A Heuristic For Data Path Synthesis Using Multiport Memories

Imtiaz Ahmad and *C. Y. Roger Chen*

Department of Electrical and Computer Engineering
Syracuse University, Syracuse, NY 13244

Abstract

Recently there is a trend for the designer to group registers into register files for efficiently implementing large VLSI chips. Multiport memories provide an effective way for such an implementation. Interconnection minimization (such as multiplexers and tri-state buffers) has become more difficult with the use of multiport memories. In this paper, a heuristic is presented which performs functional units and connection allocation tasks simulatneously to get better results for application specific designs assuming that registers have already been grouped to multiport memories. Experiments on benchmarks show very promising results.

1 Introduction

In designing VLSI chips in order to reduce the design time, explore the design space more rapidly, and obtain a design with fewer errors, high-level synthesis has become an active area of research since the past few years [1]. High-level synthesis is the realization of a register transfer level description from the functional specification described in a hardware description language like VHDL [2]. The target model for a behavior synthesizer consists of two parts: a *data path* and a *control unit*. A data path, composed of functional units (FUs), storage devices and interconnects, is capable of performing all data manipulating functions specified in the behavioral description. The function of a control unit is to control the flow of data in the data path by issuing control signals at correct times.

Data path synthesis consists of two major tasks: *scheduling* and *allocation*. Scheduling determines the number of FUs of each type, the number of control steps needed and the control step in which to perform each operation. During allocation, registers are allocated for variables, operations are assigned to FUs and connections are established between registers and FUs.

Data path allocation can be further divided into three mutual dependent subtasks: *register allocation*, *functional unit allocation*, and *connection allocation*. The first subtask binds variables to registers (register files) in such a way that the lifetimes of those variables bound to a same register will not overlap. Its primary objective is to minimize the number of registers used. The second subtask assigns operations to FUs while making sure that no more than one operation in any control step is assigned to any FU. The third subtask is to find a connection pattern that requires least expensive hardware (such as buses, multiplexers and tri-state buffers).

In designing large/complicated VLSI chips, in order to maintain a structural design and reduce the design effort, recently there is a trend for the designer to group registers into register files [3,4] for efficient implementation. Multiport memories provide an effective way for such an implementation. Multiport memories have the advantage of reducing part count by providing simultaneous access to the data by more than one FU at a time and special purpose algorithms in signal processing can take advantage of multiple ports of such memories [5]. A multiport memories based design will be more structured, modular and dense, and require less chip area. Due to a smaller number of parts, the generated design can be tested more easily. All these advantages of using multiport memories are worthwhile only if multiport memories can satisfy the following three key requirements:

- A high level of integration.
- A large number of ports.
- A high operating speed.

Recent improvements in technology for the developments of multiport memories fulfills all the above requirements. A multiport RAM compiler with flexible layout and port organization has been reported in [6] and is very useful for data path synthesis to generate fast compact multiport RAMs, where exact pitch matching is required. For example, the address access time reported is 5 nanoseconds for a 1K three port RAM generated by this compiler. Another pipelined, time sharing access technique to generate high speed and high density multiport memories is reported in [7]. The availability of high density and high speed multiport memories, and all of the above advantages motivate the use of multiport memories in data path synthesis.

0-8186-2610-0/92 $3.00 © 1992 IEEE

In this paper, we address the functional unit and connection allocation tasks, assuming that scheduling and assignment of registers to multiport memories have been performed, with the objective to design a data path with a minimum number of multiplexers and tri-state buffers. The paper is organized as follows: An overview of related work is given in Section II, and the target architecture is presented in Section III. Section IV describes functional unit and connection allocation. Experimental results are reported in Section V and concluding remarks are made in Section VI.

2 Related Research

Several different approaches have been proposed to solve the functional unit and connection allocation problems. FACET [8] uses a formal clique partitioning technique to solve these problem. HAL [9] uses force directed scheduling and a weighted clique partitioning algorithm to synthesize a data path. MAHA [10] uses a critical path determination for the allocation task. SPLICER [11] uses a branch-and-bound search for solving these subtasks. SPAID [12] uses heuristic algorithms to map the graph into a multi-bus multifunction unit architecture. LYRA and ARYL [13] use a weighted bipartite matching approach to solve the FU allocation subtask and a greedy approach for connection allocation. STAR [14] and FLORA [15] use a branch-and-bound search for each subtask. Stok [16] use a simulated annealing approach to reduce the interconnection. Other approaches have been implemented in systems like EASY [17], CATHEDRAL-II [18], HYPER [19] and PARBUS [20]. All these approaches use either isolated registers or single port register file.

The interconnection task using isolated registers is shown to be NP-complete [21] and this task becomes more difficult with the use of multiport memories because in order to establish a connection between a register and FU terminal, we need to select a memory port also. Very few systems have reported the use of multiport memories. MIMOLA [22] has mentioned the use of multiport memories based only on the simultaneous access requirements without actually showing any methodology or technique for doing that. Balakrishnan *et al.* [23] have also reported a 0-1 integer linear programming technique to group a maximal number of registers to only one multiport memory at a time. The registers not included in the multiport memory can either be synthesized as isolated registers or be grouped into other multiport memories by repeatedly applying the same algorithm which does not result in a minimum number of multiport memories. Their interconnection minimization approach is valid for only one multiport memory at a time and moreover, they assume that the information to which FU terminal a register is assigned is given. This assumption does not allow the freedom of connecting a register to any terminal of a FU accessing this register, and hence does not result in minimum interconnection [8,29].

Chen *et al.* [24] reported a heuristic to only group registers into multiport memories assuming that multiport memories have a same type of ports; that is, their approach does not allow a mixed type of ports such as some ports are read only, while other ports are read/write. Their approach does not deal with the operation and connection allocation task. In the technique reported by Wilson *et al.* [25] registers are allocated to multiport memories one-by-one, which is not necessarily an optimal solution. Wilson *et al.* [26] presented an approach to find a minimal number of connections between memory ports and FU terminals. They assume that FU allocation has been performed and the information to which Fu terminal a register will be connected is given. Moreover, their approach deal with each memory separately for connection allocation and do not consider the interactions between memories and all FU terminals to get better results.

MAP [27] overcomes some of these problems and deals with the grouping of registers into a minimum number of multiport memory modules and the minimization of interconnection hardware. Due to the constraints of using 0-1 integer linear programming formulation, their approach deals with multiplexers and tri-state buffers minimization in the connection allocation task in two different phases, and hence does not guarantee globally optimal solutions. However, some of the main weaknesses of this and other approaches [23-26] are:

- They assume that operation assignments have been performed.
- They model the problem as 0-1 integer linear programming problem, which is not practical for problems of large sizes.

In this paper, a heuristic is presented, which overcomes all these drawbacks and perform FU and connection allocation tasks simultaneously to get better results.

3 Target Architecture

The inputs to our system for the application specific design include:

- A code sequence like Value Trace [8].
- The set of available FUs with known capability.
- The set of available multiport memories with known numbers of ports and type of each port.

The synthesized final architecture can be either in linear topology or in random topology. In this paper,

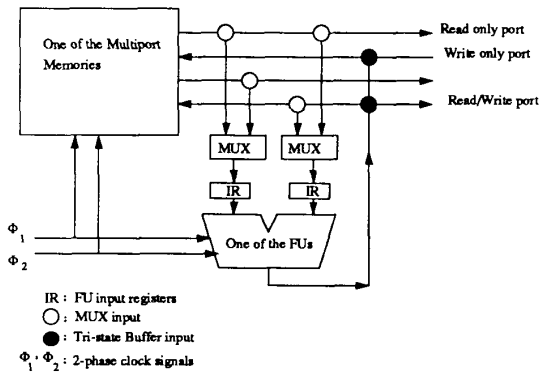

Figure 1: Target architecture.

we focus mainly on linear topology architectures because they have the advantage of reducing interconnection cost drastically [12,19] and are very effective for signal processing applications. In random topology, multiplexers are used between registers and FU inputs/outputs. The technique described in this paper is equally applicable to random topology architectures as well.

The architecture used in linear topology as shown in Figure 1 is different from SPAID [12] and STAR [14]; it consists of a set of FUs, a set of multiport memories, a set of buses which connect memories to FUs through multiplexers. Multiport memories may consist of different numbers of ports and each port may be read only or write only or both read/write. There is a unique bus for each port. The output of each FU is connected through tri-state buffers to those buses which are connected to write only or read/write ports. The decision to trade off between tri-state buses and multiplexer steering logic is sensitive to the relative costs of them in module library [28]. A two phase clocking scheme is used; in one phase data is transferred from memories to the FUs input registers and in the other phase data is transferred from FUs to memories.

4 FU and Interconnect allocation

After registers are grouped into memories, operations are assigned to FUs and connections are allocated between memories ports and FUs terminals. With the use of multiport memories, the operation and interconnection allocation problem has become a different and more complicated problem than the conventional one, where multiport memories are not used. This is due to the fact that in order to establish a connection from a register to a FU, we need to properly assign registers to the ports of memory in different

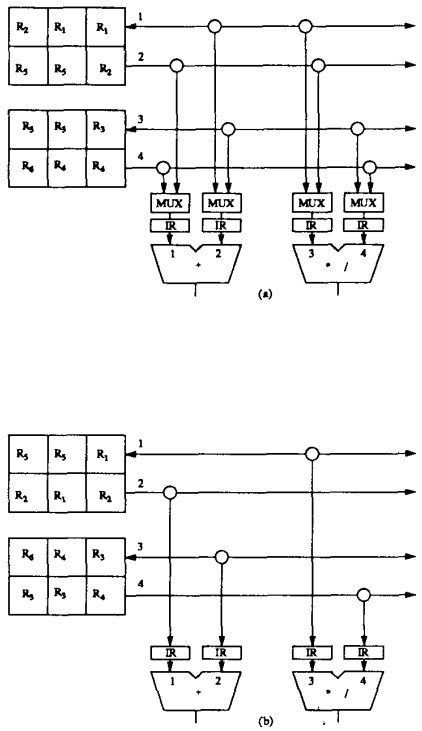

Figure 2: Registers assignment to memory ports by (a) using a straightforward method (b) applying the technique discussed in this paper

control steps in order to save interconnection. As an illustration, consider the partial data path as shown in Figure 2 for the following example, in which registers have already been grouped to memories, with two different assignments of registers to ports of memories.

$$1: \quad R_1 \leftarrow R_2 + R_3, \quad R_4 \leftarrow R_1 * R_4,$$
$$2: \quad R_5 \leftarrow R_4 + R_1, \quad R_6 \leftarrow R_3 / R_5,$$
$$3: \quad R_2 \leftarrow R_2 + R_6, \quad R_4 \leftarrow R_3 * R_5.$$

Using straightforward assignment we need four 2x1 multiplexers at the inputs of FUs as shown in Figure 2(a) and no multiplexers at the input of FUs by applying the technique described in this paper as given in Figure 2(b). Therefore, it is very important to make an intelligent assignment of registers to memory ports to reduce the interconnection.

In order to take into consideration the interconnection cost while assigning operations to FUs, we construct a two-dimensional array, $C[i = 1...n][j = 1...n]$ (where n is the total number of operations in the con-

trol sequence), to hold the value of mutual uncorrelation between each pair of operation op_i and op_j according to:

$$C[i][j] = \alpha_2 * x'_{ij} + \beta_2 * y'_{ij} \quad (1)$$

where $x'_{ij} = 1$ if the output variables of op_i and op_j are not bound to registers in a same memory, else $x'_{ij} = 0$ and y_{ij} represents the number of input registers of these two operations which are not bound to a same memory. α_2 and β_2 are the parameters to tune the cost function. The value of mutual uncorrelation between each pair of operations indicates to what extent these operations should not be mapped to a same FU, if both of these operations can be performed by a same FU. In order to facilitate the incorporation of special constraints introduced by the multiport memories, we have selected the criteria of unfavorism instead of favorism used by many researchers in this area.

To allocate registers to memory ports and interconnection of ports to FUs, the following criteria must be met.

1. Any register which is accessed in a control step should be assigned to at least one of the ports of the memory in which it resides.
2. At most one register can be assigned to a port in any control step.
3. If in any control step k, a register R_a is accessed by functional unit FU_b and register R_a is assigned to $Port_c$, then there must be a connection between $Port_c$ and FU_b in that control step.
4. If in any control step s_k, FU_b writes to a register R_a and this register resides in memory M_j, then there must be a connection between FU_b and one of the write or read/write ports of M_j to which R_a is assigned.

Each port of a memory and each terminal of an FU are assigned integers m $(1 \leq m \leq p)$ and l $(1 \leq l \leq t)$ respectively, where p is the total number of ports of all memories and t is the total number of terminals of all FUs. A 0-1 connection array $I[m = 1...p][l = 1...t]$ holds the information about connection pattern between memory ports and FU terminals. The assignment of registers to memory ports in each control step is also recorded.

We have expanded the bipartite graph matching technique reported by Huang *et al.* [13] (where they apply it to the operation allocation only and did not allow the use of multiport memories) to allocate operations and interconnection simultaneously one control step at a time starting from the control step in which maximum resources are utilized. The vertices of the weighted bipartite graph represent the FUs and the operations in the control step. If function unit FU_j is capable of performing operation op_i, then there is an edge e_{ij} between corresponding vertices. The weight w'_{ij} associated with an edge e_{ij} is calculated according to:

$$w'_{ij} = \alpha_3 * \sum_{\forall k | op_k \in OP_j} C[i][k] + \beta_3 * L_{ij} \quad (2)$$

where OP_j holds the operations which FU_j can perform but have not been allocated so far and L_{ij} represents the minimum number of extra connections required in this control step for assigning op_i to FU_j taking into account the registers assignment to memory ports and the commutativity of operations to reduce the number of extra connection required. Then this problem is solved by applying the well-known Hungarian method [32]. Here we have merged the multiplexer minimization phase and tri-state buffers minimization phase into one phase to generate more reduction in interconnection, while previously these phase were treated separately [27] (as discussed in Section II). After each control step the connection array holding the connection pattern is updated depending upon the outcome of the assignment of operations to FUs. In this way data path is incrementally synthesized by taking into account the effect of already allocated FUs and connections and possible future assignments to produce good results.

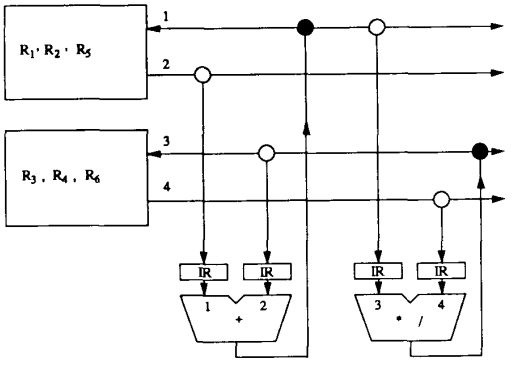

Figure 3: Data path for the example.

The final data path for the running example produced by this technique is given in Figure 3 and the registers assignment to memory ports is given in Figure 4. As mentioned earlier, with the above technique the final synthesized data path is more structured and uses minimum interconnection hardware (such as multiplexers, tri-state buffers).

CS#/#Pt		1	2	3	4
1	r	R_1	R_2	R_3	R_4
	w	R_1		R_4	
2	r	R_5	R_1	R_4	R_3
	w	R_5		R_6	
3	r	R_5	R_2	R_6	R_3
	w	R_2		R_4	

r: Read cycle w: Write cycle
CS#: Control step number
Pt#: Memory port number

Figure 4: Registers assignment to memory ports.

1: $R_3 \leftarrow R_1 + R_2$, $R_{12} \leftarrow R_1 * R_7$,

2: $R_5 \leftarrow R_3 - R_4$, $R_7 \leftarrow R_3 * R_6$,

3: $R_8 \leftarrow R_3 + R_5$, $R_{11} \leftarrow R_{10}/R_5$,

4: $R_{14} \leftarrow R_{11} + R_8$, $R_{15} \leftarrow R_{12} * R_9$,

5: $R_1 \leftarrow R_{14} - R_{13}$, $R_2 \leftarrow R_{11} * R_{15}$.

Figure 5: Code sequence for example 1.

5 Experimental Results

We have implemented the proposed heuristic in C on a SUN SPARCstation 1. We have tested our system using a number of well-known benchmarks for the application specific design. The results produced by our technique are better in most of cases or at least as good as previous 0-1 integer linear programming techniques. It is due to the fact that we have merged the operation and connection allocation tasks, which allows us to explore the design space more efficiently than previous approaches where they do not consider operation allocation and the connection allocation task is further divided into two subtask, multiplexer minimization and tri-state buffers minimization.

We present results from five different design examples and compare them with previous approaches. The comparison with Wilson *et al.* [25] is to demonstrate the quality of our proposed technique since it deals with multiport memories, where as the comparison to systems, which do not deal with multiport memories such as STAR [14], ADPS [29] and HAL [9], is to demonstrate that multiport memories can indeed significantly reduce the interconnection cost by using our proposed technique.

First example is a code sequence adopted from [25] as shown in Figure 5. By applying the technique discussed in this paper final synthesized data path for this example is given in Figure 6 and assignment of registers to memory ports is shown in Figure 7. Comparison of results with their approach is given in Table 1. It can be seen that the results produced by applying our approach requires less interconnect as compared to their approach.

The second is a fifth order wave digital elliptic filter adopted from [9]. This filter has been used by several synthesis systems, and it was a benchmark for the 1988 High-Level Synthesis Workshop. The available hardware consists of two adders which operate in a single control step, and a single pipelined multiplier operating in two control steps. It contains 26

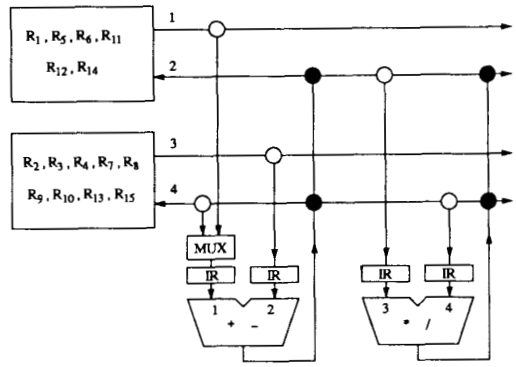

Figure 6: Data path for the example 1.

CS#/Pt#		1	2	3	4
1	r	R_1	R_1	R_2	R_7
	w		R_{12}		R_3
2	r		R_6	R_4	R_3
	w		R_5		R_7
3	r	R_5	R_5	R_3	R_{10}
	w		R_{11}		R_8
4	r	R_{11}	R_{12}	R_8	R_9
	w		R_{14}		R_{15}
5	r	R_{14}	R_{11}	R_{13}	R_{15}
	w		R_1		R_2

r: Read cycle w: Write cycle
CS#: Control step number
Pt#: Memory port number

Figure 7: Registers assignment to memory ports.

Table 1: Data Path allocation results of example 1

Systems	Our Technique	Wilson [25]
#ALUs	2	2
#MUX inputs	2	8
#Buses	4	n/a
#Tri-state Buf.	4	n/a
#Registers	15	15
#Multiport RAMs	2	2

Table 2: Data path allocation results of the wave filter.

Systems	Our Technique	STAR
#Multipliers	1	1
#ALUs	2	2
#MUX inputs	8	17
#Buses	5	5
#Tri-state Buf.	4	16
#ROMs	1	1
#Registers	14	13
#Control Steps	19	19
#Multiport RAMs	2	N/A

add operations and 8 multiply operations scheduled into 19 control steps. Coefficients are assumed to be stored in a ROM. Comparison of results with previous approaches is given in Table 2. The synthesized data path uses less interconnect hardware as compared to previous approaches.

Our third example is a sixth order elliptic band pass filter taken from [29]. It contains 10 add operations, 6 subtract operations and 12 multiply operations scheduled into 11 control steps. The available hardware consists of one multiplier, two multifunction ALUs; one of them can perform addition and subtraction, and the other is capable of performing addition, subtraction and multiplication. All FUs operate in a single control step. Comparison of results with ADPS [29] is shown in Table 3 with the assumption that coefficients are stored in a ROM. It can be seen that the result produced by using our technique requires much less interconnect hardware than previous approaches.

Our fourth example is Fast Discrete Cosine Transform (FDCT) adopted from [30]. It contains 50 operations, 17 add operations, 12 subtract operations and 21 multiply operations. It is scheduled into 12 control steps with two adders, two subtractors and two multipliers, all operating in single control step. For this example, 0-1 integer linear programming based techniques are not suitable because due to the large

Table 3: Data path allocation results of band pass filter.

Systems	Our Technique	ADPS
#Multipliers	1	1
#ALUs	2	2
#MUX inputs	10	27
#Buses	5	N/A
#Tri-state Buf.	6	N/A
#ROMs	1	N/A
#Registers	11	11
#Control Steps	11	11
#Multiport RAMs	2	N/A

Table 4: Data path allocation results of FDCT

Systems	Our Technique	Douglas[31]
#Multipliers	2	2
#Adders	2	2
#Subtractors	2	2
#MUX inputs	8	53
#Buses	6	N/A
#Tri-state Buf.	12	N/A
#ROMs	2	N/A
#Registers	9	33
#Control Steps	12	13
#Multiport RAMs	2	N/A

number of operations and variables generated, it results in a very large number of constraints, which can easily exceed the capability of many existing integer linear programming packages. Comparison of results with the only previous approach which reported results for this example by Douglas et al. [31], is given in Table 4. The synthesized data path uses much less interconnect hardware as compared to their approach.

Our fifth example is a differential equation adopted from HAL [9] and has been used in many synthesis systems. It contains 6 multiply operations, 2 add operations, 2 subtract operations and 1 comparison operation. It is scheduled into 8 control steps with one pipelined multiplier, operating in two control steps, and one multifunction ALU capable of performing addition, subtraction and comparison in a single control step [9]. The comparison of results with HAL [9] is given in Table 5 with the assumption that constants are applied directly.

6 Conclusions

We have presented a technique for the functional units and connection allocation subtasks in data path

Table 5: Data path allocation results of the differential equation.

Systems	Our Technique	HAL
#Multipliers	1	1
#ALUs	1	1
#MUX inputs	8	10
#Buses	3	N/A
#Tri-state Buf.	2	N/A
#Registers	5	5
#Control Steps	8	8
#Multiport RAMs	1	N/A

synthesis using multiport memories for application specific design to reduce the interconnect hardware. This problem is solved by a heuristic which modeled it as a bipartite graph matching problem and solved by the well-known Hungarian method [32]. Experimental results have shown that multiport memories can indeed significantly reduce the interconnection (such as buses, multiplexers and tri-state buffers) by using our technique. The proposed multiport memory based architecture is very effective for signal processing applications and for synthesis of application specific multiprocessor systems.

References

[1] M. C. McFarland, A. C. Parker, and R. Camposano, "The High-Level Synthesis of Digital Systems," *Proc. of IEEE*, vol. 78, no. 2, pp. 301-318, Feb. 1990.

[2] IEEE, "IEEE Standard VHDL Language Reference Manual," *IEEE std* 1076-1987, 1987.

[3] G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. of Research and Development*, vol. 34, no. 1, pp. 37-58, Jan. 1990.

[4] L. Ya-min and L. San-Li, "Optimizing Allocation of Multiport Register File For Parallelism in RISC and Its Implementation," *Microprocessing and Microprogramming*, vol. 26, pp. 113-117, 1989.

[5] J. R. Mick, "Introduction to IDT's four-port RAM," Application Note AN-45, Integrated Device Technology, 1989.

[6] H. Shinohara *et al.*, "A Flexible Multiport RAM Compiler For Data Path," *IEEE J. of Solid-State Circuits*, vol. 26, no. 3, Mar. 1991.

[7] K. Endo, T. Matsumura and J. Yamada, "Pipelined, Time-Sharing Access Technique For an Integrated Multiport Memory," *IEEE J. of Solid-State Circuits*, vol. 26, no. 4, Apr. 1991.

[8] C-J. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Path in Digital Systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-5, no. 3, pp. 379-395, Jul. 1986.

[9] P. G. Paulin and J. P. Knight, "Scheduling and Binding Algorithms For High-Level Synthesis," *23rd ACM/IEEE Design Automation Conference*, pp. 1-6, Jun. 1989.

[10] A. C. Parker, J. Pizarro and M. Mlinar, "MAHA: A Program For Data Path Synthesis," *23rd ACM/IEEE Design Automation Conference*, pp. 461-466, Jul. 1986.

[11] B. M. Pangrle, "SPLICER: A Heuristic Approach to Connectivity Binding," *25th ACM/IEEE Design Automation Conference*, pp. 536-541, Jun. 1988.

[12] B. S. Haroun and M. I. Elmasry, "Architectural Synthesis For DSP Silicon Compiler," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-8, no. 4, pp. 431-447, Apr. 1989.

[13] C. Y. Huang, Y. S. Chen, Y. L. Lin and Y. C. Hsu, "Data Path Allocation Based on Bipartite Weighted Matching," *27th ACM/IEEE Design Automation Conference*, pp.499-504, Jun. 1990.

[14] F. S. Tsai and Y. C. Hsu, "Data Path Construction and Refinement," *IEEE International Conference on Computer-Aided Design*, ICCAD-90, pp. 308-311, Nov. 1990.

[15] T. Y. Liu and Y. L. Lin, "FLORA: A Data Path Allocator Based on Branch-and-Bound Search," *Journal of VLSI Integration*, vol. 11, no. 1, pp. 43-66, Mar. 1991.

[16] L. Stok, "Interconnect Optimization For Multiprocessor Architectures," *CompEuro-90*, Tel Aviv, pp. 461-465, May 1990.

[17] L. Stok and V. D. Born, "EASY: Multiprocessor Architecture Optimization," *Proc. of International Workshop on Logic and Architecture Synthesis for Silicon Compilers*, ed. G. Saucier and P. M. Mclellan, Grenoble, pp. 313-328, May 1988.

[18] H. D. Man, J. Rabaey, P. Six and L. Claesen, "Cathedral-II: A Silicon Compiler For Digital Signal Processing," *IEEE Design and Test*, pp. 13-25, Dec. 1986.

[19] M. Potkonjak and J. Rabaey, "A Scheduling and Resource Allocation Algorithm For Hierarchical Signal Flow Graphs," *26th ACM/IEEE Design Automation Conference*, PP. 7-12, Jun. 1989.

[20] C. Ewering, "Automatic High Level Synthesis of Partitioned Buses," *IEEE International Conference on Computer-Aided Design*, ICCAD-90, pp. 304-307, Nov. 1990.

[21] B. M. Pangrle, "On the Complexity of Connectivity Binding," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. CAD-11, pp. 1460-1465, Nov. 1991.

[22] P. Marwedel, "The MIMOLA Design System: Tools For the Design of Digital Processors," *21st ACM/IEEE Design Automation Conference*, pp. 587-593, Jun. 1984.

[23] M. Balakrishnan, A. K. Majmudar, D. K. Banerji, J. G. Linders and J. C. Majithia, "Allocation of Multiport Memories in Data Path Synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. CAD-4, pp. 536-540, Apr. 1988.

[24] C. H. Chen and G. E. Sobelman, "Single-port/Multiport Memory Synthesis in Data Path Design," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp. 1110-1113, Sheraton Hotel, New Orleans, LA, May 1-3, 1990.

[25] T. C. Wilson, D. K. Banerji, J. C. Majithia and A. K. Majumdar, "Optimal Allocation of Multiport Memories in Data Path Synthesis," *32nd Midwest Symposium on Circuits and Systems*, Urbana IL, pp. 1070-1073, Aug. 1989.

[26] T. C. Wilson, D. K. Banerji, A. Basu, J. C. Majithia and A. K. Majumdar, "Port Assignment in Multiport Memories For Interconnect Minimization in Data Path Synthesis," *IFIP TC10/WG10.5 Workshop on Logic and Architecture Synthesis*, Paris, France, pp. 59-68, Jun. 1990.

[27] I. Ahmad and C. Y. Roger Chen, "Post-Processor For Data Path Synthesis Using Multiport Memories," *IEEE International Conference on Computer-Aided Design*, ICCAD-91, pp. 276-279, Nov. 1991.

[28] K. Kucukcakar and A. C. Parker, "Data Path Tradeoffs Using MABAL," *27th ACM/IEEE Design Automation Conference*, pp. 511-516, Jun. 1990.

[29] C. A. Papachristou and H. Konuk, "A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm," *27th ACM/IEEE Design Automation Conference*, pp. 77-83, Jun. 1990.

[30] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*, Academic Press, 1990.

[31] D. M. Grant and P. B. Denyer, "Memory, Control and Communications Synthesis for Scheduled Algorithms," *27th ACM/IEEE Design Automation Conference*, pp. 162-167, Jun. 1990.

[32] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, 1982.