y'_i)^4$ respectively.
(D) Finally, compute $Y_m = Y'_m/2$.

We show the improved routine of Algorithm 6 in Algorithm 7.

Algorithm 7. *Proposed m-repeated elliptic doublings routine.*
input: Elliptic curve point $(X_0, Y_0, Z_0), m$ *and EC domain parameter* a.
output: Elliptic curve point $(X_m, Y_m, Z_m) = 2^m(X_0, Y_0, Z_0)$.

```
701 $Y'_0 := 2Y_0$
702 $W_0 := aZ_0^4$
703 $M_0 := 3X_0^2 + W_0$
704 $S_0 := X_0(Y'_0)^2$
705 $T_0 := (Y'_0)^4$
706 $X_1 := M_0^2 - 2S_0$
707 $Y'_1 := 2M_0(S_0 - X_1) - T_0$
708 $Z_1 := Y'_0 Z_0$
709 for $i := 1$ to $m - 1$
710   $W_i := T_{i-1}W_{i-1}$
711   $M_i := 3X_i^2 + W_i$
712   $S_i := X_i(Y'_i)^2$
713   $T_i := (Y'_i)^4$
714   $X_{i+1} := M_i^2 - 2S_i$
715   $Y'_{i+1} := 2M_i(S_i - X_i) - T_i$
716   $Z_{i+1} := (Y'_i)Z_i$
717 next $i$
718 $Y_m := Y'_m/2$
```

Table 2 shows the number of multiplications and additions required for the above algorithms. Our method eliminates $2m-2$ multiplications and $5m-2$ additions compared with the m times elliptic doublings specified in the IEEE P1363 draft.

Table 2. Number of multiplications and additions.

m-repeated elliptic doublings	Multiplication	Addition
Algorithm 5 (IEEE P1363 draft)	$10m$	$13m$
Algorithm 6	$8m+2$	$14m-1$
Algorithm 7 (Proposed)	$8m+2$	$8m+2$

4 Implementation

4.1 DSP and Development Tools

For the implementation, we used the DSP TMS320C6201 [18] (Texas Instruments). The DSP consists of eight parallel-operation functional units including two 16-bit multiplication units, and has a performance of 1600 MIPS at 200 MHz. The instruction processing system is of the VLIW/pipeline type and can execute conditional operations. And the maximum instruction code size is 64 Kbytes.

As the development tools, an assembler and C compiler are provided. We implemented arithmetic routines such as modular multiplication, addition, and subtraction in assembly language. Their performance greatly affects the total performance, because they are performed frequently. Other routines were written in C for easy implementation.

4.2 Implementation of RSA and DSA

We used the following methods:

- Modular multiplication with the Montgomery multiplication method [14] described in section 2.
- Modular exponentiation with m-ary method [7] for $m = 4$.

4.3 Implementation of ECC

We used following methods:

- Modular multiplication with the Montgomery multiplication method [14] described in section 2.
- Fast elliptic doubling with the method described in section 3, combined with the technique for increasing speed in case EC domain parameter $a = 0$.
- Elliptic addition based on IEEE P1363 draft [17].
- The base point exponentiation with fixed-base comb method [11], specified using two 5-bit precomputed tables.
- Random point exponentiation in combination with sliding-window exponentiation [11] with a 4-bit precomputed table and signed-binary [7] of the exponent.

4.4 Code Size

We implemented RSA, DSA and ECDSA based on above method, and the total instruction code size was 41.1 Kbytes. Since TMS320C6201 allows a maximum instruction code size of 64 Kbytes, this implementation can deal with RSA, DSA and ECDSA without reloading.

4.5 Performance of RSA, DSA, and ECC

Table 3 shows the performance of the RSA and DSA implementation. Table 4 shows the performance of the ECC implementation including the exponentiation on a random point. We measured the 100 times average clocks and figured the running time at 200 MHz.

In Table 3, we used $e = 2^{16} + 1$ for the RSA verification key, and Chinese remainder theorem for RSA signing.

In Table 4, the exponent of a random point has a same length as that of EC domain parameter p. The ECDSA scheme is based on the IEEE P1363 draft. Table 4 also shows the bit length of the order of the base point which affects the performance of ECDSA.

Table 3. Performance of RSA and DSA @ 200 MHz.

	RSA		DSA	
	1024bit	2048 bit	512 bit	1024 bit
Sign	11.7 msec	84.6 msec	2.62 msec	7.44 msec
Verify	1.2 msec	4.5 msec	4.82 msec	14.5 msec

Table 4. Performance of ECC @ 200 MHz.

	EC domain parameter p	160-bit	192-bit	239-bit
$a \neq 0$	Order of the base point	151-bit	192-bit	239-bit
	Exponentiation on a random point	3.09 msec	4.64 msec	8.47 msec
	ECDSA sign	1.13 msec	1.67 msec	2.85 msec
	ECDSA verify	3.97 msec	6.28 msec	11.2 msec
$a = 0$	Order of the base point	160-bit	185-bit	232-bit
	Exponentiation on a random point	2.88 msec	4.15 msec	7.60 msec
	ECDSA sign	1.09 msec	1.50 msec	2.66 msec
	ECDSA verify	3.78 msec	5.50 msec	9.78 msec

5 Conclusion

We proposed fast implementation methods of Montgomery multiplication and m-repeated elliptic doublings, which are efficient for any EC domain parameters and suitable for the server systems. Our methods are efficient not only for DSP, but also for any other processors.

Construction of our Montgomery multiplication method is suitable for the implementation on various pipeline processors. Furthermore, our method is also effective for the implementation on non-pipeline processors, because it computes all carries within a single-precision value.

Our m-repeated elliptic doublings method eliminates $2m - 2$ multiplications and $5m - 2$ additions compared with m times elliptic doublings specified in IEEE P1363 draft. This method is efficient on any processors. As the multiplication is faster in comparison with addition, our method is more effective.

We implemented RSA, DSA and ECC with our method on the latest DSP TMS320C6201(Texas Instruments). The performance is 11.7 msec for 1024-bit RSA signing, 14.5 msec for 1024-bit DSA verification and 3.97 msec for 160-bit ECDSA verification.

References

1. Paul Barrett, “Implementing the Rivest, Shamir, and Adleman Public-Key Encryption Algorithm on a Standard Digital Signal Processor”, Advances in Cryptology-CRYTO’86(LNCS 263), pp.311-323, 1987.
2. E.F.Brickell, “A Survey of Hardware Implementations of RSA”, Advances in Cryptology-CRYPTO’89(LNCS 435), pp.368-370, 1990.
3. D.Chudnovsky and G.Chudnovsky, “Sequences of numbers generated by addition in formal groups and new primality and factoring tests”, Advances in Applied Mathematics, 7, pp.385-434, 1987.
4. Richard E.Crandall, “Method and apparatus for public key exchange in a cryptographic system”, U.S. Patent, 5,159,632, 27 October 1992.
5. W.Diffie and M.Hellman, “New directions in cryptography”, IEEE Transactions on Information Theory 22, pp.644-654, 1976.
6. S.R.Dusse and B.S.Kaliski Jr., “A Cryptographic Library for the Motorola DSP56000”, Advances in Cryptology-Eurocrypt’90(LNCS 473), pp.230-244, 1991.

7. Daniel M.Gordon, "A Survey of Fast Exponentiation Methods", Journal of Algorithms 27, pp.129-146, 1998.
8. J.Guajardo and C.Paar, "Efficient Algorithms for Elliptic Curve Cryptosystems", Advances in Cryptology-CRYPTO'97(LNCS 1294), pp.342-356, 1997.
9. N.Koblitz, "Elliptic curve cryptosystems", Mathematics of Computation 48, pp.203-209, 1987.
10. Çetin Kaya Koç, Tolga Acar, B.S.Kaliski Jr., "Analyzing and Comparing Montgomery Multiplication Algorithms", IEEE Macro, Vol.16, No.3, pp.26-33, June 1996.
11. Alfred J.Menezes, Paul C.van Oorschot and Scott A.Vanstone, "HANDBOOK of APPLIED CRYPTOGRAPHY", CRC Press, 1997.
12. V.S.Miller, "Use of elliptic curves in cryptography", Advances in Cryptology-CRYPTO'85(LNCS 218), pp.417-426, 1986.
13. Atsuko Miyaji, "Method for Generating and Verifying Electronic Signatures and Privacy Communication Using Elliptic Curves", U.S. Patent, No.5,442,707, 15 August 1995.
14. P.L.Montgomery, "Modular Multiplication without Trial Division", Mathematics of Computation, Vol.44, No.170, pp.519-521, 1985.
15. R.L.Rivest, A.Shamir and L.Adleman, "A Method of obtaining digital signature and public key cryptosystems", Comm. of ACM, Vol.21, No.2, pp.120-126, Feb.1978.
16. FIPS 186, "Digital signature standard", Federal Information Processing Standards Publication 186, U.S.Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, 1994.
17. IEEE P1363/D9(Draft Version 9) Standard Specifications for Public Key, http://grouper.ieee.org/groups/1363/.
18. Texas Instruments, "Digital Signal Processing Solutions Products - TMS320C6x", http://www.ti.com/sc/docs/products/dsp/tms320c6201.html.