Integrating Floorplanning In Data-Transfer Based High-Level Synthesis

Shantanu Tarafdar
Synopsys Inc.
700 East Middlefield Rd.
Mountain View, CA 94040

Miriam Leeser Zixin Yin
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA 02115

Abstract

Modern digital systems move and process vast amounts of data. Designing good ASIC architectures for these systems requires efficient data routing and storage. A high-level synthesis (HLS) system must consider spatial aspects of the architecture it synthesizes to achieve this. In this paper, we discuss using floorplanning information in the main HLS flow. Our HLS system, Midas, incorporates floorplanning and formulates HLS using a data-transfer model. Midas synthesizes an architecture whose data storage and transfer subsystems are spatially integrated with its execution unit. Midas also generates a high-level floorplan for the architecture, which contains the shapes and coordinates of its components and routing channel specifications for its buses. Our experiments comparing Midas's architectures to those generated by a HLS system that does not use the data-transfer model or floorplanning show that Midas's architectures are smaller and yet allow for large amounts of simultaneous data motion and storage.

1 Introduction

Advances in fabrication technology allow digital integrated circuits (ICs) to contain an increasing number of transistors. Computer-aided design (CAD) tools are needed to manage the increasing design complexity. In addition, high-throughput, memory-intensive digital systems, like those in multimedia applications, place heavy demands on IC timing and area, making the interdependence between layout and IC architecture design more critical today than ever before.

We address the importance of data-issues and floorplanning in high-level synthesis (HLS). HLS is a branch of CAD for ICs that synthesizes an architecture for an IC given its intended behavior. In modern IC's, interconnect and storage dominate the IC area, yet most HLS systems do not consider IC layout during synthesis, and treat data-issues as secondary. Midas, our HLS system, uses the data-transfer (DT) model [1, 2] and floorplanning in the core HLS flow. The DT-model formulates HLS around data-transfers in the behavior as opposed to operations as is more traditional. The floorplanners – a global one and an incremental one – provide Midas with the shapes and placement of components and buses very early in the HLS flow. The result is a high-level floorplan and an architecture in which data is stored close to where it is produced and used. Our experiments show this methodology yields architectures with smaller layouts than a methodology that ignores floorplanning.

ICCAD98, San Jose, CA, USA

2 Previous work

While most HLS systems ignore the floorplanning aspects of the architectures they synthesize, there are some notable exceptions. Kurdahi et al. [3] propose a layout predictive model, for use in high-level design automation tools, which accounts for a variety of register transfer level design styles and uses floorplanning. In BUD [4], operations in the design are clustered hierarchically and each cluster is scheduled, allocated, and then floorplanned. BUD evaluates the generated design and re-synthesizes until an acceptable design is achieved. The IBA (interleaved binder/allocator) HLS system uses a module called Fasolt [5] as a post-processing step to re-optimize a previously synthesized and then floorplanned architecture. 3D [6] is an HLS system that first schedules and binds operations, then performs floorplanning and finally makes changes to the architecture to reduce wiring delays computed from the floorplan. BINET [7] is a binding algorithm that performs incremental binding and floorplanning on a previously scheduled behavior using a network flow approach.

In contrast to the above approaches, our HLS system, Midas, tightly integrates the utilization of floorplanning information in the main HLS flow. Floorplanning is *not* a post-processing option. Midas uses floorplanning to take into account the area cost of a data-transfer, to measure data-transfer congestion at points in the floorplan or in the schedule, and to project more accurate area estimates for the architecture being synthesized. These design metrics guide all aspects of HLS in Midas: scheduling, allocation, and binding.

3 The DT-model

Midas aims at elevating the status of data issues in HLS by formulating the classic HLS subproblems like scheduling and binding in terms of data-transfers [1, 2]. In an architecture, data must be bound to storage elements and data transfers (DTs) must be bound to terminals of functional units, ports of storage elements, and interconnect. The DT-

model in Midas provides a framework in which to do this concurrently with operation scheduling and binding.

In HLS, a dataflow graph (DFG) is extracted from the input behavioral description. The nodes of the DFG represent operations and the directed edges represent data dependencies between operations. In traditional HLS systems, scheduling and binding are formulated in terms of operations. Midas formulates these subproblems in terms of data-transfers instead. A data-transfer (DT) is a *set* of operations corresponding to the movement of a single instance of data. It contains the operation sourcing the data and all the operations using the data. In an unscheduled DFG, the operation represented by each node with outgoing edges is the source operation of a *primary* DT and the operations of the node's children are the destinations operations of the DT. An operation can be a member of more than one DT. Figure 1 shows DTs extracted from a DFG.

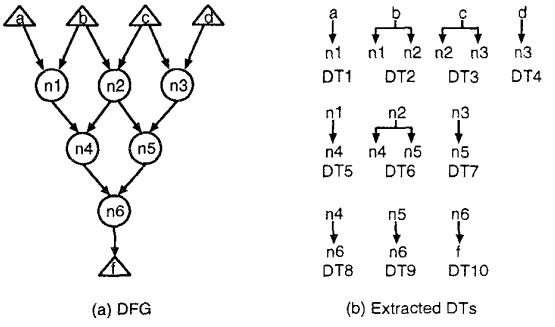

Figure 1: Extracting DTs from a DFG

DTs have a cost associated with them. The cost of transferring data can be modelled as the product of the area of the interconnect needed and the length of time the interconnect has to be reserved for the transfer. The sum of the costs of DTs scheduled in the same control step (cstep) is the *cumulative* DT-cost at the cstep and reflects the interconnect requirement of the architecture. A schedule in which each cstep has roughly the same cumulative DT-cost is one that efficiently utilizes the available interconnect in the ASIC. Figure 2 is a snapshot of the floorplan of an architecture at one cstep in the execution of the behavior. The arrows are DTs and the areas of the arrows represent the relative costs of the DTs.

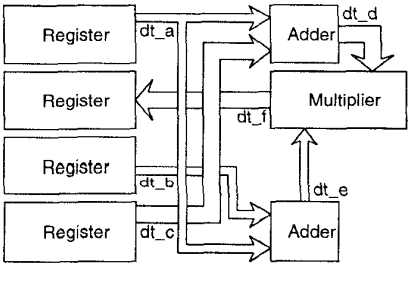

Figure 2: DT-costs in a cstep during execution

The *scheduling directive* of the DT-model states that all the operations of a DT must be scheduled in the same cstep. The directive allows the storage requirements to be modelled explicitly in Midas. If all the operations of a primary DT are scheduled in the same cstep, the data generated by the source is consumed immediately and does not need to be stored. Often, it is impossible to schedule all the operations of a DT in the same cstep due to resource or timing constraints. The primary DT must be *partitioned* into a set of *secondary* DTs, each of which meets the scheduling directive. The operations of the primary DT are distributed among the secondary DTs. DT partitioning implies the data represented by the primary DT must be stored in the architecture. Each secondary DT generated requires a read or write of the data from a storage element and has a storage read or write added to its set of operations. Figure 3 illustrates a general case of DT partitioning.

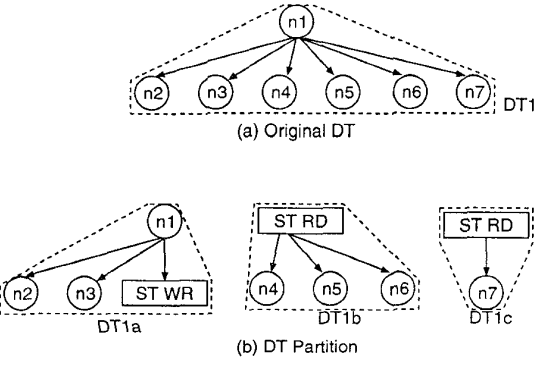

Figure 3: A partition of a primary DT

DT partitioning directly impacts storage and interconnect requirements. Partitioning gives rise to a requirement to store data, which may increase the size of storage. Each of the secondary DTs formed by partitioning a DT gives rise to a storage access requirement, which can increase the number of ports on storage elements, and an interconnect requirement, which can increase the number of buses in the synthesized architecture. In general, DT partitioning is to be avoided since it has the potential to increase storage size, the number of storage ports, and the number of buses.

DT partitioning is a core HLS step in Midas since DTs that cannot meet the scheduling directive must be partitioned before they can be scheduled. Therefore, Midas *has* to consider storage and interconnect in its main HLS flow. Since Midas has floorplanning information available, it uses this information to influence DT partitioning. For example, DTs that meet the scheduling directive, but which would have to be scheduled in csteps with high cumulative DT-costs, can be *selectively* partitioned. This allows the secondary DTs generated to be scheduled in csteps with lower cumulative DT-costs, thereby equalizing the cumulative DT-cost across the schedule and making more efficient use of the interconnect available.

The scheduling directive makes a DT correspond to a data transfer that will occur in the synthesized architecture. In addition, a DT neatly encapsulates a data broadcast. These allow Midas to make more accurate predictions regarding data accesses and transfers which in turn helps to guide Midas's synthesis algorithms. This encapsulation of data broadcasts is difficult to do in the conventional operation-based model for HLS.

DT binding is tightly coupled with floorplanning and combines operation binding and data binding. These are usually decoupled in the operation-based model. When a DT is bound, its operations are bound to functional units

and the data itself to terminals on those functional units. If the DT is a secondary DT, the data must be bound to a storage element and the storage read or write to a port of the storage element. Finally, the transfer of data must be bound to a bus in the architecture. Binding a DT completely specifies the routing of data in the architecture at a structural level. Minimizing the area of this routing helps minimize overall ASIC area.

4 Midas

Midas performs time-constrained HLS and minimizes ASIC area given an upper bound on the execution time for the behavior. It synthesizes its architectures incrementally in the body of a synthesis loop and uses deterministic algorithms. The hardware model it uses is shown in Figure 4. The execution unit is made up of functional units, the storage unit is made up of distributed, multi-ported register files, and the data-transfer subsystem is a network of buses connected to the components through multiplexers. These components are spatially integrated which means that functional units are placed close to the register files with which they interact and buses tend to be many, but short. Midas outputs both an architecture for the behavior and a high-level floorplan for the architecture.

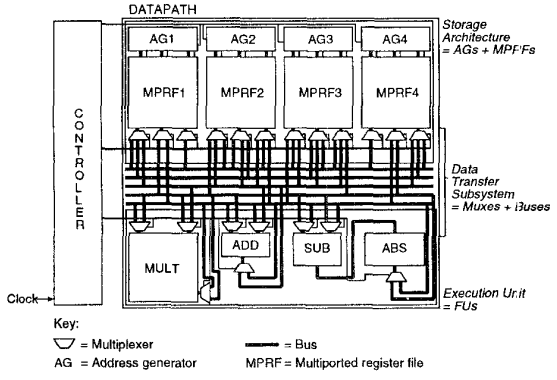

Figure 4: Midas's hardware model

Figure 5 illustrates the synthesis flow in Midas. The DFG is first transformed into the DT-domain and then Midas enters its main synthesis loop. Each time through the loop design metrics are evaluated, DTs are partitioned, and a single DT is selected, scheduled and bound. At the end of each loop, the partial design is floorplanned.

Midas uses two floorplanners – a global floorplanner and an incremental floorplanner. The global floorplanner completely rebuilds the floorplan. It is used at the end of each synthesis loop to generate a floorplan for the partial architecture. This floorplan is used in the next iteration through the synthesis loop in the evaluation of design metrics and as an input to the incremental floorplanner. The incremental floorplanner modifies an existing floorplan based on changes in the architecture caused by recent scheduling and binding actions. It is used to guide the DT binding algorithm.

The floorplan generated by the global floorplanner is used to compute the projected ASIC area and DT-costs, both of which are critical in guiding DT partitioning, scheduling and binding.

Projected ASIC area represents an estimate of the area of the final architecture. Midas starts with the existing partial floorplan and pads it to account for additional resources that Midas expects will be required by the final architecture. The additional resource utilization is computed using distribution graphs that Midas maintains.

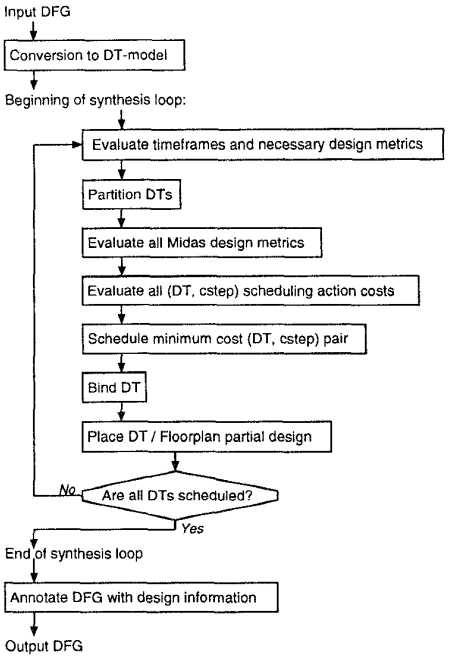

Figure 5: HLS flow in Midas

Computing the cost of a DT requires that the area of interconnect needed by the DT be known. For a bound DT, this is the area of the bus to which the DT is bound which can be acquired from the routing specification of the bus in the partial floorplan. For an unbound DT, Midas uses the worst-case scenario that the DT requires a bus that spans the diagonal of the partial floorplan.

DT-costs guide DT partitioning in deciding whether to partition a DT and how to partition it. DTs that would have to be scheduled in csteps which already have high cumulative DT-costs are partitioned to allow the secondary DTs generated to be scheduled at csteps with lower cumulative DT-costs. This prevents DT congestion in any part of the schedule.

During DT selection and scheduling, Midas evaluates the effect of each potential scheduling action on the projected ASIC area and cumulative DT-costs. Every DT can be scheduled at certain csteps in the schedule. Each such (DT, cstep) pair is a potential scheduling action. Among other things, Midas attempts to minimize the increase in the projected ASIC area and cumulative DT-costs caused by the selected scheduling action.

The incremental floorplanner is used in DT binding. Midas uses a branch-and-bound heuristic to bind a DT to functional units, storage elements, terminals, ports, and a bus. The cost function guiding the branch-and-bound search is the area returned by the incremental floorplanner. This area is a quick estimate of the impact of the suggested binding actions on the area of the floorplan. The incremental floorplanner allows Midas to favor binding a data-transfer to components that are close to one another.

Resource allocation is bundled into the binding algorithm. In addition to having the option of binding a DT to available resources, Midas presents the binder with the choice of introducing a new resource. The incremental floorplanner computes if doing so will result in a lower overall floorplan area due to a reduction in interconnect area. This enables the binder to trade redundant components for area, something that is impossible without floorplanning information.

5 Global floorplanner

The global floorplanner is based on McFarland's Fast Floorplanner [8]. The inputs to the floorplanner are a list of blocks in the architecture, a list of possible shapes for each block, a list of buses in the architecture, and a list of blocks that each bus is connected to. A block can be a functional unit or a register file. Multiplexers in the architecture are assumed to be bundled into the shape list for the block to which they are connected. The floorplanner outputs the coordinates and selected shape of each block and a routing specification for each bus.

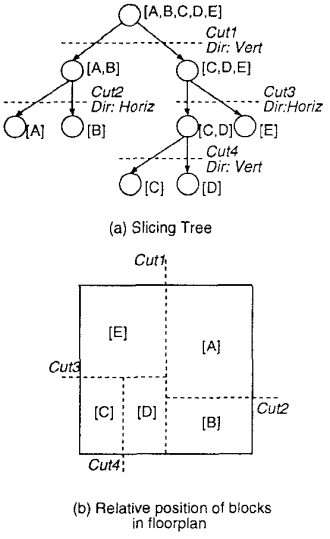

Figure 6: The relation of a slicing tree to the floorplan

The global floorplanner uses a slicing-tree mechanism. The set of blocks is bipartitioned recursively until each set contains only one block. The sets formed in the bipartitioning sequence form the nodes of the slicing tree. Each of the sets formed during recursive bipartitioning corresponds to a section of the ASIC floorplan. The two sets formed by any bipartitioning action lie on opposite sides of a "cut" on the floorplan. Figure 6 illustrates a slicing tree and its relation to a floorplan. The floorplanner uses the Kernighan-Lin [9] bipartitioning algorithm and minimizes the number of buses that connect blocks on the two sides of a cut. This has the effect of placing interconnected blocks close to one another on the floorplan. For each cut, the floorplanner decides which buses must be broadcast over the cut based on the whether the bus is entirely contained on one side of the cut or not. Buses that must be broadcast are allocated routing channels. A shape is selected for each block in the floorplan so as to minimize the overall area of the floorplan. This is done by propagating combinations of possible shapes for the blocks upwards from the leaves of the slicing tree [8]. When the possible shapes of two sides of a cut are combined, additional area is added to the combined shapes to account for routing channel allocations at the cut. Once the shapes have been computed, the floorplanner traverses the tree from top to bottom, computing the coordinates of each block and each routing channel. The result is block placement *and* bus placement in the form of the shape and coordinates of each block and the global routing channel specifications for each bus. Figure 7 shows a possible floorplan for the slicing tree in Figure 6.

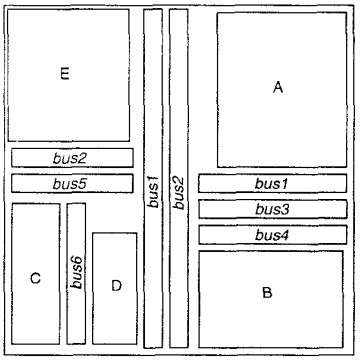

Figure 7: A sample floorplan

The global floorplanner is called at the end of Midas's synthesis loop and is used to update the partial floorplan for changes made in the architecture during an iteration of the synthesis loop. The floorplan generated by the global floorplanner at the end of the last iteration of the synthesis loop becomes the high-level floorplan output by Midas. The partial floorplan generated by the global floorplanner is used to evaluate the design metrics that guide DT partitioning, scheduling, and binding in Midas. The area of the partial floorplan is used as the starting point in computing the projected area of the final ASIC architecture. The area of a bus in the floorplan is available from the routing channel specification and is used as the area component of the DT-cost of DTs bound to the bus. The cumulative DT-cost is used to compute the scheduling cost and to guide DT-partitioning. The partial floorplan is also used as an input to the incremental floorplanner that is used by the DT binding algorithm. The partial floorplan generated by the global floorplanner is integral to the HLS flow in Midas and Midas's ability to synthesize architectures whose components are spatially integrated.

6 Incremental floorplanner

The incremental floorplanner modifies a floorplan generated by the global floorplanner to accomodate changes in the architecture. Changes include newly allocated blocks and buses, changes to the shapes of blocks due to the growth in size of register files or multiplexers, or terminals added to existing buses. The incremental floorplanner operates on the slicing tree generated by the global floorplanner. It first replaces all the blocks in the slicing tree that have been changed, and updates information at all cuts to reflect any changes to the connectivity of existing buses or any new buses. Finally, each newly allocated block is placed in the

slicing tree to minimize the increase in buses that cross cuts. After the slicing tree has been updated, bus broadcasting, shape generation and coordinate computation is performed in the same way it was in the global floorplanner.

The incremental floorplanner is used in Midas's DT binding algorithm to guide the branch-and-bound search. In the course of the search, binding actions are evaluated. Each binding action can change the shape functions of components in the architecture or the connectivity of buses. These changes are input to the incremental floorplanner along with the partial floorplan from the previous iteration of the synthesis loop. The area returned by the incremental floorplan guides the branch-and-bound search. The DT binding selected is the one that results in minimum floorplan area.

The incremental floorplanner is critical to Midas's resource allocation strategy. Unlike many other HLS systems, Midas does not allocate a minimum number of resources. Instead, Midas allocates resources to minimize floorplan area. Sometimes adding extra resources can lead to lower interconnect requirements and this reduces the overall area of the floorplan. Minimizing the number of functional units, storage elements, and buses typically leads to buses that are long, that have many terminals and consume large areas of the floorplan. The incremental floorplanner allows Midas to trade extra resources for a decrease in overall area.

7 Incremental versus global floorplanning

The global and incremental floorplanners serve very different functions and are therefore designed very differently. The global floorplanner generates the partial floorplan that is used to evaluate design metrics that guide DT partitioning, scheduling, and binding in subsequent iterations of the synthesis loop. It is important that it generates a good floorplan that converges to the final high-level floorplan. It is called only once in each iteration of the synthesis loop and, therefore, the quality of its result is more important than its speed. The incremental floorplanner is used to provide a one-step lookahead function to guide the binding algorithm. It is called once for every node in the binder's branch-and-bound search tree. It is therefore imperative that the incremental floorplanner be fast. Since the incremental floorplanner's output is not used as the partial floorplan, its quality is not as critical as the floorplanner's speed as long as the quality is not excessively poor. The global floorplanner is called after binding has been performed to "refresh" the floorplan.

While the incremental floorplanner uses a simple greedy heuristic, we find that for the one-step lookahead it needs to perform, it is adequate in guiding the DT binding algorithm. In most cases, the area of the floorplan generated by the incremental floorplanner during DT binding matches the area of the partial floorplan obtained during the subsequent run of the global floorplanner. The actual topologies of the floorplans may differ but the areas are usually very close.

The combination of incremental and global floorplanners in Midas allows floorplanning to be integrated in the HLS flow without sacrificing speed or quality. The DT-model used by Midas exploits this spatial information and guides the synthesis algorithms to the spatially integrated datapath architecture Midas seeks to generate.

8 Results

We tested Midas by synthesizing architectures and floorplans for each of five input DFGs: an arithmetic expression, a biquad filter, the differential equation solver benchmark, the elliptic filter benchmark, and a motion estimator. For each input, we varied the maximum allowed length of the schedule between its minimum possible and 30 control steps. The measure of merit for a synthesized architecture was the area of its high-level floorplan. The area was expressed as a multiple of the unit area in the scalable technology library used.

In order to provide a control against which to evaluate Midas, we constructed an alternate HLS system as a competitor for Midas. This HLS system used the same scheduling algorithm as Midas except it did not use the DT-model or floorplanning information. Binding and floorplanning were secondary steps. Register allocation and merging minimized the number and sizes of register files needed in the architecture. Functional unit allocation minimized the numbers of each type of functional unit. Operation and variable binding were done with the aim of reducing the number of inputs to the multiplexers at the terminals of functional units and ports of register files in the architecture. Bus allocation and data-transfer binding were done to minimize the number of buses and the number of terminals on each bus. The final step was floorplanning, using the global floorplanner.

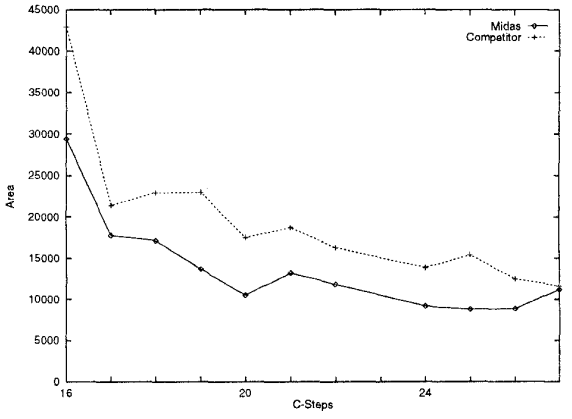

Figure 8: Results of elliptic filter runs

Figure 8 shows the results of the runs on the elliptic filter benchmark. In the graph, the solid line represents the areas of the architectures synthesized by Midas. The dotted line corresponds to architectures synthesized by the competitor. As can be seen, Midas performs better consistently. A closer examination of the floorplans synthesized showed that Midas opted for more buses than the competitor system did, but ensured they were short ones. In addition, Midas traded a larger storage unit for shorter data-transfers. The buses in the architectures generated by the competitor system were few, but had many more terminals than the buses in Midas's designs. The sum of the areas of the storage elements was smaller than the sum of register file areas in Midas, but extra routing was required to reach them. The execution units synthesized by both systems had identical numbers of components. Midas was able to exploit spatial locality between storage and functional units while scheduling and binding. The synthesis runs on the elliptic filter benchmark clearly showed the advantages of floorplanning information used early in HLS and underlined the potential of Midas's approach.

Figure 9 shows the results of runs on the motion estimator. Even though the results are not as dramatic as the

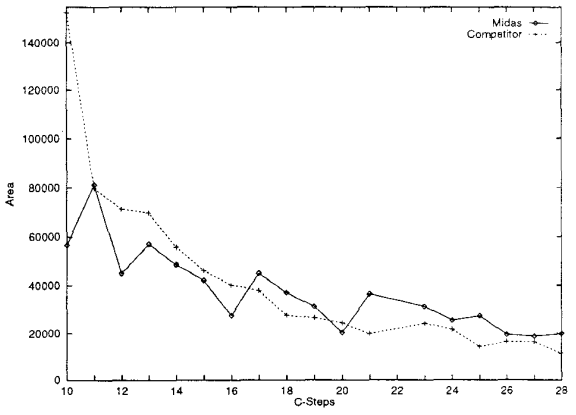

Figure 9: Results of motion estimator runs

ones obtained for the elliptic filter benchmark, we see Midas performs better for architectures with smaller schedule lengths. We observed this trend in architectures for all of the other DFGs. Midas synthesizes smaller floorplans when the schedule length is constrained to be very short.

Shorter schedules require the synthesized architecture to exhibit greater parallelism, in turn implying a larger number of components and buses. Midas exploits floorplanning information to schedule and bind DTs so that communicating components are close to one another. This reduces routing area and produces a more compact floorplan. The margin for this type of improvement increases as the number of components and, therefore, the binding search space grows. As a result, we see that Midas tends to perform better for architectures with shorter schedule lengths.

Table 1 shows the resource counts and floorplan areas for architectures with different schedule lengths that were synthesized for the elliptic filter example by both Midas and the competitor. The areas of I/O pads, adders and multipliers in our technology library tend to be smaller than those of register files and interconnect that the architectures require. In most of the architectures, we see that Midas has a higher number of these smaller resources. Midas trades this for lower register file and interconnect area. Midas's integration of floorplanning information and its elevation of data issues in HLS allows it to make this tradeoff early in the HLS flow, thereby reducing floorplan area. In the architectures for the first two schedule lengths, we see that the only difference in resource counts is that Midas's architectures have a larger number of buses. It seems counter-intuitive that their floorplan areas are smaller. On a closer examination, we found that the buses in Midas's architectures are short ones with half of them being connected to only two components. The overall effect is a reduction in floorplan area.

Custom architectures for high-throughput, data-intensive applications motivated our focus on data-issues and physical design characteristics. These applications typically have tight execution time constraints. Our results indicate that the use of the DT-model and floorplanning enables Midas to generate smaller floorplans, especially for behaviors that must execute in short lengths of time as is often required by high-throughput, data-intensive applications.

CSteps	I/O		Add		Mul		Bus		Area	
	M	C	M	C	M	C	M	C	M	C
16	4	4	4	4	2	2	18	12	29412	42912
17	4	4	4	3	2	2	11	10	17760	21402
18	4	3	4	3	2	2	10	10	17136	22952
19	4	3	3	3	2	2	9	9	13680	22995
24	3	2	2	1	2	2	8	7	9165	13860
26	2	2	2	2	2	1	6	7	8844	12485
27	2	1	2	2	2	1	7	9	11154	11596

Key:
M = Midas-generated architecture
C = Competitor-generated architecture

Table 1: Architectures for elliptic filter

9 Conclusions

We have presented a floorplanning approach to HLS in the form of our HLS system, Midas. Midas formulates HLS using the DT-model, which allows Midas to encapsulate data broadcasts and accesses and to incorporate floorplanning in the main HLS flow, making it an integral part of DT partitioning, scheduling and binding. The global floorplanner in Midas performs a slower, but more optimal floorplanning at the end of each synthesis iteration while an incremental floorplanner provides a fast, but acceptably accurate prediction for effects of synthesis actions on the floorplan. The combination of the two allows floorplanning to be included in the HLS flow while still ensuring quality and speed. Midas generates an architecture for the input behavior and a high-level floorplan for the architecture that contains the shapes and placements of all the components and routing channels. The architectures Midas synthesizes tend to have many short buses instead of a few long ones and tend to trade extra components for a reduction in overall area. The floorplanning approach that Midas uses in conjunction with the DT-model results in lower area floorplans, especially when synthesizing architectures for designs with tight schedule length constraints.

References

[1] S. Tarafdar and M. Leeser, "The DT-Model: High-Level Synthesis Using Data Transfers," in *Proceedings of the 35th ACM/IEEE Design Automation Conference*, 1998.

[2] S. Tarafdar, *A Data-Transfer Model For High Level Synthesis And Its Application In Storage And Interconnect Optimization.* PhD thesis, Cornell University, 1998.

[3] F. Kurdahi, D. D. Gajski, C. Ramachandran, and V. Chaiyakul, "Linking Register-Transfer and Physical Levels of Design," *IEICE Transactions on Information and Systems*, September 1993.

[4] M. C. McFarland and T. J. Kowalski, "Incorporating bottom-up design into hardware synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 9, pp. 938–950, September 1990.

[5] D. W. Knapp, "Fasolt: A Program for Feedback-Driven Data-Path Optimization," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 677–695, June 1992.

[6] J.-P. Weng and A. C. Parker, "3D Scheduling: High Level Synthesis with Floorplanning," *28th ACM/IEEE Design Automation Conference*, pp. 668–673, 1991.

[7] M. Rim, A. Majumdar, R. Jain, and R. De Leone, "Optimal and Heuristic Algorithms for Solving the Binding Problem," *IEEE Transactions on VLSI Systems*, vol. 2, pp. 211–225, June 1994.

[8] M. McFarland, "A Fast Floor Planning Algorithm for Architectural Evaluation," *Proceedings of the International Conference on Computer Design*, pp. 96–99, October 1989.

[9] W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technical Journal*, vol. 49, pp. 291–307, 1970.