A Hardware/Software Codesign Method for Pipelined Instruction Set Processor Using Adaptive Database

Nguyen Ngoc Binh, Masaharu Imai, Akichika Shiomi, and Nobuyuki Hikichi*

Department of Information and Computer Sciences
Toyohashi University of Technology
Toyohashi, 441 Japan
Tel: +81-532-47-0111
Fax: +81-532-48-9079

* Software Technology Department
Software Research Associates, Inc.
Tokyo, 170 Japan
Tel: +81-3-3942-4405
Fax: +81-3-3942-4416

e-mail: peas@imaisun.tutics.tut.ac.jp

Abstract— This paper proposes a new method to design an optimal pipelined instruction set processor using a formal HW/SW codesign methodology. First, a HW/SW partitioning algorithm for selecting an optimal pipelined architecture is introduced briefly. Then, an adaptive database approach is presented that enables to enhance the optimality of the design through very accurate estimation of the performance of a pipelined ASIP in HW/SW partitioning. The experimental results show that the proposed methods are effective and efficient.

I. Introduction

One of the key issues in the ASIP (Application Specific Integrated Processor) design is the optimization of the instruction set and CPU architectures of ASIPs. Because the performance of an ASIP is heavily affected by the choices of these architectures, it is essential to select the optimum instruction set and CPU architectures with the maximum performance under the constraints of chip area and power consumption for example. These designs used to be done by computer architects taking advantage of their experience and intuition, but in the ideal ASIP development environment, they should be automatically performed by the system.

In order to realize the ideal ASIP development environment, an integrated design system for ASIPs named **PEAS-I** (**P**ractical **E**nvironment for **AS**IP development – type I) was proposed and its prototype has been developed [1]. The PEAS-I system accepts a set of application programs written in C language and their associated data and some design constraints. The system profiles the application programs and generates the CPU core design as well as the application program development environment, such as compiler and simulator. The PEAS-I system employs a formal method to synthesize an optimal instruction set processor by solving Instruction set implementation Method Selection Problems (IMSP) [2]. IMSP-2 (IMSP type 2) [2] and IMSP-2P (for Pipeline) [3] are resource-constrained design with the consideration of resource sharing.

The HW/SW partitioning addressed in the previous PEAS-I is based on the fixed expected execution cycles of software implemented operations, which have been applied to any application program so far. In practice, however, their variation depends on input data of the operation. Therefore, it is difficult to estimate the performance accurately. As a result, the generated design might not be optimal becasuse of the misselection of Functional Units (FUs). In this paper we propose an approach to enhance the optimality of selected architecture with an accurate performance estimation. This is one of the most distinguished features of PEAS-I compared to other HW/SW codesign systems such as ASIA [4] and CASTLE [5].

The architecture of an ASIP synthesized by the PEAS-I system is based on the GNU C Compiler (GCC) abstract machine model [6]. The PEAS-I CPU is pipelined architecture with four stages: IF (Instruction Fetch & decode), EX (EXecution), MEM (MEMory access) and WR (Write back to Register file), respectively. Please refer to Ref. [7] (or [3]) for more detail. The essence in IMSP algorithms is selection of FUs to be added to minimum HW components called 'Kernel' to achieve the design goal for given design constraints (in particular, the highest performance of a designed ASIP for gate count and power consumption constraints).

The GCC Register-Transfer Language (RTL) operations are divided into **primitive** and **basic** operations. The primitive operations contain the minimum operations that can be included in the ASIP chip so that it can execute any C program. The primitive operations should be implemented in HW 'Kernel', which consists of an ALU, a one-bit shifter, and a register file. The basic operations contain other C operators that are not included in the primitive operations. A basic operations can be implemented using some HW choice (such as fast or slow hardware modules) or using a software subroutine (run-time

routine) that uses primitive operations and some other basic operations.

The remainder of the paper is organized as follows: Section 2 presents a HW/SW partitioning problem and algorithm briefly. Section 3 describes the adaptive database approach. Section 4 demonstrates the effectiveness and efficiency of the proposed method. The last section gives conclusions and future work.

II. HW/SW Partitioning Algorithm

IMSP-2 does not consider pipeline characteristics such as pipelined FUs and pipeline hazards. Hence, we cannot use it to design an optimal pipelined ASIP. In particular, it cannot deal with the pipeline execution cycles of the ASIP accurately. In this section, we describes a method to evaluate pipeline hazards accurately when all the basic operations are implemented in HW.

A. Definitions and Notations

The HW/SW partitioning problem in the current version of PEAS-I is defined as follows [2]:

For a whole set of all candidate instructions representing a given application domain, select a set of implementation methods which maximizes the performance of the CPU under the constraints of chip area and power consumption, taking into account the functional module sharing relation among instructions.

In order to formalize IMSP-2P we need the following definitions and notations:

(1) "n" denotes the total number of basic operations to be considered.

(2) "x_i" denotes an implementation method that realizes operation $\#i$, where x_i may be HW choice or SW, $0 \leq i \leq n$. Then $X = (x_0, x_1, ..., x_n)$ is a combination of implementation methods to be considered. (x_0 denotes HW 'Kernel' to implement all primitive operations and SW modules).

(3) "$a(x_i)$" and "$p(x_i)$" denote the area and power consumption required for implementation method x_i respectively, where $0 \leq i \leq n$.

(4) "A_max" and "P_max" denote the available chip area and the maximum power consumption allowable for the computing modules in the ASIP chip.

(5) "N" denotes the total number of basic blocks in the application program's GCC RTL code.

(6) "$t(B_j, X)$" denotes the execution cycles needed to execute basic block B_j using a combination of implementation methods X, where $1 \leq j \leq N$.

(7) "F_j" denotes the execution frequency count of basic block B_j in the given set of application programs, where $1 \leq j \leq N$.

(8) "c_j" denotes clock cycles needed to define control (e.g., branch delay) from block B_j to another one, where $1 \leq j \leq N$. Here, it is assumed that all branches are taken and delay slot scheduling is not performed.

(9) "b" denotes execution cycles reduced by un-taken branches in execution of the given application program.

B. IMSP-2P Formalization

Find a solution vector

$$\mathrm{X} = (x_0, x_1, \cdots, x_n)$$

which minimizes the objective function:

$$T(X) = \sum_{j=1}^{N} \{F_j \times (t(B_j, X) + c_j)\} - b \ , \qquad (1)$$

subject to the constraints

$$\sum_{x_i \in S} a(x_i) \leq A\_max, \qquad (2)$$

$$\sum_{x_i \in S} p(x_i) \leq P\_max, \qquad (3)$$

where

$$S = \bigcup_{i=0}^{n} \{x_i\} \qquad (4)$$

C. Outline of the IMSP-2P Solver

The key point in computing Eq.(1) is how to get $t(B_j, X)$. We have developed a HW/SW partitioning-oriented pipeline scheduling algorithm [8] to estimate $t(B_j, X)$ for basic block B_j under configuration X. The pipeline control hazards are addressed in introducing the coefficients c_j. Note that the number of clock cycles due to control hazards is equal to $\sum_{j=1}^{N}(F_j \times c_j) - b$. The pipeline scheduling algorithm detects and resolves all types of data hazards and structural hazards by ensuring that no more than one instruction can be issued or completed at each control step.

The IMSP-2P can also be solved using the branch-and-bound method as IMSP-2 can. One of the most important issues in solving problems efficiently by this method is to find a tight lower-bound function to prune as many non-optimum solutions as early as possible. For more detail please refer to Ref's. [3] and [7]. Note that coefficient b was not introduced in the previous IMSP-2P formalization in Ref. [3]. Please note that the module sharing capability and heuristic reordering are the same as in the IMSP-2 solver.

The input to the IMSP-2P solver includes information from the Application Program Analyzer (APA), in the

previous section, and a **module information database** of HW/SW modules for implementing basic operations [7]. The output of the IMSP-2P solver includes the optimum implementation method of each basic operation and pipelined schedules of basic blocks. The instruction set of the designed ASIP will include the primitive operations as default and those basic operations that are selected to be implemented in HW. The algorithm automatically integrates the functional module sharing basic operations into one HW module whenever possible.

III. Adaptive Database Approach

So far it was assumed that the execution cycles of a SW implemented operation is fixed to the expected execution cycles for any application program. Estimation errors for designs with SW implementations range from 15% to 30% for the IMSP solvers as shown in the next section. As a result, the selected architecture as well as the instruction set may not be optimal. In this section we propose a method to generate an adaptive database of expected execution cycles of SW implemented operations for a given application program with associated input data.

A. Getting the Input Data of Each Basic Operation

Using the application program analyzer (APA) of the PEAS-I system to profile the given application program we get a sequence of all input data of each basic operation through running the program. We denote G_i as a sequence of all input data of basic operation $\#i$ $(1 \leq i \leq n)$. Let G_i be

$$G_i \;=\; (g_i^{(1)}, g_i^{(2)}, ..., g_i^{(f_i)}) \;,$$

where f_i is the number of elements in G_i as the same execution frequency count of basic operation $\#i$ in execution of the application program. Note that $g_i^{(j)}$ can be a tuple depending on the number of operands of basic operation $\#i$. Then, the number of cycles needed to execute SW implemented basic operation $\#i$ with input data $g_i^{(j)}$ is denoted as $\tau_i(g_i^{(j)})$.

B. Evaluation of Execution Cycles

We can use the run-time subroutine for each basic operation $\#i$ and run it with each $g_i^{(j)}$ to get $\tau_i(g_i^{(j)})$ by using the PEAS-I simulator, as performed in Ref. [9]. However, the time needed for getting all $\tau_i(g_i^{(j)})$ is about as same as the time for running the APA with all operations to be implemented in SW. This method is very time consuming. In order to reduce the computation time, we introduce formulae to estimate execution cycles of SW modules for basic operations with given input data.

Let A and B be operands in the instruction of the form '$OP \quad C,\ A,\ B$' with the meaning $C \leftarrow A\ OP\ B$, where OP is an operation. We define

$$s(A) = \begin{cases} 1, & \text{if } A < 0\ , \\ 0, & otherwise, \end{cases}$$

and $s(B)$ is defined similarly. $left(B)$ denotes the position of the leftmost '1' in the binary presentation of B (counting from the right on the left). $one(B)$ denotes the number of 1's in the binary presentation of B. Investigating assembly codes of SW modules in the current PEAS-I system, we have found the following formulae. Please note that these formulae depend on the algorithms used in run-time routines for basic operations.

$$\tau_{mul}(A,B) \;=\; 26 + s(A) + s(B) + 7 * \max\{0, left(|B|) - 1\} \tag{5}$$

$$\tau_{umul}(A,B) \;=\; 13 + 7 * \max\{0, left(B) - 1\} \tag{6}$$

$$\tau_{div}(A,B) \;=\; 278 + s(A) + s(B) + one(|A|/|B|) \tag{7}$$

$$\tau_{udiv}(A,B) \;=\; 263 + one(A/B) \tag{8}$$

$$\tau_{mod}(A,B) \;=\; 278 + s(A) + s(B) \tag{9}$$

$$\tau_{umod}(A,B) \;=\; 263 \tag{10}$$

$$\tau_{ashr}(A,B) \;=\; \tau_{ashl}(A,B) \;=\; \tau_{lshl}(A,B) \;=\; \tau_{lshr}(A,B) \;=\; 20 + \sum_{j, BIT_j(B)=1} (2^{j+1} - 1) \tag{11}$$

$$\tau_{extendhi}(A) \;=\; 10 + BIT_{15}(A) \tag{12}$$

$$\tau_{extendqi}(A) \;=\; 8 + 3 * BIT_{15}(A) \tag{13}$$

$$\tau_{z\_extendhi}(A) \;=\; 2 \tag{14}$$

$$\tau_{z\_extendqi}(A) \;=\; 1 \tag{15}$$

where $BIT_j(B)$ is the value of bit j in the binary presentation of B (bit 0 is the rightmost bit).

C. Evaluation of Average Execution Cycles

The average execution cycles of SW implemented basic operation $\#i$ for the given application program are defined as follows:

$$\overline{\tau}(\#i) \;=\; \lfloor \frac{\sum_{j=1}^{f_i} \tau_i(g_i^{(j)})}{f_i} \rfloor \qquad (16)$$

These values replace the old values in the database to run the IMSP solvers. The revised database is adaptive to the application program.

In order to reduce the computation time, we define U_i as a set of different (unique) elements in G_i as follows:

$$U_i \;= \bigcup_{j=1}^{f_i} \{g_i^{(j)}\} \qquad (17)$$

and define $k_i(u)$ as frequency count of u in G_i. Then, Eq.(16) can be rewritten as follows:

$$\overline{\tau}(\#i) \;=\; \lfloor \frac{\sum_{u \in U_i} k_i(u) \times \tau_i(u)}{f_i} \rfloor \qquad (18)$$

Note that $\sum_{u \in U_i} k_i(u) = f_i$ and $|U_i|$ is much smaller than f_i as shown later in Table 2.

IV. Experiments and Results

The IMSP-2P algorithm with the adaptive database approach has been implemented in C and examined on a workstation. A set of sample programs has been performed to evaluate the effectiveness and efficiency of the algorithm.

A. Sample Programs

The sample programs used in the experiments are as follows:
(1) ESS : Equation System Solver program, which solves a system of two linear equations using Cramer's rule.
(2) IMC : Inverse Matrix Calculator program that computes the inverse of a non-singular 3×3 matrix using Cramer's rule.
(3) *diffeq* : A program for solving a second order differential equation from Ref. [10].

These sample programs were fed to APA of the PEAS-I system. The code optimization was performed by the GCC [6].

B. Module Library

We use a module library with both non-pipelined FUs and pipelined FUs such as multipliers and dividers generated using a high-level synthesis system called PARTHENON [11] and cell library VSC470.lib (0.8μmCMOS) from VLSI Technology, Inc. A 16 MHz clock was assumed in the design of HW modules. The database contains 14 basic operations, each of them has different implementation methods ranging from 2 to 11. The number of leaf nodes in the search tree is of $11^2 \times 7^4 \times 2^8 = 74,373,376$. The whole search tree ranges from 8.2×10^7 to 1.5×10^8 nodes depending on the order of variables (x_i's) to be examined. Therefore, it is necessary to have an efficient strategy to explore the search space to get an optimal solution in a reasonable time.

TABLE I
Expected execution cycles of SW implementations

Basic Operation	*#Cycles*	*Basic Operation*	*#Cycles*
div	216	mul	96
udiv	202	umul	91
mod	214	trunchi	2
umod	201	truncqi	1
ashl	31	extendhi	10
ashr	31	extendqi	9
lshl	31	z_extendhi	2
lshr	31	z_extendqi	1

Part of the HW module information database used in the experiments is described in Ref's. [3] and [7]. Another part of the module database contains the expected numbers of execution cycles for each basic operation to be implemented by software subroutine. This part is shown in Table 1, where the values were obtained as the average of execution cycles in running the corresponding subroutine with random input data. So far, these values are considered as fixed (therefore, non-adaptive) to any application program in the HW/SW partitioning process.

C. Effectiveness

We demonstrate the IMSP-2P using the adaptive database approach for the IMC, ESS and *diffeq* programs.

The IMSP-2P selected the optimum partitioning for different values of area constraint. The power consumption was ignored to simplify the experimental cases.

FUs needed to implement basic operations for these sample programs are a multiplier, a divider, a barrel-arithmetic shifter, an extender and so on, where the multiplier and the divider can be pipelined or non-pipelined.

Figures 1 and 2 show estimation errors by IMSP-2 and IMSP-2P, respectively, using the fixed (non-adaptive) database. Note that IMSP-2P can estimate the execution cycles much more accurately than IMSP-2. That is, estimation errors are below 1.3% for designs with gate count constraints exceeding 22 Kgates, where all operations can be implemented in HW. On the other hand, the estimation errors by IMSP-2 range from 5% to 20% because of

TABLE II
GENERATION OF ADAPTIVE DATABASE

i	*Basic Operation*	IMC *program*			ESS *program*			*diffeq program*		
		f_i	$\|U_i\|$	$\overline{\tau}(\#i)$	f_i	$\|U_i\|$	$\overline{\tau}(\#i)$	f_i	$\|U_i\|$	$\overline{\tau}(\#i)$
1	mul	720	562	52	180	153	52	868	852	48
2	div	653	314	279	377	130	280	147	117	283
3	mod	383	46	278	317	70	278	147	117	278
4	ashl	1648	48	21	1226	42	21	829	85	21
5	extendqi	1080	36	8	1809	42	8	409	66	8
6	z_extendqi	1193	37	1	1825	48	1	512	72	1

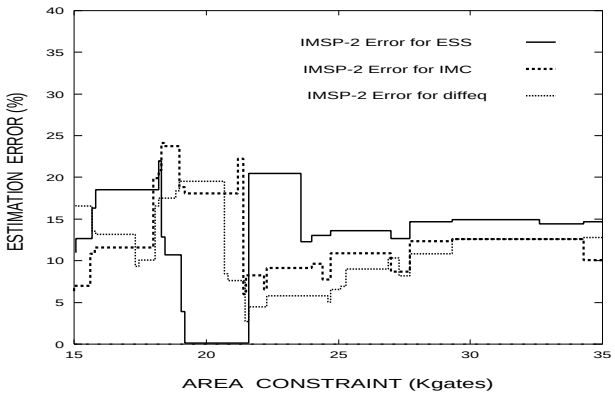

Fig. 1. Estimation errors by IMSP-2

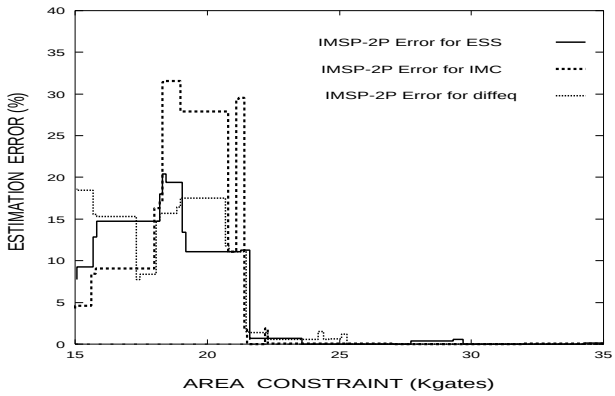

Fig. 2. Estimation Errors by IMSP-2P

not taking into account the pipeline characteristics. In addition, for designs with SW inplementations, the execution cycle estimation reported by IMSP-2P will contain some error up to 32% as in the case of IMSP-2 besause the execution cycles of SW modules depend on the application program.

The analyzed results as well as statistics of the experiment for generating adaptive database are shown in Table 2, where there are 6 basic operation types met in these application programs. Note that the number of unique input data (i.e, $|U_i|$) is much smaller than the frequency count (i.e., f_i) for basic operations #3 – #6.

Using the generated adaptive database the IMSP-2P solver selected the first optimal solutions exactly, whereas IMSP-2 may not. Note that the performance estimation errors by the IMSP-2P solver are below 1.1%, 2.7%, 1.3% for any area constraint as shown in Figure 3 for IMC, ESS, *diffeq*, respectively.

D. Efficiency

The proposed method is quite efficient. Table 3 shows the SPARCstation 10 (SS-10) CPU time in seconds for performing each experiment. The CPU time needed for APA to profiles an application program is below 20s. Then, an adaptive database can be generated within 0.1s. An optimal ASIP architecture is decided by the so-called Architecture Information Generator (AIG) with the IMSP

TABLE III
CPU TIME IN SECONDS

CPU* time for	ESS	IMC	*diffeq*
APA	19.7	16.5	6.2
Adaptive DB	0.1	0.1	0.1
AIG (IMSP-2P)	2.6	2.3	2.1
Total	22.4	18.9	8.4

* SPARCstation 10

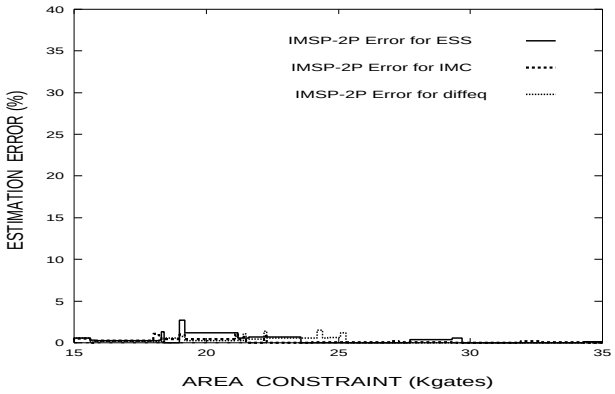

Fig. 3. Estimation errors by IMSP-2P using adaptive database

solvers. For a given area constraint, AIG with the IMSP-2P solver has taken CPU time of 2.6s, 2.3s, and 2.1s, which is average for IMC, ESS, and *diffeq*, respectively.

Thus, PEAS-I accepts as input a C written application program and design constraints. Using the proposed method, we can get an optimal pipelined ASIP architecture within a few seconds (below 25s) as shown in Table 3 for the total time.

V. Conclusion and Future Work

We have proposed an efficient method to design an optimal pipelined instruction set processor in the PEAS-I system. The method with IMSP-2P is introduced as a HW/SW codesign problem formalization and as an extension of IMSP-2. The IMSP-2P selects the implementation method of the operations that implement a pipelined ASIP instruction set so that the performance goal is maximized under the given gate count and power consumption constraints. Then, we proposed an adaptive database approach to reduce the execution cycle estimation error. The effectiveness of the IMSP-2P algorithm was demonstrated through design examples. The method estimates the performance of a designed pipelined ASIP accurately for most designs. The algorithm is so efficient that the optimal solution was obtained within a few seconds on a conventional workstation.

Using the adaptive database and introducing the new formula for execution cycle estimation in IMSP-2P we enhance the optimality of the solutions and reduce performance estimation error remarkably. Experimental results show that IMSP-2P gave the first optimal solution for any gate count constraint with performance estimation errors of below a few %. The primary goal of selecting the first optimal architecture with low estimation error has been achieved in the PEAS-I HW/SW Codesign system.

Our future work includes the development of a HW/SW partitioning algorithm for pipelined ASIP design with the least gate count under a given execution cycle and power consumption constraints, as well as for the design with the lowest power consumption under gate count and execution cycle constraints.

Acknowledgments

Authors would like to express their thanks to NTT Communication Science Laboratories, VLSI Technology, Inc., Science Create, Co. Ltd., Japan, for their kind assistance. This research is supported in part by Grant-in-Aid for Scientific Research No's. 07558038 and 07680353 from the Ministry of Education, Science and Culture, Japan.

References

[1] Sato, J., Alomary, A., Honma, Y., Nakata, T., Shiomi, A., Hikichi, N., and Imai, M.: " PEAS-I: A Hardware/Software Codesign System for ASIP Development," *IEICE Trans. A*, Japan, Vol. E77-A, No. 3, pp. 483 – 491, Mar. 1994.

[2] Alomary, A., Nakata, T., Honma, Y., Imai, M., and Hikichi, N.: " An ASIP Instruction set Optimization Algorithm with Functional Module Sharing Constraint," *Proc. of ICCAD'93*, pp. 526 – 532, 1993.

[3] Binh, N.N., Imai, M., Shiomi, A., Hikichi, N. and Sato, J., "Extension of Instruction Set Design for Pipelined Architecture in PEAS-I System," *Proc. of DA Symposium*, IPSJ, Japan, Vol. 94, No. 5, pp. 211 – 216, Aug. 1994.

[4] Huang, I-J. and Despain, A.M., "Synthesis of Instruction Sets for Pipelined Microprocessors,"*Proc. of DAC'94*, pp. 5 – 11, 1994.

[5] Wilberg, J., et al., "Design Flow for Hardware/Software Cosynthesis of a Video Compression System," *Proc. of Codes/CASHE '94*, Grenoble, France, 1994.

[6] Stallman, R.: *Using and Porting GNU C Compiler*, Free Software Foundation, Version 1.40, 1991.

[7] Binh, N.N., Imai, M., Shiomi, A., and Hikichi, N., "A Hardware/Software Partitioning Algorithm for Pipelined Instruction Set Processor," *Proc. of EURO-DAC'95*, 1995. (to appear)

[8] Binh, N.N., Imai, M., Shiomi, A., Hikichi, N., Honma, Y., and Sato, J., "An Efficient Scheduling Algorithm for Piplined Instruction Set Processor and Its Application to ASIP Hardware/Software Codesign," *IEICE Trans. A*, Vol. E78-A, No. 3, pp. 353 – 362, Mar. 1995.

[9] Binh, N.N., Imai, M., Shiomi, A., and Hikichi, N., "Optimal Instruction Set Design through Accurate Execution Cycle Estimation of Software Modules," *Proc. of the 8th Karuizawa Workshop on Circuits and Systems*, IEICE, Karuizawa, Japan, pp. 79 – 84, Apr. 1995.

[10] Paulin, P.G., Knight, J.P., and Girczyc, E.F.: "HAL: A Multi-paradigm Approach to Automatic Data Path Synthesis," *Proc. of DAC'23*, pp. 263 – 270, 1986.

[11] Nakamura, Y., Oguri, K., Nagoya, A.: "Synthesis from Pure Behavioral Descriptions," *in High-Level VLSI Synthesis*, Camposano, R., and Wolf, W., eds, pp. 205 – 229, Kluwer Academic Publishers, 1991.