Proc. Design, Automation and Test in Europe Conference, 13-16 March 2001, Munich, Germany

Heuristic Datapath Allocation for Multiple Wordlength Systems

George A. Constantinides, Peter Y.K. Cheung
Electrical and Electronic Engineering Department
Imperial College
London SW7 2BT

Wayne Luk
Department of Computing
Imperial College
London SW7 2AZ

Abstract

This paper introduces a heuristic to solve the combined scheduling, resource binding, and wordlength selection problem for multiple wordlength systems. The algorithm involves an iterative refinement of operator wordlength information, leading to a scheduled and bound data-flow graph. Scheduling is performed with incomplete wordlength information during the intermediate stages of this refinement process. Results show significant area savings over known alternative approaches.

1 Introduction

This paper presents a heuristic solution to the combined scheduling, resource binding, and wordlength selection problem for multiple-wordlength systems, introduced as an ILP formulation in [5].

Traditionally the wordlength problem for DSP applications has been to find a single uniform system wordlength, which satisfies both the conflicting requirements of design area/speed/power and acceptable rounding and truncation signal distortion. The idea of a single uniform wordlength is consistent with the DSP processor model of computation where a single, or multiple, pre-designed fixed-wordlength computational units are responsible for all operations. When synthesizing custom hardware implementations, we are freed from such constraints. It is possible to use different wordlength functional units for different operations, in order to minimize the area requirements [3, 14] or power consumption [9].

Recent research into multiple-wordlength systems has concentrated on deriving fixed-point implementations from floating-point or infinite-precision descriptions, and includes [2, 3, 14, 16]. However there has been little research [4, 14] on high-level synthesis for multiple-wordlength systems. The use of multiple wordlengths has a significant impact on the traditional problems of high-level synthesis: scheduling, resource binding, and module selection. This arises from two factors. Firstly, each computational unit of a specific type, for example 'multiplier', cannot be assumed to have equal cost in a multiple precision system since area and power consumption scale with operator wordlength [4]. Secondly, the choice of wordlength for an operation can impact on the latency of that operation. Larger bit-parallel multipliers may have longer latency, or need to be pipelined to a greater extent than smaller bit-parallel multipliers in order to maintain the same clock frequency. The existence of multiple wordlengths therefore complicates the resource binding problem, and also increases the interaction between binding and scheduling of operations.

One approach to high-level synthesis for multiple-wordlength systems is to modify the resource-binding stage, by altering a standard clique partitioning algorithm on the compatibility graph [14] to select cliques by sorting nodes in descending order of wordlength. Another approach to resource-binding for multiple-wordlength systems has been to perform a constructive 'wordlength-blind' colouring on the conflict graph (the complement of the compatibility graph) and then refine this colouring using pairwise operations based on wordlength information [4]. Neither of these approaches consider adequately the effect of wordlength on operation latency, and therefore on scheduling. This problem was first examined in [5], where a formal description the problem was proposed, and an ILP model was derived. However it was also noted in [5] that the size of this ILP grows rapidly with the number of operations. This is the motivation for the polynomial-complexity heuristic solution proposed in the present paper. It should also be noted that the scheduling, resource binding and wordlength selection problem can be recast as a scheduling, resource binding and module selection problem [7], where different module types correspond to different wordlengths. However this is a very general expression of the problem, where the number of different operation types can be as large as the number of different operations. Also the common assumption [13] that in a module library area inversely scales with latency, is not true in our case, as latency and area both scale with

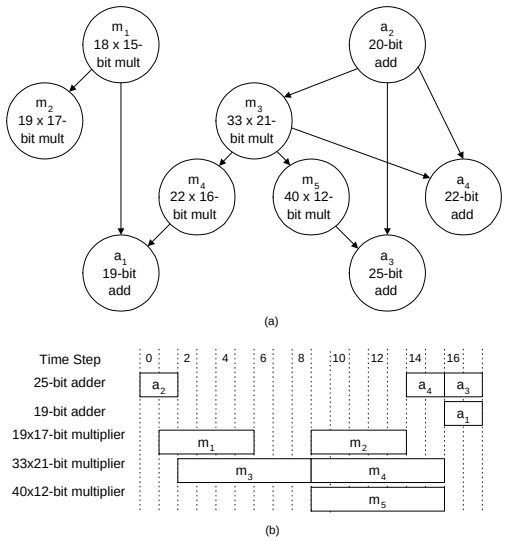

Figure 1. (a) A multiple wordlength sequencing graph and (b) its scheduling, resource binding, and wordlength selection

Table 1. Selected mathematical notation

O	set of operations
R	set of resource wordlengths
C	compatibility (directed) edges
H	wordlength (undirected) edges
$P(O,S)$	sequencing graph (data-dependencies)
$G(V,E)$	wordlength compatibility graph
$G'(O,C)$	compatibility subgraph
Y	set of operation types
N_y	resource constraint on type $y \in Y$ operations
$\ell(o)$	the latency of the resource to which operation $o \in O$ is bound
L_o	the upper-bound on the latency of operation $o \in O$ (from $G(V,E)$)
λ	user-specified overall latency constraint

wordlength.

A motivational example sequencing graph [7], representing data-dependencies, together with an area-optimal scheduling, binding and wordlength selection is illustrated in Fig. 1. The latency of all adders is two cycles, whereas the latency of an $n \times m$-bit multiplier is given by the empirical formula $\lceil (n+m)/8 \rceil$ derived for implementation at a fixed clock-rate on the SONIC reconfigurable computing platform [12]. Note that in Fig. 1(b) resources can execute operations up to the wordlength of the resource, even if implementation in a larger resource leads to a longer latency.

Section 2 of this paper introduces the proposed heuristic, section 3 illustrates solution quality and execution time results compared to alternative approaches in the literature, and section 4 concludes this paper. A table of the notation introduced and used in this paper is shown in Table 1.

2 Proposed heuristic

The proposed heuristic operates by exploiting the relationship between wordlength information and latency of each operation. The latency of each operation is refined downwards as the algorithm progresses, until the overall user-specified iteration latency constraint is satisfied. A pseudo-code overview of the heuristic is shown below. The intuition is that using the largest possible range of latencies at the start allows the greatest possible resource sharing. Latency information is only refined if the result violates the overall latency constraint. Upper-bounds are used in scheduling as the resulting bindings will then never violate the schedule. The resource set R, introduced in section 2.1, is calculated from the set of operations O.

Algorithm DPAlloc
Input: Sequencing graph $P(O,S)$, constraint λ
Output: Scheduling, binding, and wordlength information for each operation

```
while( no feasible solution ) do
    calculate resource set covering each operation;
    find upper-bounds L_o on latency of each
        operation o ∈ O;
    schedule P(O,S) using latency upper-bounds L_o;
    perform binding and wordlength selection;
    if( solution violates latency constraint )
        refine wordlength information;
    else
        record this as a feasible solution;
end while;
```

2.1 Wordlength compatibility graph

The model underlying our heuristic algorithm is the *wordlength compatibility graph* $G(V,E)$. This graph represents information on wordlength sizes, resource types, and schedule-derived information on time-compatibility between operation pairs.

The vertex set can be partitioned into two subsets $V = O \cup R$, where O represents the set of operations, and R represents the set of resource-wordlength types, for example '16×16-bit multiplier', '12-bit adder'. An algorithm for extracting all possible resource types from the set of operations is given in [5].

The edge set can also be partitioned into two subsets $E = C \cup H$. H is a set of undirected edges $\{o,r\}$, where

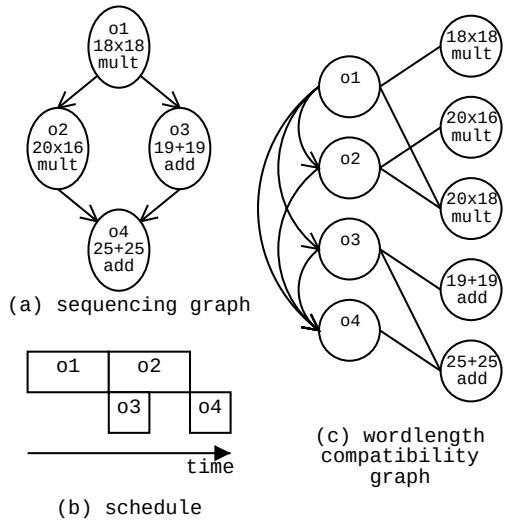

Figure 2. A wordlength compatibility graph

$o \in O$ and $r \in R$, indicating that operation o can be performed by resource-wordlength type r. Initially, this simply corresponds to the resource being of sufficient wordlength to cover the operation and of the same type, i.e. 'multiplier', or 'adder'. Later, the edges reflect the refinement of wordlength information in Algorithm DPAlloc. C is a set of directed edges (o_1, o_2), where $o_1, o_2 \in O$, indicating that o_1 is scheduled to complete execution before o_2 is scheduled to start execution. This is a transitive orientation on the subgraph $G'(O,C)$ [11], which will become important in section 2.3. A simple wordlength compatibility graph is shown in Fig. 2(c), corresponding to the simple sequencing graph and schedule shown in Figs. 2(a,b).

2.2 Scheduling with incomplete wordlengths

Standard resource-constrained scheduling typically uses specified upper limits on the number of resources of each resource type. In list scheduling [7], this constraint is checked at each control step before deciding whether to schedule a new operation. Our scheduling algorithm is introduced by comparison with this standard approach. Let $e_{o,t}$ be defined as in (Eqn. 1). Then given a set of control steps T, a set of operations $O_y \subseteq O$ of type $y \in Y$, and the maximum number of resources N_y of type y, we can formally express this standard constraint in (Eqn. 2).

$$e_{o,t} = \begin{cases} 1, & \text{if operation } o \text{ executes during control step } t \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

$$\max_{t \in T} \sum_{o \in O_y} e_{o,t} \leq N_y \tag{2}$$

In the case of multiple wordlength systems, this constraint is too relaxed to guarantee that no more than N_y resources of type y will be used. As an example, consider an iteration of the wordlength refinement process on Fig. 2, where due to latency constraints the undirected edge $\{o1, \text{'}20 \times 18 \text{ mult'}\}$ has been deleted. Under these circumstances, we cannot schedule the graph using one multiplier even though (Eqn. 2) can be satisfied for $N_{mult} = 1$. We propose the following alternative resource constraint calculation, which utilizes the incomplete wordlength information, inherent in any wordlength compatibility graph where there is at least one $o \in O$ with more than one edge $\{o,r\} \in H$.

Before any scheduling, a minimum cardinality subset $S \subseteq R$ is found such that $\forall o \in O : \exists s \in S : \{o,s\} \in H$. We refer to the set S as the *scheduling set*. Define $O(r) = \{o \in O | \exists \{o,r\} \in H\}$ and similarly $S(o) = \{s \in S | \exists \{o,s\} \in H\}$. Then our proposed scheduling constraint is given in (Eqn. 3).

$$\sum_{r \in S} \max_{t \in T} \left\{ \sum_{o \in O(r)} \frac{e_{o,t}}{|S(o)|} \right\} \leq N_y \tag{3}$$

Note that (Eqn. 3) is at least as strict as (Eqn. 2), which is a degenerate case of the former under the condition $|S| = |Y|$, the smallest possible scheduling set. This corresponds to the case when each operation of type $y \in Y$ could be performed by a single resource of type y with large enough wordlength. Under these conditions, (Eqn. 3) gives an exact bound on the number of resources. Similarly if there is a single edge in H from each operation, representing full wordlength information, then $\forall o \in O : |S(o)| = 1$, and the bound is exact.

As the possibilities for the wordlengths are refined during algorithm execution, so the 'balance' on the left-hand-side of (Eqn. 3) shifts from the 'max' operator to the outer '$\sum$' operator, to reflect this tighter constraint. Operations belonging to more than one scheduling-set member, i.e. those $o \in O$ with $|S(o)| > 1$ are accounted for by 'sharing' their usage equally between each of the elements of $S(o)$, hence the division in (Eqn. 3).

2.3 Combined binding and wordlength selection

Once a start control step has been assigned for each operation, resource binding and complete wordlength selection can go ahead. Any derived resource binding will not violate scheduling latency constraints, since the upper bounds were used in performing the scheduling.

The problem is therefore to choose a set of resources, and a mapping from operations to resources that covers all operations $o \in O$ while incurring minimum cost. We approach this problem by partitioning the subgraph $G'(O,C)$ into a set of cliques K, where each clique $k \in K$ satisfies the

constraint given in (Eqn. 4). This constraint captures the information that there must be a single resource-wordlength capable of performing all operations in the clique. The cost of this resource binding binding is given in (Eqn. 5).

$$\exists r \in R : \forall o \in k : \exists \{o, r\} \in H \tag{4}$$

$$\sum_{k \in K} \min_{r \in R : \forall o \in k : \exists \{o,r\} \in H} cost(r) \tag{5}$$

This is a special case of the well known set-covering or weighted unate covering problem [10], defined by (Eqn. 6). In our case there is one row of $\mathbf{A} = \{a_{ko}\}$ for each $o \in O$, and one column for each possible clique $k \in K^+$, $K^+ = \{O_1 \subseteq O | \exists r \in R : \forall o \in O_1 : \exists \{o, r\} \in H\}$. An entry $a_{ko} = 1 \Leftrightarrow o \in k$, and the cost $\mathbf{c}_k$ defined is as the corresponding summation term in (Eqn. 5).

$$\begin{aligned} &\text{minimize } \mathbf{c}^T \mathbf{x} \\ &\text{subject to } \mathbf{A}\mathbf{x} \geq \mathbf{1} \end{aligned} \tag{6}$$

We therefore extend a known heuristic for solving the unate covering problem in polynomial time [1]. We do not explicitly construct the matrix $\mathbf{A}$, since the number of columns can be exponential in $|O|$. Instead, we use an implicit approach, polynomial in $|O|$ shown below.

Algorithm BindSelect
Input: A scheduled wordlength compatibility graph
Output: A set of cliques $\{k_i\}$ with their associated resource types r_i

$i \leftarrow 1$;
while(still uncovered nodes) **do**
  Find a maximum clique p_r of uncovered nodes satisfying (Eqn. 4) for each $r \in R$;
  Choose $r \in R$ such that $|p_r|/cost(r)$ is maximum;
  Set $k_i := p_r$;
  Determine whether k_i can be 'grown' to cover $k_{i'}$, $i' = 1, \ldots, i-1$
    If so, delete $k_{i'}$ from the set of cliques;
  $i \leftarrow i + 1$;
end while;

At each iteration, the choice of clique is restricted to only those that are of maximum size with respect to (Eqn. 4). These can be found in linear time [11], since the subgraph of $G'(O,C)$ induced by the vertex set $O(r) \subseteq O$ is a transitively oriented graph for all $r \in R$. Since all cliques of a given $r \in R$ are of equal cost, only those with maximum size are candidates for selection in Algorithm Bindselect. The other modification to the heuristic presented in [1] is a compensation for the greedy nature of the selections. After each selection is made, it is checked whether the selected clique could be grown to cover any other cliques previously selected, in which case those superfluous cliques are now deleted.

2.4 Refining wordlength information

If the scheduling described in section 2.2 results violation of the user-specified latency constraints, then the next phase of Algorithm DPAlloc is to refine the wordlength information in order to meet the violated constraint. The first step of this process is to find a subset of nodes for which reducing their latency may lead to a reduction of the overall latency. This subset is determined by both scheduling and binding information, hence we refer to this subset as the *bound critical path*, Q^b to distinguish it from the standard critical path which is determined completely by sequencing precedence [7].

In order to determine the bound critical path, we augment the edge set S of the sequencing graph $P(O,S)$ with an additional set of edges S^b, as defined in (Eqn. 7), where $start(o)$ represents the scheduled start step of operation $o \in O$ and $\ell(o)$ represents the latency of the resource to which o is bound. The bound critical path is then defined to be the subset Q^b of operations with equal *ALAP* and *ASAP* times with respect to the augmented sequencing graph.

$$\begin{aligned} S^b = \{ \quad & \{(o_1, o_2)\} : start(o_1) + \ell(o_1) = start(o_2) \text{ and} \\ & o_1 \text{ and } o_2 \text{ are bound to the same resource } \} \end{aligned} \tag{7}$$

Once the bound critical path Q^b is established, we find the candidate subset of the bound critical path $W = \{o \in Q^b | start(o) + L_o \leq \lambda\}$ which finishes before the iteration constraint λ. At least one operation in this set must have its upper-bound latency reduced in order to schedule within the iteration period constraint. Reducing the latency of operations that are not members of this set but are members of Q^b may be necessary, but will not be sufficient to schedule the entire sequencing graph within the time required.

If there is more than one operation within this candidate set W, the operation o is chosen which would, on reduction of its latency upper-bound, lose the smallest proportion of edges in the set $\{\{o_1, r\} \in H | \exists \{o, r\} \in H\}$. Ties are broken by favouring those operations currently bound to a resource with latency less than the operation's upper-bound latency L_o.

Once an operation o is selected for refinement, all edges $\{\{o, r\} \in H | \ell(r) = L_o\}$ are deleted from the edge set H, before rescheduling.

3 Results

To our knowledge, this is the first heuristic in the literature to address the combined scheduling, binding, and

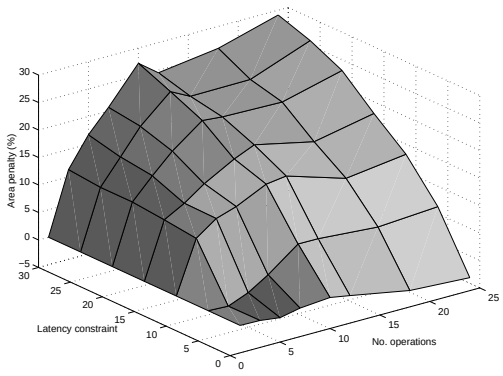

Figure 3. Variation of area penalty for [4] (over our heuristic) with number of operations and latency constraint

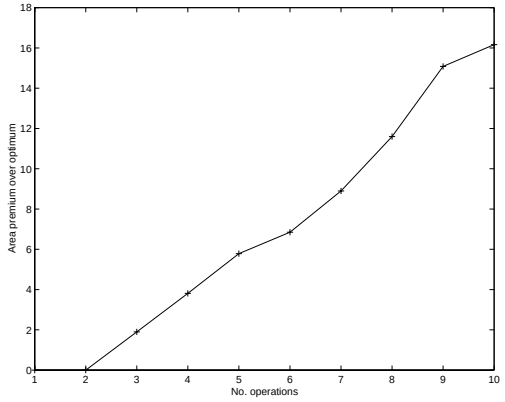

Figure 4. Variation of area premium (%) for our heuristic (over optimum [5]) with number of operations

wordlength selection problem. In this section, we compare the quality of results obtained and execution times to the optimum solution achieved by [5]. We also compare solution quality to the optimal branch-and-bound approach for resource binding and wordlength selection presented in [4]. This is a two-stage scheduling/binding approach based on sharing only resources that can be grouped together without increasing the latency of the operation.

For comparison of solution quality, we have generated 200 random sequencing graphs for each problem size $|O|$ between 1 and 24 using an adaptation of the TGFF algorithm [8]. The minimum possible latency λ_{min}, was found for each graph, from which various latency constraints were created, corresponding to a 0% to 30% relaxation of λ_{min}. A datapath was then generated for each of these graph/constraint combinations. The increase in implementation area of using the two-stage approach [4] solution over the heuristic presented in the present paper was found for each graph/constraint combination, assuming the area model presented in [5]. These data are plotted in Fig. 3. Each point represents the mean of the two hundred representative designs. These results illustrate that for designs with even a small 'slack' in terms of latency constraints, significant improvements can be made by performing the scheduling, binding, and wordlength selection in an intertwined manner. The area improvements come from increased resource sharing due to implementing small wordlength operations in larger wordlength resources with longer latency. Even for relatively small graphs, area improvements of tens of percent are possible.

Fig. 4 illustrates the increase in implementation area of using the heuristic presented in this paper over the optimum combined problem [5]. This is shown only for small problem size and minimum latency constraint $\lambda = \lambda_{min}$, as the ILP solution execution time scales rapidly as the latency constraint is relaxed (see Table 2 and below).

Table 2. Variation of execution time for 200 graphs with λ/λ_{min} for heuristic and ILP solution

λ/λ_{min}	heuristic (secs)	ILP (mins:secs)
1.00	3.02	2:07.09
1.05	3.51	4:05.21
1.10	3.73	15:55.56
1.15	3.52	>30:00.00

The variation of execution time with problem size for 200 graphs using the ILP model (executing on 'LP Solve' [15]) and the heuristic algorithm is shown in Fig. 5, illustrating the polynomial complexity of the heuristic against the exponential complexity of the ILP. All execution times are measured on a Pentium III 450 running Linux. Over the range of 1 to 10 operations, the relative increase in area ranges from 0% to 16% whereas the ILP solution takes between one and two orders of magnitude greater time to execute. An important point not brought out by these results is the scaling of execution time with overall latency constraint. The number of variables in the ILP model scales with the latency constraint, making the execution time highly dependent on this parameter [5]. This is illustrated in Table 2 for 200 9-operation sequencing graphs. For the heuristic presented in this paper, execution time does not scale with the latency constraint. Thus the one to two orders of magnitude illustrated in Fig. 5 are under conditions most favourable to the ILP-based solution.

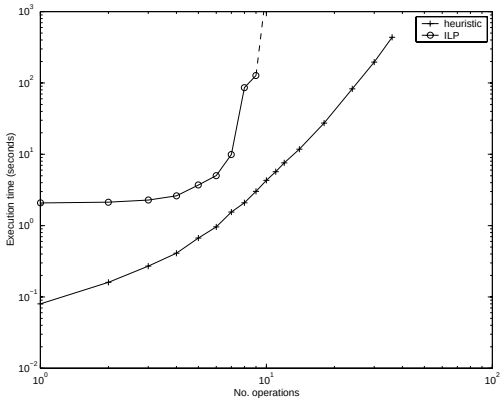

Figure 5. Variation of execution time with number of operations for heuristic and ILP solution

4 Conclusion

A heuristic has been presented for combined scheduling, resource binding and wordlength selection of multiple wordlength systems. This heuristic addresses the current lack of algorithms for high-level synthesis of operations with multiple precisions. We have demonstrated algorithms for scheduling using incomplete wordlength information, combined resource binding and wordlength selection, and refining wordlength information. These algorithms provide a powerful framework for datapath allocation, resulting in significant area savings over more traditional approaches.

In this work, the wordlength of each operation has been specified *a-priori*, either by hand or from output-error specification by a further design automation tool such as Synoptix [3, 6]. Future work should include investigation of the interaction between high-level synthesis of multiple wordlength systems and the derivation of wordlength information from output-error specifications.

Our current work on multiple-wordlength systems involves the extraction of wordlength information automatically from certain classes of nonlinear system.

Acknowledgements

The authors would like to acknowledge the support of Hewlett-Packard Laboratories and the Engineering and Physical Sciences Research Council, U.K.

References

[1] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, August 1979.

[2] R. Cmar, L. Rijnders, P. Schaumont, S. Vernalde, and I. Bolsens. A methodology and design environment for DSP ASIC fixed point refinement. In *Proc. Design Automation and Test in Europe*, München, 1999.

[3] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Multiple precision for resource minimization. In *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE Computer Society, April 2000.

[4] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Multiple-wordlength resource binding. In H. Gruenbacher and R. Hartenstein, editors, *Field-Programmable Logic: The Roadmap to Reconfigurable Systems*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[5] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Optimal datapath allocation for multiple-wordlength systems. *IEE Electronics Letters*, 36(17):1508–1509, August 2000.

[6] G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Roundoff-noise shaping in filter design. In *Proc. IEEE International Symposium on Circuits and Systems*, May – June 2000.

[7] G. DeMicheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.

[8] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task graphs for free. In *Proc. CODES/CASHE'98*, pages 97–101, 1998.

[9] M. Ercegovac, D. Kirovski, and M. Potkonjak. Low-power behavioural synthesis optimization using multiple precision arithmetic. In *Proc. 37th Design Automation Conference*, 1999.

[10] R. S. Garfinkel and G. L. Nemhauser. *Integer Programming*. John Wiley and sons, New York, 1972.

[11] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980.

[12] S. D. Haynes, J. Stone, P. Y. K. Cheung, and W. Luk. Video image processing with the Sonic architecture. *IEEE Computer*, 33(4):50–57, April 2000.

[13] M. Ishikawa and G. D. Micheli. A module selection algorithm for high-level synthesis. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 1777–1780, 1991.

[14] K. Kum and W. Sung. Word-length optimization for high-level synthesis of digital signal processing systems. In *Proc. IEEE International Workshop on Signal Processing Systems SIPS'98*, pages 569–678, 1998.

[15] H. Schwab. lp_solve. ftp://ftp.es.ele.tue.nl/pub/lp_solve, 1997.

[16] M. Willems, V. Bürsgens, H. Keding, T. Grötker, and M. Meyer. System-level fixed-point design based on an interpolative approach. In *Proc. 34th Design Automation Conference*, pages 293–298, June 1997.