Efficient Algorithms for Acceptable Design Exploration *

Chantana Chantrapornchai
Department of Mathematics
Silpakorn University
Nakorn Pathom 73000
Thailand
ctana@su.ac.th

Edwin H.-M. Sha
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
esha@cse.nd.edu

Xiaobo (Sharon) Hu
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
shu@cse.nd.edu

ABSTRACT

In this paper, we present an efficient approach to find effective module selections under resource, latency, and power constraints. The framework contains two phases: choosing a resource configuration, and determining a module binding for each resource. The first phase applies *inclusion scheduling* to estimate generic resources required. In the second phase, *module utility* measurement is used to determine module selections. A heuristic which perturbs module utility values until they lead to superior selections according to design objectives are also proposed. The experiments on well-known benchmarks show the effectiveness of the approach when comparing the obtained module selections with the results from enumerating all module selections, as well as MSSR and PSGA.

Keywords: Inclusion scheduling, Module selections, Design exploration, Module Utility, Acceptable designs

1. INTRODUCTION

In high-level synthesis, module selection, resource binding and scheduling are critical steps towards creating a superior design [7]. These three steps are highly dependent on one another. This is due to the fact that operations sharing the same resource must have the same module and that the execution delays of the operations depend on the chosen module. For today's IC systems, the cost of solving the combined scheduling, binding, and module selection problem by exhaustive search is prohibitive.

A number of research results have been published on the module selection and scheduling/binding problem. Jain developed an integer linear programming (ILP) formulation to solve an optimal module selection problem [6]. However, in this formulation, schedule and resource binding are not considered. Timmer et.al. used a mixed integer linear programming (MILP) approach to select modules and then applied resource-constrained list scheduling to check if the time constraint is met [10]. The number of integer variables of the MILP, nevertheless, increases as the size of eligible module set increases. Ramachandran and Gajski used the probability table to investigate module selection while constructing a schedule [8]. However, they do not consider resource constraints while scheduling. Torbey and Knight applied a genetic algorithm to solve the scheduling and storage optimization [11]. They, however, do not integrate the module selection into their formulation.

In this paper, we propose a novel approach to solve the combined scheduling, binding and module selection problem. We develop a powerful model, called *acceptability function*, which enables designers to consider multiple design criteria simultaneously. Such a model represents a design objective and embeds either fixed (crisp) or soft (fuzzy) constraints as well as a tradeoff among conflicting criteria based on a user's willingness to accept a design. Considering the immense module selection space, we focus on designing a technique that efficiently identifies high-quality design solutions yielding high acceptability. The key to our approach is the use of a *module utility* metric together with *inclusion scheduling* introduced in [2, 3].

Using the utility metric can give alternative solutions to designers. By selecting modules with high utility values, a small set of superior solutions (called "elite set") can be generated. Since the elite set is very small compared to the size of enumerated solutions, inspecting each design in the set can be done in a short period of time. Thus our approach allows designers to focus on multiple initial designs.

2. MODELS

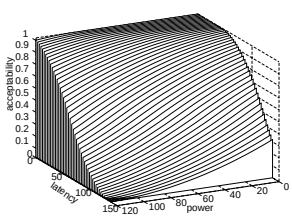
Operations and their dependencies in an application are modeled by a vertex-weighted directed acyclic graph, called a *Data Flow Graph* (DFG), $G = (\mathcal{V}, \mathcal{E}, \beta)$, where each vertex in the vertex set $\mathcal{V}$ corresponds to an operation and $\mathcal{E}$ is the set of edges representing data flow between two vertices. Function β defines the type of operation for node $v \in \mathcal{V}$.

Besides DFG, other constraints for constructing the system are characterized by a tuple $S = (\mathcal{F}, \mathcal{M}, A, \mathcal{Q})$, where $\mathcal{F}$ is the set of functional unit types available in the system. $\mathcal{M} = \{M_f | \forall f \in \mathcal{F}\}$, where each M_f contains a set of eligible modules for functional unit f, e.g., A is a function mapping from $M_f \in \mathcal{M}$ to a set of tuples $(a_1, \ldots, a_k)$, where a_1 to a_k

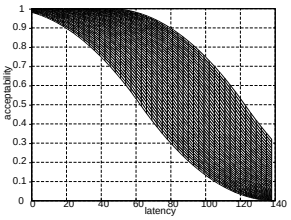
*This work was supported in part the NSF under grant number MIP 95-01006, MIP-9612298 and MIP-9701416.

represent attributes of a particular module. In this paper, $A(m) = (a_1, a_2)$ where a_1 refers to the latency attributes of module m while a_2 refers to the power consumption of module m. Finally, $\mathcal{Q}$ is a function that defines the degree of a system being acceptable for different system attributes. If $\mathcal{Q}(a_1, \ldots, a_k) = 0$ the corresponding design is totally unacceptable while $\mathcal{Q}(a_1, \ldots, a_k) = 1$, the corresponding design is definitely acceptable.

Using a function $\mathcal{Q}$ to define the acceptability of a system is a very powerful model. It can define certain constraints, express certain design goals, and model the tradeoff between two criteria. As an example, Figure 1(a) depicts an example of $\mathcal{Q}$ concerning the tradeoff graphically. Figure 1(b) shows the projection of the 3-dimensional acceptability model to the latency and acceptability plane. In this figure, each z curve represents a projection of $\mathcal{Q}$ function to a latency-acceptability plane. An inner curve (tighter latency constraint) corresponds to larger power values. Based on the acceptability model, a design with high acceptability implies an optimized design towards certain goals.



(a) z curve acc. with tradeoff



(b) Latency-acc. projection of Fig. 1(a)

Figure 1: Various kinds of acceptability functions

Based on the above model, the combined scheduling/binding and module selection problem we intend to solve can be formulated as follows:

Given a specification containing $S = (\mathcal{F}, \mathcal{M}, A, \mathcal{Q})$, *and* $G = (\mathcal{V}, \mathcal{E}, \beta)$, *find the minimum number of functional units and select modules* $m \in M_f$ *for functional unit* f, *based on the module utilities, while maximizing the acceptability degree of the solutions executing graph* G.

3. SYSTEM CHARACTERISTICS

The proposed approach contains two major phases:

Phase I– Determine the functional unit configuration (the number of functional units for each operation type)

Phase II– Determine the module utility values based on the configuration derived in Phase I.

In Phase I, we apply inclusion scheduling to efficiently estimate the maximum performance that one could achieve under a particular resource configuration among various module selections. First an initial number of resources is provided. Then inclusion scheduling constructs a representative schedule to efficiently derive possible latency and powers. After that, its maximum acceptability value among all pairs are estimated. If the value is still less than the desired acceptable value, the number of functional units and configurations are changed. After exploring several configurations, the best configuration is finalized and used as inputs to phase II. Details of inclusion scheduling can be found in [2,3]. The algorithm is modified to handle multiple criteria simultaneously.

In phase II, we produce a utility measure for each module. Ideally, modules resulting in superior designs should have higher utility values. A heuristic which analyzes modules ' quality so as to improve the module utility assignment is proposed. Finally, possible design solutions can be derived by selecting the combinations of the modules with high utility values.

The following section presents a heuristic to adjust module utility.

Module Utility Adjustment

We develop a concept of module utility metric. Each module with a utility value, which represents the usefulness of a module with respect to a given acceptability function. Ideally, the utility values of modules are either 1 or 0. The design using those modules with utility value of 1 should be of the highest quality. However, in reality, the usefulness of one module is dependent on other modules. A module may be present in both good and bad designs. To model such uncertainty, we allow the utility value of a module to be any real number between 0 and 1. Based on the utility values, we model the relationship between a functional unit and the module that implements this functional unit by a fuzzy set of possible modules for the unit as follows: Assume that there are 10 possible modules of adders ADD ={add1, add2,...,add10} and 10 possible modules of multipliers MUL = {mult1, mult2,...,mult10}. Let $\mu_{f_i}(m)$ be the utility value of module m for functional unit f_i, $m \in$ ADD for f_1, f_2 and $m \in$ MUL for f_3. It follows that $\mu_{f_i}(m)$ can be considered as the membership function of f_i. In other words, we treat each functional unit as if it has fuzzy execution times and powers, where each execution time and power pair is simply the attributes of a module. Therefore, fuzzy set theory can be used to operate arithmetic operations between two fuzzy sets [12].

Since the initial assignment of utility values may not lead to proper selection of modules in achieving the given design goal, in phase II, the utility values will gradually be adjusted to obtain better module selections. Intuitively, we attempt to give high utility values to modules which contribute to most highly acceptable latency and power pairs and assign low utility values to the modules contributing to latency and power pairs with low acceptability values. We modify the calculation of the schedule latency and power to also tally the number of module references for each latency and power value. Let the number of reference to a module by functional unit f be $\mathsf{freq}(f, m)$. For a given latency and power pair (t, p), we compute the positive contribution of m by

$$\sigma_+(f, m) = \sum_{\forall (t,p) \text{ s.t. } \mu(t,p) = \mu_f(m)} \mathsf{freq}_{t,p}(f, m)\mu_{\mathsf{acc}}(t, p) \quad (1)$$

From Equation (1), a higher $\sigma_+(f, m)$ value indicates that using m can potentially lead to systems with higher acceptability values. Similarly, we compute the negative contribu-

tion of m by

$$\sigma_-(f,m) = \sum_{\forall(t,p)\ \text{s.t.}\ \mu(t,p)=\mu_f(m)} \text{freq}_{t,p}(f,m)(1-\mu_{acc}(t,p)) \quad (2)$$

From Equations (1) and (2), an adjustment of the utility value for each module of functional unit f is estimated by Equation (3).

$$\text{adj}_f(m) = \frac{\sigma_+(f,m) - \sigma_-(f,m)}{\sigma_+(f,m) + \sigma_-(f,m)} \quad (3)$$

The term $\text{adj}_f(m)$ is a relative change to current $\mu_f(m)$ value. From Equation (3), if $\text{adj}_f(m)$ is negative, a module tends to cause more bad latency and power pairs. Then $\mu_f(m)$ should be decreased. On the other hand, if $\text{adj}_f(m)$ is positive, $\mu_f(m)$ is increased.
In our experiments, we have used the following method to calculate new $\mu_f(m)$, denoted by $\mu'_f(m)$:

$$\mu'_f(m) = \begin{cases} \mu_f(m) \times \text{adj}_f(m) + \mu_f(m) \\ \quad \text{if} \quad 0 < \text{adj}_f(m) \le 1 \\ \\ \frac{\mu_f(m)}{2} + (1 + \text{adj}_f(m)) \times \frac{\mu_f(m)}{2} \\ \quad \text{if} \quad -1 \le \text{adj}_f(m) < 0 \end{cases} \quad (4)$$

Since the value of $\text{adj}(m)$ is always between $[-1,1]$, if $\text{adj}_f(m)$ equals 1, we double the value of $\mu_f(m)$ and if $\text{adj}_f(m)$ equals -1, $\mu_f(m)$ is reduced by half. If $\text{adj}_f(m)$ is between $(-1,0]$, the change of $\mu_f(m)$ is proportional to half of $\mu_f(m)$ and if $\text{adj}_f(m)$ is between $(0,1)$, the change of $\mu_f(m)$ is proportional to $\mu_f(m)$. After the adjustment for all modules is made, $\mu'_f(m)$ are normalized with respect to the highest one, i.e., $\text{norm}(\mu_f(m)) = \frac{\mu'_f(m)}{\max_m \mu'_f(m)}, \forall m \in M_f$. If $\mu'_f(m)$ is the same as $\mu_f(m)$ from the previous iteration for every m, the adjustment is no longer needed.
By normalizing $\mu_f(m)$, $\text{norm}(\mu_f(m))$ produces a relative utility among all modules eligible for implementing f. After successively updating $\mu_f(m)$, some module utility value becomes zero. A module whose utility value is zero indicates that its contribution to superior solutions is not so significant as others. Such a module is then excluded from the elite set. The elite set is formed by inspecting the rest of the modules and choosing the ones with higher utilty values.

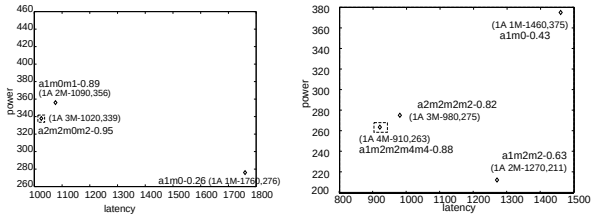
4. EXPERIMENTAL RESULTS

The proposed approach in this chapter has been implemented in a software package called WIZARD. In this section, we summarize our results on selected benchmarks. Finally, the comparison of WIZARD and other schemes: PSGA [1] and MSSR [5] is presented.
We tested our approach on examples including Discrete Cosine Transform (DCT), and Voltera filter. We randomly assign the initial values of these modules in such a way that the modules with the smallest power is the best. As comparison, we exhaustively generate the schedules for all the the module configurations for these benchmarks. Then, we consider the acceptability of the latencies and powers of these schedules and record the best acceptability.
Table 1 shows the comparison of the estimation obtained by WIZARD and the exhaustive approach for various tests. The design goals assumed for each benchmark are 5t+p and 5t+3p respectively. In Column "WIZARD est.", we present maximum acceptability obtained by inclusion scheduling in phase I for various resource configuration for each benchmark. Column "Exhau" displays maximum acceptability values obtained by the exhaustive approach. These values are close to each other. More importantly, they show the similar trend when increasing adder and/or multiplier units. The highlighted numbers in Column WIZARD show the maximum acceptability for each benchmark leading to the suggested functional unit configurations (see the corresponding rows). In these experiments, the worst case CPU time WIZARD used for these estimations is about 200 seconds while using the exhaustive approach may take up to 50,000 seconds.

Table 1: Estimating functional unit configuration

#FU	spec.	DCT (5t + p)		Voltera (5t + 3p)	
		WIZARD est.	Exhau	WIZARD est.	Exhau
2	1ADD 1MUL	0.34	0.34	0.46	0.48
3	1ADD 2MUL	0.98	0.93	0.77	0.67
	2ADD 1MUL	0.38	0.38	n/a	n/a
4	1ADD 3MUL	**0.99**	0.99	0.88	0.86
	3ADD 1MUL	0.39	0.4	n/a	n/a
	2ADD 2MUL	0.98	0.95	n/a	n/a
5	4ADD 1MUL	0.39	0.4	n/a	n/a
	3ADD 2MUL	0.98	0.94	n/a	n/a
	2ADD 3MUL	0.99	0.99	n/a	n/a
	1ADD 4MUL	0.99	0.99	**0.9**	0.9
6	4ADD 2MUL	0.97	0.94	n/a	n/a
	1ADD 5MUL	n/a	n/a	0.9	0.9
Avg. Err		1.2%		2.8%	

Figures 2(a)–2(b) depict the design characteristics (latency and power) and their acceptabilities derived for these benchmarks for various configurations. We only show the best module selection whose acceptability is the highest for each case. In practice, one only needs to explore the module selections for the specific resource configuration derived from phase I as highlighted in the box.



(a) DCT (5t + p) (b) Voltera (5t + 3p)

Figure 2: Latency, power, and acceptability for various benchmark designs based on selected modules

We also compare our results with some existing module selection schemes including: MSSR [5] and PSGA [1]. Both schemes consider fixed latency and area constraints and their objective is to find a design which minimizes area under latency constraint. In MSSR, the search for the module selection starts by considering the modules with the largest area. Then, it iteratively investigates modules with smaller areas until the latency constraint is violated. One drawback of the approach is that it assumes a fixed schedule during each

iteration of changing modules. PSGA is a mixed genetic algorithm-based and heuristic. It models a chromosome associated with a node priority, selected modules, and number of functional units. A list-based algorithm is used as a basis to calculate the fitness value of each chromosome. Each type of functional units contains only 3 modules.

We modified WIZARD so that it works with latency and area criteria. The acceptability function is defined as a crisp constraint on latency axis within region (0,x), for a given latency x, and as a linear fuzzy constraint on the area axis within region (0,y), where y is the estimated maximum area. In this case, by inspecting the graph, we use $y = 1072$. The initial number of functional units for the experiments were computed using [9].

Column "lat", these numbers are used as an x value under each test. WIZARD uses an acceptability function which favors a design with less area under each latency constraint. According to the benchmark characteristic, we determine the maximum number of functional units to be seven. Since there can be at most 4 parallel multipliers, the estimated loose upper bound on area will be 1072. Column "module set" under MSSR, PSGA, and WIZARD shows the module selection results for each approach. The scheduling method in [4] is used to find the schedule after the module selection is found. Columns "latency" and "area" presents the schedule latency and area of corresponding module selections. From the table, it is obvious that WIZARD can give a design which has smaller latency and/or area.

Table 2: Comparison results for Digital Elliptic Filter

latency	MSSR			PSGA			WIZARD		
constr	module set	lat.	area	module set	lat.	area	module set	lat.	area
14	4 a1 2 m1	14	576	3 a1 2 m1	14	560	4 a1 1 m1	14	320
18	2 a1 1 m1	18	288	2 a1 1 m1	18	288	2 a1 1 m1	16	288
30	1 a1 1 m1	27	272	1 a1 1 m1	26	272	1 a1 1 m1	26	272
60	3 a1 4 m2	60	176	3 a1 4 m2	59	176	3 a1 4 m2	58	176
300	2 a3 1 m2	288	36	n/a	n/a	n/a	2 a3 1 m2	256	36
1050	2 a3 4 m3	1040	12	2 a3 4 m3	976	12	2 a3 4 m3	960	12

Comparing the efficiency of these methods, WIZARD used only an average CPU time of 1.75 seconds on Ultra Sparc 30 to find the module selections while MSSR takes 9.5 seconds to compute the above solutions on DEC3100. For PSGA, as reported in [1], PSGA computes on average 2400–4800 schedules to arrive at good solutions. On the other hand, WIZARD performs 1-2 scheduling iterations (the predicted initial number of functional units are off by one at most) and each of the scheduling process generates 20-500 possible latency and area values to find the above solutions. The time to compute this would be equal to calculate approximately 20-1000 schedules. Therefore, the saving in the scheduling time is more than 80%.

5. CONCLUSION

We have presented an efficient module selection framework that determines both the type and number of modules to use. The framework evaluates module selections by taking scheduling/binding, and resource sharing into account. It can also handle more than one design attribute such as latency and power constraints. Our framework consists of two main steps: resource configuration estimation and module selection for the derived configuration. Both phases rely on the inclusion scheduling as well as a utility measure. Inclusion scheduling is a technique to gather information for resource approximation and module usefulness evaluation. The utility metric is a measurement that determines whether a module is a good one or not. A collection of the modules with high utilities forms a good module selection to achieve a given design goal.

6. REFERENCES

[1] I. Ahmad, M. K. Dhodhi, and C.Y.R. Chen. Integrated scheduling, allocation and module selection for design-space exploration in high-level synthesis. *IEEE Proc.-Comput. Digit. Tech.*, 142:65–71, January 1995.

[2] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient scheduling for imprecise timing based on fuzzy theory. In *Proceedings of the Midwest Symposium on Circuits and Systems*, pages 272–275, 1998.

[3] C. Chantrapornchai, E. H. Sha, and X. S. Hu. Efficient algorithms for finding highly acceptable designs based on module-utility selections. In *Proceedings of the Great Lake Symposium on VLSI*, pages 128–131, 1999.

[4] C. Chantrapornchai, S. Tongsima, and E. H. Sha. Imprecise task schedule. In *Proceedings of the International Conference on Fuzzy Systems*, 1997.

[5] M. Ishikawa and G. De Micheli. A module selection algorithm for high-level synthesis. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1777–1780, 1991.

[6] R. Jain. MOSP: Module selection for pipelined designs with multi-cycle operations. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 212–215, 1990.

[7] G. D. Micheli. *Synthesis and optimization of digital circuits*. McGraw-Hill, Inc, 1994.

[8] L. Ramachandran and D. Gajski. An algorithm for component selection in performance optimized scheduling. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 92–95, 1991.

[9] A. Sharma and R. Jain. Estimating architectural resources and performance for high-level synthesis applications. *IEEE Transactions on VLSI systems*, 1(2):175–190, June 1993.

[10] A. Timmer, M. Heijligers, L. Stok, and J. Jess. Module selection and scheduling using unrestricted libraries. In *Proceedings of the European Design Automation Conference*, pages 547–551, 1993.

[11] E. Torbey and J. Knight. Performing scheduling and storage optimization simultaneously using genetic algorithms. In *Proceedings of the Midwest Symposium on Circuits and Systems*, 1998.

[12] L. A. Zadeh. Fuzzy Logic. *Computer*, 1:83–93, 1988.