A Systematic Approach to Bit Recursive Systolic Array Design

KuoJuey R. Liu and K. Yao
Electrical Engineering Department
University of California, Los Angeles
Los Angeles, CA 90024-1594

ABSTRACT

In this paper we propose a systematic approach to bit-level recursive systolic array design. This approach provides a systematic way to design a recursive computational architecture instead of a bit slice architecture. Since the relationship is much stronger at the bit-level than at the word level and most relations can be described as shift-and-operate computations, we can formulate these kinds of relations as recursive equations. From the recursive equations we can build the systolic array without deriving the dependence graph of the bit-level computation. Some design examples for bit recursive systolic array are also presented.

1 Introduction

The bit level systolic array was first proposed by McCanny and McWhirter [1]. They argued that the computations involving sum-of-product need not necessarily be built at the multiply/add (inner product cell) level, instead they can be built at the bit level consisting only of gated full adders, unless the intermediate results are needed. Since then various specific DSP operations, such as convolution and correlation, inner product computation, Winograd Fourier transform algorithm, and rank order filtering, have been built at the bit level [3].

From the practical point of view, there are some advantages for the implementation at the bit level. For example, the system can be clocked at very high rates because the delay is only due to some logic gates; the size of the basic cells is small and so is the overall system; for fault tolerance application, we can by-pass or reconfigure some faulty cells without sacrifying too much silicon area; etc..

However, the design methodology at the bit level systolic array has rely on heurestic approaches. The bit slice concept of the bit level array is to build a one-bit basic array processor system and concatenate them either in parallel or in serial to form a multi-bit system. This approach may result in needing many chips as in the word level, which also may require much silicon area. In this paper, we propose a systematic procedure to develope a bit level recursive systolic array. Such an approach provides a simple and systematic way to design a one-chip recursive computational architecture instead of a bit slice architecture.

In the next section, some approaches to systolic array are introduced. In Section 3, the concept of bit recursibility and bit-level recursive equation are discussed. In Section 4, a systematic algorithm is proposed for the bit recursive systolic array. Finally some design examples are presented in Section 5.

CH2603-9/88/0000/0685$01.00 © 1988 IEEE

2 Mapping to Systolic Array

Various approaches to map computational algorithms onto systolic array structures have been proposed [7,14,15,16,17]. Generally they can be classified into three categories: functional transformation, retiming, and dependence mapping. Our method for the bit recursive systolic array design belongs to the dependence mapping category. The dependence mapping consists of several steps: step 1 is to map an algorithm to a dependent graph (DG); step 2 is to map the dependent graph to a signal flow graph (SFG); step 3 finally maps the SFG onto an array processor.

All these approaches can work either at the word or at the system level. However, these approaches are not easily applicable at the bit level because too many bit parameters are involved and thus resulting in very complicated data dependent graphs. For example, an inner product for two N-component vectors yields one scalar result. At the word level, this computation involves only $2N + 1$ scalar word parameters. However, at the bit level it becomes $(k+m)N + (k+m)$ binary bit parameters, where k and m are the number of bits for each word. Moreover, the dependent graph for an inner product at the bit level is more complicated to analyze. Here we propose a systematic approach for bit level systolic array design. Finally, we will show that our approach is actually a modified scheme of the above word-level design methodology.

3 Bit Recursibility

In most computations, there are various degrees of dependencies among the operands. This dependent relationship is much stronger at the bit level than at the word level. Based on the fact that most of the algebraic computations in the bit level consist only of simple shift and add (or other simple logical) operations, these operations occur again and again for different bits. In most cases, all the bits for a given word take all the same operations. This fact motivate us to find a recursive operation to perform the computations.

Definition 1: A computation is said to be *bit recursivable* if it can be realized in the bit level systolic array and there is a recursive relation at the bit level so that the overall bit level system can be built in a recursive computational manner.

The bit recursivable means that if we can find a bit recursive relation at the bit level, we can reduce the chip area by using bit serial I/O's instead of bit I/O's parallel at some data flow directions.

Definition 2: An architecture is said to be *directional bit hierarchical* if the bit flows for some variables are bit parallel and never interact or join together.

As shown in Fig. 1, the computational architecture is bit hierarchical in the $Y = \{y_i\}$ direction.

Lemma 1: If there exists some recursive relations for a computational algorithm at some directions then all these directions are bit hierarchical.
Proof: A bit recursivable computation is associated with a bit recursive equation which describes the bit order relationship. If it is not bit hierarchical at these recursivable directions, there must be some intersections of the bit flows such as that shown in Fig. 2. If we take the

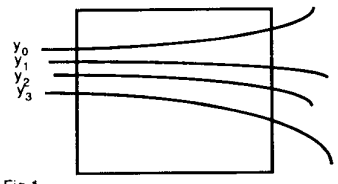


Fig.1 A computational structure

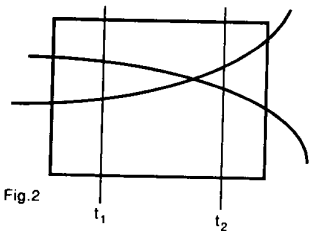


Fig.2

Figure 1. Bit hierarchical data flow of $\{y_i\}$ in a computing system.
Figure 2. The intersection of two bit data flows results in different recursive relations at different instances t_1 and t_2.

bisection at two different instances t_1 and t_2 where t_1 is ahead of the intersection and t_2 is after the intersection, then the order bit relations at these two instances are different. Thus, it contracdict the recursive relation which only a unique ordered bit relation is permitted. □

For many algebraic computations, the manipulations at the bit level consist of many shift-and-operate operations. Therefore, we can describe this kind of shift-and-operate computation by a first order bit recursive equation. Let Z be the output and X and Y be the input variables. A general first order bit recursive equation can be described as

$$\underline{Z_b}(i+1) = a(i) \cdot \underline{Z_b}(i) + b(i) \cdot (x_i \otimes \underline{Y_b}(i)) + \underline{C_b}(i), \tag{1}$$

where $\underline{Z_b}$ is the bit vector form of Z given by

$$\underline{Z_b}(N) = \{\cdots, z_{N-1}, \cdots, z_1, z_0\},$$

z_i and x_i are binary representations of Z and X respectively, while $a(i)$ and $b(i)$ are scalar constants. The $+$, $\cdot$, and $\otimes$ are some logical or arithematic operators.

A special case of this recursive equation is when $a(i) = 1, b(i) = 2$, and $\underline{C_b}(i) = \underline{0}$, for all i, and $\cdot, \otimes$, and $+$ represent multiplication, and, addition operations respectively. Then equation (1) becomes

$$\underline{Z_b}(i+1) = \underline{Z_b}(i) + 2(x_i \; AND \; \underline{Y_b}(i)). \tag{2}$$

Equation (2) represents a shift-and-add operation for the product of X and Y. It should be noted that since the carry propogates from lower significant bit to higher significant one, for simplicity and speed consideration, the bit recursive equation usually starts from least significant bit.

Lemma 2: If a computation is bit hierarchical at some directions and there exist some bit recursive relations at these directions, then the associated bit-level dependence graph for this computation is directional shift-invariant at these directions.

Proof: In these directions, the dependence arcs corresponding to the nodes along these directions are invariant with respect to the node position according to the recursive relations. Therefore, the bit-level dependence graph is directional shift-invariant along these directions. □

Theorem: A computation is bit recursivable if it satisfies all the following conditions:

1. It has directional bit hierarchical property;
2. Recursive relation exists for these bit hierarchical directions;
3. One-bit systolic array can be built at these hierarchical directions.

4 A Systematic Algorithm Design

Given a computation with some word level parameters, in order to find out whether the computation is bit recursivable, we propose the following procedure:

Step1. Pick a parameter which interacts with the other parameters as an one-bit parameter using the least significant bit. If the parameter is a vector, then consider all the components as one-bit components and take only the least significant bit. Assume all other's parameters are all bit-parallel parameters.

Step 2. According to the given computation, find the algebraic (or logical) relationship of these bit parameters and derive the one-bit systolic array. If it is not systolizable, then go to step 1 and try to pick up another parameter.

Step 3. Increase one bit for the one-bit parameter and then find the recursive equation for the computation. If there is a serial recursive equation that exists between this new input bit and the original least significat bit, modify by feedback some links of the one-bit systolic array obtained in Step 2 to satisfy this recursive equation. Otherwise, go to Step 1 to pick up another parameter.

Step 4. If we cannot find a satisfactory recursive relation in Step 3 or built a one-bit systolic array in Step 2 for some of the parameters, then the computation is not bit recursivable.

Suppose in Step 1, we pick up one parameter and by using this one bit parameter, it does not significantly reduce the complexity of the bit level dependence graph. Then we can pick up some other parameter and repeat this procedure. In each step, the parameter to be reduced to one bit can be randomly selected, but is required such that the complexity is reduced and the operation systolizable.

It can be easily seen that by comparing to the word level systolization procedure, the procedure we propose for the bit level systolization requires no more than the assumption that the dependence graph at the bit level exists and the dependence graph is shift invariant at bit direction(s) we choose to reduce. Once these conditions are satisfied, we then derive the recursive relation which is equivalent to the projection from DG to SFG for the word level systolization procedure. The iterations find the maximum degree of recursivability that can be achieved.

5 Design Examples

In this section, some design examples for basic signal processing operations such as multiplier, inner product, and convolution/correlation are studied.

5.1 Multiplier

Suppose that A and B are real valued scalar inputs and C be the multiplication output with M, N and $M+N$ bits respectively. For simplicity of presentations, assume that $M = N = 4$. Let's first assume that B is a one-bit binary input b_0 and $A = (a_3, a_2, a_1, a_0)$, where a_i is the binary representation of A. Let $\underline{C_i}$ be the ouput bit vector form at the i-th iteration. The first multiplication result is

$$\underline{C_1} = (c^3_{(1)}, c^2_{(1)}, c^1_{(1)}, c^0_{(1)}) = (a_3b_0, a_2b_0, a_1b_0, a_0b_0) \tag{3}$$

The systolic array for computing $\underline{C_1}$ is shown in Fig.3. In order that for b_0 not being broadcasted, a_i should be arranged in a skew fashion. Since it is systolizable for this one-bit case, we could proceed to find whether the bit recursive relation exists at b_i direction. Now increases the bit number of B to a two-bit variable (b_1, b_0). Then the multiplication becomes

$$\underline{C_2} = (c^4_{(2)}, c^3_{(2)}, c^2_{(2)}, c^1_{(2)}, c^0_{(2)}) = (0, \underline{C_1}) + (\underline{R_1}, 0) = \underline{C_1} + 2 \cdot \underline{R_1}, \tag{4}$$

where

$$\underline{R_i} = (a_3b_i, a_2b_i, a_1b_i, a_0b_i), \qquad i = 0, 1, 2, 3.$$

As we can see the recursive relation between $\underline{C_1}$ and $\underline{C_2}$ does exist so that the systolic array in Fig.3 can be modified to satisfy two-bit input by propagating the output of each bit stage to the next (lower) bit stage and by feedback the carry so that it could be accumulated. The input $\{a_i\}$ also need feedback to perform the recursive operation. The modified architecture is shown in Fig. 4.

Generally, we can derive the recursive equation as

$$\underline{C_i} = \underline{C_{i-1}} + 2 \cdot \underline{R_{i-1}}. \tag{5}$$

It is obvious that the architecture in Fig. 4 satisfies this recursive equation. The final version is shown in Fig. 5.

In order to demonstrate the similarity between this approach and the word-level approach, we derive the dependence graph for bit-level multiplication as shown in Fig. 6. We can see that it is a direction invariant dependence graph so that it can be projected at any direction. If we project at direction (0,1) and then apply the cut-set systolization procedure we can obtain same systolic array as above. In our approach, we do not consider the existence of the direction invariant dependence graph, instead we presume such DG exist and

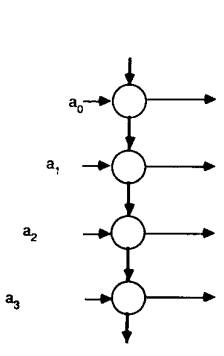


Fig.3

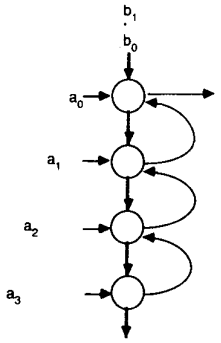


Fig.4

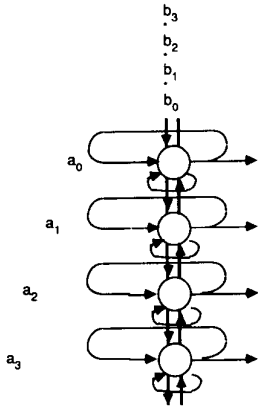


Fig.5

Figure 3. One-bit systolic array for multiplication.
Figure 4. Modified one-bit architecture for two-bit multiplication.
Figure 5. A multi-bit systolic multiplier. The carry and input links are feedbacked to achieve the recursive relation.

can project at the (0,1) direction. First, we check the existance of systolizability for one-bit case and then we try to derive the recursive equation. If both conditions are satisfied, we thus confirm the projection at the (0,1) direction is feasible.

When the number of parameters is small, there is not much difference between these two approach. However, as the number of parameters increases, the dependence graph becomes complicated and is not easy to derive. Next example shows one such limitation.

5.2 Inner Product

The inner product of two L-component vectors $\underline{A}$ and $\underline{X}$, with m and n number of bit for each component respectively, results in a scalar output Y of at most $(m+n)$ bits. For simplicity of presentation, we assume each component of $\underline{A}$ and $\underline{X}$ is a three bit parameter. Initially, assume the components of vector $\underline{A}$ are all one-bit scalar $(m=1)$, then the inner product of this $\underline{A}$ and $\underline{X}$ is given by

$$\underline{Y_1} = (y^2_{(1)}, y^1_{(1)}, y^0_{(1)}) = (T(\sum_{i=0}^{L} a^0_i x^2_i), T(\sum_{i=0}^{L} a^0_i x^1_i), T(\sum_{i=0}^{L} a^0_i x^0_i)) \tag{6}$$

where the operator T is defined as adding carries propogated from lower significant bits and taking the least significant bit of its content. The corresponding systolic array is shown in Fig. 7. After $L+3$ (or generally $L+n$) cycle we obtain the output $\underline{Y_1}$.

Since it is systolizable for the one-bit case, we then proceed to check the two-bit case $(m=2)$ and determine the existence of the recursive relation. It can be easily found that $\underline{Y_2}$ satisfies the following recursive equation.

$$\underline{Y_2} = (y^3_{(2)}, y^2_{(2)}, y^1_{(2)}, y^0_{(2)}) = (0, \underline{Y_1}) + (\underline{R_1}, 0) = \underline{Y_1} + 2 \cdot \underline{R_1} \tag{7}$$

where

$$\underline{R_i} = (T(\sum_{j=0}^{L} a^i_j x^2_j), T(\sum_{j=0}^{L} a^i_j x^1_j), T(\sum_{j=0}^{L} a^i_j x^0_j)), \qquad i = 0,1,2.$$

We can see that the recursive equation (7) has the same form as equation (4). Therefore, the inner product could be design as a bit level linear array structure. But the problem is that the throuthput will be severely hindered since the latency for computing each recurrance

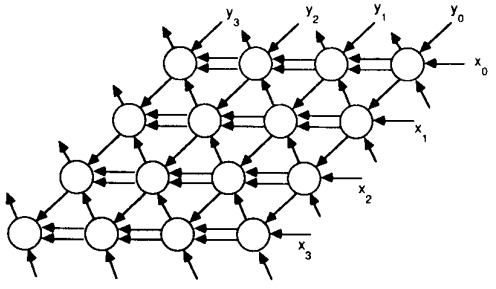


Fig.6

Figure 6. Dependence graph for multiplication.

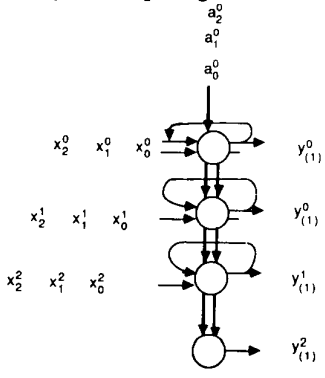


Fig.7

Figure 7. One-bit systolic array for inner product.

require $L+3$ (or generally $L+n$) clock delays and thus it requires a totally of $2(L+3)$ (or generally $m(L+n)$) clocks to compute equation (7). That is, for each $L+3$ clocks cycle we obtain only one recurrance output. The problem becomes severely when m is increased.

From equation (7), we could find a faster way to design the inner product if we put more linear arrays to perform the recursive equatin (7). As in Fig. 8, we put one more linear array to perform the recursive equation (7). The latency now becomes $(L+4)$ instead of $2(L+3)$ as before. The general recursive equation is

$$\underline{Y_i} = \underline{Y_{i-1}} + 2 \cdot \underline{R_i} \tag{8}$$

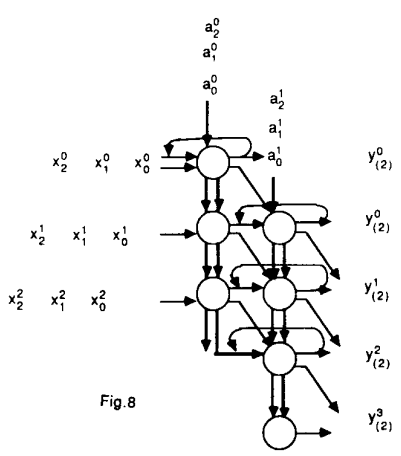
and the final systolic array to perform the inner product is shown in Fig. 9. This architecture takes only $(L+m+n-1)$ clocks to obtain the result. Even though it runs fast, the tradeoff is that it requires more silicon area.

5.3 Convolution and Correlation

The convolution or correlation are given by the form

$$y(k) = \sum_{i=1}^{L} a(i)x(k \pm i) \tag{9}$$

Basically, this is a kind of inner product operation but with x_k moving. Therefore, we prefer an efficient pipelined structure to deal with the moving data. The architectures discussed in subsection **5.1** and **5.2** are not feasible because of non-efficient pipeline rates. Let us assume that a_i and x_i had r and m bits respectively. If we further assumed a_i is a one-bit parameter, the bit equations for ouput y is as follows:



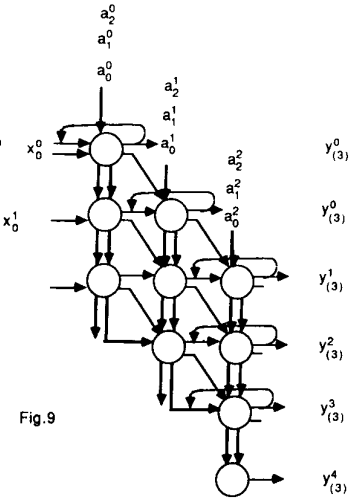


Figure 8. Two-bit systolic array for inner product.
Figure 9. Multi-bit recursive systolic array for inner product.

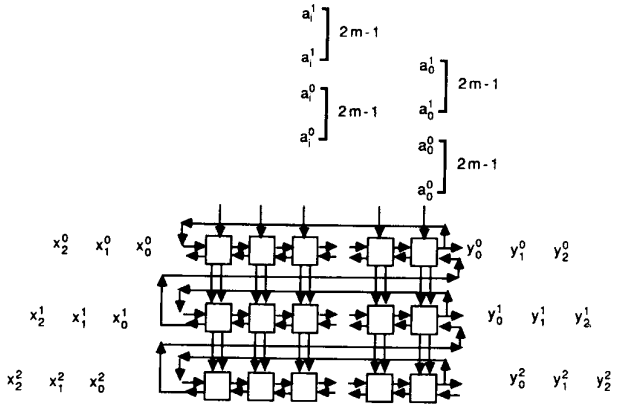


Figure 10. Multi-bit recursive systolic array for convolution/correrlation.

$$\begin{aligned} y_{(1)}^{m-1} &= T(a_0^0 x_0^{m-1} + a_1^0 x_1^{m-1} + \cdots + a_L^0 x_L^{m-1}) \\ y_{(1)}^{m-2} &= T(a_0^0 x_0^{m-2} + a_1^0 x_1^{m-2} + \cdots + a_L^0 x_L^{m-2}) \\ &\vdots \\ y_{(1)}^{1} &= T(a_0^0 x_0^0 + a_1^0 x_1^0 + \cdots + a_L^0 x_L^0) \end{aligned} \tag{10}$$

and the bit vector form of y is

$$\underline{Y_1} = (y_{(1)}^{m-1}, y_{(1)}^{m-2}, \cdots, y_{(1)}^{0})$$

A bit recursive equation for $\underline{Y_i}$ can be derived as

$$\underline{Y_{i+1}} = \underline{Y_i} + 2 \cdot \underline{R_i} \tag{11}$$

where

$$\underline{R_i} = (r_i^{m-1}, r_i^{m-2}, \cdots, r_i^1, r_i^0)$$

and

$$r_i^j = T(a_0^i x_0^j + a_1^i x_1^j + \cdots + a_L^i x_L^j)$$

In [1],[4] McCanny and McWhirter proposed an one-bit correlation systolic array which performs equation (10). Then according to equation (9), the multi-bit system can be designed in a bit slice fashion. By using the one-bit correlation systolic array proposed by McCanny as a basic building block and the recursive equation (11), we can reconfigure some feedback lines to achieve the multi-bit convolution/aorrelation array. As is shown in Fig. 10, each output is feedback to output variables fan in port of the next stage and input is feedback into it's input variables fan in port. The number of processors required is $(m + log_2 L)L$ and the computation time is propotional to $(2m-1)r + L$. If the one-bit correlation systolic array is designed in the way that variable a flow horizontally and y flow vertically, then the same architecture as in [4] can be also achieved.

6 Conclusion

Comparing equations (5), (8), and (11), we can see that the above design examples have similar recursive equations, but the designs are not unique. Indeed the designs are quite dependent on the one-bit systolic array and the criterion required by the specific computation. Since the cell delay at bit level is significantly less than that of the word-level cell such as inner-product cell, the system can be clocked at a very high rate so that the bit serial operation will not hinder much of the time complexity performance. And, of course, the area can be saved by reducing some degree of parallism into bit recursive operation.

This paper describes a systematic approach in designing a bit level systolic array. Once the one-bit architecture is systolizable and there is some kinds of relationship at the bit level, we can easily find the multi-bit architecture by this approach. The efficiency of the multi-bit architectue depends on how we design the one-bit architecture.

7 Acknowledgement

This work was partially supported by the UC MICRO program.

References

[1] J. V. MaCanny and J. G McWhirter, "Implementation of signal processing functions using 1-bit systolic array," Electronics Letters, 18, No 6, pp.241, Mar. 1982.

[2] A. Corry and K. Patel, "Architecture of a CMOS correlator," Proc. ISCAS, pp.522,1983.

[3] J. V. MaCanny and J. G McWhirter, "Some systolic array developments in the United Kingdom," IEEE Computer,Vol. 20, pp.51, July, 1987.

[4] J. V. MaCanny et al., "Optimised bit level systolic array for convolution," IEE Proc. Vol 131, Pt. F, No. 6, Oct. 1984.

[5] J. S. Ward et al, "Bit-level systolic array implementation of the Winograd Fourier transform algorithm," IEE Proc. Vol.132, Pt. F, No.6, Oct. 1985.

[6] R. B. Urquhart and D. Wood, "Efficient bit-level systolic arrays for inner product computation," GEC J. Res., 1984, 2, pp.52.

[7] S. Y. Kung, **VLSI Array Processors**, Prentice-Hall, 1987.

[8] J. C. White et al. "A high speed CMOS/SOS implementation of a bit level systolic correlator," Proc. ICASSP, pp.1161, 1986.

[9] J. G. McWhirter and J. V. McCanny, **VLSI Technology and Design**, Academic Press (London), 1987.

[10] H. T. Kung, "Why systolic architectures?" Computer, Vol.15, No. 1,pp.37, Jan. 1982.

[11] I-Chen Wu, "A fast 1-D serial-parallel systolic multiplier," IEEE Trans. Comp. Vol. C-36, pp.1243, Oct. 1987.

[12] S. Y. Kung et al., "Mapping digital signal processing algorithms onto VLSI systolic/wavefront arrays," Proc. 20th Asilomar Conf. on Signal, System & Computers, pp.6, 1986.

[13] J. V. MaCanny et al., "Use of unidirectional data flow in bit-level systolic array chips," Electronics Letters, Vol.22, No.10, May, 1986.

[14] U. Weiser, A. Davis, "A wavefront notation tool for VLSI Design", in VLSI Systems And Computations, Computer Science Press, Rockville, MD, pp. 226-234, 1981.

[15] C. E. Leiserson, J. B. Saxe, "Optimizing synchronous circuits", Proc. 22nd IEEE Symp. on Foundations of Computer Science, pp. 23, 1981.

[16] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays", Proc. of the IEEE, Vol. 71, No. 1, pp. 113, 1983.

[17] P. Quinton, "The systematic design of systolic arrays," IRISA Rept., March, 1983.