Systolic Number Radix Converters

B. PARHAMI
Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA.

Hardware number radix converters may be required at the input and output interfaces of high-speed arithmetic processors implemented in VLSI. Systolic hardware realisation of the radix conversion process along with hardware implementation details are discussed in this paper. The main results are that it is always possible to accept one input digit in a continuous stream of* k*-digit operands on every clock tick and that the total conversion delay (from the acceptance of the first digit of an operand to the emergence of the first digit in the corresponding* k′*-digit result) is* k+k′/b *clock ticks, where* b *is a design tradeoff parameter affecting the cell complexity, intercell communication, and possibly the clock period in the systolic converter.

Received March 1988, revised June 1988

1. INTRODUCTION

Radix conversion algorithms are well known (see, e.g. section 4.4 in ref. 7) and can be easily implemented in software, hardware, or firmware. When the digits of the input operand are generated serially, with the most significant digit appearing first, the preferred radix conversion method is based on repeated multiplications and additions using arithmetic operations in the new radix.

To convert a given unsigned k-digit radix-r number $x_{k-1}x_{k-2}\ldots x_1x_0$ representing the value X into a radix-r' number X', the following recursive formula (known as Horner's rule) is used:

$$X' = ((\ldots(0+x_{k-1})r+x_{k-2})r+\ldots)r+x_1)r+x_0 \quad (1)$$

This implies initialising X' to 0 and performing the following recursion step for $i = k-1, k-2, \ldots, 1, 0$:

$$X' = rX'+x_i \quad (2)$$

If X' is to be represented with k' digits, an overflow may result for some input operands. Overflow detection is also easy to implement since overflow occurs iff $X \geqslant r'^{k'}$. In fact, overflow detection can be incorporated into the computations implied by (2) if the computation is done modulo $r'^{k'}$ and an overflow flag is set whenever the correct X' value exceeds $r'^{k'}$.

In the remainder of this paper, we will present a systolic version of this algorithm which is suitable for direct VLSI implementation.[8,9] Motivation for this work comes from widespread application of special-purpose VLSI processors in areas such as signal processing.[6,10] Clearly the high speed of such processors will be of little value if not matched by the I/O conversion rate. So, with high input or output data volumes, we may need on-the-fly I/O converters that can keep up with the required processing throughput. Even when the I/O data volume is such that conventional hardware can provide the desired conversion rate, the inclusion of a special hardware 'feature' may prove to be cost-effective.

At present, the use of such special-purpose hardware aids would be limited to decimal-binary and binary-decimal conversions, as non-decimal radices that are not powers of 2 have no significant application. However, the design principles presented are quite general and can be applied to other radices if such radices prove to be useful in future (e.g. through advances in the practical application of multiple-valued logic[5,15]). At any rate, the design presented in this paper can be viewed as an addition to the repertory of tools and techniques for arithmetic system designers. Like any other tool or technique, practical use of this design is governed by performance requirements and application-specific cost/benefit analyses.

2. A SYSTOLIC RADIX CONVERSION ALGORITHM

Direct hardware realisation of the above algorithm results in excessively slow and/or costly hardware. This is because each execution of (2) requires one multiplication of a k'-digit radix-r' number by r and one addition of a radix-r digit to the result. Both of these operations involve carry propagation. Thus, a low-cost ripple-carry implementation requires $O(k')$ time units for each step and $O(kk')$ time units for the whole conversion. Assuming that k' has been selected to accommodate all possible inputs with no overflow, the overall time is $O(k^2)$. Even with costly carry lookahead hardware, the radix conversion time is still not linear in the number of digits k. We are thus led to the following systolic version of the above conversion procedure.

Algorithm 1 (systolic number radix conversion). To convert a given k-digit radix-r number $x_{k-1}x_{k-2}\ldots x_1x_0$ into a k'-digit radix-r' number $x'_{k'-1}x'_{k'-2}\ldots x'_1x'_0$, initialise all t_j ($j = 1, 2, \ldots, k'$), all x'_j ($j = 0, 1, \ldots, k'-1$), and T to zero and compute sequentially for $i = k-1, k-2, \ldots, 1, 0$:

$t_0 := x_i$; (**radix conversion phase**)

for $j = 0, 1, \ldots, k'-2, k'-1$ **do in parallel**
  $u_0 := rx'_0+t_0$; $u_j := r(x'_j+t_j), j > 0$;
  $t_{j+1} := \lfloor u_j / r' \rfloor$;
  $x'_j := u_j - r't_{j+1}$
endfor;
$T := T+t_{k'}$.

Then repeat the following steps sequentially until $t_1 = t_2 = \ldots = t_{k'-1} = 0$:

$t_0 := 0$; (**transfer propagation phase**)

for $j = 0, 1, \ldots, k'-2, k'-1$ **do in parallel**
$u_j := x'_j + t_j$;
$t_{j+1} := \lfloor u_j / r' \rfloor$;
$x'_j := u_j - r' t_{j+1}$
endfor;
$T := T + t_{k'}$.

If at the end $T \neq 0$, apply an overflow handling procedure. Otherwise, $x'_{k'-1} x'_{k'-2} \ldots x'_1 x'_0$ is the correct result. ⊗

That Algorithm 1 is correct should be obvious if one observes that the radix conversion phase implements the multiplication of the partial result by r, followed by the addition of the next operand digit, in parallel using a transfer-save scheme which can be viewed as a generalisation of the carry-save principle. At any given time, the partial result is composed of an interim 'digit' vector $x'_{k'-1} x'_{k'-2} \ldots x'_1 x'_0$ and a 'transfer' vector $t_{k'} t_{k'-1} \ldots t_2 t_1$, together representing the value

$$X = \sum_{i=0..k'-1} x'_i r'^i + \sum_{i=1..k'} t_i r'^i = t_{k'} r'^{k'} + \sum_{j=0..k'-1} (x'_j + t_j) r'^j \tag{3}$$

where t_0 is assumed to be zero. The last summation in (3) can be interpreted as corresponding to a positional radix-r' redundant number system[14] with the expanded digit set $\{0, 1, \ldots, r+\tau-1\}$, τ being the largest possible value for the transfer digit t_j. To multiply this partial result by r, we compute the 'position products' $r(x'_j + t_j)$, and decompose them into new transfers t_{j+1}^{new} and new interim result digits $x_j'^{\text{new}}$ satisfying:

$$r(x'_j + t_j) = r' t_{j+1}^{\text{new}} + x_j'^{\text{new}} \tag{4}$$

The transfer propagation phase is required for getting rid of nonzero transfers in order to obtain the correct $(k'+1)$-component radix-r' result $T x'_{k'-1} x'_{k'-2} \ldots x'_1 x'_0$, where T represents the accumulation of outgoing transfers. If $T = 0$, then the correct k'-digit converted result is at hand. The case $T \neq 0$ represents an overflow.

Example 1. The 4-digit vector 5 3 8 4 represents the decimal number 5384; i.e. $r = 10$, $k = 4$. Algorithm 1 finds a 6-digit radix-8 representation of this number (i.e. $r' = 8$, $k' = 6$) as follows:

j:	6	5	4	3	2	1	0	
t_j:	0	0	0	0	0	0	5	$= x_3$ ← Start conversion
x'_j:		0	0	0	0	0	0	
u_j:		0	0	0	0	0	5	
t_j:	0	0	0	0	0	0	3	$= x_2$
x'_j:		0	0	0	0	0	5	
u_j:		0	0	0	0	0	53	
t_j:	0	0	0	0	0	6	8	$= x_1$
x'_j:		0	0	0	0	0	5	
u_j:		0	0	0	0	60	58	
t_j:	0	0	0	0	7	7	4	$= x_0$
x'_j:		0	0	0	0	4	2	
u_j:		0	0	0	70	110	24	
t_j:	0	0	0	8	13	3	0	
x'_j:		0	0	0	6	6	0	← Start propagation
u_j:		0	0	8	19	9	0	
t_j:	0	0	1	2	1	0	0	
x'_j:		0	0	0	3	1	0	
u_j:		0	1	2	4	1	0	
t_j:	0	0	0	0	0	0	0	← Done (all zeros)
x'_j:		0	1	2	4	1	0	← Result

The final value of T, which is obtained by simply adding the values in the t_6 column, is zero. Thus, no overflow exists and the 6-digit result 0 1 2 4 1 0 is correct. ⊗

Note that Algorithm 1 directly defines a systolic hardware realisation of the conversion process. Such a systolic number radix converter is shown in Fig. 1. The first k clock ticks are devoted to the radix conversion phase, where on each clock tick the cell C_0 receives x_i $(i = k-1, k-2, \ldots, 1, 0)$ as the incoming transfer, the cell C_j $(j = 0, 1, \ldots, k'-1)$ computes u_j, t_{j+1}, x'_j, and finally the cell $C_{k'}$ accumulates $t_{k'}$ by adding it to the current value of T. Then, the transfer propagation phase starts which requires no more than k' clock ticks. In fact, if one uses exactly k' ticks, the test $t_1 = t_2 = \ldots = t_{k'-1} = 0$ can be avoided and this is indeed necessary for the design to be systolic. The total conversion time is thus $k + k'$ clock ticks.

There is a major flaw in the above suggested implementation which may not be apparent at this point. Ignoring this flaw for the moment, it is easy to see that the systolic number radix converter of Fig. 1 receives the digits $x_{k-1}, x_{k-2}, \ldots, x_1$, and x_0 of the input operand serially (most significant digit first) and produces the digits of the converted result in parallel after $k + k'$ clock ticks. To allow pipelining of multiple conversions, we can provide a second row of cells C'_j to which the values of x'_j and t_j are transferred upon the completion of the radix conversion phase. Then, provided that $k' \leqslant k$ (implying that $r' \geqslant r$), the transfer propagation phase can be completed by the second row of cells while the first row processes the next input operand. Finally, if serial rather than parallel output is desired, a third row of cells C''_j can be employed to hold the converted number while the digits are shifted out serially (Fig. 2). With this arrangement, one k-digit input operand can be converted every k clock ticks if $k' \leqslant k$ or every $\max(k, k')$ clock ticks in general. Obviously, C_j, C'_j, and C''_j can be regarded as a single cell with a larger number of registers and connections rather than as three separate cells.

Conversion of the design shown in Fig. 2 to a design that accepts the input operand and generates the radix conversion result from right to left (least significant digit first) is straightforward. To do this, the bottom row of cells is omitted and instead two top rows are added for input buffering and shifting. As one operand is shifted into the top-row buffer cells, the previous operand in the second row starts its serial movement (most significant digit first) into the third row for conversion. The total conversion delay is still $k + k'$ because the generation of result digits can start immediately after the conversion process without a need to wait for transfer propagation. We have chosen to present our results in terms of a left-to-right processing order because it matches the structure of the algorithm and is thus easier to understand.

The overflow indicator T has been defined in Algorithm 1 as the sum of $t_{k'}$ values. In practice, a binary overflow indication is sufficient. Thus, T can be initialised to **false** and $T := T + k_{k'}$ replaced by: **if** $t_{k'} \neq 0$ **then** $T :=$ **true**. The corresponding hardware implementation consists of a flip-flop in the leftmost cells $C_{k'}$ which is set by any nonzero $t_{k'}$ value. With this convention, the result digits can be taken from $C''_{k'}$ in the design of Fig. 2 (rather than from $C''_{k'-1}$) for a fully serial output where the first output digit is a binary overflow indicator.

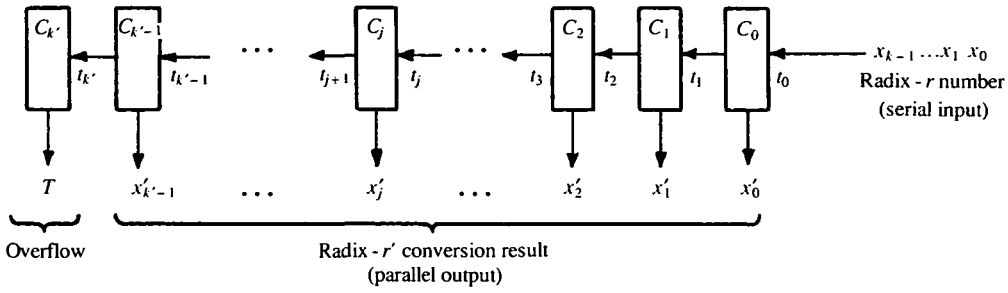

Figure 1. Hardware design of a serial-in parallel-out systolic number radix converter.

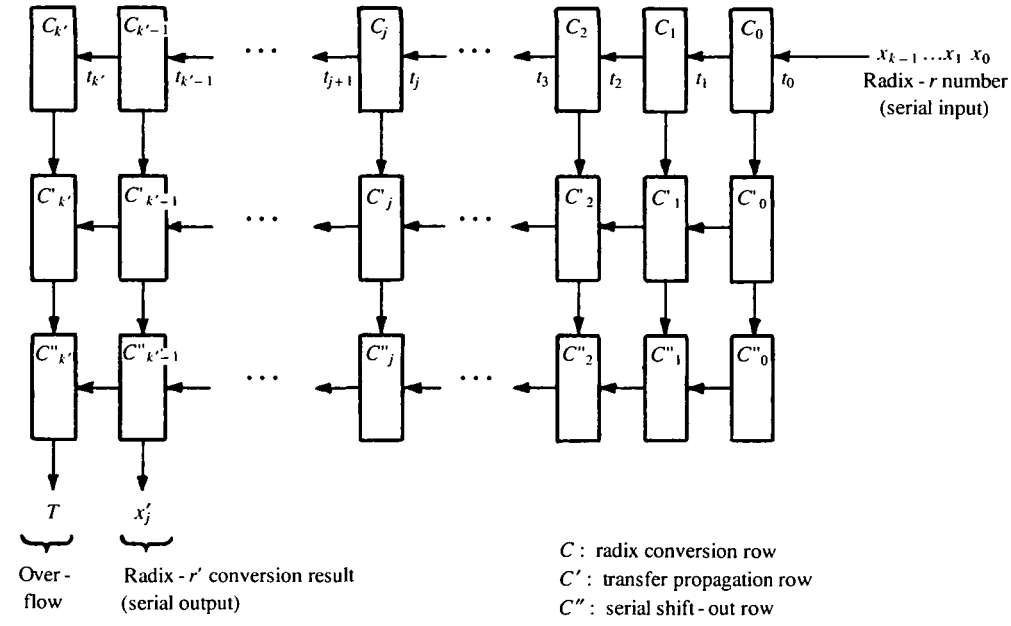

Figure 2. Hardware design of a pipelined serial-in serial-out systolic number radix converter.

3. PRACTICAL HARDWARE REALISATION

Even though Algorithm 1 'works' in all cases, its hardware realisation is impractical for $r \geqslant r'$. To see why, let us investigate the range of values for t_{j+1} in Algorithm 1. We have $t_0 = x_t$ and for $j \geqslant 0$:

$$t_{j+1} = \lfloor u_j / r' \rfloor \tag{5}$$

Noting that in the radix conversion phase, where larger transfer values are generated, $u_0 = rx'_0 + x_t$ and $u_j = r(x'_j + t_j)$ for $j > 0$, we find:

$$t_{j+1} \leqslant \max\left(\lfloor r(x'_j + t_j)/r' \rfloor, \lfloor (rx'_0 + x_t)/r' \rfloor\right) \tag{6}$$

Substituting the worst-case values of $r-1$ and $r'-1$ for x_t and x'_j in (6), we get:

$$t_{j+1} \leqslant \max\left(\lfloor r(r'-1+t_j)/r' \rfloor, \lfloor (rr'-1)/r' \rfloor\right) \tag{7}$$

It is easy to see that if $r \geqslant r'$, the right hand side of (7) is greater than t_j. So, the magnitude of transfer digits is not bounded and can grow with the number of digits in the operand. This makes the design of the cells for the systolic converter of Fig. 1 specific to a particular value of k and also implies a large number of wires between adjacent cells in any particular design. On the other hand, if $r < r'$, then (7) can be used to calculate the range of transfer digits. Let

$$0 \leqslant t_j \leqslant \tau \tag{8}$$

define the range of transfer digit values in the conversion process. Then, substituting the worst-case value τ for t_j in inequality (7) and taking the equality $t_0 = x_t$ into account, we get for $0 \leqslant j \leqslant k'$:

$$t_j \leqslant \max\left(r-1, \lfloor r(r'-1+\tau)/r' \rfloor, \lfloor (rr'-1)/r' \rfloor\right) \tag{9}$$

Considering that $\lfloor (rr'-1)/r' \rfloor = r-1$, and that $\lfloor r(r'-1+\tau)/r' \rfloor = r-1+\lfloor (r\tau + r' - r)/r' \rfloor \geqslant r-1$, we must have:

$$\tau \geqslant \lfloor r(r'-1+\tau)/r' \rfloor \tag{10}$$

From the point of view of hardware cost, it is advantageous to select the minimal value satisfying (10) for τ. To find this minimal value from (10), we first note that

$$\tau^{\text{sol}} = \lfloor r(r'-1)/(r'-r) \rfloor = r + \lfloor r(r-1)/(r'-r) \rfloor \tag{11}$$

is a solution of (10). This is easily verified by writing

$$r(r'-1) = (r'-r)\,\tau^{\text{sol}} + \delta \quad \{0 \leqslant \delta < r'-r\} \tag{12}$$

and substituting in (10). To find τ^{min}, we replace τ in (10) by $\tau^{sol}-\varepsilon$ and look for the maximum value of ε which satisfies the resulting inequality. Straightforward manipulation and the use of the identity $\lfloor z \rfloor = -\lceil -z \rceil$ yields:

$$\varepsilon \leqslant \lceil (r\varepsilon - \delta)/r' \rceil \qquad (13)$$

The maximal value of $\varepsilon \geqslant 0$ satisfying (13) is generally very small and can be found by inspection. Therefore

$$\tau^{min} = \tau^{sol} - \varepsilon^{max} \qquad (14)$$

where τ^{sol} is given by (11) and ε^{max} is the largest ε value satisfying (13).

Example 2. Consider $r = 8$, $k = 6$, $r' = 10$, and $k' = 4$; note that this is the inverse of the conversion in Example 1. Inequality (10) becomes:

$$\tau \geqslant \lfloor 8(10-1+\tau)/10 \rfloor = 7 + \lfloor (4\tau+1)/5 \rfloor \qquad (15)$$

The solution $\tau^{min} = 32$ can be obtained directly from (15) or by using (11) to find $\tau^{sol} = 36$ and then solving the following version of (13) for the maximum value of ε:

$$\varepsilon \leqslant \lceil (8\varepsilon - 0)/10 \rceil = \lceil 4\varepsilon/5 \rceil$$

The above inequality yields $\varepsilon^{max} = 4$ and

$$\tau^{min} = \tau^{sol} - \varepsilon^{max} = 32.$$

We see that even though transfer digit values are bounded for this example, their large magnitudes can make the hardware implementation slow and/or excessively costly. ⊗

We now present a method for reducing the hardware complexity of the systolic converter through a tradeoff between cell and interconnection complexities. We introduce the method through an example and then state it in general terms. The intuitive idea is that the larger the value of r' in (11), the smaller the values of τ^{sol} and τ^{min}.

Example 3. Consider $r = 8$, $k = 6$, $r' = 100$, and $k' = 2$. Comparing these values of those of Example 2, we see that the new radix r' is the square of the corresponding value in Example 2. If we convert a number into radix $r' = 100$, then the result can be easily converted into radix 10 by simply breaking each radix-100 digit D into two radix-10 digits d and d' such that $D = 10d + d'$. Inequality (10) becomes:

$$\tau \geqslant \lfloor 8(100-1+\tau)/100 \rfloor = 7 + \lfloor (2\tau+23)/25 \rfloor \qquad (16)$$

Inequality (16) yields $\tau^{min} = 8$. We see that the range of transfer digit values for this example is much smaller than that obtained in Example 2. ⊗

Example 3 leads to the following general algorithm for designing systolic number radix conversion circuits.

Algorithm 2 (design of systolic number radix converters). To design a systolic hardware circuit for converting a k-digit radix-r number $x_{k-1}\, x_{k-2} \ldots x_1\, x_0$ into a k'-digit radix-r' number $x'_{k'-1}\, x'_{k'-2} \ldots x'_1\, x'_0$ when either $r \geqslant r'$ or $r < r'$ but (10) yields an unacceptably large value for τ^{min}, consider converting the number into an intermediate k''-digit radix-r'' form $x''_{k''-1}\, x''_{k''-2} \ldots x''_1\, x''_0$, with $k'' = \lceil k'/b \rceil$ and $r'' = r'^b$, for $b = 2, 3$, etc. until the range $[0, \tau]$ for the transfer digit values, obtained from (10), becomes acceptable. Then convert each digit x''_j of the intermediate form into a sequence of b digits $x'_{jb+b-1}\, x'_{jb+b-2} \ldots x'_{jb+1}\, x'_{jb}$ of the final result. ⊗

One must note that the optimal value for b in Algorithm 2 is generally very small. This is because larger values for b may yield a small reduction (if at all) in the range of transfer digit values, while at the same time making the logic in each cell of the systolic converter exceedingly complex. The following examples accentuate this point.

Example 4. Referring to Example 3, we note that if we had selected $r'' = 1000$, we would have found $\tau^{min} = 8$. Thus, there is no reduction in the range of transfer digit values and the optimal design must be selected by comparing the two implementations with $b = 1$ and $b = 2$. ⊗

Example 5. To design a hardware unit for conversion from $r = 10$ to $r' = 8$ (same as the parameters of Example 1), we find the following:

$$b = 2 \Rightarrow r'' = 64, \quad \tau^{min} = 11$$
$$b = 3 \Rightarrow r'' = 512, \quad \tau^{min} = 10$$
$$b = 4 \Rightarrow r'' = 4096, \quad \tau^{min} = 10$$

Again, the diminishing return of higher values for b is clearly evident. The best choice here is probably $b = 2$. ⊗

An intuitive explanation of why transfer digit values are unbounded in the case of $r > r'$ is as follows. In converting an integer into a smaller radix, the number of digits tends to grow. However, in the systolic hardware realisation of Fig. 1, which was obtained directly from Algorithm 1, only j result and transfer digits are computed after processing of j input digits. In other words, no information has reached C_j and the cells to its left. The result digits are always adjusted to be within bounds. Thus, the excess value of the number, which will eventually determine the values of the jth and more significant result digits, must be absorbed by the transfer digits.

When a systolic radix converter is designed according to Algorithm 2 with $b > 1$, two methods of operation can be envisaged. If it is important to accept input operands at the highest possible rate (one conversion per k clock ticks), then b output digits must be generated in parallel during each of $\lceil k'/b \rceil$ successive clock ticks with $k - \lceil k'/b \rceil$ idle cycles between conversions during which no output is generated. On the other hand if fully serial operation is desired (no more than one output digit can be absorbed per clock cycle), then there must be $k' - k$ idle cycles between conversions when no input is accepted. This yields a conversion rate of one per $\max(k, k')$ clock ticks. The above discussion leads to the following results.

Result 1 (conversion throughput). It is possible to design a digit-serial (most significant digit or least significant digit first) systolic number radix converter composed of k' simple cells such that it accepts a continuous stream of k-digit input operands at the rate of one digit per clock tick and produces b digits of a k'-digit conversion result (with $k'/b \leqslant k$) on successive clock ticks after some initial delay. ⊗

Result 2 (conversion delay). It is possible to design a digit-serial (most significant digit or least significant digit first) systolic number radix converter composed of k' simple cells (or k'/b somewhat more complex cells, where b is a design parameter) such that the input-to-output delay for converting a k-digit radix-r number to a k'-digit radix-r' number is $k + k'$ (or $k + k'/b$) clock ticks. ⊗

4. CONCLUSION

We have considered the number radix conversion problem and presented a method for designing systolic hardware radix converters capable of processing arriving operand digits in real time: one digit processed or generated per clock tick, depending on whether we convert to a larger or a smaller radix and on some implementation details as discussed at the end of Section 3. The conversion delay was shown to be linear in both the input and output operand lengths.

Although the linear delay $k+k'$ (or more generally $k+k'/b$) may seem substantial, there are two redeeming factors. Firstly, in many applications, throughput is of primary importance and the input-to-output delay is at best a secondary concern.[3] Secondly, in any digit-serial implementation, the conversion delay is of the form $k+\Delta$. Thus, speed can be gained over the proposed systolic implementation only to the extent that Δ can be made smaller than k'/b. Since every digit of the conversion result is a function of all input digits, the best that one can hope for is a Δ that is logarithmic in k. Even assuming that such fully parallel circuits are realisable for arbitrary k, the larger clock cycles required for their operation will nullify much (if not all) of the gain. In fact, as stated in the beginning of Section 2, straightforward implementation of the conversion process using conventional building blocks such as adders and multipliers will result in a delay that is quadratic in k.

Even though it is clear from our previous discussion that the systolic converter is composed of relatively simple cells, an example quantifying the cell complexity may be helpful. In the simplest form, each cell stores an interim result digit, receives an incoming transfer, and produces a new interim result digit plus an outgoing transfer. Considering the octal-to-decimal conversion of Example 2 and assuming the use of binary encoding for all values, each cell will have $4+6$ inputs (a decimal digit plus an incoming transfer with a value of at most 32) and must compute $4+6$ Boolean functions of the 10 logic variables. The modification of Example 3 results in $8+4$ inputs (2 decimal digits plus an incoming transfer with a value of at most 8) and 12 Boolean functions. Both versions can be easily implemented with random logic, programmable logic devices, or read-only memories. However, the second version might be preferable because it implies half as many cells and a shorter input-to-output delay.

It is interesting to note that a systolic number radix converter could be designed by combining three previously published principles, if an intermediate signed-digit[1] representation of the result were employed. A systolic signed-digit multiplication[2] for multiplying by the constant r, a generalised version of systolic counting[11] for incrementation by x_i, and on-the-fly conversion of the signed-digit result into conventional representation.[4] However, such a design is considerably more complex than the direct implementation suggested here, while having the same time complexity. In fact, our implementation also uses a redundant number representation scheme to reduce multiple propagation chains to a single one at the very end. The transfer-save number representation scheme used here is a special case of generalised signed-digit (GSD) number systems which have been discussed in detail elsewhere.[12,13] It may be worth noting that the design presented here was discovered as a special case of general GSD-to-GSD converters.[13]

Acknowledgement

The research reported in this paper was carried out while the author was a Visiting Professor at the School of Computer Science, Carleton University, Ottawa, Canada, K1S 5B6.

REFERENCES

1. A. Avizienis, Signed-digit number representation for fast parallel arithmetic. *IRE Transactions on Electronic Computers* **EC-10**, 389–400 (1961).
2. P. T. Balsara and R. M. Owens, Systolic and semi-systolic digit serial multipliers. *Proceedings of the Symposium on Computer Arithmetic, Como, Italy*, pp. 169–173. (May 1987).
3. P. Denyer and D. Renshaw, *VLSI Signal Processing: The Bit-Serial Approach*. Addison-Wesley, Reading, MA (1985).
4. M. D. Ercegovac and T. Lang, On-the-fly conversion of redundant into conventional representations. *IEEE Transactions on Computers*, **C-36**, 7, 895–897 (July 1987).
5. D. Etiemble and M. Israel, Comparison of binary and multivalued ICs according to VLSI criteria. *Computer*, **21**, 4, 28–42 (Apr. 1988).
6. L. B. Jackson, *Digital Filters and Signal Processing*. Kluwer, Boston (1986).
7. D. E. Knuth, *The Art of Computer Programming*; vol. 2, *Seminumerical Algorithms*. Addison-Wesley (1981).
8. H. T. Kung, Let's design algorithms for VLSI systems. *Proceedings of the Caltech Conference on VLSI*, pp. 65–90. (Jan. 1979).
9. H. T. Kung, Why systolic architectures? *Computer*, **15**, 1, 37–46 (Jan. 1982).
10. S.-Y. Kung, R. E. Owen and G. J. Nash (editors), *VLSI Signal Processing II*. IEEE Press, New York (1986).
11. B. Parhami, Systolic up/down counters with zero and sign detection. *Proceedings of the Symposium on Computer Arithmetic, Como, Italy*, pp. 174–178. (May 1987).
12. B. Parhami, A general theory of carry-free and limited-carry computer arithmetic. *Proceedings of the Canadian Conference on VLSI, Winnipeg, Canada*, pp. 167–172. (Oct. 1987).
13. B. Parhami, Conversion between generalized signed-digit and conventional representations. *Proceedings of the Second Annual Parallel Processing Symposium, Fullerton, CA*, pp. 95–106. (Apr. 1988).
14. B. Parhami, Generalized signed-digit number systems: a unifying framework for redundant number representations. *IEEE Transactions on Computers*, **39**, 1, 89–98 (Jan. 1990).
15. K. C. Smith, Multiple-valued logic: a tutorial and appreciation. *Computer* **21**, 4, 17–27 (Apr. 1988).