Using Feedback To Improve VLSI Designs*

C. W. Liew
Department of Computer Science
Rutgers University
Piscataway NJ 08855
liew@cs.rutgers.edu

February 26, 1995

Abstract

VLSI HLS design is a typical design optimization problem with a focus on generating good solutions. Typical expert design systems incorporate a large amount of domain knowledge to generate good initial solutions. These systems are unable to use information gleaned from analysis of the solutions (feedback) to generate better solutions. This paper describes a new technique called Constrained-Redo that uses feedback to improve both the power and coverage of an existing design system, the DAA system.

1 Introduction

The VLSI High Level Synthesis (HLS) problem consists of creating a register level transfer (RTL) description of a chip that satisfies a given behavioral specification, while simultaneously maximizing the speed of the chip and minimizing area usage. The input to an HLS system is an algorithmic (functional) description in a high level language (e.g., VHDL) of a chip, either a microprocessor or a digital filter. The output is a register transfer level (RTL) description of the chip consisting of a specification of components, e.g., adders, multipliers and registers, connections between the components and a controller that determines when the components execute and when data is transferred between components. The behavior of the RTL description must satisfy the functional requirements embedded in the algorithmic description. Examples of chips specified using this scheme range from simple circuits, e.g., digital filters, to complex microprocessors, e.g., Motorola MC68000. A more detailed description of High Level Synthesis can be found in [15].

Optimization is a very important part of the VLSI design process. Much time and design effort is spent on determining how to generate a solution that is optimized for a particular combination of criteria, e.g., AREA or TIME or both. VLSI HLS is a well-understood problem so generating correct solutions is easy but finding a good solution remains difficult.

The Design Automation System (DAA) [10, 14] is a VLSI design system that designs microprocessors. The system incorporates a large amount of domain knowledge in the form of complex evaluation functions, optimizations and rule bases for selecting appropriate library modules. One weakness of the system that is also present in many other systems lies in its inability to use information (feedback) gleaned from analysis of its solutions to improve its own performance.

*Also appears in the Proceedings of the 11th IEEE Conference on AI Applications, 1995
The knowledge of how to analyze the solutions is present either in textbooks or in the minds of VLSI designers who can critique the designs. This paper describes how feedback based from the analysis of solutions has been used to improve both the power and coverage of the DAA system. The technique is called Constrained-Redo and it uses information output by critiques of an existing solution to guide the design system to generate a new, different and hopefully better solution. The Constrained-Redo technique is part of the Feedback Directed Optimization (FDO) framework described in [12]. The extended system (FDO/DAA) has been tested on several standard microprocessor designs and it has improved many of the solutions that DAA generated.

2 The DAA System

DAA is a system designed to solve the problem of automated synthesis of digital hardware from behavioral descriptions. The system was originally developed at CMU as part of the CMU DA project and later extended at AT&T as part of the Algorithms to Silicon Project.

The input to the system is a behavioral description in the ISPS language [1]. The behavioral description is then compiled into a data-flow graph where the nodes represent the computations that are to be performed and the edges represent the connections that are necessary for data values to be passed between the computations.

The VLSI synthesis task is solved by carrying out a sequence of tasks. First, the computations are scheduled so that there is no conflict, i.e., a computation is not assigned a time slot that is earlier than the computations that generate the data it uses. The values from the computations are then assigned to either registers or memory. The abstract computational and storage elements are then interconnected to support the requisite data transfers. The computational elements are then mapped to hardware modules via a clustering algorithm (Section 2.1). The controller is then designed and the modules are then placed and interconnected on a chip. The final solution generated by the DAA system is a description of a microprocessor consisting of a controller, hardware modules (e.g., adders, multipliers, registers and multiplexers) and the interconnections between the modules.

2.1 The Clustering Module

One of the main components of the DAA system is the clustering module. Its function is to map abstract computational elements onto actual hardware modules, e.g., adders, multipliers and multiplexers. This problem is difficult in that an abstract element can be mapped onto several different hardware modules and several elements can be mapped to the same module. Finding the optimal mapping is NP-complete, thus the clustering module uses a heuristic algorithm.

Once the operations have been scheduled and their values assigned (see previous section), new designs are produced by partitioning the operations and distributing them and the values they produce among a number of clusters to be then used for floorplanning. The number of ways $N$ operations can be partitioned into subsets is $2^N$ so an exhaustive search for the best partition is impractical. The heuristic method used is based on a metric that measures the relative distance of every pair of operations. The metric defines a distance between each pair of operations such that two operations are “close” if there is a great advantage to having them share the same data paths while they have a large distance if there is little advantage to putting them together.

The distance between operations depends on three factors: their common functionality, degree
of interconnection and potential parallelism. More precisely the distance between two operations $x$ and $y$ is defined to be (from [14]):

$$
dist(x, y) = -S_1 \times f\text{prox}(x, y) - S_2 \times c\text{prox}(x, y) + N \times S_3 \times \text{par}(x, y)
$$

(1)

where

$$f\text{prox}(x, y) = \frac{f\text{cost}(x) + f\text{cost}(y) - f\text{cost}(xy)}{f\text{cost}(xy)}$$

$$c\text{prox}(x, y) = \frac{\text{commconn}(x, y)}{\text{totalconn}(x, y)}$$

and

$$\text{par}(x, y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ can be done in parallel} \\ 0 & \text{otherwise} \end{cases}$$

$f\text{cost}(x \ y \ z \ldots)$ is the cost, based on delay and area, of the minimal number of function units required to do all the operations in the list. Therefore $f\text{prox}$ is the ratio of the shared functionality of $x$ and $y$ to the total functionality of both. $c\text{prox}$ is the ratio of the data-flow connections shared by $x$ and $y$ (e.g., $y$’s output is $x$’s input, or some value is an input to both $x$ and $y$) to the total data-flow connections to either $x$ or $y$. The $S$ factors are weights that show the significance of the terms they multiply to the complete design. $S_1$, for example, is the area of the functional unit required to do operations $x_1$ and $x_2$ divided by the total area of the design. $S_3$ is the probability of either $x_1$ or $x_2$ being executed in one major cycle of the hardware divided by the average number of steps in the cycle.

The metric has the effect of ranking the operations based on (1) their parallelism and (2) the estimated area of the hardware implementation. $N$ is the exponent specified by the user for the system’s objective function $AT^N$.

Initially a distance matrix is computed that gives the distance for each pair of operations. Using the method of [7], the system forms a hierarchical clustering tree that is based on the distances in the distance matrix. The leaves represent the operations or points to be clustered and the heights in the tree correspond to distances between points. Points are joined at a height corresponding to the distance between the points. For individual points, the height represents the distance between two specific operations while for subclusters, the height is the average distance between operations in the subcluster. Thus the distance is more “precise” for individual operations and the likelihood of combining subclusters in a non-optimal fashion increases with the number of operations within a subcluster.

Different configurations are formed by cutting the tree at different levels. For each branch that crosses the cut line, a separate cluster is formed containing the operations in that branch.

Figure 1 shows an example of a cluster tree. The numbers on the left show the initial indices of the operations into the distance matrix. There are 16 operations so there are 16 indices (points). The result of each clustering is represented by the number on the right of the pairings (the index of the new cluster in the distance matrix). The dashed line on the right (the cut-line) shows how the 16 points would be partitioned into four clusters. The first cluster is at 55 and is composed
Figure 1: The cluster tree for a balanced partitioning of 16 operations

of the points [55,53,27,23]. The other three clusters are at: (2) 51 with [51,18,17,40], (3) 50 with [50,48,46,38] and (4) 42 with [42,36,33,21].

The cluster tree guides the search of the design space. A series of different configurations is produced by starting with the cut line at the root and moving it toward the leaves. Each time a join is crossed, the new set of clusters defined by the cut line is formed and the resulting design is evaluated. The process continues until some stopping criterion is reached, such as no improvement in the design for a predetermined number of configurations. The best design seen in the search is restored and can be further refined by dividing it or recombining some of the clusters in it.

Figure 2 shows the results of evaluating several different configurations of the cluster tree for an AMD2901 microprocessor. The total area and average cycle time are used to compare the current design with past designs. The overall quality of a design is determined using a designer specified objective function of the form $AT^N$, i.e., $A \times T^N$ where $A$ is the total area, $T$ is the average cycle time and $N$ is an integer between 0 and 5. If $N = 0$, then only the AREA usage is used to measure the quality of the design. The designs in Figure 2 are evaluated with an objective function of $AT^0$. Level 1 is a design that consists of a single cluster and Level 2 is a 2 cluster design and so forth thereby stepping from a “serial” design with one cluster to a “parallel” design where every function is implemented by a separate hardware module.
Level 1: Average cycle time: 1712.50 ns.
Total area 1.980e+06 u^2.
AT^N: 1.980e+06

Level 2: Average cycle time: 1500.00 ns.
Total area 1.699e+06 u^2.
AT^N: 1.699e+06

Level 3: Average cycle time: 1300.00 ns.
Total area 1.936e+06 u^2.
AT^N: 1.936e+06

Level 4: Average cycle time: 1300.00 ns.
Total area 2.130e+06 u^2.
AT^N: 2.130e+06

Best configuration found at level 2.

Figure 2: Results of evaluations with N = 0

2.2 Evaluating a Design

Each design is evaluated by first scheduling the operations onto control steps using a priority list scheduler [2]. The highest priority is assigned to operations that are on the longest path through each linear block of operations. The length, width and area of each cluster is determined using information about the numbers and sizes of registers, multiplexers and wires. This information is estimated using a computation of the maximum storage required between any two control steps as well as the maximum number of interconnections required.

The clusters are now laid out on a floorplan using a min-cut partitioning algorithm that is based on [5]. An approximate floor plan is generated based on the dimensions of the leaf clusters and their component modules. The area of the bounding box for the floor plan is the total area of the design. The average cycle time of the design is calculated using the probabilities for the control steps \( p_i \) and the minimum clock period. The minimum clock period is the maximum delay over all the control steps.

3 Using Critique Knowledge to Improve DAA’s Solutions

One of the known problems with the solutions generated by the DAA system is that they sometimes use too many multiplexers for a single module. This can arise when the loads for identical functional units are unbalanced. The loads are considered to be unbalanced if for two or more modules of the same type, the number of operations that are mapped to it differs by more than 50% from the smallest to the largest. Multiplexers are used to select one of several lines on input and output. Each hardware module can have inputs from several sources and outputs to multiple sinks. Multiplexers ensure that for each operation the data is retrieved from the correct source and reaches the correct sink.

For example in the case of a generated solution that has two adders, nine out of ten addition operations may use one adder while only one operation uses the other adder. So one adder would require two 8-input multiplexers while the other would require none. A better solution might balance the load and have six operations use one adder and the remaining four operations the
other adder. The cost of multiplexers lies in the additional area for the multiplexers and increased delay in signals reaching the module, thereby possibly increasing the clock cycle length (time).

<table>
<thead>
<tr>
<th># of inputs</th>
<th>Area</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>192</td>
<td>13</td>
</tr>
<tr>
<td>16</td>
<td>384</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 1: Costs of Muxes vs # of Inputs for 16 bit Data

Table 1 shows the area and delay costs for several sizes of multiplexers. The costs assume that the data is 16 bits wide, i.e., each input consists of 16 bits of data. The table shows that the area used grows linearly in the number of inputs but the delay grows faster. The impact of balancing the load on the modules depends on whether the module will affect the speed of the critical path.

3.1 Critiquing the Solution

The FDO/DAA system proceeds by first critiquing the best solution generated using the DAA approach. The solution is then analyzed to determine which parts of the solution might be improved. The analysis modules also generate suggestions as to how to modify the solution. The analysis of the best DAA design is carried out in the following steps:

1. Find the most likely module type to focus on.
2. Find the operations that use the specified module type.
3. Find a better partitioning of the operations

The analysis method that was implemented in FDO/DAA looks for two or more functional units of the same type, e.g., adders and multipliers, in the best solution. It then checks to see if there is a large disparity between the number of inputs to each of the units. If there is a disparity, the method will try to find a better allocation of the operations to the function units. Further details about the analysis process can be found in [12].

As an example, if we set the objective function to AT^0 for the design of an AMD2901, then the generated solutions are shown in Figure 2. The best solution generated by DAA is the one generated at level two of the cluster tree with two clusters. The FDO/DAA analysis module has a pre-specified list of types of modules that it knows how to load balance. The module ranks the types by the number of units present in the solution. The operations that use that module type are then collected and then repartitioned amongst the units using the same clustering algorithm. The major difference is that in this case, the only viable clustering candidates are those operations that use the specified module type. Currently, the FDO/DAA system looks for imbalances in the following module types: AND, XOR, and NOT (inverters).

In the case of the AMD2901, the highest ranked module type is the AND module and there are four units with a total of sixteen operations that use them. Figure 1 shows the cluster tree generated for partitioning all sixteen operations equally amongst the four units. This clustering of
the sixteen operations into four clusters constitutes the feedback or recommendation generated by the analysis module.

Several other analysis methods were tried with the DAA system. These analysis methods that were tried were domain independent. They were mainly techniques that selected a "good" subcluster from the best design that DAA produced and used that subcluster as a seed point for the next design. The techniques differed mainly in how a subcluster was determined to be good. For example, one method examined each cluster and calculated the difference between the evaluation of the cluster and the sum of the evaluations of the subclusters that make up this cluster. This gives some idea of the "incompatibility" between the subclusters as a more incompatible pair would have a higher difference.

The intent of these experiments was to see if the clustering algorithm was susceptible to focusing on small local optima such that a slight perturbation would result in an improved solution. The results of the experiments are discussed in Section 5.

3.2 Using Feedback

Feedback in the FDO/DAA system consists of a set of desirable clusters. These clusters constitute the suggestions from the analysis module of how to (re)cluster a small subset of the operations and thus improve the solution. Each cluster consists of a number of points (or functional operations) that fit together, i.e., generate an efficient implementation if they are implemented as a single hardware cluster. The question is how can the system use this information to try and generate a better solution.

The Feedback Directed Optimization approach as applied to the clustering algorithm involves restarting the clustering algorithm with these clusters as new seed points. The Constrained-Redo technique is implemented as a module that is executed before the clustering algorithm. The goal is to have every generated solution contain the desired clusters. The assumption is that the clustering algorithm heuristics are generally good and that they will generate a better solution if given a better starting point that overcomes some of the weaknesses of the heuristics.

The input to Constrained-Redo is a set of desired clusters and the output is a partially clustered set of operations that is consistent with having partially executed the clustering algorithm. Each cluster in the set of desired clusters consists of a list of operations that make up the cluster. The Constrained-Redo algorithm clusters the operations in the list into a single cluster and at the same time updates the distance matrix using the same update function as the clustering algorithm. This is repeated for each cluster in the set of desired clusters. The end result of executing Constrained-Redo is similar to executing the clustering algorithm for a few iterations to form the desired clusters. The key is to ensure that the distance matrix is updated correctly.

Using our example with the AMD2901 microprocessor, the Constrained-Redo algorithm would generate the initial four clusters specified in the feedback and shown in Figure 1. The clustering algorithm is then started with the cluster tree (Figure 1) and would generate clusters based on the initial set of four clusters.

The clustering algorithm normally starts with every operation assigned to a separate cluster, i.e., every operation is a cluster. The analysis modules find desirable clusters and starts the clustering algorithm with those clusters in place, i.e., with a different set of seed points. The assumption is that with the desirable clusters in place, the clustering algorithm will be less affected by the "horizon effect" of only examining pairs of clusters. The effect of starting the clustering algorithm with the desired clusters in place is that every design generated by the clustering algorithm will have incorporated the desired clusters as part of it.
The clustering algorithm itself has been modified to accept a partially clustered starting point, i.e., to start out with the desired clusters. The clustering algorithm starts with these clusters and iterates until all the operations have been clustered into a single cluster.

4 Implementation Details

The DAA system is broken down into several modules, one of which is the clustering module. The extensions to the system making it the FDO/DAA system consists of two modules, one to analyze the best solution generated by DAA and another to implement the Constrained-Redo technique and use the feedback from the analysis. This cycle of analysis followed by redesign adds an iterative loop to the clustering module.

The DAA system consists of approximately 100,000 lines of 'C' code with a small library of parameterized hardware modules. The clustering module itself is made up of two files totaling 2200 lines of 'C' code. The additions and modifications to the clustering module consists of 200 lines of code, about 10% of the new clustering module. Most of the development effort was spent in understanding the module and developing the domain independent and domain specific analysis methods.

5 Experimental Results

The DAA system has been tested against an extensive suite of benchmark tests. These benchmarks are behavioral descriptions of processors that range over a number of different processors ranging from small toy specifications to medium sized processors that have been commercially used, e.g., MOSTEK M6502.

The FDO/DAA system was evaluated on the same benchmarks to determine if adding the FDO concepts and techniques improved the performance of the system in terms of the quality of the generated solutions. The FDO/DAA system was evaluated using a number of analysis methods to determine if (1) what effect each of the methods would have and (2) if any were clearly better. All suggestions generated by the analysis methods were tried out in succession.

The domain independent analysis methods did not significantly improve the performance of FDO/DAA as compared to DAA. Most of the solutions were not improved by the use of feedback. This was mostly to be expected as the evaluation function incorporates a large amount of domain knowledge. It would have been surprising if a weak domain independent method would have discovered weaknesses in the clustering algorithm. The results of these tests are not shown because they did not exhibit any significant improvement in the best design.

Table 2 shows a comparison of the best solutions generated by DAA and FDO/DAA for three microprocessors. The three microprocessors were chosen because they were commercially implemented and used. The first column shows the name of the benchmark specification. The three microprocessors are the Advanced Micro Devices AM2901, Advanced Micro Devices AM2910 and the Mostek 6502. The second and third columns show the quantitative evaluation for the best solution that DAA and FDO/DAA generated, respectively. A higher quality solution has a lower quantitative evaluation, i.e., the evaluation is indicative of the “cost” of the solution. The evaluation is based on the objective function of $AT^N$ with $N$ ranging from zero to five. If $N = 0$, then the system evaluates only the AREA usage and if $N = 5$, it evaluates only the TIME usage. For all other values of $N$, the system evaluates the resource usage as specified by the objective function. The fourth column shows the setting of $N$. The last column shows the percentage
improvement in the quality of the solution generated by the FDO/DAA system. This number is computed by evaluating the improvement as a percentage of the original cost as determined by the objective function.

As can be seen, the FDO/DAA system produces better solutions in many of the test cases. The FDO/DAA system was not able to produce any faster designs when speed was of overriding concern, i.e., \( N = 5 \). This can be partially explained by the fact that all three instruction sets were designed at a time when most of the design knowledge pertained to designing fast uniprocessors. As a result, the behavioral specifications are biased towards an implementation with a single fetch-decode-execute cycle, and this proves to be a general limiting factor. If the fastest design is implemented using a few hardware modules, i.e., one or two clusters, then reclustering would not have any impact and the results from FDO/DAA tend to validate that belief.

The evaluation function (Equation 1) used by the clustering algorithm was developed by Kowalski and McFarland. It was based on data obtained on experiments with the Mostek M6502 microprocessor. It would be expected therefore that DAA would perform best and FDO/DAA poorly on the M6502 benchmark. For four of the six test cases, the FDO/DAA system was not able to generate a better solution. If we rank the benchmarks by the number of times that the system could find a better solution, then the FDO/DAA system scored worst on the M6502.

<table>
<thead>
<tr>
<th>Processor</th>
<th>DAA</th>
<th>FDO/DAA</th>
<th>N</th>
<th>( \Delta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD2901</td>
<td>1.699e+06</td>
<td>1.493e+06</td>
<td>0</td>
<td>12.1%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>1.920e+06</td>
<td>1.825e+06</td>
<td>0</td>
<td>4.9%</td>
</tr>
<tr>
<td>M6502</td>
<td>1.182e+07</td>
<td>1.139e+07</td>
<td>0</td>
<td>3.6%</td>
</tr>
<tr>
<td>AMD2901</td>
<td>2.632e+09</td>
<td>2.298e+09</td>
<td>1</td>
<td>12.7%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>1.497e+09</td>
<td>1.459e+09</td>
<td>1</td>
<td>2.5%</td>
</tr>
<tr>
<td>M6502</td>
<td>5.436e+10</td>
<td>4.692e+10</td>
<td>1</td>
<td>13.7%</td>
</tr>
<tr>
<td>AMD2901</td>
<td>2.989e+12</td>
<td>2.989e+12</td>
<td>2</td>
<td>0.0%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>1.222e+12</td>
<td>1.105e+12</td>
<td>2</td>
<td>9.6%</td>
</tr>
<tr>
<td>M6502</td>
<td>2.532e+14</td>
<td>2.532e+14</td>
<td>2</td>
<td>0.0%</td>
</tr>
<tr>
<td>AMD2901</td>
<td>2.696e+15</td>
<td>2.270e+15</td>
<td>3</td>
<td>15.8%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>8.134e+14</td>
<td>7.840e+14</td>
<td>3</td>
<td>3.6%</td>
</tr>
<tr>
<td>M6502</td>
<td>1.111e+18</td>
<td>1.111e+18</td>
<td>3</td>
<td>0.0%</td>
</tr>
<tr>
<td>AMD2901</td>
<td>2.147e+18</td>
<td>1.956e+18</td>
<td>4</td>
<td>8.9%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>5.915e+17</td>
<td>5.186e+17</td>
<td>4</td>
<td>12.3%</td>
</tr>
<tr>
<td>M6502</td>
<td>4.878e+21</td>
<td>4.878e+21</td>
<td>4</td>
<td>0.0%</td>
</tr>
<tr>
<td>AMD2901</td>
<td>7.625e+02</td>
<td>7.625e+02</td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td>AMD2910</td>
<td>7.134e+02</td>
<td>7.134e+02</td>
<td>5</td>
<td>0.0%</td>
</tr>
<tr>
<td>M6502</td>
<td>4.170e+03</td>
<td>4.074e+03</td>
<td>5</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Table 2: A comparison of DAA and FDO/DAA
benchmark. On the other test processors, the FDO/DAA system was able to find a better solution at least 50% of the time. This showed that the FDO approach was able to extend the coverage of a finely tuned heuristic to cover several more cases.

The improvement gained by using the FDO/DAA system has to be evaluated in the context of the state of VLSI expert systems. The DAA system is one of several systems at the forefront of the field that incorporate large amounts of design domain knowledge. An improvement of 10% is considered a significant gain for the system because many man years have been spent constructing and refining the system. However, these systems still cannot match the abilities of human designers.

6 Discussion

The experimental results (Section 5) showed that the FDO approach and in particular the Constrained-Redo technique improved the performance of the DAA system. The system was able to generate good solutions for a larger number of solutions. In addition it was able to improve the solutions to some of the problems on which the system was trained or customized.

The FDO approach and the Constrained-Redo technique have been shown to be suited for application to large systems like DAA. Some of the reasons for the applicability are:

- treating the problem solver like a “black box”: The FDO approach views the problem solver as a system that accepts a problem specification and generates a solution, and whose internal workings are not easily accessible. The main requirement of the problem solver is that it accept a partial solution as part of the input. For example, the clustering module in DAA accepts some pre-clustered operations and goes on to cluster the rest. This allows the analysis modules to focus only on the quality of the solution and not concern themselves with the actual internal details and decisions.

The advantage of this approach is that the techniques do not require the problem solver to generate design records. Design records can be costly to capture in terms of additional computation time and in the amount of space required especially for large problems. In addition, many existing systems like DAA do not capture design records and it would require a major overhaul and modification to enable the system to generate and save design records.

- ease of modification/update: The technique has been shown to be easily added to an existing system, even one as complex as DAA. This is important because there are many large, tested knowledge based systems that would be difficult to rewrite or extend if the changes involved substantial changes in the program. The Constrained-Redo technique consists mainly of adding modules around the system thereby making it easier to add to an existing system.

- use of available domain knowledge: In many of these domains, there is usually a large amount of domain knowledge available for analysis of the solutions. The FDO approach and Constrained-Redo technique enabled the DAA system to utilize the knowledge. The experimental results showed that the addition of that knowledge improved the performance of the system. The results also showed that purely domain independent methods are unlikely to improve the solutions.

The knowledge that was present in the analysis modules cannot be easily added to the cluster generation heuristics, in particular the distance metric (Equation 1), for the following
reason. It would take more knowledge than is available to determine (1) how much weight to assign to the new function and (2) when multiplexers play a role. In the experiments with FDO/DAA, balancing the loads did not always lead to better solutions. Frequently it resulted in a worse solution. The best case for balancing the loads is when there is a large imbalance for a particular type of modules. This cannot be easily predicted a priori and is easier to detect after the solution has been generated.

7 Related Work

There has been much research done in the field of AI and Design but most of it has been focussed on the area of functional correctness, i.e., generating designs that are correct. The field of VLSI High Level Synthesis is sufficiently well understood such that the focus is more on improving the quality of the solutions. The FDO approach can be viewed as an intra-problem learning technique wherein the system improves its performance by analyzing the solutions that it generates. Earlier work on problems like the Traveling Salesman Problem (TSP) [13] or in planning (HACKER [19]) used feedback from analysis to directly modify the solution. This approach is not applicable to VLSI High Level Synthesis because the components interact strongly and make it difficult to modify the solution without causing it to be incorrect. The work that is most closely related to the FDO research falls under two general categories.

Using design histories to improve performance: There are many learning techniques that have been tried and tested on design systems. Explanation Based Generalization (EBG) [16] and other design history based mechanisms (Chronological Backtracking [18], Dependency Directed Backtracking [3], ATMS [4] and Case Based Reasoning (CBR) [9]) have not been used with systems like DAA because of the difficulty in obtaining design records. The FASOLT [8] system was able to use feedback to improve its performance but relied on the human designer to give specific instructions on how to use the feedback. In addition, the FASOLT system was only able to naively generate low quality initial designs and then improve on these designs whereas the DAA system uses domain knowledge to generate good initial designs and the FDO approach then improves on these good designs.

Generating better evaluation functions: In the previous section (Section 6), we discussed the difficulties involved in improving or modifying a complex evaluation function like the one shown in Equation 1. There has been some work on improving evaluation functions but they have either been working with simpler functions [17] or the work has been done by human designers, e.g., [11, 14].

8 Conclusion

This paper has described how the Feedback Directed Optimization (FDO) approach, in particular the Constrained-Redo technique, has been applied to an existing VLSI design system. The experimental results in Section 5 have shown that the extended system (FDO/DAA) outperforms the DAA system on a number of standard benchmarks over a set of objective functions. In addition, the Constrained-Redo technique was applied to a large, complex system and the extensions were added with little change made to the original system. This shows that the approach may be applicable to other large existing systems.

We intend to apply our work to large systems in other fields. One of the current projects is to apply the Constrained-Redo technique to a simulator for a jet engine nozzle design system [6].
The hope is that the feedback will enable the system to more quickly converge on an optimal solution.

Acknowledgements

The research described in this paper could not be done without the generosity of AT&T in providing access to the source of the DAA system. Ted Kowalski was extremely generous with his time and knowledge in describing DAA. Haym Hirsh provided constructive feedback on an earlier draft of this paper. The members of the Rutgers CAP group, especially Lou Steinberg, made many useful suggestions in the early stages of the project. The research reported in this paper was supported by the Defense Advanced Research Projects Agency through NASA grant NAG2-645. The opinions expressed in this paper are those of the author, and do not reflect any policies, either expressed or implied, of any granting agency.

References


