Applications of Attributed-Behavior Synthesis

Lawrence F. Arnstein and Don Thomas
Carnegie Mellon University, Pittsburgh Pennsylvania

The goal of this paper is to demonstrate some of the types of design problems that can be addressed using the attributed-behavior approach to high-level synthesis based design. In one example, assertions are used to enforce a template hierarchy in an effort to exploit data-flow regularity during high-level synthesis. In another, assertions are used to account for a probable effect of logic synthesis on register transfer level results. The advantage of using the attributed-behavior approach to addressing design problems at the algorithmic level is the flexibility that stems from the use of a single general purpose synthesis tool and modeling language.

In attributed-behavior design, the engineer can exercise control over a high-level synthesis tool by establishing temporal and structural relationships between operations in an algorithmic model that must be reflected in the register transfer level model it produces[1]. This paper centers around ARFILT and DIFFEQ, two benchmark examples that are frequently found in high level synthesis literature, and fully described in [3].

We first present results to show that Marionet [1], our attributed-behavior synthesis tool, performs well in the absence of design assertions. We then show that the attributed-behavior approach to design specification and synthesis is expressive and flexible enough to address a range of issues that may arise in the course of design space exploration.

1 AR-FILTER

First, we compare the performance of Marionet with that of SAM [2] for a range of schedule lengths. We use the module library shown in Table 1 that is taken from Jain and Parker [4] because we are interested in synthesis results that are obtained in the presence of a rich module library. SAM is based on an enhancement of force-directed scheduling techniques first described by Paulin [8], in which specific module instances and interconnection information is considered in the computation of forces for scheduling. In Marionet the concept of force-directed scheduling is generalized to include module instance allocation and mapping as well as scheduling.

The cost/performance trade-offs that were realized by SAM and by Marionet are compared in Figure 1. The improvement of Marionet over SAM can be attributed to the more inclusive network based force formulation that is implemented in Marionet; it is more likely than SAM to resist using slow/cheaper functional units that can severely reduce the scheduling freedom of other operations in the graph.

TABLE 1. Module Library from [4]

Name	*Area mil²*	*Delay nS*
add_1	4200 (21 units)	340
add_2	2880 (15 units)	530
add_3	1200 (6 units)	1510
mult_1	49000 (245 units)	375
mult_2	9800 (49 units)	2950
mult_3	7100 (36 units)	7370
Clock	375 ns	

FIGURE 1. Cost/Performance Comparison

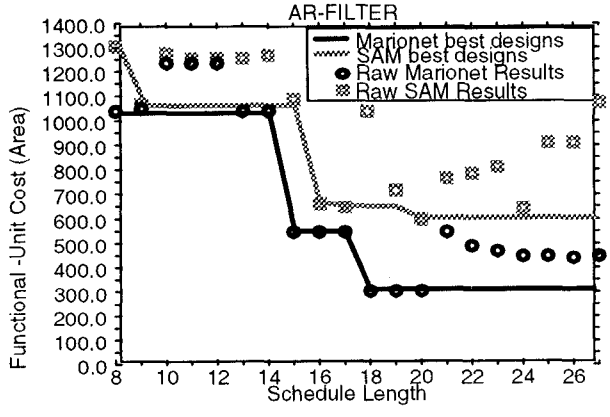
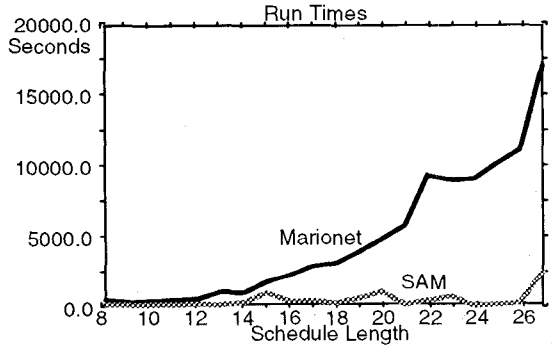


FIGURE 2. Run Time Comparison



The run-time results of Figure 2 indicate that, though Marionet performs consistently better than SAM in the presence of a rich module library, there is a significant cost in complexity. The increased complexity is due both to the constraint propagation that is necessary for the support of attributed-behavior assertions, and to the extra computation required for the more inclusive force computation.

Though Marionet has the ability to select resources from a rich module library, the complexity of the algorithm is sensitive to the size of the domains of operations (the set of possible control-step and functional unit assignments). Thus, for practical applications, it is important to keep the domains as small as possible. In the next section we examine how attributed-behavior assertions can be used with Marionet both to control complexity and to address a design issue that has heretofore been addressed only by specially designed synthesis tools. We look at two closely related problems: that of module selection and that of regularity exploitation as described by Rao and Kurdahi[9].

1.1 Module Selection

One way to reduce the complexity that results from a rich module library is to exclude some module types from consideration for some operations. Though the module selection choice can be left up to the engineer, the possibility that these choices can be made automatically based on the structure of the synthesis problem was investigated by Jain and Parker[4]. In that work, subsets of a given rich module library are ranked based on area and cost estimates derived from the structure of the given data-flow graph. A particular subset of the module library is chosen based on the cost/performance objectives of the design.

To adapt this technique to the attributed-behavior abstraction, module selection decisions made using the techniques described by Jain and Parker can be expressed as unary cost assertions that require certain operations to utilize certain resource types (not instances) from the full module library. For instance, the assertions:

```
1. (cost(*1) == 36)
2. (cost(*2) == 36)
2. (cost(*4) == 245)
3. (cost(*5) == 245)
```

Effectively selects the fastest multiplier type for operations 4 and 5 in Figure 3, while dictating that operations 1 and 2 utilize a slow multiplier. However, as unary expressions, these assertions do not specify any particular resource sharing relationship between the operations. In adhering to these assertions, Marionet will have fewer design decisions to make during synthesis, resulting in shorter run-times. in the next section a generalization of the module selection problem is discussed and some experiments are presented.

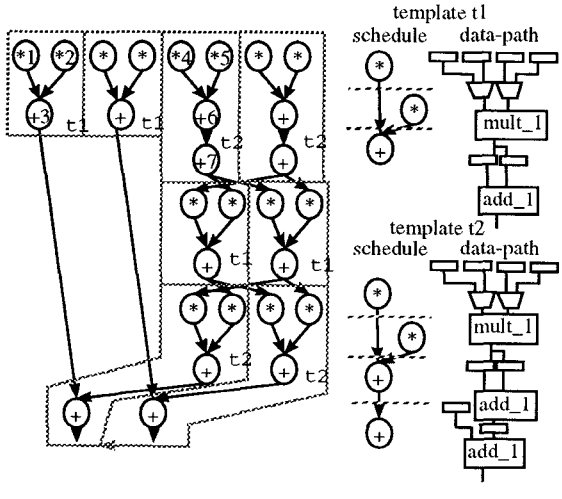
1.2 Regularity Exploitation

One problem that most synthesis heuristics suffer from is the lack of global perspective. While synthesis tools are good at extracting fine grain regularity, they may fail to exploit important coarse grained relationships. This limitation was addressed by Rao and Kurdahi [9]. They developed a special tool that identifies subgraphs that appear frequently in a given data-flow graph. A mini data-path, called a template, is designed (or synthesized) for each type of sub-graph that is identified. These templates then become the primitive module library used for synthesizing the super-graph that has a node for each subgraph in the original graph. This hierarchical approach allows for the exploitation of coarse-grained regularity that can be overlooked by many synthesis heuristics.

Note that in the trivial case, where each individual operation is its own sub-graph, this is equivalent to performing module selection as described above. In fact, like Jain and Parker, Rao and Kurdahi use a cost/performance estimation function to evaluate the effect that various template configurations have on the design space of the super-graph. There are two problems with the Rao and Kurdahi's approach: (1) the capabilities of the specialized regularity analysis and extraction tool cannot be easily combined with the capabilities of other special synthesis tools during the design process, and (2) some flexibility is lost by rigidly enforcing the template hierarchy. These two problems can be solved by performing regularity extraction at the attributed-behavior level prior to synthesis, using assertions that establish temporal and structural relationships between operations in a sub-graph.

To illustrate how the attributed-behavior abstraction and Marionet can support hierarchical regularity extraction, we apply assertions to the AR-FILTER to reproduce the template hierarchy given by Rao and Kurdahi shown in Figure 3.

FIGURE 3. A Hierarchy for AR-FILTER



To enforce this template hierarchy in Marionet, the following steps are taken: First, a primitive module library is created that does not have multipliers and adders *per se*, rather it has two types of resources, `t1` and `t2`, that can perform both add and multiply operations. The cost of each type of template is derived from the cost found for the mini data-path that implements it. The complete module library for the templates `t1` and `t2` is shown in Table 2; the costs are based on the module library shown in Table 1.

TABLE 2. Template Module Library

template	*ops*	*cost*	*delay +*	*delay **
t1	*,+	266	1 clk	1 clk
t2	*,+	287	1 clk	1 clk

The hierarchy is enforced by cost and path assertions that establish the internal scheduling and resource sharing relationships between all of the nodes in each subgraph. Additionally, a unary cost assertion is applied to one of the nodes from each sub-graph to select a template type from the module library. The assertions that should be used for sub-graphs `t1`:{*1,*2,+3} and `t2`:{*4,*5,+6,+7} of Figure 3 are shown below:

```
// assertions for template t1
1. (cost(*1,*2) == cost(*1)) // sharing...
2. (cost(*1,+3) == cost(*1)) // relationships.
3. (cost(*1) == 266) // choose template type.
4. (path(*1,*2) == 1) // establish internal...
5. (path(*1,+3) == 2) // schedule.
// assertions for template t2
6. (cost(*4,*5) == cost(*4)
7. (cost(*4,+6) == cost(*4)
8. (cost(*4,+7) == cost(*4)
9. (cost(*1) == 287)
10. (path(*4,*5) == 1)
11. (path(*4,+6) == 2)
12. (path(*4,+7) == 3)
```

There is one problem with this approach that would require some modification of our tool. Marionet will try to allocate registers and interconnections for values that are internal to the template, and thus already provided. To solve this problem, it would be necessary for Marionet to accept directives stating that some value transfers and value life-times remain un-mapped in the synthesized result. Support for such directives would be a useful feature in its own right, that would allow an engineer to use a synthesis tool to produce partial rather than complete descriptions. In this case, the partial implementation that would result is just the top level of the hierarchy.

Ignoring the extra interconnect produced by Marionet, the hierarchical design was synthesized for a range of global schedule lengths, the results are shown in Table 3 which compares the number of templates of each type required by Marionet to the number of templates of each type required by the optimal hierarchical implementation that was found by hand. A draw back of the hierarchical approach is that it tends to inhibit the realization of fine grained cost/performance trade-offs. The advantage is that the templates can be designed by hand at a lower level of abstraction for increased performance and decreased size. In this example, Marionet lags one control step behind optimal in discovering when fewer `t1` templates are needed.

TABLE 3. Hierarchical Synthesis Results

Results: Marionet v. Optimal	11-steps		14-steps		15-steps	
	t1	t2	t1	t2	t1	t2
Marionet	2	2	2	2	1	2
Optimal	2	2	1	2	1	2

These results tend to support our claim that the constraint propagation technique and force-directed synthesis formulation in Marionet are capable of the same type of hierarchical synthesis that was previously available only from a specialized tool

An advantage of the attributed-behavior approach is increased flexibility. Due to the rigid hierarchy, Rao's system does not appear to exploit the fact that the `t2` template data-path can also implement the `t1` subgraphs. Thus, no matter how long the schedule gets, the hierarchy alone requires that at least one template of each type be purchased. Additional flexibility can be given to Marionet by removing the unary cost assertions that restrict `t1` sub-graphs to the use of only `t1` templates (remove assertion 3 above). The result of this simple modification is shown Table 4. At 14-steps, two `t2` templates are sufficient for implementing all of the `t1` and `t2` subgraphs; this is discovered by Marionet for the 15-step schedule, again one step behind optimal.

TABLE 4. Flexible Hierarchy Results

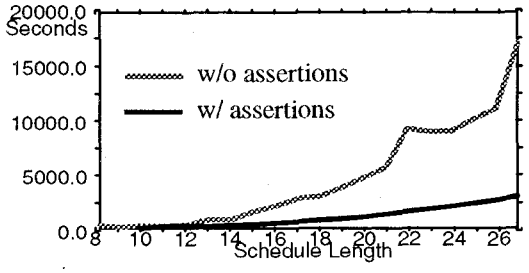
Results: Marionet v. Optimal	11-steps		14-steps		15-steps	
	t1	t2	t1	t2	t1	t2
Marionet	1	2	1	2	0	2
Optimal	1	2	0	2	0	2

To illustrate the effect of granularity extraction on the time required for synthesis, the execution times for the AR-FILTER with the hierarchy enforcing assertions are compared in Figure 4 to the execution times for the flat AR-FILTER experiments of Figure 1. The addition of assertions actually serves to reduce rather than increase the complexity of synthesis.

In the above example, assertions where used to expose regularity in a data-flow graph. By expressing hierarchy requirements in terms of attributed-behavior assertions, and graphically superimposing them on the original textual algorithmic model, the engineer can easily observe and modify the decisions made by the special

hierarchy extraction tool. New assertions can then be added to address other issues that may arise in the course of design space exploration, such as chip level partitioning[6] and external timing constraints[7][5]. The advantage of the attributed-behavior approach is that a range of design issues can be addressed using only a general purpose synthesis tool and modeling language.

FIGURE 4. Execution Times w/ Assertions



2 DIFFEQ

The example presented in this section is the difference equation solver [3] shown below that is commonly used as a benchmark for high-level synthesis tools. Again, we first compare the synthesis results obtained by Marionet with those obtained by SAM to provide support for our claim that Marionet performs well in the absence of any design assertions.

Diffeq

```
while (x < a)
begin
  t1 = u *1 dx;
  t2 = 3 *2 x;
  t3 = 3 *3 y;
  t4 = t1 *4 t2;
  t5 = dx *5 t3;
  t6 = u - t4;
  u = t6 - t5;
  y1 = u *6 dx;
  y = y + y1;
  x = x + dx;
end
```

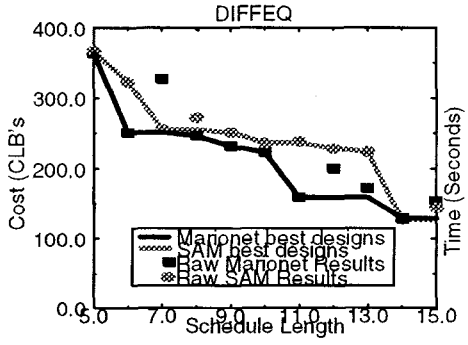
We then show that assertions can be used to address a design issue that is relevant to this example. The module library used in this example, shown in Table 5, is different from that use for the AR-FILTER above. The cost model is based on FPGA technology and the costs are given roughly in numbers of look-up tables (LUT's) required for 8-bit input functional units. As indicated in Figure 5, Marionet performs generally better than SAM, with relative run-times that look similar to the graph of Figure 2.

The inner loop of the DIFFEQ algorithm consists of a single basic block that contains only multiply, add, and subtract operations. Two of the multiply operations produce only multiples of three; these operations can be implemented by a special multiplier whose cost is more like that of an adder than that of multiplier. However, in the literature [8], and in the above experiments, it is assumed that all multiply operations must use a fully general multiplier. In the experiment below, we investigate how assertions can be used to account for the collapsibility of the multiply-by-three operations ($*_2$ and $*_3$).

TABLE 5. Module Library for DIFFEQ

Name	*Cost in LUT's*	*Delay nS*
addsub_1	9	60
add_2	15	35
add_3	20	25
mult_1	40	100
mult_2	50	80
mult_3	80	50
Clock	60 ns	

FIGURE 5. Cost/Performance Comparison



2.1 Collapsible Multipliers

Though we know that logic synthesis, when applied to the synthesized register transfer level design, can collapse multiply functional units that have constant inputs, this information does not happen to be encoded into Marionet. Ideally, to communicate this information to the synthesis tool the following attributed-behavior relationships would be expressed, where X is the cost of a general multiplier and Y is the cost of a collapsed multiplier:

```
1. if (cost(*2,*1,*4,*5,*6) ==
       cost(*1,*4,*5,*6))
   then (cost(*2) == X)
   else (cost(*2)) == Y)
2. if (cost(*3,*1,*4,*5,*6) ==
       cost(*1,*4,*5,*6))
   then (cost(*3)) == X
   else (cost(*2)) == Y
```

These rules simply state that if *2 or *3 share resources with any of the more general multiply operations then the cost of the resource is the full cost of a general multiplier, otherwise, savings can be realized. This concept is similar to the add-function cost table approach to constructing multiple function ALU's that is used in some high-level synthesis systems[10]. The fact that Marionet does not currently accept expressions of this type is a limitation of the tool rather than of the attributed-behavior approach. More generally, a primitive module library can be modeled by a set of attributed-behavior rules of this type instead of by a cost table.

Because Marionet does not currently accept rules of this type we must use assertions that are somewhat over-constraining. In this experiment, we ran the high-level synthesis tool without assertions (design A), and with the two different sets of assertions shown below (designs B and C) that can be accepted by Marionet.

Design B

1. $cost(*_2, *_1, *_4, *_5, *_6) > cost(*_1, *_4, *_5, *_6)$
2. $cost(*_3, *_1, *_4, *_5, *_6) > cost(*_1, *_4, *_5, *_6)$
3. $cost(*2) == 80$
4. $cost(*_3) == 80$

Design C

1. $cost(*_2, *_1, *_4, *_5, *_6) > cost(*_1, *_4, *_5, *_6)$
2. $cost(*_3, *_1, *_4, *_5, *_6) > cost(*_1, *_4, *_5, *_6)$
3. $cost(*_2, *_3) == cost(*_2)$
4. $cost(*_2) == 80$

The first two assertions, which are the same in both sets, effectively create two partitions of multiply operations within which resource sharing is allowed. One partition is the set $\{*_1, *_4, *_5, *_6\}$ and the other is the set $\{*_2, *_3\}$. Though resource sharing across partition boundaries is prohibited, sharing within the partitions is left to the discretion of the synthesis tool. Also, unary cost assertions have been added to ensure that the synthesis system does not try to save cost by using slower multiply functional units for $*_2$ or $*_3$.

Design B differs from Design C in that, in the latter, the two multiply-by-three operations are required to share a single resource $\{*_2, *_3\}$. If the two operations happen to be on the sequential critical-path of the design then an additional control-step will be required, as happens to be the case for this example. The results for each set of assertions for a range of schedule lengths are shown in Tables 6-8.

In filling the tables, any multiply functional unit that was used for only $*_2$ and/or $*_3$ was replaced by a hypothetical functional unit called `times_three` that has similar delay and cost characteristics as an ordinary adder from the module library. The "Raw Cost" of a design is the total cost of the multipliers allocated, whether or not they can be collapsed. The "Final Cost" is the total cost of the multipliers after accounting for collapsibility. We are only concerned with multiplier utilization in the this experiment, so the quantities and types of other functional units have been omitted from the tables. Following the tables are some interpretations of the results.

TABLE 6. Results for Design A

Design A	*No Assertions*			
Functional Unit	*A-5*	*A-6*	*A-7*	*A-8*
times_three (9)	1		2	
mult_1 (40)		1	1	3
mult_2 (50)		1		
mult_3 (80)	2	1	1	
Raw Cost	240	170	200	120
Final Cost	169	170	138	120

TABLE 7. Results for Design B

Design B	*Partitioning Only*			
Functional Unit	*B-5*	*B-6*	*B-7*	*B-8*
times_three (9)	2	1	1	1
mult_1 (40)			1	2
mult_2 (50)		1		
mult_3 (80)	2	1	1	
Raw Cost	320	210	200	160
Final Cost	178	139	129	89

TABLE 8. Results for Design C

Design C	*Partition and Sharing*			
Functional Unit	*C-5*	*C-6*	*C-7*	*C-8*
times_three (9)	-	1	1	1
mult_1 (40)	-		1	2
mult_2 (50)	-			
mult_3 (80)	-	1	1	
Raw Cost	-	160	200	160
Final Cost	-	89	129	89

2.2 Interpretation

In all cases but one (Design C-6), the raw cost for the designs with no assertions are better than the ones with assertions, for same-schedule-length designs. This tends to support the conclusion that Marionet performs well with respect to the information that is encoded into the tool.

In all cases but one (Design A-5), the final cost for designs with assertions are better than without, for the same-schedule-length designs. This tends to supports our claim that the attributed-behavior approach, even with the limited expressiveness supported by Marionet is an effective way to account for information that external to the to the synthesis tool, like the effect of collapsible multipliers.

The one case in which the final cost for no assertions is better than the one with assertions (Design A-5 compared to Design B-5) can be attributed to luck. By chance, $*_3$ in Design A-5 was bound to a multiplier that is not used for any other operations and so is collapsible. Furthermore, $*_2$ is bound to a multiplier that is in fact used

for some of the general multiply operations, so Design A-5 does not satisfy the requirements established in the assertions. Thus, the failure of Marionet to discover this good design in the presence of assertions was due to the over-specification required by restrictions on the assertions language rather than on any particular weakness in the attributed-behavior approach.

The best results, both raw and final, were obtained when the tool was forced to share the two multiply-by-three operations in Design B. Serendipitously, a better raw cost for was obtained for design C-6 than for Design A-6. This result simply indicates that the synthesis tool is not guaranteed to find an optimal design under any circumstances, but that additional information provided by the engineer can be used exposes otherwise inaccessible regions of the design space.

Though there are other distinguishing aspects of these designs such as combinational critical-path, which is lower in Design C-8 than in C-6, and the cost of multiplexors, registers, and other functional units, it is the global schedule length and multiplier costs that dominate the overall measures of design quality for this example. The important result of this experiment is that, without getting involved in intricate data-flow analysis, it is possible for the engineer to use assertions for rapid generation of design alternatives that take into account information that is external to the given synthesis tool.

3 Conclusion and Future Work

The above examples demonstrate that a single high-level modeling language and synthesis algorithm can be used to address a range of design issues that may arise during design space exploration without forcing the engineer to abandon the algorithmic perspective. Additionally, these experiments reflect our experience that the binary constraint network based force-directed synthesis algorithm implemented in Marionet is well behaved in the presence of assertions.

In future experiments we hope to show that more than one or two design issues can be simultaneously address at the attributed-behavior level for a single algorithmic model. We will also show that attributed-behavior relationships are an effective means for presenting synthesis results to the engineer in terms of the original, familiar algorithmic model. Given this type of feedback, high-level synthesis based design becomes a process of explorative refinement of an attributed-behavior specification in cooperation with a high-level synthesis tool like Marionet.

There are many issues that must be addressed during the design of VLSI circuits that vary in degree of importance from design to design and from technology to technology. Though it may not be reasonable to hope for a monolithic synthesis tool that takes all relevant factors into consideration, weighs them appropriately, and terminates quickly, it is reasonable to expect CAD algorithms to do a good job on a few closely related sub-tasks.

However, to overcome the short-sightedness of sub-task based solutions, synthesis systems have tended to become monolithic--simultaneously performing module selection, scheduling, allocation, binding, and interconnect synthesis. In fact, Marionet is an example of this trend. The difference is that by redefining the synthesis task in terms of attributed-behavior, we are hoping to encouraging the re-development of specialized tools for specific sub-tasks, some of which are discussed above. If these new specialized tools are designed to perform at the attributed-behavior level, not only by writing assertions, but by reading, responding to, and perhaps modifying existing assertions, then synthesis knowledge can again be distributed, but this time without losing the potential for cooperation among interrelated sub-tasks.

Acknowledgments

This research was funded by an NSF Graduate Research Fellowship and by NSF Grant # MIP-9112930.

References

[1]Arnstein, L.F., Thomas, D.E., "A General Consistency Technique for Increasing the Controllability of High Level Synthesis Tools", *Proceedings of the International Conference on Computer Aided Design*, November, 1993.

[2]Cloutier, R., "Force Directed Scheduling Allocation and Mapping", *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 71-76, 1990.

[3]Dutt, N., Ramachandra, C., *Benchmarks for the 1992 High Level Synthesis Workshop*, Technical Report 92-107, University of California, Irvine, Oct. 1992.

[4]Jain, R., Parker, A., Park, N., "Module Selection for Pipelined Synthesis", *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pp. 542-547, 1988.

[5]Ku, D.C., De Micheli, G., Relative Scheduling under Timing Constraints: Algorithms for High Level Synthesis of Digital Circuits, *IEEE Transactions on Computer Aided Design*, Vol 11, No 6, June 1992, pp 696.

[6]Lagnese, E., *Architectural Partitioning for System Level Design of Integrated Circuits*, Ph.D. Thesis, Carnegie Mellon University, ECE Department, 1989.

[7]Nestor, J. A., *Specification and Synthesis of Digital Systems with Interfaces*. PhD thesis, Carnegie-Mellon University, April 1987.

[8]Paulin, P., *High Level Synthesis of Digital Circuits Using Global Scheduling and Binding Heuristics*, Ph.D. Thesis, Department of Electronics, Faculty of Engineering, Carleton University, January 1988.

[9]Rao, D.S., Kurdahi, F.J., "Hierarchical Design Space Exploration for a Class of Digital Systems", *IEEE Transactions on Very Large Scale Integration Systems*, Vol 1, No 3, September 1993, pp. 282-295.

[10]Thomas, D., *The System Architect's Workbench User's Guide*, Technical Report CMUCAD-91-42, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1991.