A Self-Timed Redundant-Binary Number to Binary Number Converter for Digital Arithmetic Processors

Chin-Long Wey, Haiyan Wang*, and Cheng-Ping Wang
Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824-1226
*: Department of Mathematics

ABSTRACT

This paper presents a self-timed converter circuit which converts an n-digit redundant binary number to an (n+1)-bit binary number. Self-timed refers to the fact that the conversion is problem-dependent and requires variable conversion time to complete the operation. The propagation delay of the proposed converter circuit does not increase with the number of digits to be converted, but it is determined by the maximum number of consecutive 0's in that number. This study shows that the statistical upper bound of the average maximum number of consecutive 0's is $\log_3 n$, or 3.78 for 64-digits. This implies that the proposed self-time circuit can be approximately 17 times faster than the ripple-type converter. Thus, the proposed converter is well-suited to high-speed, long-word digital arithmetic processors.

1. Introduction

The speed of a digital arithmetic processor depends heavily on the speed of the adders used in the system [1,2]. The conventional addition of two numbers in classical binary form is usually the bottleneck for speed improvement. While other digital operations may be performed in parallel (shift, data storage, etc.), addition requires a carry propagation. Various solutions have been proposed to solve the carry propagation problem, such as *carry save* and *redundant binary* representations. Carry save adders are widely used in fast arithmetic processors and they reduce the sum of three binary numbers to the sum of two binary numbers without carry propagation. With an appropriate set of rules, the binary redundant representation can achieve carry-propagation-free addition [3]. Both redundant binary and carry save representations are in essence similar. The main difference is the fact that the former does not need to use the 2's complement form to handle negative numbers. Thus, the redundant binary representation is more nature to be used if both positive and negative numbers have to be processed [4].

Redundant binary representation has been implemented to design high performance multipliers and dividers [4,5] using carry-propagation-free additions. However, with the redundant binary representation, the adder cell requires more chip area than the conventional full adder cell, and the final results in redundant binary representation need to be converted to a binary form. Several efficient converter circuits that convert a redundant binary number to a two's complement binary number have been reported [6,7]. The *on-the-fly converter* [6] works from the most-significant digit end to the least-significant digit end achieving a carry-free conversion. However, the carry-free conversion scheme is penalized by using more gates to implement it, and the conversion time increases with the number of digits to be converted. In [7], an efficient conversion algorithm was proposed with a simple converter circuit. A redundant binary number D is decomposed into two numbers D^+ and D^-, where D^+ includes all digits of D except replacing $\bar{1}$'s to 0's, and D^- contains all digits of D except replacing 1's to 0's and $\bar{1}$'s to 1's. Thus, $D=D^+-D^-$. A set of simple conversion rules was derived to find the difference. The conversion process starts from the least-significant digit end to the most-significant digit end. Thus, the conversion time for the serial-mode converter also increases with the number of digits to be converted. To speed up the conversion, a concept similar to carry lookahead addition has also presented [7].

The objective of this paper is to present an alternative conversion algorithm and its hardware implementation. The algorithm converts an n-digit redundant binary number $D=(d_{n-1}, d_{n-2},\ldots,d_1,d_0)$, where $d_i \in \{\bar{1},0,1\}$, and $\bar{1}$ represents "-1", to an (n+1)-bit 2's complement binary number $B=(b_n,b_{n-1},b_{n-2},\ldots,b_1,b_0)$, where $b_i \in \{0,1\}$. The detail conversion algorithm will be discussed shortly. Basically, the major step in this conversion algorithm is to transform the patterns (00..01) and (00...0$\bar{1}$) to (1$\bar{1}$....$\bar{1}$) and ($\bar{1}$1...1), respectively. Note that no transformation is needed if all digits in D are non-zero. Since the number of consecutive 0's

1063-6404/95 $4.00 © 1995 IEEE

and their locations are not known in advance, a serial-type conversion is needed and a flag is used to sense the existence of 0's. It should be mentioned that all new flags are originated from the stages with non-zero-digits, i.e., 1 or $\bar{1}$, and they will pass to their left stages if the passing stages with zero-digits and stop passing if a non-zero-digit occurs. In other words, all the patterns, (00..01) or (00...0$\bar{1}$), in D are transformed simultaneously. Thus, the entire transformation is done whenever the pattern(s) with the longest length of consecutive 0's has been completely transformed. For example, if the numbers of consecutive 0's in an n-digit redundant binary number to be converted do not exceed 3, this implies that the converted binary number is valid at the end of 3 delay units instead of the worst-case n delay units. This leads to the development of a self-timed converter circuit. *Self-timed* refers to the fact that the conversion is problem-dependent and requires variable conversion time to complete the operation. This study will show that statistical results for the average longest length of the consecutive 0's in an n-digit redundant binary number. Results show that the average is 3.434 for 64-digit with a sample size of 100,000, and its statistical upper bound is 3.78. This implies that the speed-up ratio is approximately 64/3.78≈17 for 64-digit conversion.

In the next section, the proposed conversion algorithm and its hardware implementation are presented. Section 3 discusses the simulation and statistical results of the longest length of consecutive 0's in n-digit redundant binary numbers and presents the design of the self-timed converter circuit and its physical layout. Finally, a concluding remark is given in Section 4.

2. Development

Given an n-digit redundant binary number $D=(d_{n-1},d_{n-2},\ldots,d_1,d_0)$, where $d_i \in \{\bar{1},0,1\}$, the most negative number for such a representation is $(\bar{1}\bar{1}..\bar{1})=(-2^n+1)$. Therefore, it is necessary to use (n+1) bits to represent the value of an n-digit redundant binary number, where value of D, or D_v, can be represented as

$$D_v = \sum_{i=0}^{n-1} d_i \cdot 2^i \qquad (1)$$

Our goal is to efficiently convert a given number D to an (n+1)-bit 2's complement binary number $B=(b_n,b_{n-1},b_{n-2},\ldots,b_1,b_0)$, where $b_i \in \{0,1\}$ and b_n is the sign bit. This section first introduces the basic concept of the conversion algorithm, and then discusses the hardware implementation. The correctness of the conversion algorithm is verified by comparing the values of both representations.

A. Conversion Algorithm

The conversion process starts with a simple conversion step which results in an intermediate (n+1)-digit redundant binary number $C=(c_n,c_{n-1},\ldots,c_1,c_0)$. The conversion rule is

$$c_i=d_{i-1},\ i=1,2,\ldots,n;\ \text{and}\ c_0=1. \qquad (2)$$

This step simply shifts the number D to left by one digit and sets a 1 to the least-significant digit. The second step is to transform the number C to the other intermediate (n+1)-digit redundant binary number $A=(a_n,a_{n-1},\ldots,a_1,a_0)$, where a_i, $i=0,1,\ldots,n$, is either 1 or $\bar{1}$, and the transformation rules are: (The letter "x" means an non-zero digit, i.e., x=1 or $\bar{1}$.)

$$\begin{array}{ll} \text{(a)} & (c_i,c_{i+1})=(x,1) \Rightarrow (a_i,a_{i+1})=(x,1); \\ \text{(a')} & (c_i,c_{i+1})=(x,\bar{1}) \Rightarrow (a_i,a_{i+1})=(x,\bar{1}); \\ \text{(b)} & (c_i,c_{i+1},c_{i+2})=(x,0,1) \Rightarrow (a_i,a_{i+1},a_{i+2})=(x,1,\bar{1}); \\ \text{(b')} & (c_i,c_{i+1},c_{i+2})=(x,0,\bar{1}) \Rightarrow (a_i,a_{i+1},a_{i+2})=(x,\bar{1},1); \\ \text{(c)} & (c_i,c_{i+1},\ldots,c_{j-1},c_j)=(x,0,\ldots,0,1) \Rightarrow \\ & \qquad (a_i,a_{i+1},a_{i+2},\ldots,a_{j-1},a_j)=(x,1,\bar{1},\ldots,\bar{1},\bar{1}); \\ \text{(c')} & (c_i,c_{i+1},\ldots,c_{j-1},c_j)=(x,0,\ldots,0,\bar{1}) \Rightarrow \\ & \qquad (a_i,a_{i+1},a_{i+2},\ldots,a_{j-1},a_j)=(x,\bar{1},1,\ldots,1,1); \end{array} \qquad (3)$$

Since the transformation does not make any change in value, we have $A_v=C_v$. Finally, the last step is to convert A to an (n+1)-digit binary number B by the following rules:

$$\begin{array}{ll} \text{(a)} & b_i = (1+a_i)/2;\ i=0,1,\ldots,n-1,\ \text{and} \\ \text{(b)} & b_n=(1-a_n)/2. \end{array} \qquad (4)$$

This step simply duplicates B from A except replacing a $\bar{1}$ by a 0. By Rule 4(a), if a_i=1, then b_i=1; and if $a_i=\bar{1}$, then b_i=0. Rule 4(b) converts the sign bit, where a_n=1 indicates a positive number and thus b_n=0, and $a_n=\bar{1}$ means a negative number and thus b_n=1. The conversion algorithm is summarized in Algorithm 1.

```
Algorithm 1.
    Step 1.  D-C Step;
             $D=(d_{n-1},d_{n-2},\ldots,d_1,d_0) \Rightarrow C=(c_n,c_{n-1},\ldots,c_1,c_0)$;
             $d_i, c_i \in \{\bar{1},0,1\}$; $c_i=d_{i-1}$, $i=1,2,\ldots,n$; and $c_0=1$.
    Step 2.  C-A Step;
             $C=(c_n,c_{n-1},\ldots,c_1,c_0) \Rightarrow A=(a_n,a_{n-1},\ldots,a_1,a_0)$;
             $a_i \in \{\bar{1},1\}$; $c_i \rightarrow a_i$, by rule (3).
    Step 3.  A-B Step;
             $A=(a_n,a_{n-1},\ldots,a_1,a_0) \Rightarrow B=(b_n,b_{n-1},\ldots,b_1,b_0)$,
             $b_i \in \{0,1\}$; $a_i=1 \rightarrow b_i=1$; $a_i=\bar{1} \rightarrow b_i=0$;
             $a_n=1 \rightarrow b_n=0$; $a_n=\bar{1} \rightarrow b_n=1$.
```

The following example illustrates the stepwise conversion procedure: Given a 7-digit redundant binary number D=($\bar{1}0\bar{1}0010$), the *D-C step* produces a 8-digit redundant binary number C=($\bar{1}0\bar{1}00101$), the *C-A step* transforms C to a 8-digit number A=($\bar{1}\bar{1}11\bar{1}\bar{1}1\bar{1}$), and, finally, the *A-B step* generates a 8-bit binary number B=(10110010).

Theorem 1. $B_v=D_v$.

Give the redundant binary number $D=(\bar{1}0\bar{1}0010)_2$, its value in decimal number representation is $D_v = -78_{10}$ and the decimal value of $B=(10110010)_2$ is also -78_{10}. Thus, $B_v = D_v = -78_{10}$. Theorem 1 verifies the correctness of the conversion process.

Note that, from Algorithm 1, both *D-C step* and *A-B step* are very straightforward and all bit-slices can be converted in parallel. However, the bit-slice transformation in *C-A step* may not be processed in parallel. More specifically, the transformation in *C-A step* is needed only if there exist 0-digits in C. Therefore, if $(c_{i+1},c_i)=(0,1)$, by rules 3(b) and 3(c), we have $a_i=\bar{1}$, but a_{i+1} cannot be determined immediately. In fact, a_{i+1} is determined by c_{i+2}, i.e., if $c_{i+2}=0$, then $a_{i+1}=\bar{1}$, otherwise, $a_{i+1}=1$. Similarly, if c_{i+2} is a 0, we can assign the value of a_{i+1}, but a_{i+2} is still determined by c_{i+3}. The same determination process is repeated until a non-zero c_{i+k} occurs. This is the reason why the transformation in *C-A step* cannot be processed in parallel.

In this study, a conditional flag f_i, $f_i \in \{\bar{1},1\}$, is used to sense the existence of the consecutive 0's and the flag holds the predicted value for the next converted binary bit. A new flag $f_{i+1}=1$ ($\bar{1}$) will originate from the i-th digit if $c_i=\bar{1}$ (1); and it will pass to the left if the passing digit c_j has a 0. In other words, if its left digit is zero, the flag value is assigned to the next converted binary number. On the other hand, if its left digit is non-zero, a new flag has been generated. Note that the initial flag f_0 is assigned by a 1. With the use of the flags, both *D-C step* and *C-A step* can be combined and re-stated as follows and tabulated in Table 1.

```
$f_0=1;$
FOR $i=0$ to $(n-1)$
    IF $d_i=0$, THEN $f_{i+1}=f_i$; $a_i=-f_i$; ELSE $f_{i+1}=d_i$; $a_i=f_i$.
$a_n=f_n;$
```

The following example illustrates the stepwise procedure in (5) and Table 1.

		d_6	d_5	d_4	d_3	d_2	d_1	d_0
D=(		$\bar{1}$	0	$\bar{1}$	0	0	1	0)
f_i:	$\bar{1}$	$\bar{1}$	$\bar{1}$	1	1	1	1	1
a_i:	$\bar{1}$	$\bar{1}$	1	1	$\bar{1}$	$\bar{1}$	1	$\bar{1}$

Further, Table 1 can be simplified as shown in Table 2 by including the simple *B-A step*. By (4), $a_i=\bar{1}$ is converted to as $b_i=0$; The flag f_i, 1 or $\bar{1}$, can be represented by a new flag bit $g_i=(1-f_i)/2$, i.e., $g_i=0$ if $f_i=1$ and $g_i=1$ if $f_i=\bar{1}$. Therefore, the initial flag $g_0=0$ because $f_0=1$. The redundant binary digit $d_i=0$, 1, and $\bar{1}$ can be encoded as $(d_i^s,d_i^a)=(0,0)$, $(0,1)$, and $(1,1)$, where d_i^s is the sign bit and d_i^a is the absolute value. Note that, in (5), "$f_{i+1}=f_i$; $a_i=-f_i$" is equivalent to "$g_{i+1}=g_i$; $b_i=g_i$"; "$f_{i+1}=d_i$; $a_i=f_i$" is equivalent to "$g_{i+1}=(1-d_i)/2=d_i^s$; $b_i=1-g_i=\bar{g}_i$", where $\bar{g}_i$ is the bit-complement of g_i; and "$d_i=0$" can be replaced by "$d_i^a=0$". Therefore, (5) can be re-stated in terms of g_i and b_i as follows.

```
$g_0=1;$
FOR $i=0$ to $(n-1)$
    IF $d_i^a=0$, THEN $g_{i+1}=g_i$; $b_i=g_i$; ELSE $g_{i+1}=d_i^s$;
$b_i=\bar{g}_i$.
```

According to Table 2, the following example illustrates the conversion process.

		d_6	d_5	d_4	d_3	d_2	d_1	d_0
D=(		$\bar{1}$	0	$\bar{1}$	0	0	1	0)
g_i:	1	1	1	0	0	0	0	0
b_i:	1	0	1	1	0	0	1	0

B. Hardware Implementation

Based on the conversion rules in Table 2, the Boolean equations for b_i and q_{i+1} are

$$b_i = g_i \oplus d_i^a \qquad (7)$$
$$g_{i+1} = d_i^s + \bar{d}_i^a g_i$$

The Boolean equations are implemented as follows

$$b_i = \overline{\bar{g}_i \oplus d_i^a} \text{ and } \overline{g_{i+1}} = \overline{\overline{d_i^s + \overline{d_i^a + \bar{g}_i}}}$$

Figure 1(a) shows the block diagram of a converter cell (C-cell) which takes an input digit d_i encoded as (d_i^s,d_i^a) and a flag bit $\bar{g}_i$, and produces a binary bit b_i and a new flag bit $\bar{g}_{i+1}$. The Boolean equations in (8) are implemented by the circuit in Figure 1(b). Figure 1(c) illustrates an n-stage converter circuits, where n C-cells are cascaded. The flag g_i generated from the i-th C-cell is fed to the (i+1)-th C-cell. Note that the initial flag $g_0=0$, or $\bar{g}_0=1$, is fed to the right-most C-cell. Since $b_n=g_n$, an inverter is needed to complement the flag bit $\bar{g}_n$ generated from the left-most C-cell. Since the C-cell is implemented by CMOS technology and a XNOR gate can be realized by four transistors, the propagation delay of the C-cell is thus the gate delay of two NOR gates, i.e., $\Delta_{CC}=2\Delta_{NOR}$, where Δ_{CC} and Δ_{NOR} are the propagation delays of the C-cell and the NOR gate, respectively. Therefore, the propagation delay of the n-digit converter circuit is $n\Delta_{CC}+\Delta_{inv}$, or $(2n+1)\Delta_{NOR}$, where Δ_{inv} is the propagation delay of an inverter which is assumed to be the same as Δ_{NOR}.

It should be mentioned that our conversion algorithm differs from the one proposed in [7], however, both end up with having a similar hardware implementation due to the use of same encoding scheme. Moreover, the contribution of our algorithm is the finding of the fact that the worst-case delay of the converter circuit is not determined by the number of stages, but by the maximum number of consecutive 0's in a given redundant binary number D. If the numbers of consecutive 0's in a given redundant binary number do not exceed 3, then the conversion result is valid at the end of

$3\Delta_{CC}+\Delta_{NOR}$, or 7 Δ_{NOR}. Similarly, if the numbers do not exceed 5, then the conversion time is 11 Δ_{NOR}. Both cases are much better than the worst-case conversion time $2n\Delta_{NOR}$. The question that arises is what is the maximum number of consecutive 0's, in average, in n-digit redundant binary numbers. If the self-timed circuit delay is much shorter than the worst-case delay in an average sense, then it is worthwhile to develop the self-time converter circuit. The next section discusses the statistical results and presents the design and hardware implementation of a self-timed converter circuit.

3. Self-timed Converter Design and Hardware Implementation

From the transformation rules in (3), new flags are generated simultaneously at the stages with non-zero digits, i.e., $(d_i^s,d_i^a)=(0,1)$ or $(1,1)$, or $d_i^a=1$. Those flags are simultaneously passed to their left stages with zero digits, i.e., $d_i^a=0$. Each flag is kept passing until its left digits is a non-zero digit. Thus, the conversion is actually complete at the end of completing the longest flag passing. In other words, the conversion time is not the worst-case circuit delay, but the time required to complete the longest flag passing. In this section, we first discuss the simulation results and statistical upper bound of the average longest flag passing in n-digit redundant binary numbers, and then present the self-time converter circuit design and hardware implementation.

A. Statistical Results

In order to investigate the average of the maximum numbers of consecutive 0's in an n-digit redundant binary number, a simulation has been conducted by randomly choosing n-digit numbers and converting each chosen number. Various sample sizes, 10, 100, 1,000, 10,000, and 100,000 numbers have been simulated for 8-, 16-, 32-, and 64-digit numbers. Table 3 summarizes the simulation results and shows that the maximum numbers of consecutive 0's, in average of 100,000 samples, are approximately 1.5, 2.2, 2.8, and 3.5 for 8-, 16-, 32-, and 64-digit numbers, respectively.

Let L_n be the average longest length of flag passing for n-digits redundant binary numbers. Similar to the statistical analysis for self-timing addition by Burks, Goldstine, and Von Neumann in 1946, the upper bound of $L_{n,max}$ is

$$L_n \le L_{n,max}=\log_3 n \qquad (8)$$

where $L_{8,max}=\log_3 8=1.89$, $L_{16,max}=\log_3 16=2.52$, $L_{32,max}=\log_3 32=3.15$, and $L_{64,max}=\log_3 64= 3.78$. The speed-up ratio of the worst-case longest length to that the average value

$$\Theta_n = n/\log_3 n \qquad (9)$$

becomes appreciate for large n, where $\Theta_8=4.23$, $\Theta_{16}=6.35$, $\Theta_{32}=10.16$, and $\Theta_{32}=16.93$. This implies that one can build a converter circuit which can convert $n/\log_3 n$ times faster than the serial-mode converter circuit in Figure 1. If we denote Δ_{stage} as the propagation delay of a stage in an n-digit self-timed converter circuit, then the propagation delay, $\Delta_{STC,n}$, of the n-digit self-timed converter circuit is

$$\Delta_{STC,n} = \lceil L_{n,max} \rceil \Delta_{stage} \qquad (10)$$

This implies that $\Delta_{STC,8}=2\Delta_{stage}$; $\Delta_{STC,16}=3\Delta_{stage}$; $\Delta_{STC,32} = 4\Delta_{stage}$; and $\Delta_{STC,64} = 4\Delta_{stage}$. The next section is to present the self-timed converter circuit design and estimate the propagation delay of $\Delta_{STC,n}$.

B. Self-timed Converter Circuit Design and Hardware Implementation

Figure 2 shows an n-digit self-timed converter circuit. Each stage is comprised of a FG (*flag generator*) cell, an NOR gate, and a XNOR gate. A FG cell is used to generate a new flag and to sense if the stage has been processed. This implies that FG cell is operated in one of the following states: (1) the stage has not yet been processed; (2) the stage has been processed with an output flag bit 1; and (3) the stage has been processed with an output flag bit 0. Therefore, it is necessary to use two flag bits to represent these three states and to realize the cell in a two-rail way [8]. The FG cell, as shown in Figure 2(b), receives two data input bits, d_i^s and d_i^a, two input flag bits, g_i and g_i^*, and one control signal H, and produces two new flag bits, g_{i+1} and g_{i+1}^*. Note that $(g_{i+1},g_{i+1}^*)=(0,0)$ implies that the stage has not yet been processed, while $(g_{i+1},g_{i+1}^*)=(0,1)$ or $(1,0)$ indicates the stage has been processed with an output flag 0 or 1, respectively. The relationship among the input data, the input flags, and the new flags can be illustrated as follows,

$$(g_{i+1},g_{i+1}^*) = \begin{cases} (g_i,g_i^*) & \text{if } (d_i^s,d_i^a)=(0,0); \\ (0,1) & \text{if } (d_i^s,d_i^a)=(0,1); \\ (1,0) & \text{if } (d_i^s,d_i^a)=(1,1); \end{cases} \qquad (11)$$

The signal H is employed to initialize the flag bits, i.e., $g_i=g_i^*=0$, for all i, if H=0. The flag-passing is begun by releasing the inhibition on all stages, i.e., H=1, including the selected initial flags $g_0=0$ and $g_0^*=1$. The output of the NOR gate taking two inputs from g_{i+1} and g_{i+1}^* indicates if the stage has been processed. Finally, the completion signal, realized by an n-input NOR gate, is asserted as soon as all stages are processed.

In summary, the converter circuit starts with setting H=0 for all stages such that $g_i=g_i^*=0$, for all i and then resetting H=1 and initializing $g_0=0$ and $g_0^*=1$ to convert the input data. Thus, the Boolean functions for the new flag bits are

$$\begin{aligned} g_{i+1} &= d_i^s H + \overline{d}_i^a g_i \qquad (12) \\ g_{i+1}^* &= \overline{d}_i^s d_i^a H + \overline{d}_i^a g_i^* \end{aligned}$$

Figure 2(c) shows the block diagram of the FG cell which is realized in a two-rail way. Each cell takes two inverters, and six NAND gates. The propagation delay of the FG cell is $\Delta_{inv}+\Delta_{NAND3}+\Delta_{NAND2}$, where Δ_{inv}, Δ_{NAND3}, and Δ_{NAND2}, are the delays for an inverter circuit, 3-input NAND gate, and 2-input NAND gate. For simplicity of propagation delay estimation, these three primitive gates are assumed to have the same gate delay, Δ_g. Thus, the delay of a FG cell is $3\Delta_g$, but the delay time from the input flag bits to the new flag bits is $2\Delta_g$. This implies that, by (10), the average propagation delay of the n-digit converter circuit is

$$\Delta_{STC,n} = 2\lceil L_{n,max} \rceil \Delta_g + \Delta_g \qquad (13)$$

where $\Delta_{STC,8}=5\Delta_g$, $\Delta_{STC,16}=7\Delta_g$;$\Delta_{STC,32}=9\Delta_g$;$\Delta_{STC,64}=9\Delta_g$; $\Delta_{STC,256}=13\Delta_g$;and $\Delta_{STC,1024}= 15\Delta_g$;

Note the Boolean expression of the converted binary bit b_i, as shown in (7) is implemented as follows

$$b_i = \overline{g_i \oplus \bar{d}_i^a} \qquad (14)$$

and $b_n=g_n$. A XNOR gate is used to realized the function. Since $g_0=0$ is assumed in Figure 3(a), we have $b_0=d_0^a$, i.e., b_0 is connected directly to d_0^a without a XNOR gate. The NOR gate for the output flag of the stage (n-1) is omitted because the propagation delay of the stage is $2\Delta_g$ which is smaller than that of the NOR gate for the stage (n-2) and the n-input NOR gate for completion signal. Therefore, the propagation delay of the n-digit self-timed converter circuit is $\Delta_{STC,n}$ plus the propagation delay of the n-input NOR gate.

The actual performance evaluation in terms of chip area and propagation delay time for various approach depend heavily on implementation technology. In [7], a lookahead-mode converter circuit was proposed. Let $\Delta_{CLAC,n}$ denote the propagation delay of their n-digit lookahead-mode converter. Thus, $\Delta_{CLAC,8}=10\Delta_g$; $\Delta_{CLAC,16}=14\Delta_g$; $\Delta_{CLAC,256}=18\Delta_g$ and $\Delta_{CLAC,8}=22\Delta_g$, where it was assumed that $\Delta_{NANDi}=\Delta_g$, i=2,3,4,5,6. It is not attempted to compare the lookahead scheme in [7] with the proposed self-timed converter, but, in average sense, the proposed self-timed converter provides faster conversion. The proposed approach takes slightly less hardware cost than the lookahead scheme.

4. Conclusion

This paper presents a self-timed converter circuit which efficiently converts an n-digit redundant binary number to an (n+1)-bit binary number. The conversion time does not increase with the number of digits to be converted, and it is determined by the maximum number of consecutive 0's in that number. Results show that the statistical upper bound of the maximum number of consecutive 0's is 3.78 for 64-digit redundant binary numbers. Thus, the self-timed converter circuit can achieve speed improvement by 17 times faster than the serial-mode converter circuit in average. With moderate hardware cost, the self-timed converter is well-suited to the high speed, long-word digital arithmetic processors.

References

[1] K. Huang, *Computer Arithmetic: Principles, Architecture and Design*. Wiley, New York, 1979.

[2] I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[3] A Avizienis, "Signed-digit number representation for fast parallel arithmetic," IRE Trans. on Electronics Computer, Vol. EC-10, pp.389-400, September 1961.

[4] A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P.G.A. Jespers, "A new carry-free division algorithm and its application to a single-chip 1024-b RSA processor," IEEE Journal of Solid-state Circuits, Vol. 25, No. 3, pp. 748-755, June 1990.

[5] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," Proc. of 8th Symposium of Computer Arithmetic, pp.80-86, 1987.

[6] M.D. Ercegovac and T. Lang, "Fast multiplication without carry-propagate addition," IEEE Trans. on Computers, Vol. 39, no.11, pp.1385-1390, November 1990.

[7] S.-M. Yen, C.-S. Laih, C.-H. Chen, and J.-Y. Lee, "An efficient redundant-binary number to binary number converter," IEEE Journal of Solid-state Circuits, Vol. 27, No. 1, January 1992.

[8] S.H. Unger, *Asynchronous Sequential Switching Circuits*, Wiley, New York, 1969.

Table 1. Transformation Rules.

f_{i+1},a_i	$f_i=1$	$f_i=\bar{1}$
$d_i=$ 0	1, $\bar{1}$	$\bar{1}$, 1
1	1, 1	1, $\bar{1}$
$\bar{1}$	$\bar{1}$, 1	$\bar{1}$, $\bar{1}$

Table 2. Conversion Rules.

d_i	d_i^s,d_i^a	g_{i+1}		b_i	
		$g_i=0$	$g_i=1$	$g_i=0$	$g_i=1$
0	0, 0	0	1	0	1
1	0, 1	0	0	1	0
$\bar{1}$	1, 1	1	1	1	0

Table 3. Average Longest Length of Flag Passing

digits	Sample Size				
	10	100	1000	10000	100000
8	1.6	1.38	1.491	1.502	1.495
16	1.8	1.89	2.098	2.139	2.146
32	2.4	2.57	2.741	2.769	2.794
64	2.7	3.17	3.388	3.412	3.434

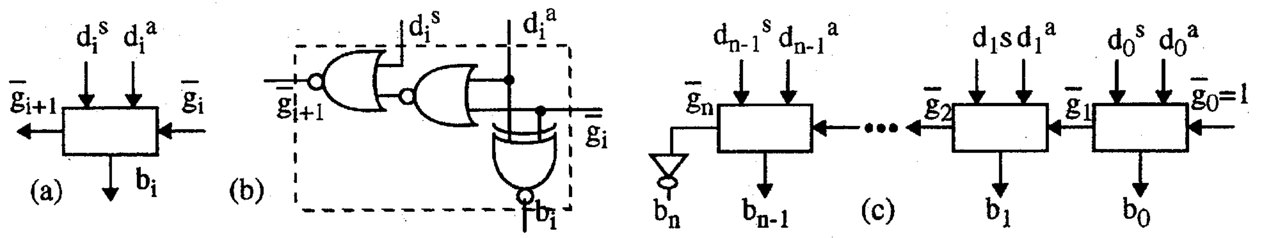

Figure 1. Converter Circuit: (a) & (b) Converter cell (C-cell); and (c) An n-digit converter circuit.

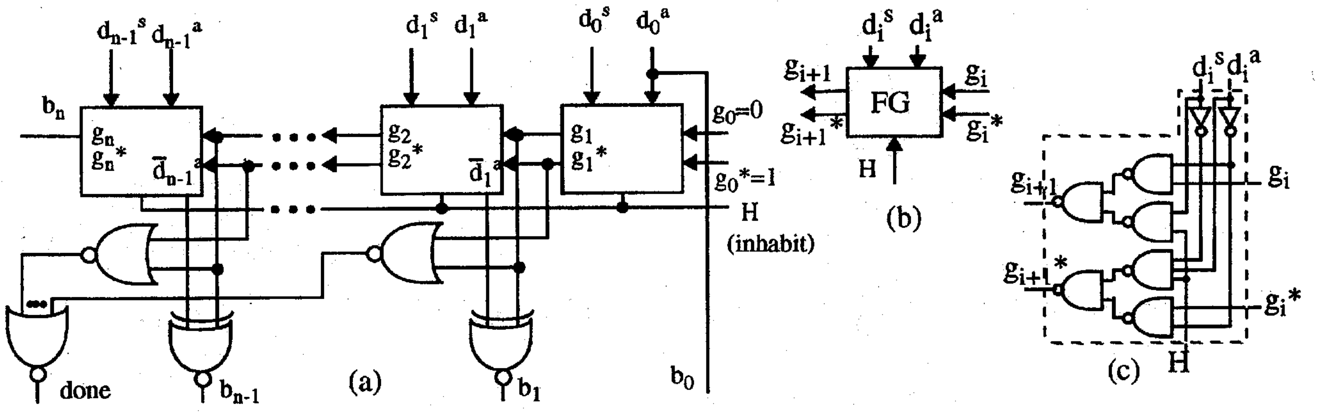

Figure 2. Self-timed Converter: (a) An n-digit Converter ; (b) Block diagram & (c) Schematic diagrm of the FG cell.

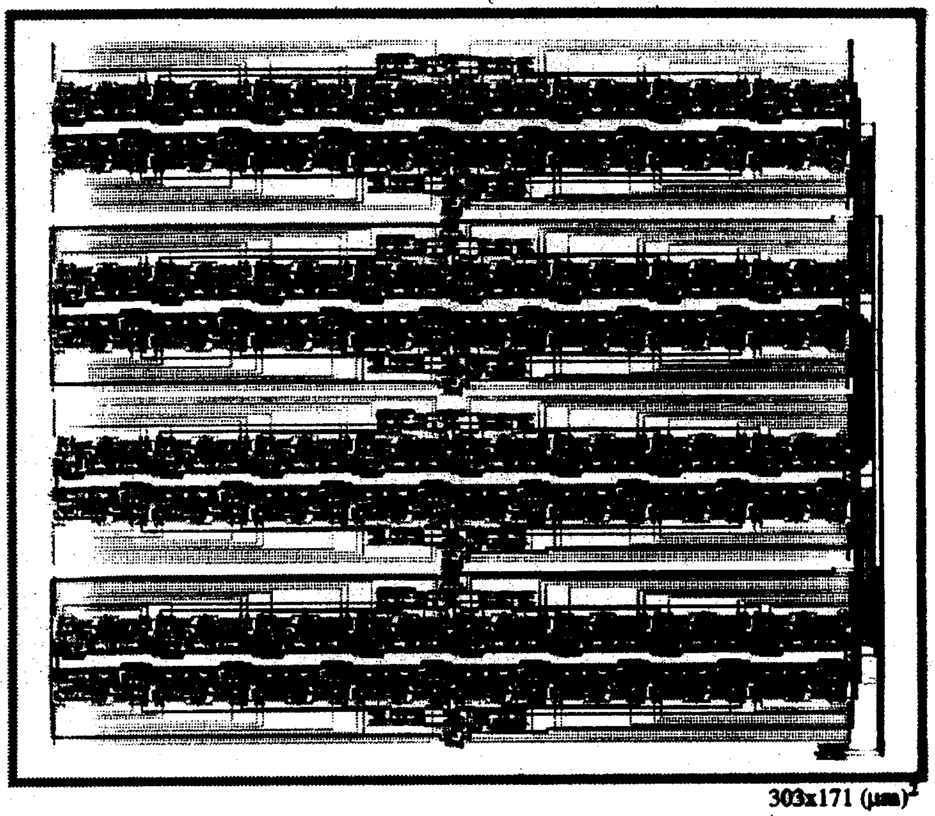

Figure 3. Physical Layout of a 64-digit Self-timed Converter Circuit.