Parallel Computing 10 (1989) 231–238
North-Holland

Pin reduction through variable duplications and substitutions in a data dependence graph

Ferng-Ching LIN and Robert CHARNG

Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, R.O.C.

Received May 1987
Revised August 1987

Abstract. Due to the packaging problem, the implementation of computing arrays on chips is constrained to certain pin limitation. This paper introduces two basic techniques for pin reduction without lowering computational efficiency in systolic designs. We exhibit the ideas through the synthesis of a systolic array with very few I/O pins for polynomial division.

Keywords. Systolic array, polynomial division, data dependence graph, space-time mapping, pin reduction.

1. Introduction

The dramatic development of the very large scale integration (VLSI) technology has made it possible to implement algorithms directly in hardware and hence promoted a great deal of interests in designing algorithmically specialized processing components. Following Kung's systolic concept [7–9], many computing arrays have been proposed to handle various compute-bound problems. These arrays are formed by regularly connected processing elements (PEs) in which data are communicated locally and operated on rhythmically. The simplicity, regularity and locality of the systolic arrays render them suitable for VLSI implementation. High performances are achieved by the concurrent use of large amount of PEs in the arrays.

Recently, there have been considerable efforts focused on systematic methods for synthesizing systolic arrays [2]. For more recent developments see [3,5,10,11,16] also. Due to the packaging problem, the implementation of computing arrays on chips is constrained to certain pin limitation. However, none of those methods deals with the pin problem. In other words, the systolic arrays designed with those methods often require a large number of external I/O connections. One way of reducing the pins is to partition the problem on a small-sized array [14,15]. But, the restricted number of PEs in the array will unavoidably downgrade the computation concurrency of the array.

In this paper, we shall adapt Moldovan's synthesis method [13,14] and provide two basic techniques for pin reduction without lowering computational efficiency in designing systolic arrays. The idea is to identify the data dependencies of a given problem to form a data dependence graph, transform it to one that represents a systolic array. An exceeding number of I/O pins will occur when there are too many data flows (in the dependence graph) which are 'orthogonal' to the target PE space. A technique which we call variable duplication will be

applied to redirect the pin-producing data flows and thereby substantially reduce the pin number. Another technique called variable substitution can be employed to replace inactive data flows by others and hence reduce the pins further. We'll exhibit these ideas through the synthesis of a systolic array with very few I/O pins for polynomial division.

Polynomial division is fundamental to algebraic computations [1], decoder implementation [12], signal/image processing [4] and synthesis and analysis of control systems [6]. Żak and Hwang have proposed a linear systolic array for this problem [17]. In their design, repeated coefficients of the divisor polynomial are required to enter each PE and coefficients of the resulting quotient polynomial are emitted from different PEs. The number of I/O pins depends on the degree of the quotient polynomial. Based on our techniques, we will ultimately reduce the pin number to just 4 when counted wordwise.

2. A systolic array with too many pins

It is well known that if $F(x) = a_0x^m + \cdots + a_{m-1}x + a_m$ and $G(x) = b_0x^n + \cdots + b_{n-1}x + b_n$ $(G(x) \neq 0)$ are two polynomials with real coefficients and of degree m and n $(0 < n \leqslant m)$ respectively, there exist unique polynomials $Q(x)$ and $R(x)$, the quotient and remainder, such that

$$F(x) = G(x)Q(x) + R(x), \quad 0 \leqslant \deg R(x) < \deg G(x).$$

Here is the ordinary sequential algorithm to compute $Q(x)$ and $R(x)$:

1. $Q(x) \leftarrow 0$.
2. If deg $F(x) <$ deg $G(x)$ then $R(x) \leftarrow F(x)$; stop.
3. $q \leftarrow$ leading coefficient of $F(x)$ / leading coefficient of $G(x)$.
4. $k \leftarrow$ deg $F(x) -$ deg G(x).
5. $F(x) \leftarrow F(x) - qx^kG(x)$.
6. $Q(x) \leftarrow Q(x) + qx^k$.
7. Go to 2.

Following the algorithm, the commonly used synthetic division can be laid out as follows:

b_0	b_1	$\cdots$	b_n			
	q_1	q_2	$\cdots$	q_{m-n+1}		
	a_0	a_1	$\cdots$	a_n	$\cdots$	a_m
		q_1b_1	$\cdots$	q_1b_n		
		$a_1^{(1)}$	$\cdots$	$a_n^{(1)}$	$\cdots$	$a_m^{(1)}$
				$\vdots$		
			$a_{m-n}^{(m-n)}$		$\cdots$	$a_m^{(m-n)}$
				$q_{m-n+1}b_1$	$\cdots$	$q_{m-n+1}b_n$
				r_0	$\cdots$	r_{n-1}

We may abstractly view this algorithm as computation activities on the index set $\{(i, j) \mid 1 \leqslant i \leqslant m+1, 1 \leqslant j \leqslant m-n+1\}$, and express them by the following recursive equations:

/* Initialization */

$$a_{i0} = a_{i-1} \quad \text{for } 1 \leqslant i \leqslant m+1,$$

$$b_{i0} = \begin{cases} b_i & \text{for } 0 \leqslant i \leqslant n, \\ 0 & \text{for } n+1 \leqslant i \leqslant m, \end{cases}$$

$$q_{0j} = 0 \quad \text{for } 1 \leqslant j \leqslant m-n+1,$$

$$c_{i0} = \begin{cases} 1 & \text{if } i = 0, \\ 0 & \text{if } 1 \leqslant i \leqslant m; \end{cases}$$

/* Propagation & Execution */

$$b_{ij} = b_{i-1,j-1},$$

$$c_{ij} = c_{i-1,j-1},$$

$$q_{ij} = \begin{cases} a_{i,j-1}/b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 1, \\ q_{i-1,j} & \text{if } c_{i-1,j-1} = 0, \end{cases}$$

$$a_{ij} = \begin{cases} 0 & \text{if } c_{i-1,j-1} = 1, \\ a_{i,j-1} - q_{i-1,j}\, b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 0; \end{cases}$$

/* Termination */

$$q_j = q_{m+1,j} \quad \text{for } 1 \leqslant j \leqslant m-n+1.$$

If computation at index point (i, j) depends on the outcome of computation at index point (i', j'), the vector $[i-i' \; j-j']^{\mathrm{T}}$ stands for a data dependency. For example, since the computation of c_{ij} depends on $a_{i,j-1}$, $q_{i-1,j}$, $b_{i-1,j-1}$, $c_{i-1,j-1}$, so $[0\ 1]^{\mathrm{T}}$, $[1\ 0]^{\mathrm{T}}$, $[1\ 1]^{\mathrm{T}}$ are all dependence vectors. From the above recursive equations, all data dependencies can be identified to form a data dependence graph, as shown in Fig. 1.

In order to map this data dependence graph into a systolic array, we seek for a linear transformation represented by a 2×2 matrix $M = [{}^{T}_{S}]$, where T is a time mapping and S is a space mapping. That is, the computation indexed by (i, j) will be performed at the time step $T[i\,j]^{\mathrm{T}}$ in the PE enumerated by $S[i\,j]^{\mathrm{T}}$. The choice of M is constrained to the conditions: $Td > 0$ for all dependence vectors d and M is nonsingular, for apparent reasons. For the

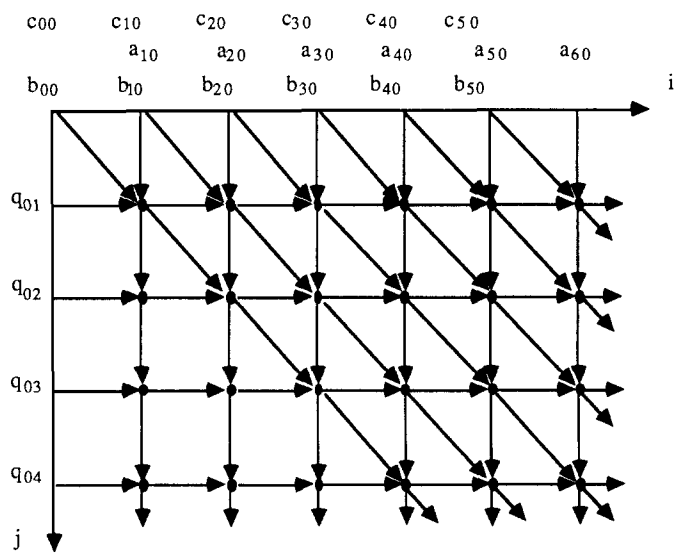

Fig. 1. Data dependence graph of polynomial division.

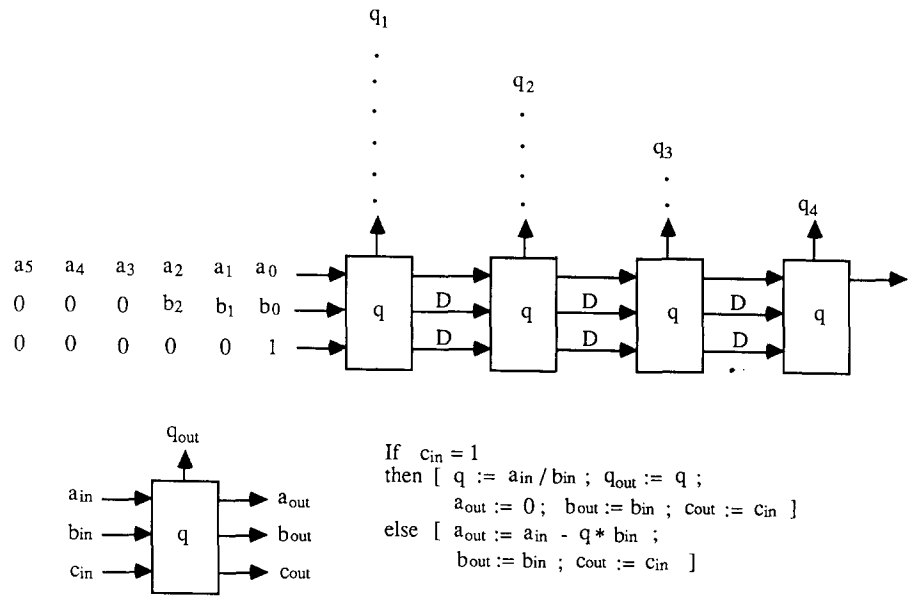

Fig. 2. A systolic array with too many I/O pins.

example above, we especially choose $T = [1\ 1]$ and $S = [0\ 1]$ for the purpose of optimization. The resulting systolic array is depicted in Fig. 2, where each D on a connection means an extra delay unit. The function of PEs, which can naturally deduced from the recursive equations, is also specified there.

The total computation time can be calculated as

$$T\begin{bmatrix} m+1 \\ m-n+1 \end{bmatrix} - T\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1 = 2m - n + 1.$$

The number of PEs in the array is

$$S\begin{bmatrix} m+1 \\ m-n+1 \end{bmatrix} - S\begin{bmatrix} 1 \\ 1 \end{bmatrix} + 1 = m - n + 1.$$

The computation time is optimal due to the following facts. First, we can find a directed path in the data dependence graph (i.e., Fig. 1) with $2(m-n)+1$ index points (1, 1), (2, 1), (2, 2), (3, 2),..., $(m-n,\ m-n)$, $(m-n+1,\ m-n)$, $(m-n+1,\ m-n+1)$ such that the sequence of the computations on these points must be preserved by any parallel implementation of the synthetic division. This implies that $2(m-n)+1$ time steps are required to produce q_{m-n+1}. Second, all the computations on the n index points $(i,\ m-n+1)$, $m-n+2 \leqslant i \leqslant m+1$, depend on q_{m-n+1}, which is generated at $(m-n+1,\ m-n+1)$, and no broadcast is allowed in a temporally local systolic array, we need n more steps to produce the remainder polynomial. Therefore, at least $2(m-n)+1+n = 2m-n+1$ time steps are required for the whole computation. Although this systolic array achieves optimal computation time, it has a severe pin problem. The coefficients of $R(x)$ are emitted all from the rightmost PE, however, different coefficients of $Q(x)$ are output from different PEs, through the upward pins. When counted wordwise, there are $m-n+5$ I/O pins in total.

3. Pin reduction techniques

The exceeding amount of pins for outputting polynomial $Q(x)$ occurs because the data flows of the variable q are 'orthogonal' to the target PE base $[0\ 1]^T$. It is therefore desirable to

'redirect' the data flows in order to reduce the pins. What we propose is a technique called variable duplication. We duplicate q_{ii} by variable q'_{ii} right after it is computed, and make q'_{ii} move along the target PE base. The recursive equations are now rewritten as follows:

/* Initialization */

$$a_{i0} = a_{i-1} \quad \text{for } 1 \leqslant i \leqslant m+1,$$

$$b_{i0} = \begin{cases} b_i & \text{for } 0 \leqslant i \leqslant n, \\ 0 & \text{for } n+1 \leqslant i \leqslant m, \end{cases}$$

$$q_{0j} = 0 \quad \text{for } 1 \leqslant j \leqslant m-n+1,$$

$$q'_{i0} = 0 \quad \text{for } 1 \leqslant i \leqslant m+1,$$

$$c_{i0} = \begin{cases} 1 & \text{for } i = 0, \\ 0 & \text{for } 1 \leqslant i \leqslant m; \end{cases}$$

/* Propagation & Execution */

$$b_{ij} = b_{i-1,j-1},$$

$$c_{ij} = c_{i-1,j-1},$$

$$q_{ij} = \begin{cases} a_{i,j-1}/b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 1, \\ q_{i-1,j} & \text{if } c_{i-1,j-1} = 0, \end{cases}$$

$$q'_{ij} = \begin{cases} a_{i,j-1}/b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 1, \\ q'_{i,j-1} & \text{if } c_{i-1,j-1} = 0, \end{cases}$$

$$a_{ij} = \begin{cases} 0 & \text{if } c_{i-1,j-1} = 1, \\ a_{i,j-1} - q_{i-1,j}\, b_{i-1,j-1} & \text{if } c_{i-1}, j-1 = 0; \end{cases}$$

/* Termination */

$$q_j = q'_{m-n+1,j} \quad \text{for } 1 \leqslant j \leqslant m-n+1.$$

Again the data dependencies can easily be identified and the formed data dependence graph be mapped by exactly the same space-time transformation as the one used in the previous section. The desired linear systolic array, as depicted in Fig. 3, has no upward output pins since the variables q_{ij} do not act as output variables any more. The number of pins becomes 5, a constant which is independent of the problem size.

Another technique which we call variable substitution can be applied to reduce the pins further. Due to fact that the variables a_{ij}, $1 \leqslant i \leqslant j \leqslant m-n+1$, are not used for any operation at all, the variables q'_{ij} can be substituted by a_{ij} there and are totally eliminated from the

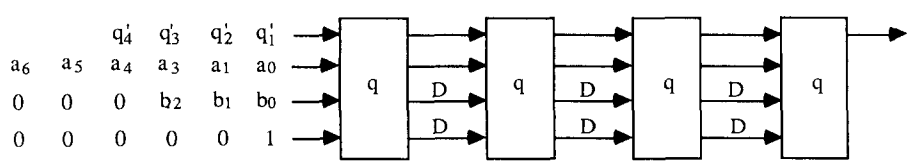

Fig. 3. A systolic array derived with variable duplication.

recursive equations. This brings us the following recursive equations for polynomial division:

/* Initialization */

$$a_{i0} = a_{i-1} \quad \text{for } 1 \leqslant i \leqslant m+1,$$

$$b_{i0} = \begin{cases} b_i & \text{for } 0 \leqslant i \leqslant n, \\ 0 & \text{for } n+1 \leqslant i \leqslant m, \end{cases}$$

$$q_{0j} = 0 \quad \text{for } 1 \leqslant j \leqslant m-n+1,$$

$$c_{i0} = \begin{cases} 1 & \text{if } i = 0, \\ 0 & \text{if } 1 \leqslant i \leqslant m; \end{cases}$$

/* Propagation & Execution */

$$b_{ij} = b_{i-1,j-1},$$

$$c_{ij} = c_{i-1,j-1},$$

$$q_{ij} = \begin{cases} a_{i,j-1}/b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 1, \\ q_{i-1,j} & \text{if } c_{i-1,j-1} = 0, \end{cases}$$

$$a_{ij} = \begin{cases} a_{i,j-1}/b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 1, \\ a_{i,j} - q_{i-1,j}\, b_{i-1,j-1} & \text{if } c_{i-1,j-1} = 0; \end{cases}$$

/* Termination */

$$q_j = a_{m-n+1,j} \quad \text{for } 1 \leqslant j \leqslant m-n+1.$$

The corresponding data dependence graph is the same as in Fig. 1. By the same space-time transformation, we reach the desired systolic design which is depicted in Fig. 4. In this design, all the input data are fed into the array through the leftmost PE and all the output data are delivered from the rightmost PE. The number of pins is reduced to 4 finally.

Let's use $F(x) = 8x^4 + 2x^3 - 2x^2 + 4x + 5$ and $G(x) = 2x^2 - 4x + 1$ as an example to demonstrate the operations of the new linear systolic array. In Fig. 5 all the intermediate results of different time steps are shown. Steps 3, 4, 5 produce the three coefficients of $Q(x) = 4x^2 + 9x + 15$ and then steps 6, 7 produce the two coefficients of $R(x) = 55x - 10$.

4. Concluding remarks

Through the synthesis of a systolic array with only 4 I/O pins for polynomial division, we have fully illustrated two basic techniques for pin reduction. Variable duplication on the data

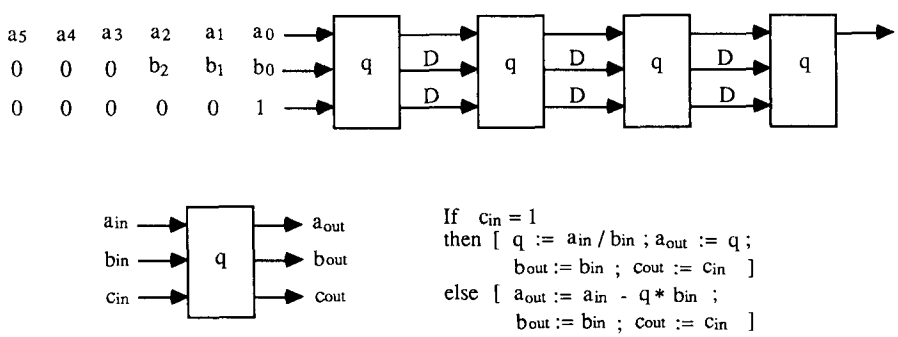

Fig. 4. The new linear systolic array.

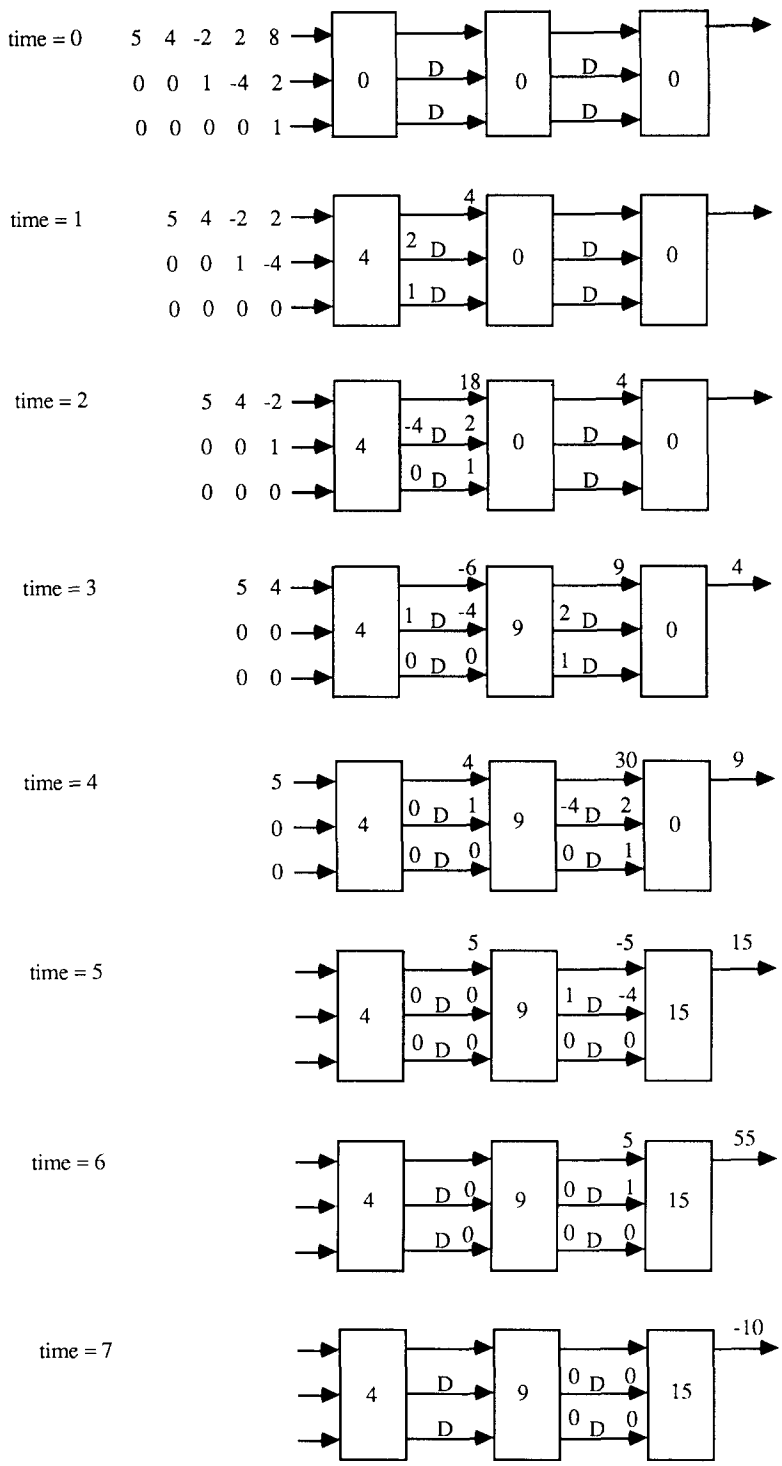

Fig. 5. Snapshots of the new linear systolic array.

dependence graph can be utilized to change data flow direction and reduce the problem-size-dependent number of pins to a constant. Variable substitution can then be applied to reuse some inactive variables and hence reduce the I/O pins further. In the example of polynomial

division, the pin reduction does not affect computation efficiency, local memory requirements and data pipelining. The pin problem has somehow been ignored by many synthesis methods. We believe that the result presented in this paper will lead to more systematic treatments on this important subject.

References

[1] A. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1975).

[2] J.A.B. Fortes, K.S. Fu and B. Wah, Systematic approaches to the design of algorithmically specified systolic arrays, *Proc. International Conference on Acoustics, Signal and Speech Processing, Vol. 1* (1985) 8.9.1–8.9.5.

[3] C. Guerra and R. Melhem, Synthesizing non-uniform systolic designs, *Proc. International Conference on Parallel Processing* (1986) 765–771.

[4] K. Hwang and S.P. Su, VLSI architecture for feature extraction and pattern classification, *J. Comput. Vision, Graphics and Image Processing* **24** (2)(1983) 215–228.

[5] O. Ibarra, S. Kim and M. Palis, Designing systolic algorithms using sequential machines, *IEEE Trans. Comput.* **35** (1986) 531–542.

[6] K. Kailath, *Linear Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[7] H.T. Kung, Let's design algorithms for VLSI systems, *Proc. Cal Tech. Conference on VLSI* (1979) 65–90.

[8] H.T. Kung, Why systolic architectures?, *IEEE Comput.* **15** (1982) 37–46.

[9] H.T. Kung and C.E. Leiserson, Systolic arrays (for VLSI), *Proc. SIAM Sparse Matrix Symposium* (1978) 256–282.

[10] S.Y. Kung, P.S. Lewis and S.C. Lo, On optimally mapping algorithms to systolic arrays, *Proc. IEEE International Symposium on Circuits and Systems* (1986) 1316–1322.

[11] F.C. Lin and I.C. Wu, Broadcast normalization in systolic design, *IEEE Trans. Comput.* **37** (1988) 1428–1434.

[12] F.J. MacWilliams and N.J.A. Sloane, *The Theory of Error-correcting Code* (North-Holland, Amsterdam, 1977).

[13] D.I. Moldovan, On the design of algorithms for VLSI systolic arrays, *Proc. IEEE* (January 1983) 113–120.

[14] D.I. Moldovan and J.A.B. Fortes, Partitioning and mapping algorithms into fixed size systolic arrays, *IEEE Trans. Comput.* **35** (1986) 1–12.

[15] J.J. Navarro, J.M. Llaberia and M. Valero, Computing size-independent matrix problems on systolic array processors, *Proc. 13th Annual Symposium on Computer Architecture* (1986) 271–279.

[16] S.V. Rajopadhye, S. Purushothaman and R.M. Fujimoto, On synthesizing systolic arrays from recurrence equations with linear dependencies, Lecture Notes on Computer Science **241** (Springer, Berlin, 1986) 488–503.

[17] S.H. Żak and K. Hwang, Polynomial division on systolic arrays, *IEEE Trans. Comput.* **34** (1985) 577–578.