Fast and Accurate Estimation of Floorplans in Logic/High-level Synthesis

Kiarash Bazargan Abhishek Ranjan Majid Sarrafzadeh

Department of Electrical and
Computer Engineering
Northwestern University
Evanston, IL 60208-3118

{kiarash,abhi,majid}@ece.nwu.edu

ABSTRACT

In many applications such as high-level synthesis (HLS) and logic synthesis and possibly engineering change order (ECO) we would like to get fast and accurate estimations of different performance measures of the chip, namely area, delay and power consumption. These measures cannot be estimated with high accuracy unless a fairly detailed layout of the chip, including the floorplan and routing is available, which in turn are very costly processes in terms of running time. As we have entered the deep sub-micron era, we have to deal with designs which contain million gates and up. Not only we should consider the area occupied by the modules, but we also have to consider the wiring congestion. In this paper we propose a cost function that is, in addition to other parameters, a function of the wiring area. We also propose a method, to avoid running the floorplanning process after *every* change in the design, by considering the possible changes in advance and generating a floorplan which is *tolerant* to these modifications, i.e., the changes in the netlist does not dramatically change the performance measures of the chip. Experiments are done in the high-level synthesis domain, but the method can be applied to logic synthesis and ECO as well. We gain speedups of 184% on the average over the traditional estimation methods used in HLS.

1. INTRODUCTION

Floorplanning is the highest level of the physical design process without which we cannot get estimations of different performance and cost measures of the design accurately. However, it is a very costly process in terms of running time. As we enter the deep-submicron (DSM) era, the designs get larger and interconnections become dominant in both the area and delay of the chip. These changes make the floorplanning a very difficult task. On the other hand, to make design decisions in the early phases of the design process, one must generate a floorplan to be able to estimate different measures of the chip.

An example of such early phases of the design is the logic synthesis process. In logic synthesis, operations such as restructuring, logic cloning and buffer insertion is applied to the design. Each of these operations might change the netlist dramatically, which in turn changes the layout significantly. We should have a fast and accurate way of assessing how the logic synthesis operations affect the layout. Another example of the early stages of the design which need floorplanning is the scheduling/binding phase in the high-level synthesis process. After each binding move the netlist might change and so would the congestion area and delay in the final layout.

In this paper we propose a method which generates a *tolerant* floorplan for a design. The area or delay of a *tolerant* floorplan would not change significantly with the changes in the netlist, because netlist changes were anticipated when generating the floorplan. By considering which blocks are likely to undergo logic synthesis changes (e.g., blocks located in a highly congested area or blocks in a path where timing constraints are not met) in advance, we can come up with a floorplan which is reasonably good for most likely moves. By taking into consideration the possible bindings of the operations to different resources (in high-level synthesis), we can weight the potential connections between any two resources based on the likelihood that such a connection is used in the final design, and do the floorplanning in a way that the nets with more probability end up having smaller length. Another example where the tolerant floorplans can be useful is the handling of the engineering change orders (ECO) which usually happens after we have done most of the design cycle. When an ECO, which is usually minor changes in the netlist, comes in we would like to change as little parts of the design as possible. If we had anticipated which units were more likely to have ECOs (units that were recently designed as opposed to IPs) when we were doing the floorplanning, then the generated floorplan most likely does not need many changes.

Throughout the paper, we have focused on the application of tolerant floorplans in high-level synthesis. Although we have only shown results in this field, our method can be used in other scenarios, some of which were discussed above. For example, our work can be directly used in [10].

High level synthesis, or HLS, has been the subject of many research studies in the area of VLSI CAD in the past decade, e.g., see [2; 4; 7]. Since it deals with the design in the algorithmic level, which is the highest level of the design process, it has the most profound impact on the final product's area and performance measures.

In deep submicron (DSM) design, parameters such as interconnect structure become dominant in the performance of the chip. A state-of-the-art CAD tool should be able to estimate layout parameters

such as area, delay and power early in the design process (scheduling, allocation and binding).

However, these measures cannot be estimated with high accuracy unless a fairly detailed layout of the chip, including the floorplan and routing is available. For this reason, some research studies have considered simultaneous floorplanning and scheduling/binding, e.g., [2; 6; 7]. Not all of them directly consider the contribution of the wires to the floorplan area.

Another HLS system which generates a floorplan during binding and scheduling processes is BINET [7]. Although BINET is fast, it has two drawbacks: (a) It generates floorplans only after binding the operations to resources at each time step. (b) Uses the method in [12] to estimate the wire area. [12] estimates the wiring area using statistical models or by comparing the design to those from previous projects which are fully routed. Obviously, these methods will not yield an accurate estimation of the wiring area.

[2] combines binding and floorplanning. It uses [11] as its floorplanner which does not directly consider area needed for wires. Instead, it uses a linear function of floorplan area (without wires) and the total wire length. Using this method, one cannot estimate the area taken by the wires connecting modules together.

The methods in [6] and [9] use force-directed floorplanners [5]. Calls to [5] are made after every scheduling, allocation or binding move. Simulated annealing methods are proved to generate more compact floorplans than force-directed ones. But as designs are getting more complex, it is less affordable to spend so much time, i.e. call the floorplanning process, for each HLS move.

In this work, we have proposed an estimation method for the area and other design parameters of the chip. We have also proposed a more accurate method for area estimation of the chip by considering the area contribution of the netlist more realistically. By considering the netlist prior to binding the nodes of the Data Flow Graph (DFG) to functional units, and the possible binding options, our system generates a tolerant floorplan. In this way, we can avoid running the floorplanning process after every HLS move. Since we know that the area does not change dramatically with a new move, we keep updating the floorplan area until it is more than x% the initial area, or y number of HLS moves have been done. (x and y will be determined experimentally.) Experimental results show that our method is 184% the speed of a traditional method on the average.

2. PROBLEM FORMULATION

Input to HLS is defined by a DFG (data flow graph) $G_{DFG} = (V, E_{DFG})$ where $V = \{v_i | i = 1, 2, \ldots, n_{ops}\}$ is the set of operations and E_{DFG} is the set of directed edges which defines the dependencies between operations.

$$E_{DFG} = \{(v_i, v_j) \mid \textit{the output of operation } v_i \textit{ is an input to operation } v_j\}$$

An example of a DFG can be found in Figure 1.

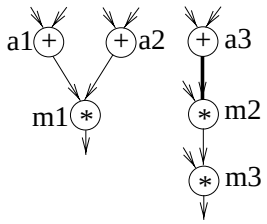

Figure 1: An example of a Data Flow Graph (DFG).

Let k be the number of operation "types" in the DFG. We define function $\mathcal{T} : V \rightarrow \{1, 2, \ldots, k\}$ to be the type of operations. Suppose we have decided to use at most a_i $(i = 1, 2, \ldots, k)$ instances of resource type i on the chip. The chip can be represented by *resource connection graph* $G_{chip} = (R, E_{chip})$ where R is the set of resource instances on the chip,

$$R = \{r_{1,1}, \ldots, r_{1,a_1}, r_{2,1}, \ldots, r_{2,a_2}, \ldots, r_{k,1}, \ldots, r_{k,a_k}\}$$

and

$$E_{chip} = \{(r_{t_1,i}, r_{t_2,j}) \mid \textit{there is a net on the chip connecting resources } r_{t_1,i} \textit{ and } r_{t_2,j}\}$$

An example of G_{chip} is shown in Figure 2b.

Operation v and resource $r_{t,i}$ are said to be "compatible" iff $\mathcal{T}(v) = t$, i.e., v can be implemented on $r_{t,i}$.

Definition 1. *The* Binding Graph G_{bind} *is a bipartite graph* (V, R, E_{bind}) *where*

$$E_{bind} = \{(v_i, r_{t,j}) \mid \mathcal{T}(v_i) = t \textit{ and operation } v_i \textit{ is scheduled to be performed on resource } r_{t,j}\}$$

A binding graph corresponding to the DFG in Figure 1 is shown in Figure 2a. Two adders and two multipliers are used. ($R_{1,1} = Add1$ $R_{1,2} = Add2$ $R_{2,1} = Mul1$ $R_{2,2} = Mul2$).

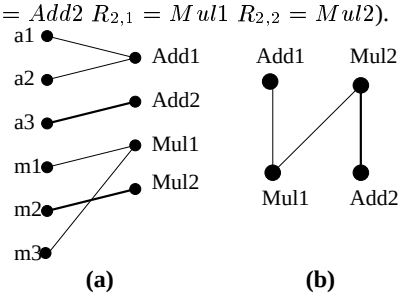

Figure 2: (a) A binding of the operations to resources. Two adders and two multipliers are used. ($R_{1,1} = Add1$ $R_{1,2} = Add2$ $R_{2,1} = Mul1$ $R_{2,2} = Mul2$). (b) The resource connection graph G_{chip} which corresponds to the binding shown on left, and the DFG in Figure 1.

Lemma 1. *The number of edges in* G_{bind} *is exactly* n_{ops}. *If* $(v_i, r_{t_1,m_1}) \in E_{bind}$ *and* $(v_j, r_{t_2,m_2}) \in E_{bind}$ *and* $(v_i, v_j) \in E_{DFG}$, *then there must be an edge* $(r_{t_1,m_1}, r_{t_2,m_2}) \in E_{chip}$.

The thick edges in Figures 1 and 2 show an example of the Lemma 1.

Definition 2. *Let function* $\mathcal{M} : E_{DFG} \rightarrow E_{chip}$ *be the* mapping *function of DFG edges to resource connections.* $\mathcal{M}((v_i, v_j)) = (r_{\mathcal{T}(v_i),m_1}, r_{\mathcal{T}(v_j),m_2})$ *iff* $(v_i, v_j) \in E_{DFG}$ *and* $(v_i, r_{\mathcal{T}(v_i),m_1}) \in E_{bind}$ *and* $(v_j, r_{\mathcal{T}(v_j),m_2}) \in E_{bind}$.

More than one DFG edge might be mapped to a resource connection. For example, both edges (a1,m1) and (a2,m1) in Figure 1 are mapped to resource connection (Add1,Mul1).

Assuming there is no preference in binding operations to resources, the probability of an operation v to be bounded to resource $r_{\mathcal{T}(v),i}$ is $\frac{1}{a_{\mathcal{T}(v)}}$. In other words,

$$p\left((v_i, r_{\mathcal{T}(v_i),j}) \in E_{bind}\right) = \frac{1}{a_{\mathcal{T}(v)}} \quad (1)$$

The probability that a DFG edge be mapped to a resource connection edge can be derived from Equation 1 as follows:

$$p\left(\mathcal{M}((v_i,v_j)) = (r_{\mathcal{T}(v_i),m_1}, r_{\mathcal{T}(v_j),m_2})\right) = \frac{1}{a_{\mathcal{T}(v_i)} a_{\mathcal{T}(v_j)}} \quad (2)$$

Lemma 2. *The probability that there is no connection between two resources in G_{chip} is equal to the probability that no two adjacent operations in G_{DFG} are scheduled to be performed on them.*

Using this lemma, we can calculate the probability that a net exists between resources R_{t_1,m_1} and R_{t_2,m_2} using Eq. 2.

$$p\left((R_{t_1,m_1}, R_{t_2,m_2}) \in E_{chip}\right) = 1 - \prod_{\substack{\mathcal{T}(v_i)=t_1,\\ \mathcal{T}(v_j)=t_2}} \left(1 - p\Big(\mathcal{M}\big((v_i,v_j)\big) = (R_{t_1,m_1}, R_{t_2,m_2})\Big)\right) \quad (3)$$

3. TOLERANT FLOORPLANS

In Section 3.1 we present an approximation for the area of a floorplan which considers the contribution of wires connecting resources as well as the area of the resources.
In Section 3.2 we show how we use probabilities calculated by Equation 3 to generate a floorplan which is tolerant to changes in the binding graph.

3.1 Area Contribution of Netlist

To estimate the area of a floorplan more accurately, we try to find the area contribution of the connections as well as the modules' area. We have used a method similar to what was used in [1] to estimate the area taken by wires.
Throughout the paper, we have focused only on slicing floorplans[11]. Furthermore, we have only considered simulated annealing as the process of floorplanning[11], because it has proved to be the best among all other floorplanning methods.
Typically, a net is routed within the rectangle whose bottom-left and upper-right corners are the centers of resources which are connected by the net. We call this rectangle "bounding box" of a net: $\mathcal{B}(net)$[1]. The dashed rectangle in Figure 3 is the bounding box of the net (3,5).

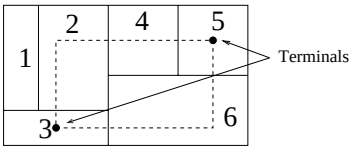

Figure 3: Bounding box of a net.

Also, we can safely assume that when the routing path of a net passes through the boundary of a resource, its area contribution to the chip can be estimated by increasing the width or height of the resource by 1. For example, if a 40×30 resource is enclosed in a 40×40 block and a bus with width 6 is routed horizontally through the block, then we consider the area contribution of the bus to the resource to be 40×6, which updates the dimensions of the resource to 40×36. In this case, the routing did not change the enclosing box of the resource.

[1] If resources are the same (e.g., the result of an adder is to be fed back to a register at its input), then $\mathcal{B}(net)$ is set to be the bounding box of the resource.

Since we don't know which blocks will be selected to route different nets, we have to use probabilistic methods to guess what the path of a net would be. There are several models which can be used. Figure 3 shows the bounding box of a net. We assume that the routing is done within the bounding box. To formulate different probabilistic models for the routing path we have to define a few terms first.

Definition 3. *Assume that a grid $G_{grid} = (V_{grid}, E_{grid})$ is superimposed on the floorplan. The edges connecting the grid points are called* segments. *The set of possible routing segments of a net through a module's bounding box is:*

$$\mathcal{S}\left((r_{t_1,m_1}, r_{t_2,m_2}), r_{t_3,m_1}\right) = \Big\{(v_1,v_2) \in E_{grid} \Big| v_1 \text{ and } v_2 \in \mathcal{B}\big((r_{t_1,m_1}, r_{t_2,m_2})\big) \bigcap BBox(r_{t_3,m_1})\Big\} \quad (4)$$

where $BBox(r_{t,m})$ is the bounding box of the resource $r_{t,m}$. The bounding box of a resource is the rectangle in the hierarchical slicing blocks which contains only that resource. The width/height of the bounding box of a resource is greater than or equal to the width/height of the resource.

There are several probability distributions that we can use to estimate whether a segment is used in routing a net or not. Three of them are shown in Figure 4. Figure 4-a shows a uniform distribution. This model assumes that the probability that the routing passes any of the segments is constant and a function of the area of the intersection of the bonding boxes of the net and the module. Figure 4-b shows a better model. In this model it is more likely for the net to be routed using segments near the border of the bounding box than using those near the center. Figure 4-c shows a more realistic model which assumes that all routings are done using at most two bends.

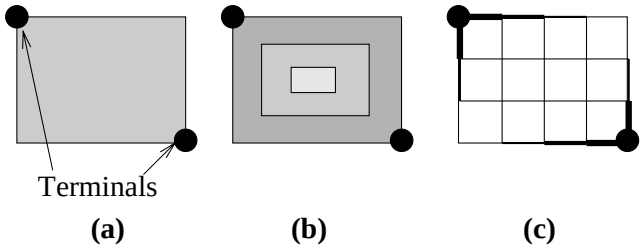

Figure 4: Different routing distribution functions. (a) uniform (b) near the net boundry routing, (c) two-bend routing.

The distribution in Figure 4-c can be calculated by considering all possible ways of routing the net using L-shaped or Z-shaped routings, and dividing number of different routes passing through a segment by the total number of routes.
Assuming that we are using any of the distribution functions in Figure 4, the area contribution of a net to a module is defined as:

Definition 4. *Let $p(s)$ be the probability that a segment is used in routing of a net. Also, let S_h and S_v be the set of horizontal and vertical segments in S respectively. The area contribution of a net (with width 1) to a resource's area is defined as functions $\mathcal{A}_w : E_{chip} \times R \to [0,1]$ and $\mathcal{A}_h : E_{chip} \times R \to [0,1]$*

$$\mathcal{A}_w\left((r_{t_1,m_1}, r_{t_2,m_2}), r_{t_3,m_1}\right) = \sum_{s \in S_v((r_{t_1,m_1}, r_{t_2,m_2}), r_{t_3,m_1})} p(s) \quad (5)$$

$$\mathcal{A}_h\left(\left(r_{t_1,m_1}, r_{t_2,m_2}\right), r_{t_3,m_1}\right) = \sum_{s \in S_h((r_{t_1,m_1}, r_{t_2,m_2}), r_{t_3,m_1})} p(s) \quad (6)$$

Calculation of functions $\mathcal{A}_w$ and $\mathcal{A}_h$ for each net and resource pair can be done in constant time, because one only needs to know the dimensions of the intersection of their bonding boxes, and if the intersection is on the boundary of the net bounding box.

The area of a floorplan can be estimated by updating the width and height of every resource $r_{t,m}$ as:

$$width(r_{t,m}) \leftarrow width(r_{t,m}) + \sum_{(r_{t_1,m_1}, r_{t_2,m_2}) \in E_{chip}} \mathcal{A}\left(\left(r_{t_1,m_1}, r_{t_2,m_2}\right), r_{t,m}\right) \quad (7)$$

$$height(r_{t,m}) \leftarrow height(r_{t,m}) + \sum_{(r_{t_1,m_1}, r_{t_2,m_2}) \in E_{chip}} \mathcal{A}\left(\left(r_{t_1,m_1}, r_{t_2,m_2}\right), r_{t,m}\right) \quad (8)$$

To update the area of a floorplan after insertion or deletion of a net, one can add or subtract the area contribution of the net to each resource on the chip, and then use floorplan sizing to find the chip area.

Although this updating process takes linear time (in terms of number of resources on the chip), it is fairly fast. The reason is that the number of resources on the chip is small at scheduling and binding phases of the design. Furthermore, we can use this updating process after each floorplanning move, because each floorplanning move calls floorplan sizing to calculate the area. By updating the area contribution of the nets, we are not slowing down the floorplanning process.

We have compared our method of estimation of the wiring area with traditional methods in Section 4.1.

3.2 Generating Tolerant Floorplans

Consider the hierarchy of blocks which represent a slicing floorplan. (For a definition of a slicing floorplan, refer to [11].) Each resource, or module, is bounded to a rectangular area, or bin. Dimensions of the bin are more than or equal to the resource dimensions. If a net passes this bin, we estimate its area contribution by increasing width and height of the resource by the amount defined in Definition 4. If the inflated resource still fits in the bin, then the area of the floorplan does not change because of this part of the routing. We refer to this situation as "hiding of a net by a resource". We have tried to generate a floorplan which "hides" as many routings as possible. Such a floorplan will most probably have the same area if we insert or delete some nets.

By anticipating which resources are likely to be connected, we can generate a probabilistic netlist which contains information about the nets, and how likely it is that each net is used. Such a probabilistic netlist is generated using Equation 3.

Using the probabilistic netlist, we run a simulated annealing floorplanning process. To calculate the area of the chip during the floorplanning, we use a combination of Equations 7 and 8 and 3:

$$width(r_{t,m}) \leftarrow width(r_{t,m}) + \sum_{(r_{t_1,m_1}, r_{t_2,m_2}) \in E_{chip}} \Big[p\left((R_{t_1,m_1}, R_{t_2,m_2}) \in E_{chip}\right) * \mathcal{A}\left(\left(r_{t_1,m_1}, r_{t_2,m_2}\right), r_{t,m}\right)\Big] \quad (9)$$

$$height(r_{t,m}) \leftarrow height(r_{t,m}) + \sum_{(r_{t_1,m_1}, r_{t_2,m_2}) \in E_{chip}} \Big[p\left((R_{t_1,m_1}, R_{t_2,m_2}) \in E_{chip}\right) * \mathcal{A}\left(\left(r_{t_1,m_1}, r_{t_2,m_2}\right), r_{t,m}\right)\Big] \quad (10)$$

It is possible though, that one uses a combination of Equation 3 and other methods to calculate the wiring cost. For example, to modify the Wong-Liu algorithm [11] to handle probabilistic netlists, all we have to do is to multiply the half-perimeter length of the wires' bounding box to their probabilities and use the sum of all such terms as part of the cost function.

The result of the floorplanning process is called FP_0. Intuitively, this floorplan has minimum area for the most probable cases. This floorplan is likely to be "tolerant" to changes in the netlist, meaning that changes in the routing will most probably not change its area.

After FP_0 is generated, our system translates each binding move to a list of insertion and deletion of nets. Examples of binding moves are swapping two operations of two resources, or moving an operation from one resource to another one. It is possible that a binding move introduces zero or more insertions, and zero or more deletions, depending on current netlist.

Each binding move is applied to the floorplan by inserting or deleting corresponding nets and calculating the floorplan area using Equation 7 and 8 (and *NOT* Equation 9 and 10). It is possible that a binding move isolates a resource from the rest of the chip. This situation happens when only one operation is assigned to a resource, and the binding move assigns the operation to a different resource. The first resource has no operation to perform, and hence it can be removed from the chip. Conversely, it is possible that a binding move reinserts a previously removed resource to the chip by moving an operation from another resource to it. In both cases, a new call to floorplanning process should take place. Otherwise, if the difference between area of the new floorplan and FP_0's area is less than ϵ% (we chose $\epsilon = 2$ in the experiments.) of FP_0's area, we do not need to perform a new floorplanning. This intuitively means that the floorplan is still consistent with the initial prediction of the netlist.

Before running the floorplanning process again, we have to update the probabilities of the netlist so that the new floorplan is different with FP_0. This can be easily done by "biasing" the probabilities of edges in the binding graph. Instead of using equal probabilities for the assignment of operations to resources (as in Equation 1), we can update the probabilities of assignment of operations to resources such that it is more likely that an operation is assigned to a resource which it is currently assigned to (in the last floorplan, just before redoing the floorplanning).

We have run experiments to compare this method with other traditional methods. The result of these experiments is shown in Section 4.2.

4. EXPERIMENTAL RESULTS

This section, which consists of two subsections, shows the result of our experiments. Section 4.1 shows the result of experiments comparing our model for area contribution of the wires to the traditional

Data Set	A	B	C
Data1	106.37%	101.17%	93.78%
Data2	125.87%	107.44%	100.11%
Data3	170.93%	106.21%	107.24%
Data4	165.67%	101.95%	99.49%

Table 1: Ratio of the areas of floorplans generated by different methods. All the area comparisons are based on what TimberWolf reports after global routing. Please refer to Section 4.1 for description of different columns.

methods. Section 4.2 compares the speed of an HLS system using "tolerant" floorplans to one which uses traditional floorplanners.
We have compared tolerant floorplans to two traditional methods. The two methods are called "Simple" and "Wong-Liu"[11]. The Simple method does not consider netlist at all, and just tries to arrange the resources in a way that the area is minimized. The Wong method uses sum of wire length of nets as part of the cost function used in simulated annealing process. By including a term corresponding to wire lengths to the cost function, Wong-Liu method tries to place highly connected resources closer to avoid running wires all over the chip.

4.1 Wiring Area Estimation

To compare our model for the wiring area to the traditional models, we implemented the Wong-Liu's simulated annealing floorplanner, but used three different cost functions: one consisting of only area ("Simple"), another one consisting both of area and the wire lenght ("Wong-Liu") and finally, one which combines the area of the modules and the area contribution of the wires as formulated in Equations 9 and 10 ("Tolerant"). Then we ran TimberWolf's global router on the floorplans generated by each of the methods and compared the final area. We allowed TimberWolf to do compaction before routing.
The area of these floorplans are shown in Table 1. We have done experiments on high-level synthesis benchmarks (see Section 4.2 for a description of them); however, since they are small, we have used our own circuits for the purpose of area comparison. The circuits contain about 30 modules with varying number of nets, ranging from 20 to 50.
Column "A" shows the ratio of the area of the simple method to the area of the tolerant floorplan, both after global routing is done using TimberWolf.
Column "B" shows the ratio of the area of the Wong-Liu method to the area of the tolerant floorplan, both after global routing is done using TimberWolf.
Column "C" shows the ratio of the estimated area (estimated by Equations 9 and 10) to the actual area reported by TimberWolf after global routing. It shows that on the average, the tolerant floorplanner can estimate the wire area with 2% error.

4.2 Tolerant Floorplans

To compare our method with the traditional methods, we have used three behavioral model HLS benchmarks, "diffeq"[8], "FIR"[2] and "ellipf"[3]. For resource dimensions, we have used two sets of libraries[2] for the resources. The libraries are shown in Table 2. Note that in Table 2, the unit of area is reported in square micron.
"Diffeq" is the hardware description for an algorithm which tries to numerically solve the differential equation $y'' + 3xy' + 3y = 0$. The DFG of this model is shown in Figure 5. Two adders and two multipliers are used to schedule the algorithm. The FIR filter is scheduled with 3 adders and 4 multiplier units. Its DFG is shown in Figure 6.

Library 1		
Fab Tech	16-bit Adder	16-bit Multiplier
1.6μm	746875	8711250
1.2μm	420000	4900000
Library 2		
Fab Tech	8-bit Adder	8-bit Multiplier
1.2μm	46875	564375
1.0μm	21250	261875

Table 2: Area of different library modules. Numbers reported in the table are measured in square micron.

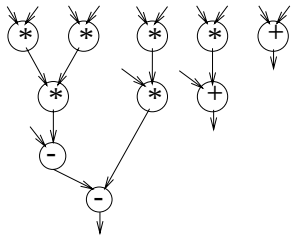

Figure 5: The DFG of diffeq benchmark.

Five adders and two multipliers were used for the elliptic filter. The DFG of "ellipf" is shown in Figure 7.
To compare different methods of floorplanning, we let the floorplanning algorithm generate an initial floorplan FP_0. The area of FP_0 is computed using Equations 7 and 8. Then we start applying binding moves to the floorplan and compute the area using the same equations, until the new area is ϵ% (We used $\epsilon = 2$ in our experiments.) more than the area of FP_0.
We report the number of binding moves applied to the floorplan as a measure of speed of the method. The greater the number of moves, the less we need to call the floorplanning process and hence the overall HLS process will be faster.
The result of the experiments can be found in Table 3. We have not reported the "Diffeq" results because none of the methods stopped after 500 moves, meaning that the circuit is too small for the wiring to change the are significantly.

5. CONCLUSION AND FUTURE WORK

We have proposed a floorplanning method which generates tolerant floorplans for a set of possible binding moves. The results show that anticipating such binding moves is very effective. An improvement to our method is the formulation of the scheduling constraints into binding probabilities. By considering time constraints, the probabilities would be more exact and hence the generated floorplans will be more reliable for area and other parameters.

6. ACKNOWLEDGMENTS

This research has been sponsored in part by a grant from the Defense Advanced Research Projects Agency under contract number

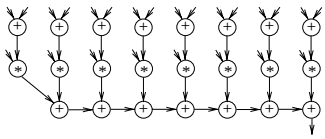

Figure 6: The DFG of FIR benchmark.

Benchmark	Simple	Wong	Tolerant
ellipf lib1 1.6μ	16	3	71
ellipf lib2 1.0μ	3	4	4
fir23 lib1 1.6μ	46	62	152
fir23 lib2 1.0μ	6	3	3
fir34 lib1 1.6μ	44	212	327
fir34 lib2 1.0μ	9	6	13
fir74 lib1 1.6μ	140	137	238
fir74 lib2 1.0μ	13	22	20
Average	34.63	56.13	103.50
Ratio	299%	184%	100%

Table 3: Number of binding moves to apply to different floorplans so that the area exceeds by 2% the initial area. FirXY is the FIR filter implemented with X adders and Y multipliers.

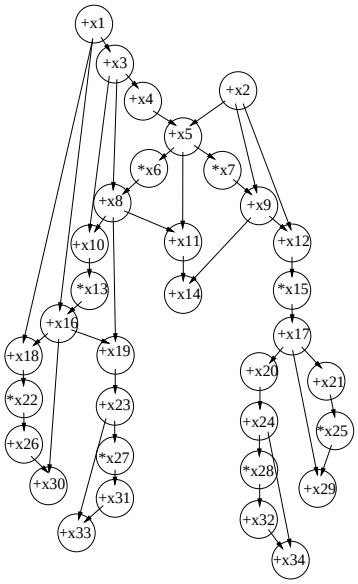

Figure 7: The DFG of ellipf benchmark.

DABT63-97-C-0035, and the National Science Foundation under grant 9527389.

7. REFERENCES

[1] W. E. Donath, R. J. Norman, B. K. Agrawal, and S. E. Bello. “Timing Driven Placement Using Complete Path Delays”. In *Design Automation Conference*, pages 84–89. IEEE/ACM, 1990.

[2] Y. Fang and D. F. Wong. “Simultaneous Functional-Unit Binding and Floorplanning”. In *International Conference on Computer-Aided Design*, pages 317–321, 1994.

[3] I. G. Harris and A. Orailoglu. “Microarchitectural Synthesis of VLSI Designs with High Test Concurrency”. In *Design Automation Conference*, pages 206–211, 1994.

[4] H. Jang and B. M. Pangrle. “A Grid-Based Approach for Connectivity Binding with Geometric Costs”. In *International Conference on Computer-Aided Design*, pages 94–99. IEEE, 1993.

[5] Y. Mori, V. Moshnyaga, H. Onodera, and K. Tamaru. “A Performance-Driven Macro-Block Placer for Architectural Evaluation of ASIC Designs”. In *Proceedings fo the 8th Annual IEEE International ASIC Conference and Exhibit*, 1995.

[6] V. G. Moshnyaga and K. Tamaru. “A Placement Driven Methodology for High-Level Synthesis of Sub-Micron ASIC’s”. In *International Symposium on Circuits and Systems*, pages 572–575. IEEE, 1996.

[7] A. Mujumdar, M. Rim, R. Jain, and R. De Leone. “BITNET: An Algorithm for Solving the Binding Problem”. *7th International Conference on VLSI Design*, pages 163–168, 1994.

[8] P. G. Paulin, J. P. Knight, and E. F. Girczyc. “HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis”. In *Design Automation Conference*, pages 263–270, 1986.

[9] P. Prabhakaran, J. Crenshaw, P. Banerjee, and M. Sarrafzadeh. “Simultaneous Scheduling, Binding and Floorplanning for Interconnect Power Optimization”. pages 428–434, January 1999. Proceedings of 1999 VLSI Design Conference, India.

[10] A. H. Salek, J. Lou, and M. Pedram. “A DSM Design Flow: Putting Floorplanning, Technology-Mapping and Gate-Placement Together”. In *Design Automation Conference*, pages 128–133. IEEE/ACM, 1998.

[11] D. F. Wong and C. L. Liu. “A New Algorithm for Floorplan Design”. In *Design Automation Conference*, pages 101–107, 1986.

[12] G. Zimmerman. “A New Area and Shape Function Estimation Technique for VLSI Layouts”. In *Design Automation Conference*, pages 60–65. IEEE/ACM, 1988.