A Robust Datapath Allocation Method For Realistic System Design*

Kyumyung Choi†
CAE, Semiconductor Business
Samsung Electronics Co.
Korea

Steven P. Levitan
Department of Electrical Engineering
University of Pittsburgh
Pittsburgh, PA 15261

ABSTRACT

We present a novel datapath allocation method. Key features of this method are its flexibility to handle accurate modeling of datapath units, use of direct objective functions, and the simultaneous optimization of all objective functions. The proposed method consists of a new binding model construction scheme and an optimization technique based on simulated annealing. Two new datapath allocation approaches have been devised based on the proposed method for two different problem environments. Experimental results show our approaches are very effective for both datapath allocation problems.

I. Introduction

Two major problems of existing datapath allocators are: First, they do not consider the interdependency among the sub-tasks of datapath allocation correctly because of the use of *sequential problem solving approaches*. Second, they use an *indirect objective functions* for their optimization criteria. This is because they are usually developed with restrictive assumptions such that their results must be modified with postprocessing steps in order to be used in diverse situations. Until these two problems are addressed, current algorithms developed for datapath allocation will not be useful in real design environments.

The majority of current datapath allocation algorithms are not suitable to solve these two problems. These include constructive approaches based on graph based algorithms, heuristic algorithms, and approaches based on ILP(integer linear programming). On the other hand, simulated annealing is a suitable technique to solve both of these problems. In the past, several datapath allocation algorithms used simulated annealing for their optimization methods[1, 2]. However, they did not solve the modeling problem correctly. Rather, they used simulated annealing with conventional modeling and indirect objective functions. In this paper, a datapath allocation method which can overcome these problems is proposed. It is capable of handling complicated models of datapath units for accurate synthesis results and able to optimize multiple objective functions concurrently.

Two new datapath allocation approaches have been devised based on the proposed method for two different complex problem environments. First, we develop a datapath allocation approach which incorporates accurate interconnection area and delay estimates, where floorplanning is tightly integrated into datapath allocation. This approach is presented in detail in [3]. Second, we develop a datapath allocation approach which handles registers, register files and multiport memories for data storage, and random and linear topologies for interconnection architectures using direct binding models and objective functions[4]. In this paper, we primarily focus on how we handle multiport memories, register files and buses using direct binding models and objective functions.

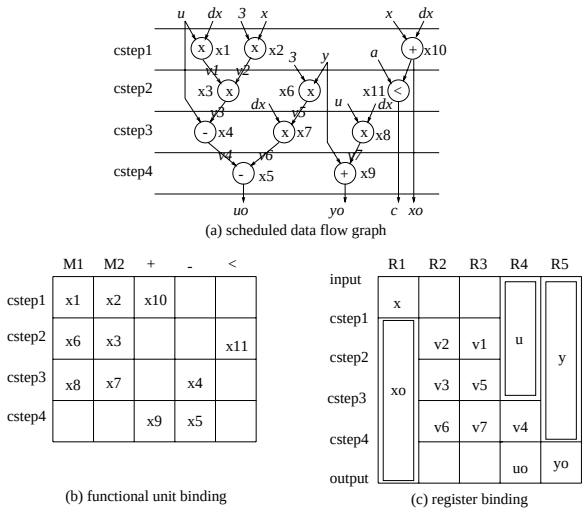

Figure 1: Conventional binding model

II. New Datapath Allocation Method

Our datapath allocation method is based on a new binding model construction scheme. The general binding model for datapath allocation has been described in several papers[1, 2]. Figure 1(b) and (c) show these binding models for datapath allocation of the standard differential equation example. We extend the idea of these two-dimensional binding models for functional unit and register allocation to all kinds of *primary datapath units*. These primary datapath units depend on the target architecture chosen as summarized in Table 1. That is, we construct a two-dimensional matrix, of which one axis are the primary datapath units and the other axis shows the control steps, for the binding of each primary datapath unit. Figure 2 shows these binding models for primary datapath units allocation, and will be explained in the next section.

Our datapath allocation method consists of two phases, initial allocation and allocation improvement. During initial al-

[1] This work was supported, in part, by the National Science Foundation under Grant MIP-9102721

[2] The author was supported, in part, by a Samsung Electronics Fellowship

Table 1: Primary and secondary datapath units

Memory architectures	Primary datapath units	Secondary datapath units
Register	Functional units, Registers	
Register file	Functional units, Register files	Registers
Multiport memory	Functional units, Ports	Registers
Interconnection architectures	**Primary datapath units**	**Secondary datapath units**
Random topology		Multiplexers, Wires, Routing channels, Interconnection delay
Linear topology	Buses	Multiplexers, Tri-state buffers

location, the initial bindings of primary datapath units are performed by efficient graph based algorithms[4]. These algorithms each produce individual near optimum solutions. As a result, the minimum numbers of primary datapath units required are obtained in almost cases. Once these numbers are determined, they are fixed during allocation improvement in order to prohibit the exploration of unnecessary regions of the solution space, and to reduce the relatively long computation time of allocation improvement.

During allocation improvement, new binding states are explored by moving or exchanging each element on our two- dimensional matrices for the bindings of primary datapath units. Accurate modeling of datapath units for specific target architecture and direct objective functions enable us to estimate the amount of improvement on *secondary datapath units* accurately during each move or exchange of elements. As a result, the optimization of secondary datapath units is performed. These secondary datapath units also depend on the target architecture chosen as summarized in Table 1.

Allocation improvement uses a modification of *simulated annealing* [5]. During allocation improvement, our scheme explores various binding alternatives and seeks an optimal solution by probabilistically exploring possible datapath structures. Because, at each step, the allocation improvement mode is chosen randomly among the possible operations which generate new binding states, our method does not sequentially optimize any of the sub-tasks during datapath allocation but performs the optimizations simultaneously. As a result, we avoid the sequential nature of other techniques which are sources of local optimum solutions.

The long computation time typical of simulated annealing methods is reduced by two schemes: First, we prohibit the system from exploring unnecessary areas of the solution space during allocation improvement by our two phase scheme. Second, we use a *dynamic cooling schedule* for the annealing process. The detailed explanation of dynamic cooling schedules for datapath allocation can be found in [4].

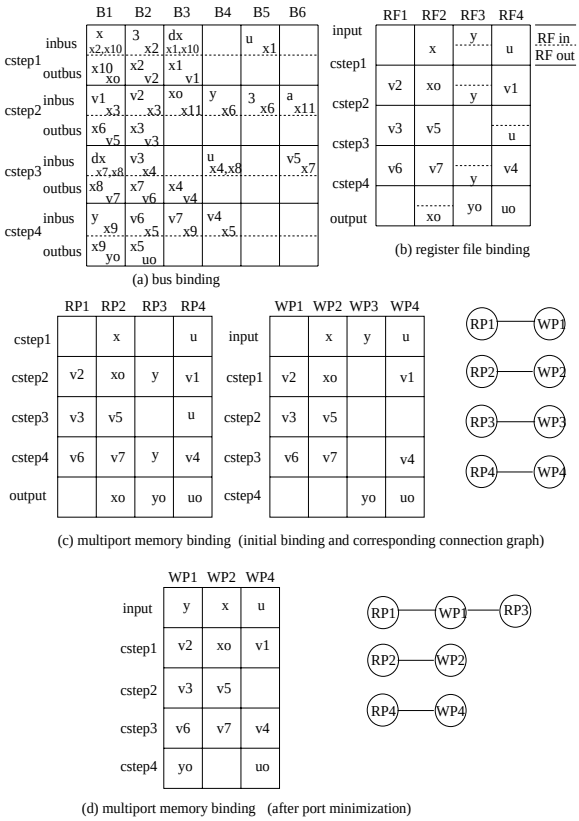

Figure 2: Binding models for primary datapath units

III. Allocation for Large Memory Architectures

In [4] datapath allocation algorithms for several diverse architectures are presented. In this section we focus on the application of the proposed binding model construction scheme for diverse memory and interconnection architectures. Also, a datapath allocation algorithm for multiport memory architecture is described.

A. New binding models

The target architectures for this approach are multiport memories and register files as well as registers for data storage. Here, a unique bus is connected to each port for multiport memory architectures, whereas both linear and random topologies are supported as the interconnection methods for register file and register architectures. For linear topology and register file architectures, any register file can be connected to any bus, as required by the allocation result.

We adopt the binding model construction scheme described in the previous section to these datapath allocation problem environments. We add binding models for buses, register files and multiport memories to the binding models for functional units and registers. Therefore, all hardware units in the datapath are treated as objects, and considered from the beginning of the allocation process for optimum final target architectures. A unique binding model for each primary datapath unit is necessary, because the binding criteria are different.

Figure 2(a) shows the bus binding model. The left-top operator (variable or constant) and right-bottom variable (operator) pairs in each square indicate that the data transfer

between them is implemented using the bus specified by the corresponding column. A two phase clocking scheme was used for this bus binding model. The binding model for a single phase clocking scheme can be drawn by a similar method. Figure 2(b) shows the register file binding model. The differences between this binding model and the register binding model of Figure 1(c) arise from the fact that we must consider a variable's *access time* for the binding of register files, whereas we must consider a variable's *life time* for the binding of registers. The *life time* of a variable is the time interval from its definition to its last use. The *access time* of a variable is the time when a variable is entered into or taken from a register file. Figure 2(c) and (d) show the multiport memory binding model. Because a group of registers can share a set of read ports and write ports in a multiport memory, whereas a group of registers can share only one read-only port and one write-only port in a register file, a register can be accessed through different ports in different control steps and a port can be shared by different registers in different control steps in a multiport memory. During datapath allocation, our system considers read ports and write ports separately and keeps their relation using a **connection graph**:

$$\begin{aligned} G &= \{V, E\} \\ V &= \{ \text{ read ports and write ports } \} \\ E &= \{ \text{ ports represented by vertices on a edge must reside in one memory module } \} \end{aligned}$$

When read/write ports are used for a target multiport memory, a read port and the corresponding write port are merged at the final step of datapath allocation. The reason why the special binding model is necessary for multiport memories is that the above mentioned sharing of ports by registers will cause a reduction in the interconnection area.

B. Algorithm for multiport memory architecture

Memory area depends on the number of ports and the number of registers in multiport memories. The minimum number of ports required is the maximum number of concurrently accessed variables in any control step. The interconnection area consists of buses, multiplexers and tri-state buffers. Because we assume a unique bus is connected to each port, the number of buses depends on the number of ports. The interconnection cost during the allocation improvement is:

$$\begin{aligned} interconnection\ unit\ cost &= P_m \times \#multiplexer\ inputs \\ &+ P_b \times \#tri-state\ buffer \end{aligned} \quad (1)$$

where P_m, P_b are weighting values which indicate the relative importance of each objective.

Initial allocation

The initial assignment of variables to ports is performed by the following procedure:

1. Assignment of variables to read and write ports.
2. Minimization of read and write ports.
3. Partitioning of ports to different multiport memories.

In step 1, we set the restriction that a write access and following read accesses of a variable are assigned to a write port and a read port. Then, we can generate the compatibility graph using the variable's access time, and the assignment of variables to ports is performed by the *clique partitioning algorithm*[6]. Also, the connection graph is initially generated, where an edge exists between a write port and the corresponding read port, as shown in Figure 2(c), in order to keep information that those ports must reside in one memory module.

In step 2, the number of read ports and write ports are minimized separately. This is performed by merging ports when there is no access conflict. For example, ports WP1 and WP3 on Figure 2(c) are merged to port WP1 on Figure 2(d). The connection graph is updated when ports are merged.

In step 3, the maximum number of permitted ports for each of the target multiport memories is examined, and ports are partitioned to different multiport memories. For example, if the architecture allows multiport memories to have 4R-3W (4 Read Ports and 3 Write Ports), Figure 2(c) and (d) are synthesized by one multiport memory. If the architecture allows 2R-2W (2 Read Ports and 2 Write Ports), it may be synthesized by two multiport memories, and {RP1, RP3, WP1} {RP2, RP4, WP2, WP4} is a possible partition.

Allocation improvement

During allocation improvement, a new binding state can be generated by any of these operations:

- move or exchange a functional unit binding
- swap the input binding of a functional unit in a commutative operation
- move or exchange a port assignment of a multiport memory

An important point to note is that we must maintain the rule that a variable resides in one memory module, when an exchange has happened between different memory modules.

Two different cost functions are considered during variable movement. If a move happens within the same multiport memory, only the interconnection unit cost function, equation 1, is considered. But, if a move happens between different multiport memories, the following memory unit cost must be added to the interconnection unit cost.

$$memory\ unit\ cost = P_r \times \#register \quad (2)$$

where P_r is weighting value which indicates the relative importance of this objective.

IV. Experimental Results

These datapath allocation algorithms have been implemented in a system called *MandM*. Table 2 shows the allocation results of the elliptical wave filter (EWF) example[7] for multiport memory architectures. The 19 control steps scheduling result[7] was used as an input. Two adders, one pipelined multiplier and a two phase clocking scheme were used[8, 9]. Table 2(a) shows that *MandM* produced better results than the other systems for almost all hardware resources except the number of tri-state buffers. Table 2(b) shows how the difference of port restrictions affects the allocation results, especially

Table 2: Allocation results of EWF for multiport memory

System	*MandM*	*Kim&Liu's*	*MAP*
Multiport module	2 (1 3R/W, 1 2R/W)	2 (1 3R/W, 1 2R/W)	2 (not specified)
Port	5	5	5
Bus	5	5	5
Register	10	12	14
Mux input	7	9	10
Tri-state buffer	7	5	5

(a) Comparison with other system results

Port restrict	Multiport module	Reg	Mux	Mux input	Tri-state buffer
1R/W	5 (5 1R/W)	13	4	13	6
2R/W	3 (2 2R/W, 1 1R/W)	10	4	11	6
3R/W	2 (1 3R/W, 1 2R/W)	10	3	7	7
5R/W	1 (1 5R/W)	9	3	6	6

(b) Result variation due to port restrictions

the interconnection cost metrics. We can see that the necessary number of registers, multiplexers, multiplexer inputs and tri-state buffers are reduced when the multiport memory modules permit more ports. However, if datapath allocation was performed sequentially as in [8, 9], where registers and interconnection units are allocated first, then registers are grouped into multiport memories (or these tasks are performed in reverse order), this improvement would not be possible.

Table 3 shows the allocation results of the EWF example for register file(RF) and linear topology architectures. A two phase clocking scheme was assumed. Table 3(a) compares results with the results of *SPAID*[10]. We used a result of our scheduler as the input. *MandM* produced better results than *SPAID* for all the different designs. Notably, *MandM* needed a far fewer number of buses, registers, and multiplexer inputs. Table 3(b) compares results with the results of *STAR*[11]. The 17 control steps scheduling result from [11] was used as an input. *MandM* produced better results than *STAR* for most hardware resources except the necessary number of register files. The reason for the improved results of *MandM* is that, in *MandM*, all allocation sub-tasks are performed simultaneously. Therefore, it is possible to optimize extensively the interconnection units which are highly interdependent with the functional units and memory units. The run times of these examples ranged from 6 to 12 CPU minutes on a SUN Sparcstation.

V. Conclusion

In this paper, we presented a novel method for datapath allocation, which is based on the need for accurate modeling of datapath units, the direct consideration of all objective functions, and the simultaneous optimization of each objective function in the binding model. Two new approaches were developed based on the proposed method for two different datapath allocation problems. The approach which was the focus of this paper can handle diverse memory and interconnection architectures in datapath allocation. Experimental results show this method is very effective not only for large embedded memory architectures, but also for a large variety of other target architectures.

Table 3: Allocation results of EWF for RF - linear topology

System	Steps	FU *	FU +	RF	Bus	Reg	Mux	Mux input	Tri-state buffer
MandM	17	3	3	7	4	10	6	17	11
SPAID		2	3	6	6	17	n/a	26	n/a
MandM	17	2P	3	7	5	11	7	21	13
SPAID		2P	3	6	6	17	n/a	26	n/a
MandM	19	2	2	5	4	10	4	11	11
MandM	19	1P	2	5	3	10	5	12	8
SPAID		1P	2	5	5	19	n/a	17	n/a
MandM	21	1	2	5	5	10	5	12	12
SPAID		1	2	6	6	19	n/a	19	n/a

(a) Comparison with *SPAID*

System	Steps	FU *	FU +	RF	Bus	Reg	Mux	Mux input	Tri-state buffer
MandM	17	2	2	5	4	9	4	12	13
STAR	17	2	2	3	5	11	n/a	16	13

(b) Comparison with *STAR*

References

[1] S. Devadas and R. Newton, "Algorithms for Hardware Allocation in Data Path Synthesis," *IEEE Trans. on Computer-Aided Design*, vol.8, no.7, pp.768-781, Jul. 1989.

[2] G. Krishnamoorthy and J. A. Nestor, "Data Path Allocation using an Extended Binding Model," *Proc. 29th DAC*, pp.279-284, 1992.

[3] K. Choi and S. P. Levitan, "Exploration of Area and Performance Optimized Datapath Design Using Realistic Cost Metrics," *Proc. ISCAS-95*, pp.1049-1052, 1995.

[4] K. Choi, "A Robust Architectural Synthesis Method for Realistic System Design," *Ph.D. dissertation*, Dept. of Electrical Engineering, Univ. of Pittsburgh, 1995.

[5] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol.220, no.4598, pp.671-680, May 1983.

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

[7] P. G. Paulin and J. P. Knight, "High-Level Synthesis Benchmark Results using a Global Scheduling Algorithm," in *Logic and Architecture Synthesis for Silicon Compilers*, North-Holland, pp.211-228, 1989.

[8] I. Ahmad and C. Y. Roger Chen, "Post-Processor For Data Path Synthesis Using Multiport Memories," *Proc. ICCAD 91*, pp.276-279, 1991.

[9] T. Kim and C. L. Liu, "Utilization of Multiport Memories in Data Path Synthesis," *Proc. 30th DAC*, pp.298-302, 1993.

[10] B. S. Haroun and M. I. Elmasry, "Architectural Synthesis for DSP Silicon Compilers," *IEEE Trans. on Computer-Aided Design*, vol.8, no.4, pp.431-447, Apr. 1989.

[11] F. S. Tsai and Y. C. Hsu, "STAR: An Automatic Data Path Allocator," *IEEE Trans. on Computer-Aided Design*, vol.11, no.9, pp.1053-1064, Sep. 1992.