COMPARISON OF FPGA IMPLEMENTATION OF THE MOD *M* REDUCTION

J-P. DESCHAMPS† and G. SUTTER‡

† *Escola Tècnica Superior d' Enginyeria, Universitat Rovira i Virgili, Tarragona, Spain, jeanpierre.deschamps@urv.net; http://www.etse.urv.es*

‡ *Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain, gustavo.sutter@ii.uam.es; http://www.ii.uam.es*

***Abstract*— Several algorithms for computing x mod m are presented, among others the reduction mod B^k-a, the pre-computation of $B^{i.k}$ mod m, a generalized version of the Barrett algorithm and a modified version of the same Barrett algorithm. The four mentioned algorithms, as well as the classical integer non-restoring division algorithm, have been synthesized and implemented within xc3s4000 components.**

***Keywords*— Arithmetic in FPGA, Galois Field, Cryptography, modular operation.**

I. INTRODUCTION

Arithmetic operations over the finite ring $Z_m = \{0, 1, ..., m\text{-}1\}$ are used as computation primitives for executing numerous cryptographic algorithms, especially those related with the use of public keys (asymmetric cryptography). Classical examples are ciphering / deciphering, authentication and digital signature protocols based on RSA-type or elliptic curve algorithms. One of the basic operations is the modulo *m* reduction. Combined with operations over the set *Z* of integers (sum, subtraction, product, and so on) it allows to perform the same operations over Z_m. A straightforward solution consists of using an integer division algorithm. Nevertheless, more efficient algorithms have been proposed (Blake *et al*, 2002; Hankerson *et al*, 2004). In this paper several algorithms are described, namely the reduction mod B^k-*a*, the pre-computation of $B^{i.k}$ mod *m*, a generalized version of the Barrett algorithm and a modified version of the same Barrett algorithm. The four mentioned algorithms, as well as the classical integer non-restoring division algorithm, have been synthesized and implemented within xc3s4000 components.

II. ALGORITHM

In this section the following problem is studied: given two naturals *x* and *m*, compute $z = x \bmod m$.

A. Integer division

A straightforward method consists of performing the integer division of *x* by *m*, that is,

$$x = q.m + z,\ z < m.$$

For that purpose, any division algorithm can be used, for example the non-restoring division algorithm (Deschamps *et al*, 2006).

Algorithm 1 – Non-restoring reduction

```
y := m*(2**(n-k));
rems(1) := x - y;
for i in 1 .. n-k loop
  if rems(i) < 0 then
    rems(i+1) := 2*rems(i)+ y;
  else
    rems(i+1) := 2*rems (i) - y;
  end if;
end loop;
if rems(n-k+1) < 0 then
   z := rems(n-k+1)/(2**(n-k)) + m;
else
   z := rems(n-k+1)/(2**(n-k));
end if;
```

The core of the algorithm is an (*n*-*k*)-step iteration. If a ripple-carry *k*-bit adder-subtractor is used, the computation time is about

$$\text{time}(n,k) \approx (n\text{-}k).k.T_{FA} \quad , \tag{1}$$

where T_{FA} is the delay of a full-adder.

B. Reduction mod B^k-*a*

Assume that $B^{k-1} \le m < B^k$, where *B* is a natural number ≥ 2. Then $m = B^k - a$ where $1 \le a \le B^k - B^{k-1}$.

Compute the following quotients q_i and remainders r_i:

$$\begin{aligned} &x = q_0.B^k + r_0, \\ &q_0.a = q_1.B^k + r_1, \\ &q_1.a = q_2.B^k + r_2, \\ &\ldots \\ &q_{s-2}.a = q_{s-1}.B^k + r_{s-1}. \end{aligned} \tag{2}$$

Multiply the second equation of (2) by (B^k/a), the third one by $(B^k/a)^2$, ... , the last one by $(B^k/a)^{s-1}$, and sum up the *s* equations; the result is

$$\begin{aligned} x = r_0 + r_1.(B^k/a) + r_2.(B^k/a)^2 + ... \\ + r_{s-1}.(B^k/a)^{s-1} + q_{s-1}.B^k.(B^k/a)^{s-1}. \end{aligned} \tag{3}$$

As $a < B^k$, that is, $B^k/a > 1$, there exists a minimum value of *s* such that

$$x < B^k.(B^k/a)^{s-1}, \tag{4}$$

and thus $q_{s-1} = 0$. Let *s* be the minimum value of *s* such that $q_{s-1} = 0$. Notice that if $r_{s-1} = 0$ then the last equation

of (2), with $q_{s-1} = 0$, is $q_{s-2}.a = 0$, that is, $q_{s-2} = 0$, so that s is not the minimum value of s such that $q_{s-1} = 0$. Thus

$$x = r_0 + r_1.(B^k/a) + r_2.(B^k/a)^2 + ... + r_{s-1}.(B^k/a)^{s-1}, \text{ with } r_{s-1} > 0. \quad (5)$$

By summing up the s equations of system (2), with $q_{s-1} = 0$, the following relation is obtained:

$$x = q_0.(B^k-a) + q_1.(B^k-a) + ... + q_{s-2}.(B^k-a) + r_0 + r_1 + ... + r_{s-1}. \quad (6)$$

Define

$$r = r_0 + r_1 + ... + r_{s-1}. \quad (7)$$

According to (6) and (7),

$$x \equiv r \bmod m, \text{ with } m = B^k-a. \quad (8)$$

Comparing (5) with (7) it is obvious that if $s > 1$, that is, if $x \geq B^k$, then $r < x$. If r is still greater than or equal to B^k, the same method can be used in order to get $r' \equiv x \bmod m$, with $r' < r$. After a finite number of iterations, a number r'' is obtained such that $r'' \equiv x \bmod m$ and $r'' < B^k$, so that $z = r'' - q.m$ where $0 \leq q \leq B$-1. In particular, if $B = 2$ then z is either r'' or r''-m. To summarize, the mod m reduction algorithm, with $m = B^k$-a, is the following:

Algorithm 2 – mod *m* reduction algorithm, with $m = B^k$-a

```
r := x mod b**k; q := x/b**k;
loop
  loop
    r := r + (q*a mod b**k);
    q := q*a/b**k;
    if q = 0 then exit; end if;
  end loop;
  q := r/b**k; r := r mod b**k;
  if q = 0 then exit; end if;
end loop;
while r >= m loop r := r-m; end loop;
z := r;
```

If B is the base (or a power of the base) of the chosen numeration system, then the division by B^k and the mod B^k reduction are trivial operations. The only non-trivial operations are multiplication by a, sums (remainder accumulation) and subtractions (final reduction). The number of executions of the internal loop body can be estimated as follows: a sufficient condition for q_{s-1} being equal to 0 is (4), which is equivalent to $s > (\log x - \log a) / (k.\log B - \log a)$. Thus $s = \lceil (\log x - \log a) / (k.\log B - \log a) \rceil$. In particular, if $x = B^n$-1, that is, the greatest n-digit B-ary number, then

$$s = \lceil (n - \log_B a) / (k - \log_B a) \rceil, \quad (9)$$

and, assuming that $\log_B a$ is much smaller than k and n,

$$s \approx n/k. \quad (10)$$

As regards the reduction rate of the algorithm, that is, the relation between an initial value x and the obtained value r after a first execution of the internal loop, notice that r is smaller than $s.B^k$, so that the number $d(r)$ of B-ary digits of r satisfies the condition

$$d(r) \leq k + \lceil \log_B s \rceil, \quad (11)$$

where s is approximately equal to (10). Thus $d(r_{max}) \approx \log_B n + k - \log_B k < \log_B n + k$.

In order to define the size of the variable $r = r_0 + r_1 + ... + r_{s-1}$, the following values are previously calculated (see (9) and (11)):

$$s = \lceil (n - \log_2 a) / (k - \log_2 a) \rceil, \; t = \lceil \log_2 s \rceil,$$

so that r can be represented as a $(k+t)$-bit number.

The core of the algorithm is an (n/k)-step iteration. Each step includes the multiplication of an $(n-k)$-bit number q by a k-bit number a, and the sum of a $(k+t)$-bit number r and a k-bit number. The computation time of the multiplier depends on the particular value of a. Nevertheless, in order to get an estimation of the computation time, it will be assumed that a parallel multiplier is used. Its computation time is about $((n-k)+2.k-2).T_{FA} \approx (n+k).T_{FA}$ (Deschamps *et al*, 2006). The step duration is approximately equal to $(n+k).T_{FA} + (k+t).T_{FA}$. If $n+2.k >> t$ then the computation time is approximately equal to

$$\text{time}(n,k) \approx (n/k).(n+2.k).T_{FA}. \quad (12)$$

C. Pre-computation of $B^{i.k}$ mod m

Assume again that $B^{k-1} \leq m < B^k$, and that x is represented in base B^k, i.e.

$$x = x_{s-1}.B^{(s-1).k} + x_{s-2}.B^{(s-2).k} + ... + x_1.B^k + x_0, \text{ where } x_{s-1} > 0. \quad (13)$$

The following values must have been previously computed:

$$b_0 = 1, \; b_1 = B^k \bmod m, \; b_2 = B^{2.k} \bmod m, \; ..., \; b_{s-1} = B^{(s-1).k} \bmod m.$$

Then $x \equiv x_{s-1}.b_{s-1} + x_{s-2}.b_{s-2} + ... + x_1.b_1 + x_0.b_0 \bmod m$, and the problem is reduced to the computation of r mod m where

$$r = x_{s-1}.b_{s-1} + x_{s-2}.b_{s-2} + ... + x_1.b_1 + x_0.b_0. \quad (14)$$

Observe that $b_i = (B^{i.k} \bmod m) < m < B^k \leq B^{i.k}$, $\forall\, i > 0$. Comparing (14) with (13), it is obvious that if $s > 1$, that is, if $x \geq B^k$, then $r < x$. If r is still greater than or equal to B^k, the same method can be used in order to get $r' \equiv x \bmod m$ with $r' < r$. After a finite number of iterations, a number r'' is obtained such that $r'' \equiv x \bmod m$ and $r'' < B^k$, so that $z = r'' - q.m$ where $0 \leq q \leq B$-1. In particular, if $B = 2$ then z is either r'' or r''-m.

To summarize, the mod m reduction algorithm, with pre-computation of $B^{i.k}$ mod m, is the following (it is assumed that the constants $b_i = B^{i.k} \bmod m$ have been previously calculated):

Algorithm 3 – mod *m* reduction, with pre-computation of $B^{i.k}$ mod *m*

```
main: loop
  --represent x as an s-digit number:
  vector_x(0) := x mod base**k;
  q := x/base**k;
  for i in 1 .. s-1 loop
    vector_x(i) := q mod base**k;
    q := q/base**k;
  end loop;
  --end of computation detection:
  one_digit := true;
  for i in 1 .. s-1 loop
    if vector_x(i) /= 0 then
      one_digit := false;
      exit;
    end if;
  end loop;
  --main computation
  if one_digit then
    exit main;
  else
    x := vector_x(0);
    internal: for i in 1 .. s-1 loop
      x := x + vector_x(i)*b(i);
    end loop internal;
  end if;
end loop main;
r := vector_x(0);
while r >= m loop r := r-m; end loop;
z := r;
```

The mod B^k reduction and the integer division by B^k are trivial operations. The only non-trivial operations are products of base-B^k digits (vector_x(i)*b(i)) and sums, as well as the end of computation detection.

Let n be the number of B-ary digits of x. According to (13), $x_{max} = B^{s.k} - 1$, so that $n = s.k$ and the number s of executions of the internal loop body is

$$s = n/k. \tag{15}$$

As regards the reduction rate of the algorithm, notice that r is smaller than $s.B^{2.k}$, so that the number $d(r)$ of B-ary digits of r satisfies the condition $d(r) \le 2.k + \lceil log_B s \rceil$, where s is equal to (15). Thus

$$d(r_{max}) \approx log_B n + 2.k - log_B k < log_B n + 2.k. \tag{16}$$

The core of the algorithm is an (n/k)-step iteration. Each of them includes the product of two k-bit numbers x_i and b_i, and the sum of two $(log_2 n + 2.k)$-bit numbers. The total computing time is approximately equal to

$$\text{time}(n,k) \approx (n/k).(log_2 n + 5.k). \tag{17}$$

D. Barrett reduction algorithms

A generalized version of the Barrett algorithm (Blake *et al*, 2002; Hankerson *et al*, 2004) is presented.

D.1 *n*-digit to (*k*+*t*)-digit reduction

Assume that m belongs to the range $B^{k-1} < m < B^k$ where B is the base (or a power of the base) of the chosen numeration system (if m is a power of B the computation of x mod m is trivial). The value of $z = x$ mod m is the remainder of the integer division of x by m, that is, $x = q.m + z$, $z < m$. The Barrett algorithm starts with the computation of an approximation q' of $q = \lfloor x/m \rfloor$ such that

$$q\text{-}a \le q' \le q. \tag{18}$$

Compute

$$r' = x - q'.m. \tag{19}$$

Taking into account that $z = x - q.m$, then, according to (18), $z \le r' \le z + a.m$. Let t be the minimum integer such that

$$B^t \ge a+1. \tag{20}$$

Then $r' \le z + a.m < (a+1).m < B^{k+t}$. Thus $0 \le z \le r' < B^{k+t}$, so that

$$r' = r' \bmod B^{k+t} = (x - q'.m) \bmod B^{k+t}. \tag{21}$$

Furthermore, according to (19)

$$r' \bmod m = x \bmod m = z. \tag{22}$$

The following algorithm, including a function *approximation* which generates an approximation q' of $\lfloor x/m \rfloor$ - see relation (18) - , computes a $(k+t)$-digit number r equivalent to x mod m:

Algorithm 4 – *n*-digit to (*k*+*t*)-digit reduction

```
q := approximation(x, m);
r := ((x mod B^(k+t)) -
     (q*m mod B^(k+t))) mod B^(k+t);
```

If $a = 2$ and $B \ge 3$, then condition (20) is $B^t \ge 3$ and is satisfied if $t = 1$. Thus $x - q'.m$ can be computed mod B^{k+1}. This case corresponds to the classical Barrett algorithm.

D.2 A first approximation of *q*

Let x and m be expressed in base B:

$$x = x_{n-1}.B^{n-1} + x_{n-2}.B^{n-2} + \ldots + x_0.B^0,$$

$$m = m_{k-1}.B^{k-1} + m_{k-2}.B^{k-2} + \ldots + m_0.B^0, \text{ where } m_{k-1} > 0.$$

The approximation q' of $q = \lfloor x/m \rfloor$ is

$$q' = \lfloor \lfloor x/B^{k-1} \rfloor . \lfloor B^n/m \rfloor / B^{n-k+1} \rfloor.$$

It can be demonstrated (Hankerson *et al*, 2004) that $q \le q' + 2$, that is $a = 2$.

According to (20) the value of t must be chosen in such a way that $B^t \ge 3$. Thus

if $B = 2$, then $t = 2$ (the computation is performed mod B^{k+2}),

if $B > 2$ (classical Barrett algorithm), then $t = 1$ (the computation is performed mod B^{k+1}).

To summarize, the following algorithm computes $z = x \bmod p$. The constant

$$c = \lfloor B^n/m \rfloor \quad (23)$$

must have been previously calculated.

Algorithm 5 – Generalized Barrett reduction

```
y := x/B**(k-1); w := y*c;
q := (w/B**(n-k+1)) mod B**(k+t);
r := ((x mod B**(k+t)) -
      ((q*m) mod B**(k+t))) mod B**(k+t);
while r >= m loop r := r-m; end loop;
z := r;
```

The division by B^{k-1} or B^{n-k+1} and the mod B^{k+t} reduction are trivial operations. The only non-trivial operations are the multiplication by m and the subtractions.

Comment In the classical Barrett algorithm (Blake *et al*, 2002; Hankerson *et al*, 2004), n is assumed to be equal to $2.k$, so that

$$c = \lfloor B^{2.k}/m \rfloor,\ q' = \lfloor \lfloor x/B^{k-1} \rfloor \lfloor B^{2.k}/m \rfloor / B^{k+1} \rfloor.$$

Assuming (best case approximation) that the first value of r is already smaller than m, the computation time is the sum of the delays of an $(n\text{-}k+1)$-bit by $(n\text{-}k+1)$-bit multiplier (computation of w), a $(k+t)$-bit by k-bit multiplier (computation of $q.m$) and a $(k+t+1)$-bit subtractor. It is approximately equal to $((3.(n\text{-}k+1)\text{-}2) + (k+t+2.k\text{-}2) + (k+t+1)).T_{FA}$. If $2.t << 3.n+k$, then

$$\text{time}(n,k) \approx (3.n+k).T_{FA}. \quad (24)$$

A drawback of the Barrett algorithm is the high cost of the multipliers. The cost of an n-bit by m-bit multiplier is proportional to $n.m$ (Deschamps *et al*, 2006). Thus, the total cost of both multipliers is proportional to $(n\text{-}k+1)^2 + (k+t).k \approx (n\text{-}k)^2 + k^2$ whose minimum value (for k smaller than n) is $n^2/2$ (when $k = n/2$).

D.3 A second approximation of *q*

In order to reduce the computation complexity (basically the computation of w), a worse approximation of q can be computed. First observe that $c = \lfloor B^n/m \rfloor$ is an at most $(n\text{-}k+1)$-digit number. Thus

$$w = y.c = c_0.B^0.y + c_1.B^1.y + ... + c_{n-k}.B^{n-k}.y,$$
$$q' = \lfloor y.c / B^{n-k+1} \rfloor = \lfloor c_0.B^{-n+k-1}.y + c_1.B^{-n+k}.y + ... + c_{n-k}.B^{-1}.y \rfloor. \quad (25)$$

Define $q'' = c_0.\lfloor B^{-n+k-1}.y \rfloor + c_1.\lfloor B^{-n+k}.y \rfloor + ... + c_{n-k}.\lfloor B^{-1}.y \rfloor$, that is

$$q'' = c_0.v_0 + c_1.v_1 + ... + c_{n-k}.v_{n-k},$$
$$\text{with } v_i = \lfloor y/B^{n-k-i+1} \rfloor,\ \forall\ i = 0, 1, ..., n\text{-}k. \quad (26)$$

Obviously $q'' \le q'$. Furthermore $q' \le q'' + c_0 + c_1 + ... + c_{n-m} = q'' + weight(c)$, where $weight(c)$ is the sum of all digits of c. Thus $q' - weight(c) \le q'' \le q'$ and $q - 2 - weight(c) \le q'' \le q$, that is, q'' is an approximation (18) of q such that $a = 2 + weight(c)$.

Algorithm 6 – Modified Barrett reduction

```
y := x/B**(k-1);
for i in 0 .. n-k loop
  v(i) := (y/B**(n-k-i+1)) mod B**(k+t);
end loop;
q := c(0)*v(0) + c(1)*v(1)) mod B**(k+t);
for i in 2 .. n-k loop
  q := (q + c(i)*v(i)) mod B**(k+t);
end loop;
r := ((x mod B**(k+t)) -
     ((q*m) mod B**(k+t))) mod B**(k+t);
while r >= m loop r := r-m; end loop;
z := r;
```

The division by B^{k-1} or $B^{n-k-i+1}$ and the mod B^{k+t} reduction are trivial operations. The only non-trivial operations are multiplications by B-ary digits c_i (a trivial operation if B=2), multiplication by m, additions and subtractions. The computation is divided into two parts. First, an $(n\text{-}k)$-step iteration computes q. The corresponding time is approximately $(n\text{-}k).(k+t).T_{FA} \approx (n\text{-}k).k.T_{FA}$. Assuming again (best case approximation) that the first value of r is smaller than m, the second part consists of a $(k+t)$-bit by k-bit product $(q.m)$ and a $(k+t)$-bit subtraction, that is, a delay equal to $((3.k+t\text{-}2) + (k+t)).T_{FA} \approx 4.k.T_{FA}$. Thus, the total time is about

$$\text{time}(n,k) \approx (n\text{-}k+4).k.T_{FA} \approx (n\text{-}k).k.T_{FA}. \quad (27)$$

E. Summary

The main results are summarized in table 1. The approximate computation time, expressed in full-adder delays, is given for every reduction method. In particular, the values obtained when $n = 2.k$ are computed: they correspond to the case where x is the result of multiplying two elements of Z_m, that is, two k-bit numbers.

Table 1. Computation time, expressed in full-adder delays, for reducing an n-bit number modulo a k-bit number

algorithm	*time*(n,k)	*time*$(2.k,k)$
non-restoring division	$(n\text{-}k).k$	k^2
mod 2^k-a	$(n/k).(n+2.k)$	$8.k$
pre-comput of $2^{i.k}$ mod m	$(n/k).(\log_2 n+5.k)$	$10.k$
Barrett	$3.n+k$	$7.k$
modified Barrett	$(n\text{-}k).k$	k^2

As long as the computation time is considered, and assuming that the approximations are reasonably good, the Barrett algorithm is the best choice. Nevertheless, as quoted above, its cost is $O(n^2)$ and could be prohibitively high for great values of n (see next section) .

III. FPGA IMPLEMENTATIONS

Reduction circuits, with $n = 2.k = 16$, 64, 256 and 1024, have been synthesized using ISE6.3i (Xilinx, 2006). The results for an xc3s4000-5 device are given in tables 2 to 6. The cost is expressed in number of slices. Apart from the logic slices, both Barrett algorithms need a lot of 18-by-18-bit multipliers. The xc3s4000-5 device contains 96 such dividers, an insufficient number for implement-

ing Barrett algorithms for great values of n. This fact is indicated by the $\propto$ symbol within the *cost* column. Reduction circuits for $n = 64$ and $m = 239$, so that $k = 8$, have also been synthesized (table 7).

Table 2. Non-restoring division: cost and computation time ($n = 2.k$)

n	*cost* (*slices*)	*minimum period* (*ns*)	*average time* (*ns*)
16	49	7	60
64	133	9	300
256	430	14	1,800
1024	1619	36	19,000

Table 3. Reduction mod 2^k-a: cost and computation time ($n = 2.k$)

n	*cost* (*slices*)	*minimum period* (*ns*)	*average time* (*ns*)
16	25	6	25
64	72	8	35
256	240	13	55
1024	918	35	140

Table 4. Pre-computation of $2^{i.k}$ mod m: cost and computation time ($n = 2.k$)

n	*cost* (*slices*)	*minimum period* (*ns*)	*average time* (*ns*)
16	42	6	50
64	144	9	75
256	536	20	160
1024	2061	62	500

Table 5. Barrett algorithm: cost and computation time ($n = 2.k$)

n	*cost* (*slices*)	*minimum period* (*ns*)	*average time* (*ns*)
16	31	8	25
64	130	10	30
256	$\propto$	-	-
1024	$\propto$	-	-

Table 6. Modified Barrett algorithm: cost and computation time ($n = 2.k$)

n	*cost* (*slices*)	*minimum period* (*ns*)	*average time* (*ns*)
16	62	9	80
64	373	17	650
256	4,245	25	3,300
1024	$\propto$	-	-

Table 7. Cost and computation time ($n = 64$ and $m = 239$)

algorithm	*cost* (*slices*)	*min. period* (*ns*)	*time* (*ns*)
non-restoring division	118	14	850
mod 2^k-a	101	14	300
pre-comput. of $2^{i.k}$ mod m	116	20	600
Barrett	546	13	50
modified Barrett	215	19	1,600

IV. COMMENTS AND CONCLUSIONS

According to both the theoretical analysis (table 1) and the practical synthesis results (tables 2 to 7), the fastest circuits are obtained with the Barrett algorithm. Nevertheless, the corresponding costs are excessive for great values of n. The second best solution, as regards the computation time, is the reduction mod 2^k-a. Actually, these conclusions are valid as long as generic reduction circuits are considered. For specific values of n and m, the pre-computation option could be an interesting alternative (chapter 8 of Deschamps *et al*, 2006). For small values of n, the best option is a block of ROM storing the 2^n pre-computed values of x mod m. In the case where the reduction is part of an algorithm including a lot of multiplications, for example an exponentiation algorithm, an alternative solution is the Montgomery product (Montgomery, 1985). It has not been studied in this paper dedicated to reduction circuits, but is one of the main topics of another (not yet published) work on finite ring and field operations.

REFERENCES

Blake, I.V., G. Seroussi and N. Smart, *Elliptic Curves in Cryptography*. Cambridge University Press (2002)

Hankerson, D., A.J. Menezes and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer (2004)

Deschamps, J.-P., G.A. Bioul, and G.D. Sutter, *Synthesis of Arithmetic Circuits*, Wiley (2006)

Montgomery, P., "Modular Multiplication without Trial Division", Mathematics of Computation, **44**, 519-521 (1985)

Xilinx Inc, http://www.xilinx.com (2006)

Received: April 14, 2006.
Accepted: September 8, 2006.
Recommended by Special Issue Editor Hilda Larrondo.