Improved ZDN-Arithmetic for Fast Modulo Multiplication

Hagen Ploog, Sebastian Flügel and Dirk Timmermann
University of Rostock
Institute of Applied Microelectronic and Computer Science
Richard-Wagner-Str. 31; 18119 Rostock; Germany
Hagen.Ploog@uni-rostock.de

Abstract

In 1987 Sedlak proposed a modulo multiplication algorithm which is suitable for smart card implementation due to it's low latency time. It is based on ZDN (zwei_drittel_N) arithmetic using an interleaved serial multiplication and reduction to calculate the product P=AB mod M. It can be shown that the maximum average reduction rate is theoretically limited to 3 bit/operation.

In this paper we propose a modified left-to-right signed digit (SD)-recoding algorithm to receive an average shift of 4.5 bit/operation. Based on the presented ideas we also propose a modified reduction algorithm giving an average reduction rate of 4.5 bit/operation, too. The speed up of our algorithms compared with the original algorithm is therefore 50 %.

1. Introduction

During the execution of the modulo multiplication of $P = AB \bmod M$ the multiplying factor B is analyzed from the left most significant bit b_{n-1} to the least significant bit b_0, with n being the number of bits in the binary representation of B. After each multiplication the intermediate result $P_i = 2P_{i-1}+Ab_i$ is reduced by the modulus M. The throughput during the serial execution can be increased if the architecture supports the skipping over the zero-elements of B since only for b_i=1 an addition of A is required. The number of skipped bits is sp. The skipping is realized by using a logarithmical barrel shifter with k-stages. Often a recoding of the multiplicand B to a binary signed-digit representation D_{SD2} with $d_i \in \{-1, 0, 1\}$ is used to minimize the average number of non-zeros-elements in D and therefore the average number of executed additions.

Wu and Hasan [1] proved that SD-recoding of an n-bit number can reduce the number of non-zeros to an average of n/3 allowing the multiplication to work in an average speed of 3 bits per operation (addition or subtraction).

Sedlak's [2] main idea is not to reduce the partial product P_i into the range [0..M-1] in one step, but in several steps during the multiplication, since $A \bmod M = (A+x \cdot M) \bmod M$. Therefore, the architecture must be able to shift the modulus M in and out of an additional buffer of a given size $x_{\max}$. The reduction itself is performed by a subtraction of the modulus M if P_{i-1} is positive, or with an addition of the modulus otherwise.

Let x_{i-1} be the current shift of M into the buffer, then x_i is calculated as $x_i = x_{i-1}+sp-sm$. Where sm is the possible dynamic reduction of x during the multiplication computed by the reduction look ahead mechanism. By adding sp to x_{i-1}, M is shifted relative to P_i and it is guaranteed that P_i is always smaller than $2^{xi}M$ and we finally get:

$$P_i = 2^{sp} \cdot P_{i-1} + d_i \cdot A \mp 2^{x_{i-1}+sp-sm} M$$

Sedlak used a three operand–adder realized in carry-save-technique. To minimize the required number of registers the redundant result is converted back to it's binary representation using a carry-propagate adder (CPA). The look ahead mechanisms for multiplication and reduction are independent of each other but coupled in that way that the average reduction rate is the same.

The average reduction rate is theoretically limited to 3 bit/operation but can not be achieved in an implementation since the throughput of the architecture is limited by several factors: the number of stages of the barrel shifters, the number of additional registers for buffering M, the number of bits to be skipped that was calculated during SD-recoding and the number of bits to be skipped computed by the reduction look ahead.

In this paper we propose a new SD-recoding algorithm to receive an average of 4.5 bit/operation for the multiplication. Based on the presented ideas we also propose a new reduction algorithm to receive an average of 4.5 bit/operation during reduction.

The paper is organized as follows: In chapter 2 we give a short review on SD-recoding and present our algorithm for simplified SD-recoding in chapter 3. Chapter 4 summarizes the reduction look ahead and presents the new algorithm. We finally conclude with chapter 5.

2. SD-recoding (left-to-right)

2.1. Standard method

It is well known that the transformation of a binary represented number B into a signed-digit represented number D_{SD2}, with $d_i \in \{-1, 0, 1\}$, can reduce the number of non-zero elements in D to an average of $n/3$ and $n/2$ in the worst case, with n being the number of bits necessary for the binary representation of B [1]. Sedlak [3] proposed a look-up-table for SD_2-recoding which is based on the following two rules:

1. $\left(\langle 1\rangle^{a}\right)_2 \mapsto \left(1, \langle 0\rangle^{a-1}, \bar{1}\right)_{SD2} \quad \forall a \geq 2$
2. $\left(\bar{1}, 1\right)_{SD2} = \left(0, \bar{1}\right)_{SD2}$

For left-to-right SD-recoding an additional borrow-signal bc is required to indicate if a '1'-chain or a '0'-chain is skipped. Table 1 summarizes all possible cases. The inputs are the next three bits of B to be analyzed and the current bc. The outputs are the corresponding d_i and the next value for bc, called bc'. Joye and Yen [4] proved that usage of Table 1 produces optimal recoded SD-numbers with the same (minimal) Hamming-weight as the corresponding canonical SD-numbers. The algorithm itself requires a leading and two trailing zeros. The algorithm starts with $bc=0$ and b_n=0.

	b_i	b_{i-1}	b_{i-2}	d	bc'
bc=0	0	0	X	0	0
	0	1	0	0	0
	0	1	1	1	1
	1	0	X	1	0
	1	1	X	X	X
bc=1	1	1	X	0	1
	1	0	1	0	1
	1	0	0	-1	0
	0	1	X	-1	1
	0	0	X	X	X

Table 1 : left-to-right SD-recoding

2.2. Simplified SD-recoding

It can be seen that using Sedlaks algorithm bit b_{i-2} is required only to detect the start of a '1'-chain (bc=0, 010/ 011) or the end of a '1'-chain (bc=1, 101/100). In comparison to the original proposed algorithm, simplified SD-recoding requires only the two next bits of B to compute the next digit of D and it is based on the same rules as the original algorithm. Obviously, we can't detect the beginning and the end of a '1'-chain correctly. But this doesn't matter until the least significant bit (LSB) of B is reached, since

$$\ldots 0100 \ldots_2 = \ldots 0(2\cdot 1)0 \ldots_4.$$

We call this mechanism deferred correction. It's worth notifying that the length of the recoded number is now the same as in the binary representation, which is not true for Sedlak's method. Clearly, we actually do not profit from this modification until now since it requires more die area without any improvements in terms of speed. The algorithm works as follows:

ALGORITHM 1:
INPUT: $(b_{m-1}, \ldots, b_0)_2$
OUTPUT : $(d_{m-1}, \ldots, d_0)_{SD}$

```
bc_{m-1} ← 0; b_{-1} ← 0
for i in m-1 downto 1 loop
  bc_{i-1} ← ⌊(bc_i + b_i + b_{i-1})/2⌋
  d_i ← bc_{i-1} + b_i − 2·bc_i
loop end
d_0 ← b_0 − 2·bc_0
```

2.3. Calculation of the speedup

D_{SD} contains a certain number of non-zeros, each requiring the multiplication look ahead to stop for an operation. In this section we will show how many operations are saved by the modification of the multiplication look ahead. Figure 1 depicts a Moore-type finite state machine (FSM) for simplified sd-recoding. Because of it's symmetry, the FSM can be cut into two identical parts. It can be seen that S_3 is equivalent to $\overline{S_7}$.

To ease further calculation we set: S_1=S_5, S_2=S_6, S_3=S_7, and S_4=S_8. To calculate the average increase of the speed-up we first compute the probability P_{add} of a second addition within the following next two bits of B:

$$\begin{aligned} P(add \mid S_3) &= P(S_3 \mid 10) + P(S_3 \mid 11) \\ &= 0.25 + 0.25 = 0.5 \\ P(add \mid S_8) &= P(S_8 \mid 10) + P(S_8 \mid 11) \\ &= 0.25 + 0.25 = 0.5 \end{aligned}$$

Next, we have to calculate the probability P_{stop} that the FSM terminates in a given state S_x because that terminating state is the next state the FSM will start from. We therefore have to compute the length of a chain between two terminating states. We first start at S_3. With "10" as input we will be back in S_3 and terminate. We will also terminate in S_3 again if we receive "010", "0010", … "0(n·0)10" as input pattern. More formally, we can say that if we start in S_3 we will end in S_3 with a probability of 50%, or:

$$\begin{aligned} P(S_3 \mapsto S_3) &= \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \cdots \\ &= \tfrac{1}{4}\cdot\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{4}\cdot\frac{1}{1-\frac{1}{2}} = \tfrac{1}{2} \\ P(S_3 \mapsto S_8(S_4)) &= \tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2}\cdot\tfrac{1}{2} + \cdots \\ &= \tfrac{1}{4}\cdot\sum_{i=0}^{\infty} 2^{-i} = \tfrac{1}{4}\cdot\frac{1}{1-\frac{1}{2}} = \tfrac{1}{2} \end{aligned}$$

In whatever state the FSM starts, it will terminate with the same probability in S_3 as in S_8. Now we are able to compute the probability of a second termination P_{H2} within the next two following bits of B:

$$P_{H2} = P(HALT \mid S_3) \cdot P(S_3 \mid add) + P(HALT \mid S_8) \cdot P(S_8 \mid add) = \tfrac{1}{2}$$

We gain no speed up if there is no termination within the next two bits, but if there is one the speed up factor is 100% because of two additions executed in a single step instead of one. The average width of the shift using this modification on simplified SD-recoding is therefore

$$(1 - P_{H2}) \cdot 3 \frac{bit}{operation} + P_{H2} \cdot 2 \cdot 3 \frac{bit}{operation} = 4.5 \frac{bit}{operation}$$

It can be shown [5] that the same modification performed on Sedlak's original algorithm leads to an improvement of $\frac{7}{16}$ which is obviously less than 0.5.

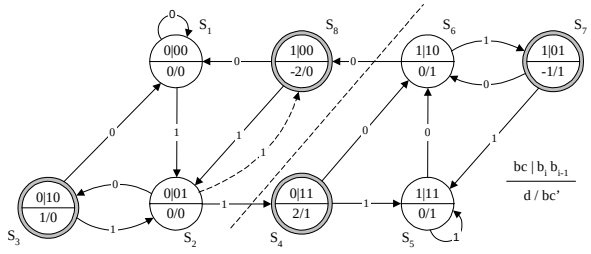

Figure 1: FSM for simplified SD-recoding

3. Modulo reduction

3.1. Standard method

The modulo reduction of $P = AB$ mod M is equivalent to finding a quotient Q such that $AB = P + QM$ is satisfied. Let $Q=q_{n-1}...q_0$, then, according to the division algorithm of Sedlak, each q_i is serially determined beginning at the MSB of Q. Let $P = AB - QM$, then M has to be subtracted if q_i=1 or added if q_i= -1. Unfortunately, no closed-form expression of the division algorithm exists, since d_iA is added in every step to the partial product P_{i-1}. To determine the next q_i merely a set of rules exists [7]:

```
ALGORITHM 2:
INPUT: M, P_{i-1}
OUTPUT : sm
  sm←0
  while (2/3)M2^{-sm} > |P_{i-1}| do
    sm = sm +1
  wend
  return sm
```

Remember, that M could be shifted in and out to an additional buffer. The actual shift into the buffer is x and has to be reduced to zero at the end of the reduction. So one comparator for each bit to shift has to be implemented. For a better understanding we now analyze the division algorithm itself without considering the shift of P_{i-1} and the addition of d_iA.

The average reduction rate of three bit per operation points to the fact that a SD-division is performed. We first show, that the canonical signed digit LSB-first serial multiplication is the inverted function to the division according to Sedlak. In [6] Reitwiesner proposed a canonical recoding algorithm for transforming a binary represented number $Q = (q_{n-1}, ..., q_0)_2$ with $q_i \in \{0, 1\}$ to a signed digit representation $D = (d_n, d_{n-1}, ..., d_0)_{SD2}$ with $d_i \in \{-1, 0, 1\}$. It is based on the same rules given in chapter 2. The main difference is that Reitwiesner's algorithm starts at the LSB instead of the MSB. The algorithm starts with bc=0 and requires two leading zeros. Table 2 summarizes all possible cases. Reitwiesner also proved that the output is the canonical representation D_{SD2} of Q. A signed-digit representation is called to be canonical if it has the minimal Hamming weight and for $0 \le i \le n$-1 the following is true: $d_i \cdot d_{i+1} = 0$.

Lemma:

The canonical signed digit LSB first multiplication is the inverted function to the division algorithm proposed by Sedlak.

Proof:

The serial multiplication starts at the LSB. In the first iteration 2^0M is added to or subtracted from P_i if $D_{SD,0} \neq 0$. During the i.th iteration $2^{i-1}M$ is added or subtracted if $D_{SD,i-1} \neq 0$. Since $D_{SD,i-1}$ could be 1 or –1 two cases can occur:

1) Let $D_{SD,i-1} = 1$.

Taking the Hamming distance of two into consideration, $D_{SD,i}$ is currently bounded by:

$$D_{SD,i} \le 2^{i-1} + 2^{i-3} + 2^{i-5} + \ldots = 2^i \cdot 0.101010\ldots_2 < \tfrac{2}{3} 2^i$$

$$D_{SD,i} \ge 2^{i-1} - 2^{i-3} - 2^{i-5} - \ldots = 2^i \cdot (0.1_2 - 0.001010\ldots_2) > \tfrac{1}{2} 2^i - \tfrac{1}{6} 2^i = \tfrac{1}{3} 2^i$$

And therefore:

$$\tfrac{1}{3} 2^i < D_{SD,i} < \tfrac{2}{3} 2^i \quad \text{and}$$

$$\tfrac{1}{3} M 2^i < P_{i-1} < \tfrac{2}{3} M 2^i.$$

with $P_{i-1} = D_{SD,i}M$.

2) Now let $D_{SD,i-1} = -1$. Then $D_{SD,i}$ is bounded by:

$$D_{SD,i} \le -2^{i-1} + 2^{i-3} + 2^{i-5} + \ldots = -2^i \cdot (0.1_2 - 0.001010\ldots_2)$$
$$< -\tfrac{1}{3} 2^i$$
$$D_{SD} \ge -2^{i-1} - 2^{i-3} - 2^{i-5} - \ldots = -2^i \cdot 0.10101\ldots_2$$
$$> -\tfrac{2}{3} 2^i$$

And therefore:

$$-\tfrac{1}{3} 2^i > D_{SD,i} > -\tfrac{2}{3} 2^i \quad \text{and}$$
$$-\tfrac{1}{3} M 2^i > P_{i-1} > -\tfrac{2}{3} M 2^i.$$

with $P_{i-1} = D_{SD,i} M$.

It could be observed that the sign of P_{i-1} and the sign of the actual MSB of $D_{SD,i}$ are always identical. This leads to:

$$\tfrac{1}{3} 2^i < |D_{SD,i}| < \tfrac{2}{3} 2^i \quad \text{and}$$
$$\tfrac{1}{3} M 2^i < |P_{i-1}| < \tfrac{2}{3} M 2^i \quad \text{or}$$
$$\tfrac{2}{3} M 2^{i-1} < |P_{i-1}| < \tfrac{2}{3} M 2^i$$

This all is true if $D_{SD,i} \neq 0$. If $D_{SD,i} = 0$ then i is reduced by one and the inequality is tested again. This is exactly how the next q_i is determined by the division algorithm according to Sedlak. The comparison with 2/3M is naming the ZDN-arithmetic [8].

bc=0				*bc*=1			action
q_{i+1}	q_i	d	bc'	q_{i+1}	q_i	d	bc'	
X	0	0	0	X	1	0	1	skip
0	1	1	0	1	0	-1	1	add / sub
1	1	-1	1	0	0	1	0	

Table 2 : Look-up table according to Reitwiesner's canonical recoding algorithm

3.2. New division

To speed up the division we will speed up the multiplication and invert the result. The basic idea on how to speed up the serially executed multiplication is already presented in chapter 2. By expanding Reitwiesener´s CSD-Recoding algorithm [6] we get Table 3 which summarizes all possible cases wherein $d_i \neq 0$ and an addition or subtraction has to be performed. If $d_i = 0$ this particular position can be skipped and i can be increased until $d_i \neq 0$.

bc_{i-1}	q_{i+3}	q_{i+2}	q_{i+1}	q_i	d_{i+2}	d_{i+1}	d_i	bc_{i+2}	bc_{i+1}	bc_i
0	X	X	X	0	X	X	0	X	X	0
	0	0	0	1	0	0	1	0	0	0
	0	0	1	1	1	0	-1	0	1	1
	0	1	0	1	1	0	1	0	0	0
	0	1	1	1	0	0	-1	1	1	1
	1	0	0	1	0	0	1	0	0	0
	1	0	1	1	-1	0	-1	1	1	1
	1	1	0	1	-1	0	1	1	0	0
	1	1	1	1	0	0	1	1	1	1
1	X	X	X	1	0	0	0	X	X	1
	1	1	1	0	0	0	–1	1	1	1
	1	1	0	0	-1	0	1	1	0	0
	1	0	1	0	-1	0	–1	1	1	1
	1	0	0	0	0	0	1	0	0	0
	0	1	1	0	0	0	–1	1	1	1
	0	1	0	0	1	0	1	0	0	0
	0	0	1	0	1	0	–1	0	1	1
	0	0	0	0	0	0	–1	0	0	0

Table 3 : Extended serial canonical signed digit LSB-first multiplication

The intermediate partial product P_i is than given by :

$$P_{i+1} = 2^i P_i + (4D_{SD,i+2} + 2D_{SD,i+1} + D_{SD,i}) \cdot M$$

P_{i+1} is positive if d (d=$4D_{SD,i+2} + 2D_{SD,i+1} + D_{SD,i}$) is positive, otherwise P_{i+1} is negative. We only perform this step of the multiplication if $b_i \neq bc_i$. If we want to invert this step, two cases can occur:
1) If $P_{i+1} > 0$, we know that:

I. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 0\ 0\ 1$ or
II. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 1\ 0\ -1$ or
III. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 1\ 0\ 1$.

2) If $P_{i+1} < 0$, we know that:

I. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 0\ 0\ -1$ or
II. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = -1\ 0\ 1$ or
III. $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = -1\ 0\ -1$.

Due to the inversion, the sign of P_{i+1} selects which of the above cases is used. In the following, we will focus on the explanation on positive P_{i+1} (case 1) only.

If d =001 (I.), we can skip over the leading zeros with a shift of 2. d now holds 1xx. If we look at the next two digits of d with respect of a Hamming weight of two, three cases can occur:

a. $d = D_{SD,i}\ D_{SD,i-1}\ D_{SD,i-2} = 1\ 0\ 0$ or
b. $d = D_{SD,i}\ D_{SD,i-1}\ D_{SD,i-2} = 1\ 0\ -1$ or
c. $d = D_{SD,i}\ D_{SD,i-1}\ D_{SD,i-2} = 1\ 0\ 1$.

The latter two cases are already handled with II. and III. Case a. indeed is case I. with a shift of 2. At least three cases have to be analysed:

1.) $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 1\ 0\ 0$ or
2.) $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 1\ 0\ -1$ or
3.) $d = D_{SD,i+2}\ D_{SD,i+1}\ D_{SD,i} = 1\ 0\ 1$.

Case 1: D_{SD} is bounded by:

$$D_{SD} \le 2^{i+2} + 2^{i-1} + 2^{i-3} + 2^{i-.5} \ldots < \left(4 + \tfrac{2}{3}\right) \cdot 2^i$$
$$D_{SD} \ge 2^{i+2} - 2^{i-1} - 2^{i-3} - 2^{i-}5 \ldots > \left(4 - \tfrac{2}{3}\right) \cdot 2^i$$

Case 2: D_{SD} is bounded by:

$$D_{SD} \le 2^{i+2} - 2^{i} + 2^{i-2} + 2^{i-4} + 2^{i-.6} \ldots < \left(4 - 1 + \tfrac{1}{3}\right) \cdot 2^i$$
$$D_{SD} \ge 2^{i+2} - 2^{i} - 2^{i-2} - 2^{i-4} - 2^{i-6} \ldots > \left(4 - 1 - \tfrac{1}{3}\right) \cdot 2^i$$

Case 3 : D_{SD} is bounded by:

$$D_{SD} \le 2^{i+2} + 2^{i} + 2^{i-2} + 2^{i-4} + 2^{i-.6} \ldots < \left(4 + 1 + \tfrac{1}{3}\right) \cdot 2^i$$
$$D_{SD} \ge 2^{i+2} + 2^{i} - 2^{i-2} - 2^{i-4} - 2^{i-6} \ldots > \left(4 + 1 - \tfrac{1}{3}\right) \cdot 2^i$$

Considering the sign of P_{i+1} we finally get:

$$\tfrac{8}{3} M 2^i \le |P_{i+1}| < \tfrac{10}{3} M 2^i \rightarrow |d| = 10\bar{1}$$
$$\tfrac{10}{3} M 2^i \le |P_{i+1}| < \tfrac{14}{3} M 2^i \rightarrow |d| = 100$$
$$\tfrac{14}{3} M 2^i \le |P_{i+1}| < \tfrac{16}{3} M 2^i \rightarrow |d| = 101$$

If $|P_{i+1}|$ is not inside the range [8/4*M*, 16/3*M*) we know that the current most significant bit of Q_{SD} is zero and the algorithm has to be started again at the next bit downward the LSB. This is realized by shifting the compare-value to the LSB as it was proposed by Sedlak.

3.3. Calculation of the speed up

As with the multiplication speed up, the speed up calculation of the modulo operation shall be performed on it's finite state machine representation. Since we do not have a closed representation of the reduction, we will use the LSB-first multiplication instead. The inverse operation is connected to the normal operation in a way that the inverse of each operation is being performed in a reverse order. All intermediate results are identical but in reverse order, too. Hence, the outcome of one operation is the starting value of its inverse operation. The numbers of steps for both operations are identical.

The Moore-type state machine of the LSB-first multiplication is given in figure 2.

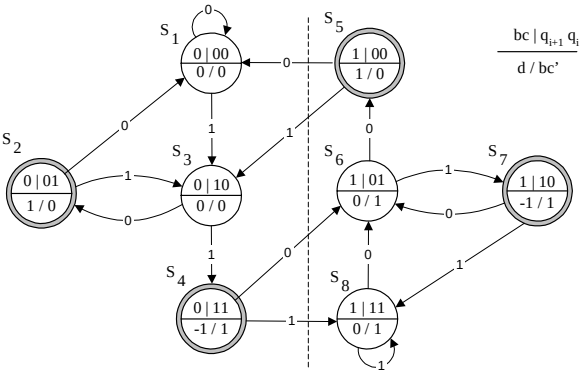

Figure 2 : FSM for LSB-first SD-recoding

Again, a speed up occurs if another operation will have to be performed within the next two digits of D_{SD}. If the original modulo look ahead terminates in either states S_2 or S_7 the chance of finishing the next look ahead within the next two steps is P=0.5. If it terminates in states S_4 or S_5 the probability is P=0.5, too.

For each of the critical states we have to compute the probability P_{stop} that the FSM terminates in that specific state. It has to be noted that since an operation is forced at states S_2, S_4, S_5, and S_7 only, the FSM will start in those states for the next operation. First, we start at state S_2. The FSM will terminate in the same state if it receives an input of "10", or "010", or "0010", and so on. Assuming the probabilities of a bit-value being P(0)=P(1)=0.5 leads to:

$$P(S_2 \mapsto S_2) = \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} + \tfrac{1}{32} + \ldots$$
$$= \tfrac{1}{4} \sum_{i=0}^{\infty} \left(\tfrac{1}{2}\right)^i = \tfrac{1}{4} \frac{1}{1 - \frac{1}{2}} = \tfrac{1}{2}$$
$$P(S_2 \mapsto S_4) = \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} + \tfrac{1}{32} + \ldots = \tfrac{1}{2}$$
$$P(S_4 \mapsto S_7) = \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} + \tfrac{1}{32} + \ldots = \tfrac{1}{2}$$
$$P(S_4 \mapsto S_5) = \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} + \tfrac{1}{32} + \ldots = \tfrac{1}{2}$$

Because of the symmetry, state S_2 can be considered equivalent to state S_7 in terms of probability, as with state S_4 and S_5, respectively. So, by starting from any state the chances for each final state are equal. This leads to the probability P_{O2} for an operation within the next two digits:

$$P_{O2} = P(S \mid S_2) \cdot P(add \mid S_2) + P(S \mid S_4) \cdot P(add \mid S_4)$$
$$= \tfrac{1}{2} \cdot \tfrac{1}{2} + \tfrac{1}{2} \cdot \tfrac{1}{2}$$
$$= \tfrac{1}{2}$$

As for the multiplication look ahead, the modification accelerates the modulo reduction by 50% from 3 bit/operation to 4.5 bit/operation.

4. Implementation

The comparison in algorithm 2 is the most time and area critical part to implement. The right result could only be obtained if all bits of the two numbers were compared. Instead of implementing a full comparator only the *j*-most significant bits were analyzed, depending of the accuracy the result should obtain. The probability of making a wrong decision is then given with $2^{-(j+1)}$. The intermediate result will be corrected during the next step of the algorithm since $A \bmod M = (A+x\cdot M) \bmod M$.

Let M be the product of two large prime numbers p and q as it is recommend for the RSA-algorithms [9]. However, with M being a constant during the runtime of the algorithm $\frac{2}{3}M$ (ZDN) is a constant, too.

Now let p and q be such primes, that the most significant bits of M look like "1100..0" and we get

$$ZDN = \tfrac{2}{3}M = \tfrac{2}{3}\cdot 1100..0 = 1000..,$$

which significantly eases the design of the comparator.

Unfortunately, the choice of such p and q will constrain the system modulus.

5. Conclusion

The modulo multiplication can be realized efficiently with respect to the number of necessary operations with a serial architecture. To receive a speed up using Sedlak's modulo multiplication algorithm a speed up of both, the multiplication and the reduction, is necessary. It was shown in this paper that both operations are based upon efficient SD-recoding of operands.

The improvements lead to a speed up of 50% to 4.5 bit per operation. The proposed optimisations could be realized with a reasonable cost of additional hardware.

Acknowledgments. We would like to thank Dr. Jean-Pierre Seifert (Infinion) for lots of valuable discussions. Also, we would like to thank the referees for careful reading and improving the quality of the presentation.

6. References

[1] H. Wu and M. A. Hasan, "Closed-Form Expression for the Average Weight of Signed Digit Representations", IEEE Transactions on Computers, Vol. 48, No. 8, August 1999

[2] H. Sedlak, United States Patent No 4.870.681, Sep. 26,1989

[3] H. Sedlak, "The RSA cryptography processor", Proc. of Eurocrypt '87, LNCS 304, Springer-Verlag, pp.95-105

[4] M. Joye and S.-M. Yen, "Optimal Left-to-Right Binary Signed-Digit Recoding", IEEE Transactions on Computers, Vol. 49, No. 7, July 2000

[5] H. Ploog, S. Flügel and D. Timmermann, "4.3 Bit/Operation bei serieller Multiplikation durch modifiziertes SD-recoding", 10. E.I.S.-Workshop, 3.-5. April, Dresden, 2001

[6] G.W. Reitwiesner, "Binary Arithmetic", Advances in Computers, vol. 1, pp.231-308, 1960

[7] H. Sedlak, "Ein Public-Key-Code Kryptographie-Prozessor", 2. E.I.S.-Workshop, GMD, Bonn, 1986

[8] E. Hess, B. Meyer, N. Janssen, "Design of Long Integer Arithmetic Units for Public-Key Algorithms", Proc. of EUROSMART Security Conference 2000, pp.325-334, 2000

[9] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Communications of the ACM, Vol. 21, No. 2, pp.120-127, February 1978