Layout-driven RTL Binding Techniques for High-Level Synthesis

Min Xu†
Fadi J. Kurdahi‡

Technical Report #96-11
April 25, 1996

†Department of Information and Computer Science
‡Department of Electrical and Computer Engineering
University of California, Irvine
Irvine, CA 92717-3425
(714) 824-8104

mxu@ics.uci.edu
kurdahi@ece.uci.edu

Abstract

The importance of effective and efficient accounting of layout effects is well-established in High-Level Synthesis (HLS), since it allows more realistic exploration of the design space and the generation of solutions with predictable *metrics. This feature is highly desirable in order to avoid unnecessary iterations through the design process. In this paper, we address the problem of layout-driven register-transfer-level (RTL) binding as this step has a direct relevance on the final performance of the design. By producing not only an RTL design but also an approximate physical topology of the chip level implementation, we ensure that the solution will perform at the predicted metric once implemented, thus avoiding unnecessary delays in the design process.*

Layout-driven RTL Binding Techniques for High-Level Synthesis

Abstract

The importance of effective and efficient accounting of layout effects is well-established in High-Level Synthesis (HLS), since it allows more realistic exploration of the design space and the generation of solutions with *predictable* metrics. This feature is highly desirable in order to avoid unnecessary iterations through the design process. In this paper, we address the problem of layout-driven register-transfer-level (RTL) binding as this step has a direct relevance on the final performance of the design. By producing not only an RTL design but also an approximate physical topology of the chip level implementation, we ensure that the solution will perform at the predicted metric once implemented, thus avoiding unnecessary delays in the design process.

1 Introduction

High-Level Synthesis (HLS) typically uses generic, abstract models of hardware during the tasks of scheduling, allocation and binding. The use of these models simplifies HLS algorithms and standardizes the output of HLS to a generic format so that it can then be implemented in a particular technology through register-transfer-level (RTL) synthesis (e.g., logic synthesis, technology mapping and physical design).

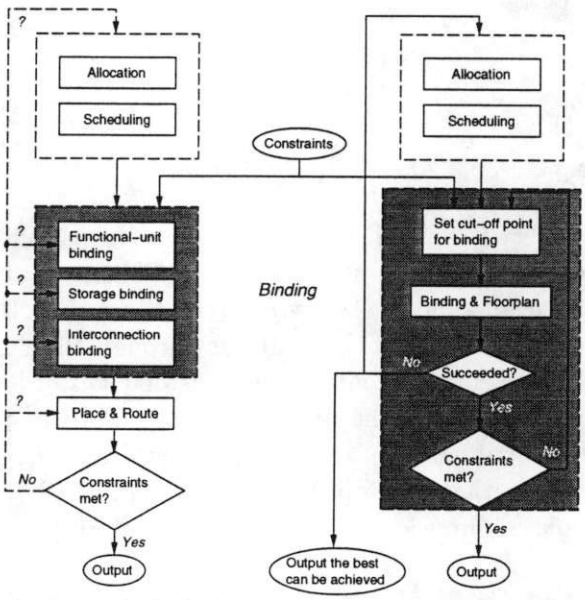

Figure 1: (a) partial flow in a typical design methodology (b) design flow in our layout driven binding techniques for HLS

However, experimental evidence indicates that there is tremendous variation in hardware attributes based not only on the target technology chosen, but also on the physical design of each implementation. BUD [1], Chippe [2] and Fasolt [3] clearly indicated the significance of interconnect and other layout effects –traditionally considered as second order in HLS– on the overall implementation area and delay. For HLS algorithms (e.g., scheduling, allocation and binding) to make effective decisions that eventually result in high-quality layouts, we need to incorporate physical design information during HLS. We must account for not only place and route effects, but also global considerations such as RT wiring, component styles, aspect ratio, floorplanning, and the combination of "all of the above". Without such information, the RTL designs may produce unpredictable results when implemented on silicon.

The work presented here proposes a paradigm to incorporate layout information into the tasks of HLS. As the first step towards solving the problem, we turn our attention to the task of binding. Binding is typically the final task in HLS which follows scheduling and allocation. In binding, there are three subtasks: (1) *functional-unit (FU) binding:* operations are assigned to hardware modules, (2) *storage binding:* values are assigned to hardware registers, and (3)*interconnection*

binding: interconnections are bound to specific buses or multiplexors.

Existing CAD systems treat binding and physical design independently. Figure 1(a) shows the flow of scheduling, allocation, binding and physical design in a typical design methodology of an automatic behavioral synthesis system. This traditional flow suffers from three major drawbacks: (1) it is not known whether the design will meet the constraints or not until the end of the time-consuming phase of place & route; (2) when the constraints are not met, it is difficult to identify where the problem comes from and at which level the design should be modified, and there is no way to identify the constraint which leads to no feasible solution; (3) the three subtasks (FU, storage, and interconnection binding) are tightly related to each other, and the deadlock situation between them is still an open problem in HLS.

In contrast with previous approaches, we incorporate physical information into the task of binding as shown in Figure 1(b). The main features of this work are the following: (1) the final result is evaluated without actually going through the time consuming phase of place & route; (2) when time constrains are met, the algorithm will output not only a structural RTL netlist, but also it's corresponding physical topology which can be carried through silicon implementation in a predictable manner; (3) whenever time constraints are not met, our binding techniques provide a means of exploring the design space in a realistic and efficient way, with this exploration, our binding techniques will provide feedback to the previous tasks if the constraints can not result in any feasible solution and output the best implementation that can be achieved; (4) we break the deadlock situation among FU, storage, and interconnection binding by performing these three subtasks simultaneously, and physical design information is taken into account as well.

While our proposed approach is valid for any technology, we benchmark the results with respect to the Field Programmable Gate Array (FPGA) design style since the ability to shorten development cycles has made FPGA an attractive alternative to standard cells and Mask Programmed Gate Arrays for realization of Application-Specific Integrated Circuits. Specifically, the Xilinx XC4000 series is assumed to be the layout design style for remainder of this paper.

2 Previous Work

3-D [4] presented an approach to the problem of binding while simultaneously considering floorplanning. Operators are assigned (and placed) as close as possible to their predecessors in order to minimize the interconnection cost. However, this approach didn't consider the cost and delay of registers, multiplexors, and wiring space overhead.

GBA [5] and BITNET [6] also considered binding with physical information. However, GBA applies only to one dimension bit-slice design, and BITNET does not consider interconnection delay.

Ewering [7] and ApplaUSE [8] addressed the binding with physical information problem by moving placement earlier before bus and register assignment, but no physical information is taken into account when FU binding is performed.

SMB [9] presented an integrated approach for minimizing critical path delay by simultaneously performing FU binding and floorplanning. But their approach has to start with a fixed floorplan and does not account for the shape and delay of multiplexors which affect the delay of the critical

path. Furthermore, it is not clear whether SMB can handle multi-cycled FUs or not.

On the other hand, our approach does not rely on any particular floorplan and we take the shape and delay of multiplexors into consideration. Furthermore, we consider clock period (register-to-register delay) in the datapath as our main process object rather than FU to register or register to FU delay as the main concern as in SMB.

3 Architecture Module and Problem Definition

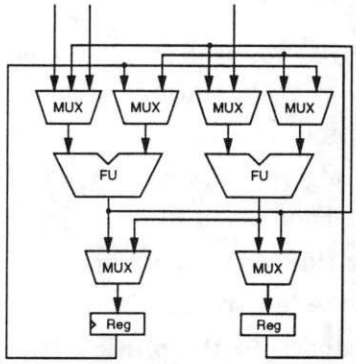

Figure 2: Architectural models

In high-level synthesis, an RTL system that consists of FUs, storages, and interconnections is synthesized from the behavioral description. In order to explore the impact of physical design information in HLS, we need to define a target architecture. In our approach, we consider two styles of target architectures: multiplexor-based and bus-based architectures. Although our approach can handle both architectures, we confine our scope to multiplexor-based architectural model (Figure 2). We also assume FUs are 2-input, 1-output combinational circuits, and registers are 1-input, 1-output circuits. Operation chaining is supported in this model by allowing connections from the output ports of some FUs directly to the input ports of other FUs. Moreover, operations can execute over several clock cycles: multi-cycled operations are possible.

Our problem can be defined as follows:

> Given (1) a scheduled data flow graph (SDFG), (2) number of FUs, registers, input and output multiplexor and (3) maximum clock period, which is usually part of the system specification, identify whether there is a feasible RTL datapath solution or not. If there is, after perform binding, generate RTL netlist and it's corresponding floorplan; Otherwise, report it to previous tasks in HLS and output the best solution that could be achieved.

The example in Figure 3 illustrates the problem. Given are a scheduled data flow graph which consists of two control steps, and allocation resource which includes 2 adders and 4 registers. The shape function and their corresponding delay information of the components can be obtained from the component library. Our algorithm will output a RTL datapath netlist with all the binding

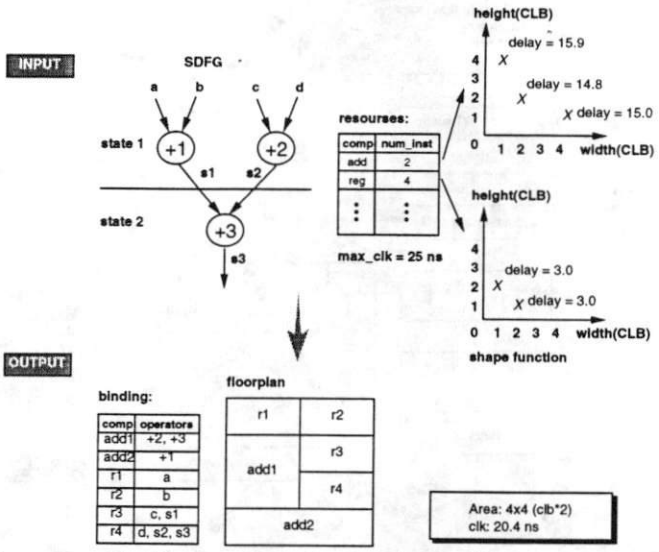

Figure 3: An example illustrating the inputs and outputs of the problem

information. Meanwhile, a corresponding floorplan and the clock period (register-to-register delay) which includes wire delay will also be generated. We assume that the controller is implemented as a Moore FSM with status and control registers. This way, the clock cycle is determined by the worst case register-to-register delay which will fall either completely inside the datapath or completely within the controller. Our work concentrates on the datapath area and delay metrics.

4 Our Approach

The flow of our algorithm is shown in Figure 4. Given a scheduled data flow graph, first, we construct a fully connected netlist in which each FU is connected to every register and each register is connected to every FU. Then, we use our physical level estimation tools ChipEst-FPGA [11], and CompEst-FPGA [12] to obtain an approximate topology of the layout. CompEst-FPGA is a component estimation tool which predicts the area and delay of a given RTL component netlist. Given a specification of a particular component as a set of Boolean equations, we use CompEst-FPGA to predict the shape function of that component. CompEst-FPGA predicts the effects of some logic synthesis tasks such as technology mapping as well as the effects of physical design. This shape function can be obtained by estimating the dimensions of a component with a varying number of rows. Additionally, CompEst-FPGA estimates the critical path delay of each configuration with wiring delay as well as false paths being taken into account. Benchmarking has shown that CompEst-FPGA can estimate area with about 2.5% accuracy and static delay with about 2-13% accuracy.

Once we have obtained a shape function for each component, ChipEst-FPGA is used to generate an approximate topology of the overall design. ChipEst-FPGA employs a partial slicing technique to generate a highly efficient approximate topology of the design, and chooses the most appropriate

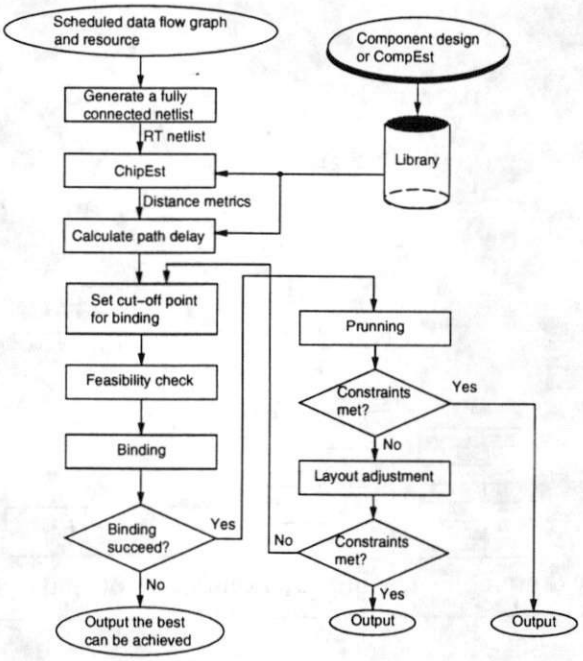

Figure 4: Layout-driven binding technique for HLS

implementation of each component. Experience has shown that component area and delay do vary (and sometimes significantly) with aspect ratio [13]. At this moment, we can get distance metrics between the different units and this step provides valuable feedback to the binding task in HLS as described later.

The backbone of our approach is a branch and bound search algorithm. We sequentially perform binding one control step at a time. Within each control step, for each operation in the step, FU and storage binding are performed simultaneously by finding a virtual binding for operation first, then for it's output variables (Ovar), and finally for it's input variables (Ivar). The actual binding will not be executed until all the virtual bindings have succeeded. The search space can be shown with a tree having three levels of hierarchy as shown in Figure 5(a). The first level is for FU, the second level is for Ovar and the third level is for Ivar. At FU level, the depth of the tree is equal to the number of operators (OPi: the ith operator) in the control step, and each path is a virtual binding for all the previous objects (an object can be FU, Ivar, or Ovar). For example, the path from root to node M means OP1 is bound to FU1 and OP2 is bound to FU2. After finishing FU binding, the binding procedure proceeds to the Ovar and Ivar level.

During the search, our algorithm can accept a seed to start with a different search order. Also, a backtracking mechanism enables the algorithm backtracking up to the higher level of virtual binding solution when the current virtual binding fails and to resume the binding process. We can see that the search space can be huge. It is unrealistic to evaluate all the possible solutions. Thus, the layout information from ChipEst-FPGA is used to confine our exploration space to a subset of the possible solutions as will be described in Section 4.2.

Let's use a three dimensional graph to express paths of register-FU-register as shown in Fig-

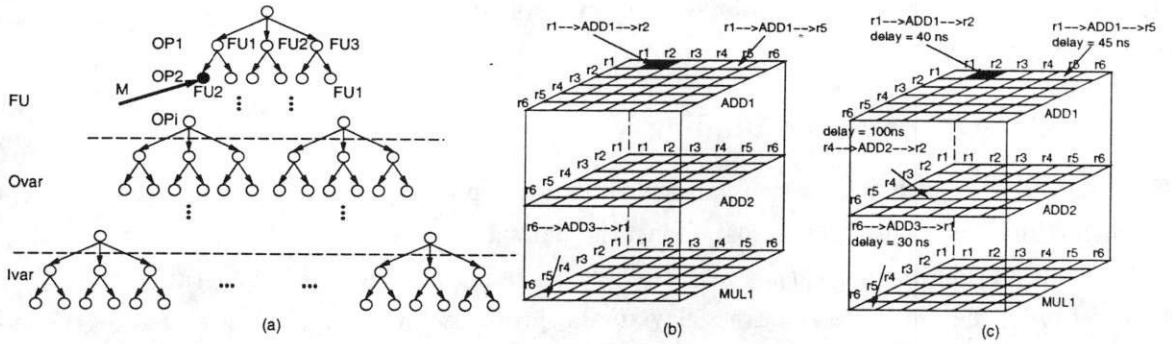

Figure 5: (a)A 3-dimensional graph for paths, (b) path delay (c) A search space tree

ure 5(b). For example, the shaded square in Figure 5(c) stands for a path from r1 to ADD1 to r2 (for operation chaining, the FU plane stands for chained operators). Using layout information, we can calculate the delay for each register-FU-register path. We also define the number which decides whether the path will be used or not as *cut-off point for binding.* Only paths with delay smaller than the cut-off point can be used in binding. Thus, by setting the cut-off point for binding, we can confine our search space to a subset of paths with delay smaller than the cut-off point. Furthermore, when the cut-off point is smaller than a certain number, there may be no sufficient paths for binding. We call this limit point the *cut-off-point threshold.*

Details of each of the steps in the overall flow shown in Figure 4 will be discussed in the following subsections.

4.1 Compute Path Delay

A typical datapath operation involves reading operands from the registers, computing the result in the FUs, and finally writing the result back into a destination register. The input multiplexors are at the input ports of FUs, and the output multiplexors are at the output ports of FUs. The path delay is determined by register-to-register delay. Based on our architectural model shown in Figure 2, we can specify the path delay by the following equation:

$$path\_delay = Tp_R + TW_{RM} + T_{IM} + TW_{MF} + T_{FU} + TW_{FM} + T_{OM} + TW_{MR} + Ts_R \quad (1)$$

where:

Tp_R and Ts_R are the propagation delay and the setup time of the register, respectively,
T_{IM} is the delay of the input multiplexor,
T_{FU} is the delay of the FU,
T_{OM} is the delay of the output multiplexor,
TW_{RM} is the wire delay from register to input multiplexor,
TW_{MF} is the wire delay from input multiplexor to FU,
TW_{FM} is the wire delay from FU and output multiplexor, and
TW_{MR} is the wire delay from output multiplexor to register

From the component library, we can get the component delay. From the distance metrics, we can calculate the wire length and use our estimation tool to get the wire delay [10]. One example is shown in Figure 5(b)

4.2 Set Cut-Off Point for Binding

Knowing all the path delays, we can set the cut-off point to decide whether the path can be used for binding (for multi-cycled operations, the partial path is identified). Let's denote the initial cut-off point for binding as CT_{init} and the cut-off point for the current iteration as $CT_{current}$. Let cr_delay_{prev} be the critical path delay of the previous binding solution and α be the factor of choosing the current cut-off point. The user can decide whether α should be equal to 10, 100, 1000... so that the tradeoff between the time spent on exploration and the number of solutions explored can be made.

The initial cut-off point and current cut-off point can be obtained by the following equation:

$$CT_{init} = \lceil MAX(Delay_{i,j,k} | i,k = 0,1,...r; j = 0,1,...f) \rceil \tag{2}$$

$$T_{current} = \lfloor cr\_delay_{prev} * \alpha \rfloor \tag{3}$$

where $Delay_{i,j,k}$ is the delay among the ith register, the jth FU to kth register, r is the number of registers, and f is the number of FUs.

Though the way of selecting the cut-off point is quite straightforward, the cut-off point plays an important role in our approach. By decreasing the cut-off point gradually, we actually categorize the binding solution into several groups. Once the cut-off point is given, we try to find a solution that meets the constraint instead of finding the best solution (i.e. the one with lowest clock cycle). Better solutions can instead be found later by further lowering the cut-off point.

The delay of the longest path is reduced every time $CT_{current}$ is calculated. In this way, we can guarantee that a different binding solution will be generated each time though the performance of the final layout may not be necessarily better.

4.3 Feasibility check

During binding, a feasibility check is needed to determine if there are enough paths with delay less than cut-off point to perform binding. Feasibility check includes two tasks: compatibility check and resource check.

The compatibility check in FU binding determines whether the operator can be bound to the FU, and in register binding, determines whether the variable to be bound is compatible with all the variables already bound to the target register by analyzing the life times of these variables. The other task is called *resource check*. Once cut-off point for binding is given, we can skip those paths with delays exceeding the constraint. We can now compute the number of FUs, input registers, output registers, and input-output registers on the remaining paths. Then we compare them with

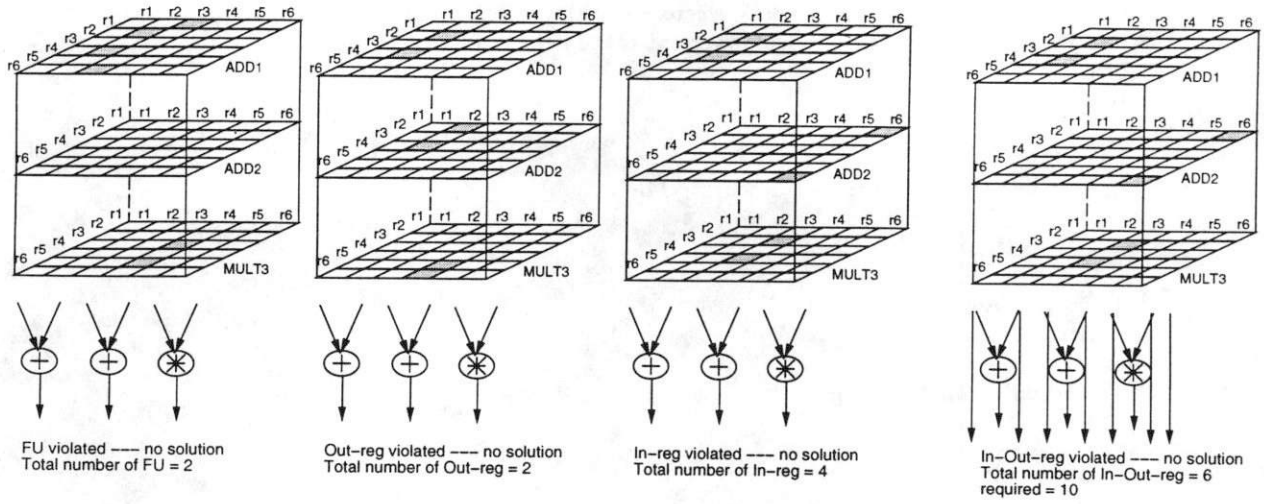

Figure 6: Some feasibility check examples

the required number of FUs, input registers, output registers and input-output registers. Then we can identify whether the available resources are sufficient to succeed the following binding. Figure 6 shows some examples where the number of resources is not sufficient. The shaded squares are the paths with delay smaller than the cut-off point. The feasibility check is carried out every time a new object (FU, Ovar, or Ivar) has been virtually bound. This speeds our search algorithm and will stop the algorithm whenever the cut-off point hits the cut-off-point threshold.

4.4 Binding

As we mentioned in Section 4, we use a branch and bound search algorithm to search for different binding possibilities one control step at a time sequentially. Within each control step, the virtual binding is carried out in the order of FU, Ovar, Ivar and multiplexors. The FU, Ovar, and Ivar binding call the same recursive binding procedure (which is outlined in Figure 7) to generate all the different possible solutions while the feasibility check in every step prunes the infeasible solutions as early as possible.

The inputs to this algorithm consist of the source object to be bound, a set of target object candidates, and the allocation resources.

The algorithm either generates an actual binding solution if it exists under the given cut-off point or reports that no feasible solution is available together with the best result that can be achieved.

We need to mention here that, for the interconnection from registers to FUs, there are two different assignments since each FU has two input ports. By assigning interconnections to the ports differently, the multiplexor cost (i.e. size and number) will be different. So our interconnection binding not only includes the compatibility check which checks whether two interconnections can share same multiplexor, but also attempts to minimize the size of the multiplexor and the number

```
Procedure: Binding (S_i, N, M, allocation resource)
  Inputs: S_i: the ith source objects (1 ≤ i ≤ N where N is the number of source objects);
      T_ij: the jth target objects (1 ≤ j ≤ M where M is the number of target objects);
      /* objects = FU, Ovar, Ivar */
  Output: Binding solution;
begin Procedure
  for (j = 1 to M)
    if (FeasibilityCheck(S_i, T_ij, allocation resource)) then
      VirtualBinding(S_i, T_ij);
      if (i + 1 ≤ N) then
        success = Binding(S_i+1, N, M);
        if (success) then
          return (True);
        else
          UnVirtualBinding(S_i, T_ij);
        end if;
      else
        ActualBinding;
        return (True);
      end if;
    end if;
  end for;
  return (False);
end Procedure
```

Figure 7: The binding algorithm

of interconnections. Basically, we try two different assignments for each interconnection, check the multiplexor cost and select the one with less cost. Once the FU and Ovar have been virtually bound, the interconnection from FU to Ovar has also been bound.

4.5 Pruning

If the binding succeeds, the algorithm will proceed to the next step: pruning. In this step, all the unnecessary interconnections will be pruned, all the unnecessary multiplexors will be deleted and finally, the size of the multiplexors will be shrunk according to the actual interconnection information. When the multiplexors are changed, new types of multiplexors may be generated. The algorithm will then update the area and timing information based on the component information in the library or by invoking CompEst-FPGA [12]. At the end of this step, an optimized RTL netlist will be generated.

4.6 Layout Adjustment

At this point, if the clock period exceeds the maximum clock period, layout adjustment will be invoked to re-run our ChipEst-FPGA on the pruned RTL netlist based on new multiplexors and interconnection information. Usually this will minimize the waste layout area and improve the performance of the final design. Figure 8 shows a layout after pruning and layout adjustment for HAL example.

After layout adjustment, if the cycle time still can not satisfy the maximum clock period constraint, we need to reset the cut-off point and redo the binding. This iteration will continue until

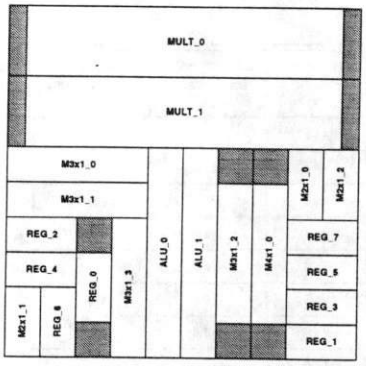

Figure 8: A layout after pruning and layout adjustment for HAL example

the cut-off point hits the cut-off-point threshold. This way, we only evaluate a set of possible solutions to see whether a final solution can be found. Our experimental results in Figure 10 show that there is a big chance we can find the solution if one exists although we only evaluate a small subset of possible solutions.

5 Experimental Results

5.1 Experimental Procedure

We have implemented our layout-driven RTL binding techniques for HLS in C on the Sun SPARC workstation. The designs used to test our binding techniques are from some well-known high level synthesis benchmarks. The first example is *the 2nd order differential equation solver* [14] which consists of 6 multiplication operations, 2 additions, 2 subtractions, and 1 comparison. In this example, we assume that multiplications are performed by multipliers while additions, subtractions, and comparison are performed by ALUs. The second example is *the 5th order elliptic wave filter (EWF)* [14] which consists of 8 multiplications and 26 additions. The third example is *Discrete Cosine Transformer (DCT)* which consists of 16 multiplications, 25 additions, and 7 subtractions. In the second and third examples, we assume that multiplications are performed by multipliers and additions and subtractions by ALUs. The bit-width of all the examples is 4.

The datapath components can be obtained from our library in which all the components' layout and timing information are pre-characterized. The components can be implemented by different tools such as Xilinx hard macro library, xblox and designware. Alternatively, we use GENUS, a generic component generator, to generate the logic equation (EQN) according to the desired functionality [15]. Then, we use Synopsys synthesis tools to optimize and synthesize the design. After synthesis, the components are translated to gate level netlist (xnf) and fed into Xilinx partitioning, placement and routing tool by giving different constraints to get different aspect ratios. For each specific placement and routing, Xilinx delay analysis tool Xdelay is invoked to get the delay information. Thus, we generate a shape function for each component similar to the one shown in Figure 3.

5.2 Results

Examples	Resources	RT–Level constraints (s)	Cut–off point for binding (ns)	Clock period w/o prunning (ns)	Clock period w prunning (ns)	Clock period w adjustment (ns)
DCT	8A, 4M, 15R	44.5	160.0	153.2	87.2	80.1
		44.5	153.0	152.3	87.6	82.2
		44.5	152.0	151.4	86.2	78.5
		44.5	151.0	150.2	86.2	78.5
		44.5	150.0	149.9	86.3	78.4
		44.5	149.0	149.0	86.3	81.0
		44.5	148.9	148.8	84.9	81.0
Area before re–floorplan: 34X34 CLB, after re–floorplan: 24X24 CLB						
EWF	3A, 2M, 12R	44.5	120.0	115.7	82.8	79.0
		44.5	115.0	114.6	82.8	77.7
		44.5	114.6	114.4	82.5	72.3
Area before re–floorplan: 22X22 CLB, after re–floorplan:14X14						
HAL	2A, 2M, 8R	44.5	85.0	82.2	69.3	64.6
		44.5	82.0	82.0	68.9	68.5
Area before re–floorplan: 14X14 CLB, after re–floorplan: 12X12 CLB						

Figure 9: Experimental Result

The first set of experiments we did was for DCT, EWF, and HAL examples. Figure 9 shows the results. The RTL constraints only include FU and register delays since they have no interconnection and layout information. We get the cut-off point for binding by using the formula 2 and 3, the clock period without pruning is the cycle time after we perform the binding and can be further used to get the next cut-off point. If the binding succeeds, we construct the actual RTL netlist and get it's actual cycle time. If this still cannot satisfy the maximum clock period constraint, we further optimize the cycle time by layout adjustment. These results clearly indicate that: (1) layout and interconnection delays are significant since they may contribute up to 50% of the overall delay; (2) by varying the cut-off point, we can explore a set of alternative binding solutions with varying clock cycle time. Our algorithm are efficient since each solution takes less than a minute of CPU time for all the cases.

the ith run / results	1st run	2nd run	3rd run	4th run	5th run	6th run	7th run
The best cycle time	80.18	81.35	81.37	80.99	81.37	80.41	80.61
The cut–off threshold	149.00	149.00	148.80	149.00	149.00	149.00	148.80

Figure 10: Experimental Result: 7 set of runs for DCT

To test the robustness of our branch and bound algorithm, we did another set of experiment using the DCT example. Given different seeds to it, the algorithm will search in a different order and may find different binding for the same cut-off point. We tried 7 different search orders and got all the results for the different cut-off points. We compared their the best case cycle time and their cut-off-point thresholds as shown in Figure 10. In each case, there is only small variation for

the best case cycle time. This shows that our small set of exploration is not only efficient, but also sufficient in finding the best solution in most cases. For the 7 sets of runs, in the worst case, 7 to 8 minutes were required to find the best solution that could be achieved.

6 Conclusion

We presented a binding approach which simultaneously binds FUs, registers and interconnections and also uses an accurate layout estimator to simultaneously produce an RTL solution and a corresponding floorplan. Future work will incorporate the controller effects into HLS using the approach proposed in [16].

7 References

[1] M. C. McFarland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions," *Proc. 23rd DAC*, pp. 474-480, July 1986.

[2] F.Brewer and D.Gajski, "Chippe: A System for Constraint Driven Behavioral Synthesis," *IEEE Trans. on CAD*, Vol. 9, No. 7, pp. 681-695, 1990.

[3] David W. Knapp, "Fasolt: A Program for Feedback Driven Data-Path Optimization," *IEEE Trans. on CAD*, Vol. 11, No. 6, pp. 677-695, 1990.

[4] P. Gupta and A. C. Parker, "SMASH: A Program for Scheduling Memory-Intensive Application-Specific Hardware," *Proc. 7th International Workshop on HLS*, pp. 54-59, May 1994.

[5] Hyuk-Jae Jang and Barry M. Pangrle, "A Grid-Based Approach for Connectivity Binding with Geometric Costs," *Proc. ICCAD*, pp. 94-99, 1993.

[6] Ashutosh Mujumdar, Minjoong Rim, Rajiv Jain, and Renato De Leone, "BITNET: An Algorithm for Solving The Binding Problem," *7th International Conference on VLSI Design*, pp. 163-168, 1994.

[7] C.Ewering, "Automatic High-Level Synthesis of Partitioned Busses," *Proc. EURODAC*, pp. 304-307, 1990.

[8] Elof Frank, Thomas Lengauer "APPlaUSE: Area and Performance Optimization in a Unified Placement and Synthesis Environment," *Proc. ICCAD*, pp. 662-667, 1995.

[9] Y.M. Fang and D.F. Wong, "Simultaneous Functional-Unit Binding and Floorplanning," *Proc. ICCAD*, pp. 317-312, 1994.

[10] C. Ramachandran, F. J. Kurdahi, D. Gajski, V. Chaiyakul, and A. Wu, "Accurate Layout Area and Delay Modeling for System Level Design," *Proc. ICCAD '92*, Nov. 1992.

[11] M. Xu and F.J. Kurdahi "Chip Level Area and Timing Estimation for Lookup Table Based FPGAs," *Tech. Report #95-31, UCI*, Aug. 1995.

[12] M. Xu and F.J. Kurdahi "Area and Timing Estimation for Lookup Table Based FPGAs," *Proc. of ED&TC*, March 1996.

[13] P.Jha, C. Ramachandran, N. Dutt, and F.J. Kurdahi, "An Empirical Study on the Effects of Component Styles and Shapes on High-Level Synthesis," *Proc. of VLSI'94*, 1994.

[14] N. Dutt and C. Ramachandran, "Benchmarks for the 1992 High Level Synthesis Workshop," *Tech. Report #92-107, UCI*, 1992.

[15] P.K. Jha, T. Hadley and N.D. Dutt "The GENUS User Manual and C Programming Library," *Technical report.* No.93-32, April, 1993

[16] C. Ramachandran, F. J. Kurdahi, "Incorporating the Controller effects during Register-Transfer Level Synthesis," *Proc. EDAC* 1994