Simultaneous FU and Register Binding Based on Network Flow Method

Jason Cong and Junjuan Xu
UCLA Computer Science Department, Los Angeles, CA 90095, USA
{cong, irene.xu}@cs.ucla.edu

Abstract – With the rapid increase of design complexity and the decrease of device features in nano-scale technologies, interconnection optimization in digital systems becomes more and more important. In this paper we develop a simultaneous FU and register (SFR) binding algorithm for multiplexer optimization based on min-cost network flow. Unlike most of the prior approaches in which functional unit binding and register binding are performed sequentially, our approach performs these two highly correlated tasks gradually and concurrently. We also present an ILP formulation of the combined functional unit and register binding problem for the optimality study of heuristics. Experimental results show that when compared to traditional binding algorithms, our simultaneous resource binding algorithm is close to optimal solutions for small-size designs (only 5% more MUX) and achieves significant reduction for MUX area (12%) and timing (10%) for a set of real-life benchmark designs.

I. Introduction

As VLSI circuits move into the era of nano-scale technology and gigahertz clock frequency, the system's area and performance have increasingly become dominated by the interconnection. Studies show that interconnects consume over 70% of the total area and over 75% of the total power for most of the FPGA designs [10][16]. The multiplexer (MUX), one of the basic data path connection elements, contributes a significant amount of these costs, especially for FPGA designs. A recent study from Altera [18], based on the analysis of 100 customer designs, stated that multiplexers account for 26% of the logic element utilization. Optimizing multiplexer is very important for the overall quality of digital designs.

The behavioral synthesis, which compiles designs specified in high-level languages into register-transfer level code, determines the main micro-architecture of designs and thus has a big impact on the design quality. Behavioral synthesis consists of three basic stages: resource allocation, scheduling, and resource binding. Allocation determines how many instances of each type of resources (functional units or registers) are needed; scheduling determines when a computational operation will be executed; resource binding assigns operations (or variables) to the resources. Each of these three steps has influence on the multiplexer utilization. In this paper we will assume that the first two steps are already finished and focus only on the third step, resource binding, for multiplexer optimization.

Specifically, the task of functional unit (FU) binding is to assign operations to functional units of the same type, and the task of register (REG) binding is to assign variables to registers. A MUX will be introduced before a register whenever more than one functional unit produces results and stores them into this register. In a similar way, a MUX will be introduced before an input port of a functional unit whenever more than one register feeds data to this port. Results of FU binding and REG binding jointly determine the final interconnection structure. To optimize MUX, both of these two tasks need to be performed carefully. Unfortunately, there is a cyclic inter-dependency between them that compromises the performance. Specifically, the assignment decision of operations to functional units is based on the knowledge of the assignment of variables to registers. On the other hand, the assignment of variables to registers can be informed only if the functional unit binding is performed beforehand. This cyclic inter-dependency was recognized in previous works, but most of these broke it up by simply solving one task completely before doing the other. This simple method has to sacrifice one task's accuracy and thus might not lead to good solutions.

Even if the result of one binding task is fixed, the problem of the other binding problem with the goal of minimizing MUX remains to be NP-Hard [12]. Because of the significant effect of interconnection on the overall chip qualities, many heuristics have been proposed to optimize interconnection. In earlier works from the 1980s, clique partitioning methods and branch-and-bound search algorithms are applied to bind functional units and registers [13][14][19]. Later on, a weighted bipartite-matching algorithm was presented to solve both register and functional unit binding [8]. The cost of edges represents the possibility that the sharing of two operations or two variables will introduce a MUX. The authors experimented with two flows in which the functional unit binding and the register binding were solved in different orders. Experimental data showed that the ordering did not impact the interconnection result much. In [9], an integrated flow was proposed to perform scheduling, allocation and binding all together in a step-by-step fashion. For each control step, the maximum possible number of operations are selected out of the ready operations and scheduled at the current step. Then a network flow is constructed to solve the concurrent functional unit binding and register binding. This method breaks up the cyclic inter-dependency in a better way than previous methods, but the drawback is that it can only perform local exploration and thus compromises the final quality due to the lack of global exploration. In [20], the min-cost network flow algorithm is applied to solve both binding tasks. Again, the inter-dependency is broken by solving one task completely before starting the other. In [4], a bipartite-based algorithm is proposed for register binding with the goal of optimizing MUX. The authors assume that functional unit binding is already done and fixed. The work in [5] solved the same problem as [4], but formulated it as a *k*-cofamily problem solved by the min-cost network flow.

Among the above-mentioned related works, none handle the cyclic inter-dependency well and may sacrifice the optimality of one or both tasks. In this paper, instead of solving these two tasks separately, we propose a simultaneous FU and register binding algorithm, named SFR, to tackle this cyclic inter-dependency without compromising any task's performance. Given a scheduled DFG and available resources, our goal is to minimize the total area of multiplexers. Specifically, two resource binding tasks are performed gradually and interactively so that both of them can make assignment decisions based on the intermediate partial binding information supplied by each other. In this way, neither of the two tasks has to sacrifice accuracy. As the binding process progresses and more binding information is gained, necessary adjustments will be made and operations and variables which are already bound together might be separated to allow for optimization. More details of the SFR binding algorithm will be provided in Section III. Also, we present an ILP formulation of the multiplexer minimization problem for the purpose of an optimality study. Both the SFR binding algorithm and the ILP formulation support general scenarios, such as multi-cycle operations, pipelined operations and chaining. Experimental results show that our synthesis flow can achieve a significant amount of savings in terms of MUX area and timing.

In the remainder of this paper, we will introduce related preliminaries and define the problem to be solved in Section II. The simultaneous binding algorithm, as well as the formal ILP formulation, will be presented in Sections III and IV. Section V shows the experimental results, and Section VI concludes this paper.

II. Preliminaries and definitions

The inputs to the resource binding problem are a scheduled DFG, and the available resources include a number of functional units and registers. A DFG is an acyclic directed graph, in which nodes represent operations, and edges represent data dependence. Data propagated along the edges are called variables. Variables produced in one clock cycle and used in another clock cycle need to be registered.

The set of operations to be bound is denoted as O, and the set of variables to be stored is denoted as V. We use O_f to represent the group of operations of type f. For two operations o_i and o_j of the same function type f, if their execution times do not overlap and operation o_i comes before o_j, we call operations o_i and o_j *compatible* with each other, and denote this compatibility as $o_i \rightarrow o_j$. Compatible operations can be bound into a single functional unit without lifetime conflict. For pipelined operations, we only consider the initiating execution time for compatibility calculation. The lifetime of a variable v is defined as the time period from the control step where v is generated to the last control step where v is consumed by other operations. Similarly, for two variables v_i and v_j, if their corresponding lifetimes do not overlap and variable v_i is produced before v_j, we call variables v_i and v_j *compatible* with each other, and denote this compatibility as $v_i \rightarrow v_j$.

We define *FU sharing ratio* as $1 - |F| / |O|$, where $|F|$ is the total number of functional units, and $|O|$ is the total number of operations to be bound. Define *REG sharing ratio* as $1 - |R| / |V|$, where $|R|$ is the total number of registers, and $|V|$ is the total number of variables to be bound. The higher the sharing ratio, the less resources utilized, and more operations or variables are grouped to a single resource instance. These definitions will be used in the synthesis algorithm in Section III.

The resource binding problem to be solved in this paper is formulated as follows:

Given: (1) A scheduled DFG $G(O, A)$, where O is the operation set, and A is the data dependence among operations, (2) A group of functional units F, and a group of registers R.

Objective: Bind the DFG with the given resources such that the total area of the required multiplexers is minimized.

Note that since the number of functional units and registers is fixed, minimizing MUX is equal to minimizing the total area.

III. Simultaneous FU and register binding algorithm

In this section we will first introduce the whole synthesis flow, and then present the construction of the core min-cost network flow, which decides which operations and variables are grouped together and share a same resource instance.

A. The SFR binding algorithm

The basic idea of the simultaneous FU and register (SFR) binding algorithm is that neither all of the functional unit binding nor all of the register binding is finished in one single step. Instead, the resource sharing ratio will start from a low value with large numbers of resources and gradually increase till the used resources come down to the given resource constraints. During this gradual binding process, both of the two binding tasks are performed concurrently and are able to access the intermediate partial information from the other. Also, necessary adjustments will be made to improve chances for optimization.

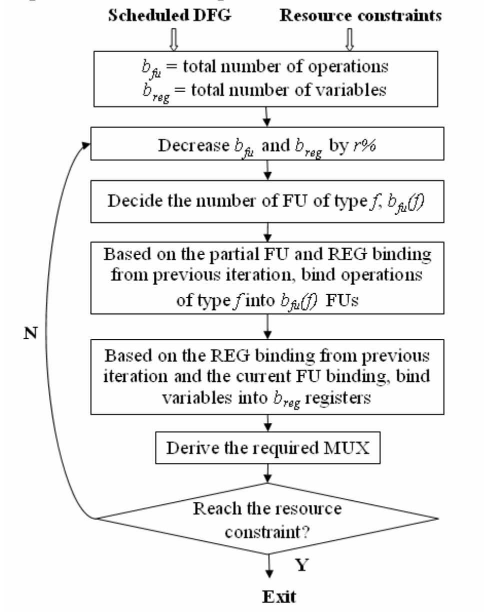
Scheduled DFG
Resource constraints
b_{fu} = total number of operations
b_{reg} = total number of variables
Decrease b_{fu} and b_{reg} by $r\%$
Decide the number of FU of type f, $b_{fu}(f)$
Based on the partial FU and REG binding from previous iteration, bind operations of type f into $b_{fu}(f)$ FUs
Based on the REG binding from previous iteration and the current FU binding, bind variables into b_{reg} registers
Derive the required MUX
Reach the resource constraint?
N
Y
Exit

Figure 1. The SFR Binding algorithm

The SFR binding algorithm is shown in Figure 1. b_{reg} is the number of registers to be used in each iteration, and b_{fu} is the total number of functional units to be used. The number of functional units of each type will be derived from b_{fu}. Initially, b_{fu} and b_{reg} are set to the total number of operations and variables, respectively. Then they are decreased gradually by r% after each iteration, where r is a value to control how fast the resource sharing ratio increases. When b_{fu} and b_{reg} reach the given resource constraints, the whole process is completed.

The calculation of $b_{fu}(f)$ for type f is the rounding of $b_{fu} \cdot |F_f| / |F|$, where F_f is the set of functional units of type f given in the resource constraints, and F is the complete set of functional units given in the resource constraints. This calculation means that each type of operations occupies proportional resources decided by the resource constraints.

After resource allocation is done, functional unit binding and register binding are performed based on the intermediate binding results from the previous iteration, which has a smaller resource sharing ratio. At this point, the two tasks can be done in any order. In Figure 1, we show the situation where functional unit binding is performed first. The current functional unit binding makes assignment decisions based on the complete binding information from the previous iteration. After the functional unit binding is finished in the current iteration, its solution is available for the subsequent register binding step in the same iteration. Therefore, the current register binding will be performed based on the register information from the previous iteration and the latest functional unit information from the current iteration. How such information guides the assignment will be discussed in the next subsection. Obviously, we may perform register binding first in the current iteration, so we still have some inter-dependency in the same iteration. However, compared to the overall cyclic inter-dependency and the traditional algorithms, our method has much less influence on the final binding solution, since these two tasks can interact with each other instead of being separated completely.

After binding is finished, MUX is generated in front of registers and functional units where they are needed. Then the whole binding information, including resource binding, MUX size and critical paths, is fed into the next iteration to guide the resource binding with fewer resources.

In this synthesis flow both functional unit binding and register binding are performed gradually and interactively. Each intermediate binding solution is gained based on previous binding results with smaller resource sharing ratios and it then provides useful information for subsequent iterations with larger resource sharing ratios. In this way, the cyclic inter-dependency between the two binding tasks is broken up and neither has to be solved completely prior to the other.

We will use a simple example to illustrate the advantage of the SFR binding algorithm over traditional algorithms. Figure 2(a) shows a scheduling solution with six operations of the same function type, o_i ($i \in [1,6]$). The arrows represent the lifetime of their output variables, v_i ($i \in [1,6]$). For the sake of simplicity, we only consider MUX before registers. We can see that all pairs of operations are compatible and can share the same functional unit. Except that variable v_1 conflicts with variable v_2, other pairs of variables are compatible and can share the same register. Assume that available resources include two functional units and two registers.

Assume there are two iterations in the SFR binding algorithm for this example, and in the first iteration both b_{fu} and b_{reg} are equal to 3. Assume we prefer to bind o_1 and o_2 together based on estimated information. Suppose the functional binding solution from the first iteration is $\{o_1, o_2\}$, $\{o_3, o_4\}$, $\{o_5, o_6\}$, and the register binding solution is $\{v_1, v_3\}$, $\{v_2, v_4\}$, $\{v_5, v_6\}$. Note that o_1 and o_2 are bound together due to estimated information, while in the subsequent register binding, their output variables, v_1 and v_2, are stored into different registers due to their conflict. The intermediate binding is shown in Figure 2(b). A two-input MUX is required in front of register R_1 and R_2, since they have two different functional units feeding data into them. Inaccurate estimation in the functional unit binding leads to these unnecessary MUX. In the second iteration with resource constraints set as two functional units and two registers, since more information about the register binding is gained, the functional unit binding has the chance to make an adjustment and split o_1 and o_2 apart. Suppose the functional binding solution from the second iteration is $\{o_1, o_3, o_4\}$, $\{o_2, o_5, o_6\}$, and the register binding solution is $\{v_1, v_3, v_4\}$, $\{v_2, v_5, v_6\}$. The complete binding solution is shown in Figure 2(c). Redundant MUX is eliminated due to the gradual binding and the interaction of the two binding tasks. In traditional algorithms, since we prefer to bind o_1 and o_2 together based on estimation, we might come to the functional unit binding of $\{o_1, o_2, o_3\}$, $\{o_4, o_5, o_6\}$, and then the register binding of $\{v_1, v_3, v_4\}$, $\{v_2, v_5, v_6\}$. The binding solution is shown in Figure 2(d). A two-input MUX is required in front of both of two registers. Inaccurate estimation, which is intrinsic in traditional algorithms, leads to this suboptimal result.

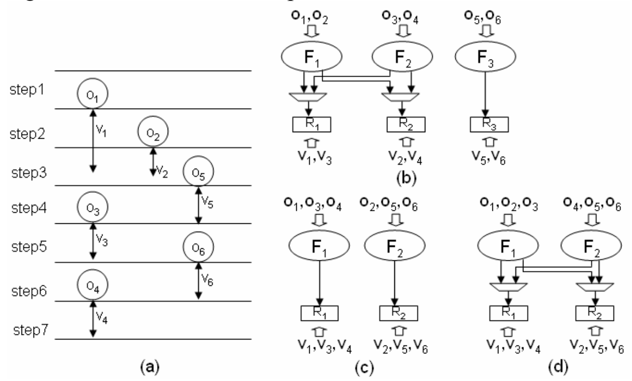

Figure 2. (a) A scheduling example, (b) binding solution from Iteration 1 in SFR, (c) binding solution from Iteration 2 in SFR, (d) binding solution from traditional algorithms

B. Network flow construction

Network flow formulation has been used widely for binding problems with the goal of optimizing all kinds of qualities of concern, such as interconnection, switching activity, and power consumption [2][5][6][11][20]. Due to its polynomial-time complexity and flexibility for considering multiple metrics in the cost function, we still utilize the network flow formulation for the resource binding with the goal of optimizing MUX.

During each iteration, the register binding and the functional unit binding of each type of operations (e.g., addition and multiplication) are performed separately. The network construction for each of the tasks is almost the same, except that the compatibility is calculated differently (as explained in Section II). We will use functional unit binding to illustrate the network construction.

A network $H = (s, t, V_n, E_n)$ is constructed based on O_f in the DFG and the compatibilities of operations. We introduce the source vertex s and sink vertex t. V_n is the vertex set, and E_n is the edge set of the network. For each operation in O_f, there is a corresponding vertex in V_n. When two operations are compatible, there is a directed edge connecting the two corresponding vertices. s is connected to all vertices, and all vertices are connected to t. The maximum capacity of each edge is 1. Both the outflow of s and the inflow of t are equal to $b_{fu}(f)$. Each edge is associated with a weight, which represents the cost of binding two operations into a single functional unit.

In the following we will use the example in Figure 3(a) to illustrate the construction of H. The constructed network is shown in Figure 3(b). From the definition of compatibility, we have that $o_1 \rightarrow o_2$, $o_1 \rightarrow o_3$, and $o_3 \rightarrow o_2$. The corresponding three edges are drawn in the network.

The cost of edges starting from s or ending at t is set as 0. The cost of edges from o_i to o_j is defined as

$C(o_i, o_j) = g(MUX, timing, consistency)$, (0)

where "*MUX*" represents the possibility of introducing MUX due to sharing, "*timing*" considers if splitting o_i or o_j will help to decrease the clock period, and the goal of "*consistency*" is to maintain the binding consistency among successive intermediate binding solutions. The possibility of introducing MUX is based on the register binding and compatibility of the input and output variables of o_i and o_j. The idea is the same as that in [5][8][20]. We will not repeat it here. MUX introduces non-negligible delay compared to other datapath components [5]. Avoiding putting them on critical paths will help to improve the timing of designs. If either o_i or o_j is on the critical paths in a previous binding solution, a non-zero weight will be added to $C(o_i, o_j)$; otherwise, the weight of "*timing*" will be 0. This means if any of these two operations is on critical paths, in order to alleviate the impact of MUX on timing, we will prefer not to bind them together at this time. If o_i and o_j do not share a same resource instance in the binding solution of the previous iteration, a non-zero weight will be added to $C(o_i, o_j)$; otherwise, the weight of "*consistency*" will be 0. This means if two operations share a common resource instance in a previous iteration, we will prefer that they stay together.

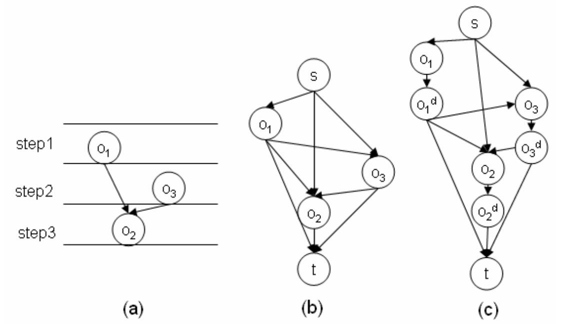

Figure 3. A scheduling and its network H and H^d

From the definition of edge costs, it is easy to see that the resource binding solution derived from the network flow with smaller total cost has a smaller MUX area, improved timing and good consistency with previous solutions.

To guarantee that only one unit flow goes through each node $o \in O_f$, we can apply a node-splitting technique, which was first adopted in [2] to guarantee that we will have a legal binding solution. This technique splits every vertex $o \in O_f$ in H into two nodes, o and o^d. There is an edge from o to o^d. All the edges coming out from o will be connected to o^d instead. Both the flow capacity lower bound and upper bound are 1 for the edge (o, o^d). We denote the network after splitting as H^d. Figure 3(c) shows the split version of H from Figure 3(b).

Lemma 1: A flow u, with $|u| = 1$, in the network H^d corresponds to a group of operations that are compatible with each other and thus can be bound to a same resource instance.

Theorem 1: The min-cost $b_{fu}(f)$-flow of H^d, *min-flow*, binds O_f to exactly $b_{fu}(f)$ functional units. €

The proofs of the above lemma and theorem are omitted due to page limitations.

The min-cost network flow can be solved by shortest-path-based algorithms [3]. After we obtain the min-cost flow, each edge (o_i^d, o_j) with a unit flow represents that operations o_i and o_j should be bound together into the same resource instance.

IV. ILP formulation

To gain a better knowledge of the quality of the proposed algorithm, we formulate the MUX optimization problem as an ILP formulation. ILP either maximizes or minimizes an objective function of a set of variables, subject to a group of linear equation and inequality constraints and integral restrictions on all of the variables.

Let $|F|$ be the total number of given functional units, and $|R|$ be the total number of given registers. We give each resource instance an identifier ranging from 1 to $|F| + |R|$. Let $CF(i)$ be the set of functional units which can execute operation o_i. Let CR be the set of all registers. Let $x_{i,k}$ represent whether operation o_i (or variable v_i) is bound to the functional unit k (or register k). If this is the case, $x_{i,k}$ is set as 1; otherwise, the value is 0. Let $c_{k,l,p}$ represent if there is connection from resource instance k to the port p of resource instance l (registers have one input, but functional units have two or more inputs). Let $m_{k,p}$ represent the number of connections to the port p of resource instance k.

Based on the above definitions, the constraint that each operation/variable is bound to one and only one resource instance is described below.

$$\sum_{k\in CF(i)} x_{i,k} = 1 \qquad \forall o_i \in O \qquad (1)$$

$$\sum_{k\in CR} x_{i,k} = 1 \qquad \forall v_i \in V \qquad (2)$$

Each resource can only be occupied by only one operation/variable at one time, which means those conflicting operations/variables cannot be bound to a same resource.

$$(x_{i,k} + x_{j,k}) \le 1 \qquad \forall !(o_i \to o_j)\ \&\&\ !(o_j \to o_i),\ \forall k \in CF(i) \cap CF(j) \ (3)$$

$$(x_{i,r} + x_{j,r}) \le 1 \qquad \forall\ !(v_i \to v_j)\ \&\&\ !(v_j \to v_i),\ \forall r \in CR \qquad (4)$$

For dataflow from o_i to o_j, suppose that o_i is bound to functional unit k, the output variable of o_i, v_i, is bound to register r, and o_j is bound to functional unit l. There will be a connection from functional unit k to register r, and from register r to the corresponding port of functional unit l. In inequation (6) below, p in $c_{r,l,p}$ is the port of o_j into which the output of o_i is fed.

$$x_{i,k} + x_{i,r} - 1 \le c_{k,r,0} \qquad \forall v_i \in V,\ \forall k \in CF(i),\ \forall r \in CR \qquad (5)$$

$$x_{i,r} + x_{j,l} - 1 \le c_{r,l,p} \qquad \forall\ (o_i, o_j) \in E\ \&\&\ o_i \text{ and } o_j \text{ not chained}, \forall r \in CR,\ \forall l \in CF(j) \qquad (6)$$

If o_i and o_j are chained, there is only one connection from functional unit k to functional unit l.

$$x_{i,k} + x_{j,l} - 1 \le c_{k,l,p} \qquad \forall\ (o_i, o_j) \in E\ \&\&\ o_i \text{ and } o_j \text{ chained}, \forall k \in CF(i),\ \forall l \in CF(j) \qquad (7)$$

$m_{k,p}$ is the summation of connections from other resource instances to the port p of resource instance k.

$$\sum_{l\in[1,|F|+|R|]} c_{l,k,p} = m_{k,p} \qquad \forall k \in [1, |F|+|R|],\ \forall p \in ports \ (8)$$

The objective function is the sum of the connections in front of all resource instances.

$$\min: \sum_{l\in[1,|F|+|R|],p\in \text{ports}} m_{k,p}$$

The description of interconnection constraints is similar to [15]. However, [15] describes minimizing the total wiring area after a preliminary floor-planning is done, while our formulation is to minimize MUX.

For a benchmark with 15 operations and 9 variables, when the resource constraint is set at five functional units and five registers, the ILP formulation comprises 190 variables and 582 equation and inequation constraints. For a benchmark with 48 operations, 33 variables, 20 functional units and 17 registers, the number of variables and constraints increases to 2145 and 17598. Solving the ILP problem of such sizes is very time- consuming and solutions may not be reached within a reasonable time.

We would like to point out that only when two or more interconnections are connected to a single port, a MUX will be introduced. Also, the MUX area is not exactly linear to its input number. Figure 4 shows the curve of MUX inputs and its area in terms of LUTs based on the Xilinx Virtex2P device [1]. The bitwidth of ports is set at 32. In the objective function, we simply minimize the total connections among resource instances, which might not lead to the optimal solution with the smallest MUX area. We observe that the total interconnections are highly correlated with the total inputs of MUX. Minimizing one objective will lead to optimizing the other, though not necessarily optimally. We also observe that when the MUX's inputs are less than six, which is the case for small designs, the curve is very close to a line. Furthermore, due to the huge number of variables and constraints for medium- and large-size designs, we can only test small designs with ILP. In this case, ILP will generate optimal or near-optimal solutions for small designs and can be a good standard for evaluating the quality of heuristics.

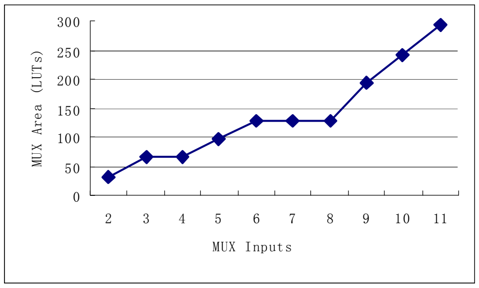

Figure 4. MUX area

V. Experimental results

The binding algorithm and ILP formulation are implemented in xPilot, a platform-based behavioral synthesis system developed at UCLA [7]. In this framework a behavioral description in C is first parsed and optimized into a control- and data-flow graph. After the scheduling and resource allocation is done, the simultaneous binding flow and the ILP binding discussed in Section III and IV are then performed. This tool is platform-based and integrates area and timing estimators targeted at different user-defined hardware platforms. In this paper, all the experimental data are based on the Xilinx Virtex2P device [1], and the set of real-life benchmarks are from [17]. These examples are data-dominated behavioral description in signal and image processing applications.

A. Comparison to previous works

To evaluate the efficiency of our simultaneous binding solution, we implemented the resource binding algorithm recently proposed in [20], which is also based on the min-cost network flow, but breaks the cyclic inter-dependency by solving one task completely before starting the other. We experimented with two different traditional flows based on the algorithm of [20]:

"*FU+REG*": Functional unit binding is performed first, followed by register binding.

"*REG+FU*": Register binding is performed first, followed by functional unit binding.

Also, we add a post-processing step after the resource binding is finished, where we redo resource binding repeatedly until the MUX area does not decrease any more. For instance, after adding the post-processing, "*FU+REG*" would be "*FU+REG+FU+REG +FU…*". The second "*FU*" will be performed based on the previous "*REG*" and the second "*REG*" will be based on the second "*FU*". The purpose of the post-processing is to understand how much iterative refinement of FU and register binding can save the MUX area.

Table 1 summaries the MUX area of six different flows in terms of LUTs. Column "*SFR*" shows the results from the simultaneous flow. Column "*SFR+Post*" shows the results of the simultaneous flow followed by the post-processing. The remaining four columns list the results from "*FU+REG*", "*REG+FU*" and the versions with post-processing. The last row lists the comparison results over "*SFR*". We can see that "*SFR*" consistently introduces less MUX than "*FU+REG*" and "*REG+FU*" (12% and 9% better on average, respectively). Also, we observe that the post-processing does work for "*FU+REG*" and "*REG+FU*" with an improvement of 4% and 1.7%. However, the post-processing cannot improve "*SFR*" further. Even after the improvement by post-processing to "*FU+REG*" and "*REG+FU*", "*SFR*" still performs better.

For the total area in terms of the number of occupied LUTs, "*SFR*" achieves a saving of 4% and 3% compared to "*FU+REG*" and "*REG+FU*" on average. Details are omitted due to page limitation.

Table 2 summaries the timing results given by the six flows in terms of clock period (*ns*). "*SFR*" still out-performs the last four columns, because the simultaneous flow is capable of making necessary adjustments to alleviate the introduced MUX on critical paths due to sharing.

Table 1. Experimental results of MUX area

Designs	SFR	SFR +Post	FU+REG	FU+REG +Post	REG+FU	REG+FU +Post
chem	23936	23936	25600	25600	24576	24576
dir	5824	5824	7168	6592	6336	6336
fei	38720	38720	42432	41408	40832	40384
honda	4672	4672	4992	4992	5248	4864
lee	2688	2688	3328	3136	3008	2944
mcm	3648	3648	4160	3968	4032	4032
wang	2624	2624	2944	2880	3072	3072
Avg.	-	0.0%	13.8%	9.9%	9.8%	8.1%

Table 2. Experimental results of clock period

Designs	SFR	SFR +Post	FU+REG	FU+REG +Post	REG+FU	REG+FU +Post
chem	15.3	15.3	15.3	15.3	15.7	15.7
dir	12.6	12.6	15.0	15.0	15.0	15.0
fei	15.0	15.0	15.0	15.0	15.0	15.3
honda	15.0	15.0	15.3	15.7	15.0	15.3
lee	12.6	12.6	15.0	15.0	15.3	15.3
mcm	12.6	12.6	15.0	15.0	15.0	15.0
wang	13.0	13.0	13.0	13.0	15.3	15.3
Avg.	-	0.0%	8.4%	8.7%	11.3%	12.0%

B. Binding consistency

One of the key factors of the simultaneous and gradual flow is that it can maintain a high consistency among the successive intermediate binding solutions. We achieve this by adding a certain weight in the cost function to prefer previous binding decisions. To show the impact of consistency on the solution quality, we also perform experiments on another cost function, which omits the weight of "*consistency*" in function (0). We denote the algorithm with this new cost function as *SFR'*.

Let $tp_{fu}(i\text{-}1)$ ($tp_{reg}(i\text{-}1)$) be the total number of pairs of operations (variables) that are bound together in the $(i\text{-}1)^{th}$ iteration. Let $cp_{fu}(i)$

($cp_{reg}(i)$) be the number of pairs of operations (variables) that share a same resource in both $(i\text{-}1)^{th}$ and i^{th} iterations. We calculate the FU binding consistency of $(i\text{-}1)^{th}$ and i^{th} iterations as $cp_{fu}(i)$ / $tp_{fu}(i\text{-}1)$, and the REG binding consistency as $cp_{reg}(i)$ / $tp_{reg}(i\text{-}1)$. Figure 5 shows the consistency data of *SFR* and *SFR'* for design *chem*. We can see that the consistency of *SFR* is most often over 90%, much higher than *SFR'*, which is usually less than 20%.

Figure 6 shows the results of the MUX area from these two cost functions. *SFR'* introduces 10% more MUX on average, even worse than the traditional bindings for some designs.

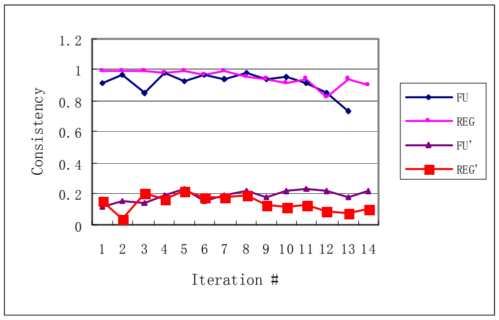

Figure 5. Consistency comparison of two cost functions

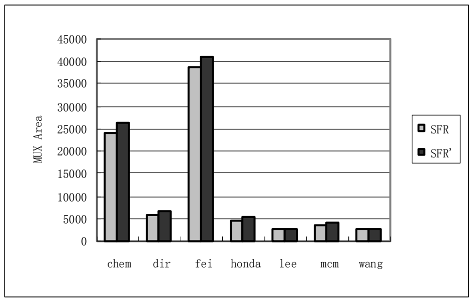

Figure 6. Experimental results of two cost functions

C. *Optimality study*

Table 3. Experimental results of optimality study

Designs	ILP	SFR	FU+REG	REG+FU
t1	1152	1152	1280	1280
t2	1088	1152	1216	1216
t3	960	960	1024	1024
t4	960	1088	1152	1152
t5	1024	1088	1088	1126
Avg.	-	5.1%	11.2%	11.9%

To study the optimality of the simultaneous flow, we tested it against the ILP formulation with small designs of around 15 operations. Table 3 lists the MUX area given by four flows. The second column shows the results from ILP, which are optimal or near-optimal in terms of MUX area. The remaining three columns show results from "*SFR*", "*FU+REG*" and "*REG+FU*". For these small designs, "*SFR*" is very close to ILP with 5% more MUX on average, while the two traditional flows are about 11% worse.

VI. Conclusions and Future Work

In this paper we present a simultaneous FU and register binding algorithm, SFR, to optimize MUX, which can break up the cyclic inter-dependency between functional unit binding and register binding without sacrificing the performance of either task. ILP of the binding problem is formulated to evaluate the optimality of heuristics. Experiments show that our simultaneous synthesis flow is close to optimal solutions and achieves significant reduction for MUX area (12%) and timing (10%) for a set of benchmark designs, compared to traditional binding algorithms.

When it comes to medium- and large-size designs, the ILP formulation cannot be solved in reasonable time due to the huge number of variables and constraints. Also, the real MUX area is replaced by the MUX input number in the ILP formulation, which compromises the optimal solutions. How to design algorithms that can give the lower bound or optimal solution of MUX area will be our future work.

Acknowledgements

This work is supported by National Science Foundation under grants CCF-0096383 and CCF-0530261 and Semiconductor Research Corporation under grant 2006-TJ-1400.

References

[1] Xilinx Web Site, http://www.xilinx.com.

[2] J. M. Chang and M. Pedram, "Register Allocation and Binding for Low Power," *Design Automation Conference*, 1995.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, 1993.

[4] D. Chen, J. Cong, and Y. Fan, "Low-Power High-Level Synthesis for FPGA Architectures," *International Symposium on Low Power Electronics and Design,* Seoul, Korea, pp. 134-139, Aug. 2003.

[5] D. Chen, and J. Cong, "Register Binding and Port Assignment for Multiplexer Optimization," *ASPDAC*, pp. 68-73, January 2004.

[6] D. Chen, J. Cong, and J. Xu, "Optimal Module and Voltage Assignment for Low-Power," *ASPDAC*, pp. 850~855, 2005.

[7] J. Cong, Y. Fan, G. Han, W. Jiang, and Z. Zhang, "Platform-Based Behavior-Level and System-Level Synthesis," *Proceedings of IEEE International SOC Conference*, pp. 199-202, Sept. 2006.

[8] C.Y. Huang, et al. "Data Path Allocation Based on Bipartite Weighted Matching," *Design Automation Conference*, 1990.

[9] T. Kim and C.L. Liu, "An Integrated Data Path Synthesis Algorithm Based on Network Flow Method," *Proc. of the IEEE Custom Integrated Circuits Conference*, 1995.

[10] F. Li, D. Chen, L. He and J. Cong, "Architecture Evaluation for Power-efficient FPGAs," *ACM International Symposium on FPGA*, Feb. 2003.

[11] C. G. Lyuh and K. Taewhan, "High-level Synthesis for Low-Power Based on Network Flow Method," *IEEE Trans. on VLSI Systems*. 2003. 11(3): 364~375.

[12] Barry Pangrle, "On the Complexity of Connectivity Binding," *IEEE Tran. on Computer-Aided Design*, Vol.10, No.11, Nov. 1991.

[13] B. M. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding," *Design Automation Conference*, pp. 536-541, Jun. 1988.

[14] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis," *Design Automation Conference*, pp. 263-270, Jul. 1986.

[15] M. Rim, R. Jain, and R. D. Leone, "Optimal Allocation and Binding in High-Level Synthesis," *Design Automation Conference*, 1992.

[16] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," *ACM International Symposium on FPGA*, Feb. 2002.

[17] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *Trans. on VLSI Systems*, 1995.

[18] J. Stephenson and P. Metzgen, "Logic Optimization Techniques for Multiplexers," *Altera Literature*, 2004.

[19] C-J. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Path in Digital Systems," *IEEE Tran. on CAD of ICAS*, Vol.CADJ, No.3, pp 379-395, Jul. 1986.

[20] H.W. Zhu and C.C. Jong, "Interconnection Optimization in Data Path Allocation Using Minimal Cost Maximal Flow Algorithm," *Microelectronics*, Vol.33, No.9, pp 749-59, Sept. 2002.