Register Binding and Port Assignment for Multiplexer Optimization

Deming Chen and Jason Cong
Computer Science Department
University of California, Los Angeles
{demingc, cong}@cs.ucla.edu

Abstract - Data path connection elements, such as multiplexers, consume a significant amount of area on a VLSI chip, especially for FPGA designs. Multiplexer optimization is a difficult problem because both register binding and port assignment to reduce total multiplexer connectivity during high-level synthesis are *NP*-complete problems. In this paper, we first formulate a *k*-cofamily-based register binding algorithm targeting the multiplexer optimization problem. We then further reduce the multiplexer width through an efficient port assignment algorithm. Experimental results show that we are 44% better overall than the left-edge register binding algorithm on the total usage of multiplexer inputs and 7% better than a bipartite graph-based algorithm. For large designs, we are able to achieve significantly better results consistently. After technology mapping, placement and routing for an FPGA architecture, it shows considerably positive impacts on chip area, delay and power consumption.

I. Introduction

As VLSI systems are becoming increasingly complex as driven by Moore's law [1], and the advances in Deep Sub-Micron (DSM) technologies have made possible multi-million gate designs, circuit designers are facing a tremendous challenge of interconnect-centric design flow. It has been shown that the area of multiplexers and interconnects has by far outweighed the area of functional units and registers. This is especially the case for FPGA designs because a larger amount of transistors have to be provided in the wiring channels and logic blocks to provide programmability for signal transition. Studies show that interconnects alone contribute 70-80% of the total area [2] and 75-85% of the total power [3] [4] for most of the FPGA designs.

A multiplexer (MUX) is a standard complex gate often used in data-path logic to provide multiple connections between functional units (FUs) and registers. The multiplexer area is not linear with its input number. Table 1 shows some characterization data of a 24-bit Carry Look-ahead adder, an 18-bit Wallace-Tree multiplier and some MUXes when they are mapped and fit into FPGAs [5]. A CLB under the *Area* column is a logic block containing four LUTs, and an LUT can implement any logic function of up to four variables in the experiment. A *k*-to-1 MUX selects one of the *k* inputs of the MUX to drive the MUX output. We can observe that the area, delay and power data of a 32-to-1 MUX are almost equivalent to the 18-bit multiplier. These data show how expensive a wide MUX is in terms of chip area, delay and power consumption. It thus motivates us to design highly effective algorithms to reduce the amount and sizes of the multiplexers generated during high-level synthesis. As a result, we reduce the complexity of the connections between functional units and registers and in return reduce the requirement of interconnects during the placement and routing in the later physical design stages.

Given a scheduled data flow graph (DFG), the process of data path generation mainly consists of functional unit binding, register binding, and connection allocation steps. Functional unit binding assigns operations to physical functional units, and register binding assigns variables to registers. Both the operation-to-unit and the variable-to-register mappings determine the multiplexing requirements of the register transfer level (RTL) design. Lastly, connection allocation connects the functional units and registers together. We use the term *connection* to refer to this type of connectivity. Some works use interconnect or interconnection for the same concept. The total connections determine the total MUX inputs, or MUX connectivity. Connection allocation tries to reduce

Functional Unit and MUX	Implementation	Area (CLB)	Delay (ns)	Power (w)
add24bit_cla	Carry look-ahead	26	11.8	0.010
mul18bit_wall	Booth-recoded Wallace	280	14.8	0.308
mux24bit_2to1	Synopsys design	6	0.6	0.002
mux24bit_8to1	Synopsys design	66	4.6	0.023
mux24bit_32to1	Synopsys design	276	10.9	0.240

Table 1: Characterization of adder, multiplier and MUXes

the MUX requirement through connection sharing and/or port assignment. In this work, we assume the DFG is already scheduled, and the functional unit binding is also finished. We concentrate on register binding and port assignment for MUX reduction – both of these problems were proved to be *NP*-complete to solve [6].

There is extensive literature on binding and allocation problems for high-level synthesis [7] [8] [9]. We first review works done to minimize connectivity through functional unit binding and/or register binding. Works in [10] and [11] used a clique partitioning method to reduce register and connection usage. They designed heuristic algorithms to solve the clique partitioning problem due to its high complexity. In [12], a branch-and-bound search algorithm was applied to bind registers and allocate connections while binding functional units. In [13], a breakthrough approach using a weighted bipartite-matching algorithm was presented to solve both register and functional unit binding. The weights on the bipartite graph represented the connection cost when variables or operations were assigned to registers or functional units. It generated optimal number of registers for non-hierarchical DFGs. It compared with [10], [11], and [12] and showed better results in terms of total MUX inputs. Trying to further improve connection solutions, [14] presented an integer linear programming formulation to minimize MUX and wire area. It is optimal with regard to its objective function. The main concern is its high complexity. Authors in [15] presented an integrated data path synthesis flow for scheduling and binding to reduce total connections. Although it had better connection results compared to [13], larger numbers of registers were reported. In [16], a min-cost max-flow algorithm was presented to carry out functional unit binding and register binding for connection reduction. Although the formulation was efficient, it also suffered an increased number of registers than the minimum required. It did not report comparison results with other published algorithms.

The port assignment problem was studied in [17] and [18]. The work in [17] performed global permutation of all the inputs for a functional unit during the MUX generation. The work in [18] designed an integer linear programming algorithm. The complexity of both algorithms is a concern.

In this paper, we present a *k*-cofamily-based algorithm to carry out the register binding task, which guarantees to maintain the minimum number of registers while reducing the MUX usage. We also implement a port assignment algorithm that further reduces the total MUX inputs efficiently after the register binding is done. In the following, Section 2 provides the definitions and problem formulation. Section 3 presents the *k*-cofamily algorithm, and Section 4 presents the port assignment algorithm. Section 5 shows our experimental results, and Section 6 concludes this paper.

II. Definitions and Problem Formulation

The data path of a high-level design can be represented by a DFG, $G = (V, A)$. Let $V = \{v_1, v_2, ..., v_x\}$, $A = \{a_1, a_2, ..., a_y\}$, and $a = \{v_m, v_n\}$ represents the edge from v_m to v_n. Set V corresponds to operations and set A corresponds to data flowing from one operation to another. After scheduling, the life time of each edge (data value) in the DFG is the time during which the data value is active (valid) and is defined by an interval [birth time; death time]. A compatibility graph $Gc = (Vc, Ac)$ for these edges can then be constructed, where vertices correspond to data values (or variables), and there is a directed edge $a_c = (v_i, v_j)$ between two vertices if and only if their corresponding life times do not overlap, and variable v_i comes before v_j. In such a case, we call variables v_i and v_j *compatible* with each other and they can be bound into a single register without life time conflicts. Let w_{ij} denote the weight of the edge a_c, which represents the cost of binding v_i and v_j into a single register.

We now introduce several important concepts in combinatorial theory on partially ordered set, which will be used later in our algorithm. A *partially ordered set (POSET) P* is a collection of elements with a binary relation $\leftarrow$ defined on $P \times P$, which satisfies the following conditions [19]:

1) *reflexive*, i.e., $x \leftarrow x$ for all $x \in P$;
2) *antisymmetric*, i.e., $x \leftarrow y$ and $y \leftarrow x \Rightarrow x = y$;
3) *transitive*, i.e., $x \leftarrow y$ and $y \leftarrow z \Rightarrow x \leftarrow z$;

We say that x and y are *related* if we have either $x \leftarrow y$ or $y \leftarrow x$. An *antichain* in P is a subset of elements such that no two of them are related. A *chain* in P is a subset of elements such that every two of them are related. A *k-family* in P is a subset of elements that contains no chain of size $k + 1$, and a *k-cofamily* in P is a subset of elements that contains no antichain of size $k + 1$ [20]. We can associate weights for *k*-cofamilies, where the minimum weighted *k*-cofamilies are especially important to us. Details will be explained in Section III.

The problem of register binding for MUX reduction can be formulated as the following:

Instance: A scheduled DFG graph $G = (V, A)$, a set of registers R, a set of functional units U, a functional unit binding $\{f_U: v \rightarrow u \mid$ *for all* v, *where* $v \in V$ *and* $u \in U\}$, and a positive integer N.

Question: Is there a register binding $\{f_R: a \rightarrow r \mid$ *for all* a, *where* $a \in A$ *and* $r \in R\}$ such that the number of connections between registers and functional units is $\leq N$?

The problem of port assignment for MUX reduction can be formulated as the following:

Instance: A scheduled DFG graph $G = (V, A)$, a set of registers R, a set of functional units U, a functional unit binding $\{f_U: v \rightarrow u \mid$ *for all* v, *where* $v \in V$ *and* $u \in U\}$, a register binding $\{f_R: a \rightarrow r \mid$ *for all* a, *where* $a \in A$ *and* $r \in R\}$, and a positive integer N.

Question: Is there a port assignment, i.e., for the two input-registers of every operation bound to a functional unit u, which register should be connected to which port of u, such that the number of connections between registers and u is $\leq N$?

As shown in [6], both of these problems were proved to be *NP*-complete.

III. Register Binding with *k*-cofamily Formulation

A. Problem Reduction

In this section, we formulate the register binding problem for MUX reduction as a problem of calculating the minimum weighted cofamilies of a POSET. To obtain such a cofamily, the problem is then reduced to calculating the minimum cost flow in a network.

Given a compatibility graph $Gc = (Vc, Ac)$, let POSET $Pc = \{v_1, v_2, ..., v_n\}$ such that Pc contains all the vertices of Gc, and the compatibility relation defined in Ac can be the relation $\leftarrow$ on the elements of Pc. It is easy to show that the compatibility relation is reflexive, antisymmetric, and transitive. By such, an edge $a_c = (v_i, v_j)$ of Ac represents a relation on the two elements of Pc as $v_i \leftarrow v_j$. Therefore, there is a one-to-one correspondence between one node in Vc and one element in Pc, and between one edge in Ac and one $\leftarrow$ in Pc. We also assign the weight on a_c to the relation $v_i \leftarrow v_j$. Our objective for register binding is as follows: find a subset of Ac that covers all the vertices in Vc in such a way that the total sum of the weights of all the edges in the subset is the minimum with the constraint that all the vertices can only be bound into as many as k registers.

Theorem 1: A register binding on a compatibility graph Gc into k registers is equivalent to find k disjoint chains in the POSET Pc, and each chain contains all the variables bound into one of the k registers.

Theorem 1 can be illustrated from a simple example. In Fig. 1 (a)[1], the solution with an optimal number of registers (in this case, two) is obtained by the partition of two disjoint chains (dashed ovals) in the POSET. Variables in one disjoint chain can then be bound into one separate register.

Therefore, register binding of the nodes in Gc into k registers with the minimum total weight is equivalent to finding k disjoint chains in the POSET Pc with the minimum total weight. There is an important fundamental result on partially ordered sets due to Dilworth [21], which indicates that any k-cofamily in a POSET P can be partitioned into at most k disjoint chains. We have the following Corollary.

Corollary 1: The minimum weighted k-cofamily with at least one antichain of size k in a POSET Pc can be partitioned into exactly k disjoint chains with the minimum total weight to cover every element in Pc.

[1] Edges {1, 4} and {1, 5} are there due to the transitive property of the relations.

Therefore, our goal becomes to find the minimum weighted *k*-cofamily in *Pc*. In [22], an algorithm based on network flow theories was presented to calculate the maximum node-weighted k-cofamilies. Next, we will show how we can convert the calculation of the minimum edge-weighted *k*-cofamily into the calculation of the minimum cost flow in a network.

First, we construct the split graph *G(Pc)* associated with *Pc* as follows: for each element v_i in *Pc*, we introduce two vertices x_i and y_i in *G(Pc)*. We introduce a direct edge (x_i, y_j) in *G(Pc)* if $v_i \leftarrow v_j$. Moreover, we introduce two more vertices *s* (source) and *t* (sink) in *G(Pc)* and add edges (s, x_i) and (y_i, t) for each $1 \le i \le n$. Fig. 1 (b) shows the corresponding split graph of POSET *Pc* of Fig. 1 (a). We choose the capacity of each edge *e* to be 1 and the cost of each edge *e*, denoted as *d(e)*, to be

$$d(e) = \begin{cases} 0, & \text{if } e = (s, x_i) \text{ or } (y_i, t) \\ w_{ij}, & \text{if } e = (x_i, y_j) \\ -1, & \text{if } e = (x_i, y_i) \end{cases}$$

Theorem 2: Let *Pc* be a POSET of *n* elements. Let *k* be the minimum number of registers required to bind all the *n* corresponding variables in the compatibility graph *Gc*. Then, *Pc* has a *k*-cofamily that covers all the *n* elements with minimum total weight if and only if the split graph *G(Pc)* has a (*n*-*k*)-flow of the minimum weight[2].

Our task then becomes to find the minimum cost flow in the network *G(Pc)*. It can be obtained through capacity

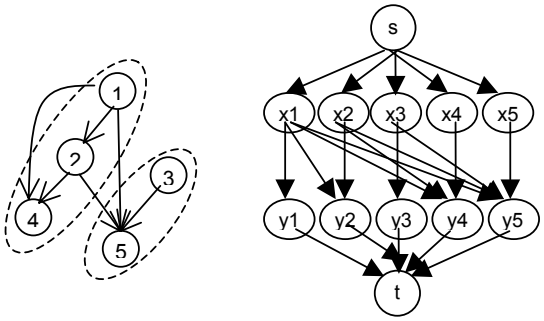

Fig. 1: (a) A POSET *Pc*. (b) The split graph *G(Pc)*.

scaling and successive shortest path computation and has running complexity $O(|E| \log U (|E|+|V| \log |V|))$ [23] [24], where *U* is an upper bound on the largest supply/demand and largest capacity in the network. In our case, $U = n - k$. After we obtain the minimum cost flow, each edge with a unit flow in *G(Pc)*, $e = (x_i, y_j)$, represents that variables v_i and v_j should be bound together into the same register. If there is a flow for $e = (x_i, y_i)$, it means that v_i occupies a register just by itself[3].

B. Cost Function Formulation

In this section, we provide some details for calculating the edge weight w_{ij} if v_i and v_j are to be bound together. A MUX occurs in

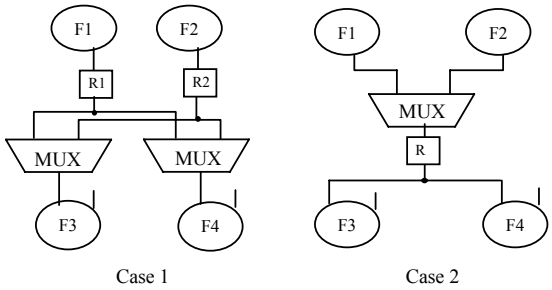

Fig. 2: One example of multiplexing situations

two situations: 1) it is introduced before a register *r* when more than two functional units produce results and store them into this register; 2) it is introduced before a port *p* of a functional unit when more than two registers feeding data to this port. Different register binding will produce different multiplexing situations.

Fig. 2 shows an example. Case 1 binds the two variables driven by functional units F1 and F2 into two separate registers. By such, it saves a MUX between the connections of F1/F2 and their output registers. However, two more MUXes will be required for connections of the two registers R1/R2 to the *fanout* functional units (*fanout_FUs*) F3 and F4. On the other hand, Case 2 binds the two variables from F1 and F2 into a single register, and as a result, a MUX is generated between F1/F2 and register R. Yet, it is a better solution than Case 1 because there are no MUXes required between R and fanout_FUs F3/F4. Notice that if F1 and F2 are actually the same functional unit there will not be a MUX generated in Case 2, which makes Case 2 an even better solution. However, there are situations where Case 1 is better than Case 2, especially when F1 and F2 are different. A simple case happens when none of the fanout_FUs in Case 1 requires a MUX to connect to both register R1 and R2 so it uses one less MUX than Case 2. In the real situation, it is hard to predict which case is better because it all depends on the original DFG data flow, scheduling results, and the functional unit binding solution.

The cost function is defined as follows:

$$w_{ij} = -(N_{mux} + \partial \cdot T_{r\_f} + \beta \cdot T_{fu}) - L$$

where N_{mux} is the number of MUXes saved (or MUXes wasted, i.e., N_{mux} becoming negative) by binding v_i and v_j into a single register (Case 2) than not binding them into a single register (Case 1); T_{r_f} is the total number of connections between register R1/R2 and the fanout_FUs; T_{fu} is the total number of fanout_FUs involved during this tempted binding of v_i and v_j; *L* is a large positive constant[4]; α and β are positive scaling constants. The term T_{r_f} is trying to capture the overall connectivity situation of the fanout_FUs so that some global optimization criteria can be considered. It is needed because not all of the fanout_FUs that connect to R1/R2 require a MUX on their ports for the signals driven by R1/R2. If its value is large, Case 2 is preferred, i.e., v_i and v_j are preferred to be bound into a single register to reduce the total connectivity of fanout_FUs. Term T_{fu} is trying to capture the overall connectivity from another angle, i.e., if more fanout_FUs are involved, Case 2 is preferred. Nonetheless, N_{mux} is set as the overwhelming factor in this cost function because it directly reflects the MUX usage of this binding. The smaller the cost, the better to bind v_i and v_j together.

[2] Proof is omitted. Interested readers please refer to [22], where a maximum weighted *k*-cofamily was computed for node-weighted partially ordered set. A related theorem was presented in [22].

[3] v_i is not compatible with any of the other variables.

[4] *L* guarantees $w_{ij} < -1$ (check the formulation of *d(e)*).

IV. Port Assignment

Port assignment is an important technique for reducing MUX connections between functional units and registers. However, effective heuristics have not been proposed to practically tackle this difficult problem during high-level synthesis. In this section, we will apply a greedy algorithm for port assignment. We will show later that our algorithm is very effective.

In [6], an important lemma proves that finding the minimum connectivity port assignment is equivalent to minimizing the number of input-registers that are connected to both ports of a functional unit u. An optimal solution will be automatically obtained for u if there are no input-registers that drive both ports of u. We will use this lemma to guide our port assignment solutions.

We observe two cases where a register is connected to both ports of u. In Fig. 3, the register in Case 1 contains variables v_1 and v_2 and the operations on the functional unit u is v_1+x_1 and v_2+x_2. Because of the bad port assignments of x_1 and x_2, this register has to drive both ports of u and renders a total of four connections. The register in Case 2 contains a variable v, and the three operations together with the current port assignments force v to drive both ports of u and render a total of five connections.

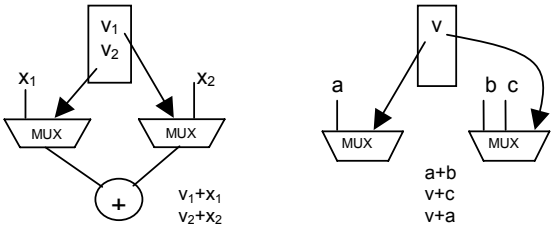

Fig. 3: Two cases of register connecting to both ports

Fortunately, we observe that the connection number of both cases can be reduced using a simple operation of operand swapping. For Case 1, we can swap the port assignments of v_1 and x_1, and for Case 2, v and c. The solutions are illustrated in Fig. 4.

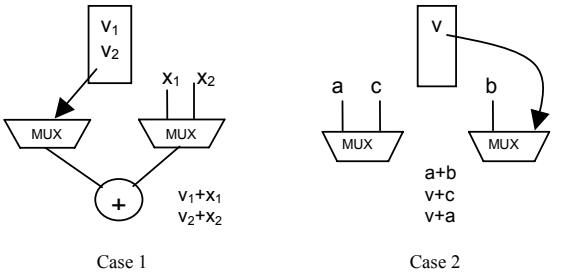

Fig. 4: Operand swapping for the two cases

In our port assignment algorithm, we first provide a solution of a random port assignment. We then find all the registers that drive both ports of their corresponding functional units and perform operand swapping. Fig. 5 provides an outline of the algorithm to handle the Case 2 situation. Case 1 is handled following a similar fashion. The actual implementation deals with other details and complications, such as a combination of Case 1 and 2 in a single register, which are omitted in the outline. The worst complexity is $O(n^2)$ where n is the total MUX input number for the two ports of the targeted functional unit.

There are some situations where operand swapping will not help. For example, if we have an operation as v_1+v_2 in Case 1, the register has to drive both ports. For Case 2, if we encounter a series of circular operations such as $a+b$, $b+c$, and $c+a$, then the register has to drive both ports too. We check these situations first so the operand swapping procedure can exit early if these situations are encountered.

Terminology:

- *opr_pair*, *o* and *o~*: the two operands for a single operation, e.g. addition
- *port_pair*, *p* and *p~*: the two ports on the functional unit *u*, e.g. an adder
- *target_register*, *R*: a register under consideration
- *dual_signals* of *R*, *{d_1, d_2, ..., d_S}*: the signals in *R* that are driving the two ports of *port_pair* simultaneously in the initial solution
- *remnant_signals* of *R*, *rsig_set*: a set that contains the other signals in *R* that only drive one port of the *port_pair*

Case A: $S = 1$

1. Find all the *opr_pairs* that contain *d_1*. It is possible that there are more than one *d_1~*.
2. Find the port to which all the signals in *rsig_set* are assigned in the initial solution. Suppose it is port *p*.
3. Swap *d_1* from port *p~* with all the *d_1~* from port *p*. Make sure there is no new *dual_signals* created for *R* or for other registers that drive *u*.
4. If step 3 is successful, *R* will only drive port *p* and total MUX input number is reduced by 1 for *u*.

Case B: $S > 1$

For each *d_i* that belongs to the set *{d_1, d_2, ..., d_S}*, perform operations 1 to 3 as in the Case A. If all the swapping is successful, *R* will only drive port *p* and total MUX input number is reduced by 1.

Fig 5: An outline of the operand swapping

V. Experimental Results

We compare our algorithm with an enhanced version of the weighted bipartite-matching algorithm [5]. It was based on the original bipartite-matching algorithm published in [13]. As shown in the Introduction, [13] has been consistently winning on MUX

Benchmarks	Node No.	Edge No.	Min Reg. No.
aircraft	2283	4680	528
chem	347	731	112
dir	148	314	50
feig_dct	548	1899	282
honda	97	214	34
mcm	94	252	54
pr	42	134	31
steam_u4mul	220	472	55
u5ml_12	547	1144	163
wang	48	134	30

Table 2: Node, edge and minimum register numbers

input reduction compared to other algorithms when the variables are bound into the minimum number of registers required. In [5], the cost function for the weighted bipartite graph was refined. It was counting the inputs of the involved MUXes directly instead of counting number of input-registers as in [13] when computing the cost of assigning variable v_i to register r_j. Although more input-registers generally mean more MUXes, these two parameters are not always correlating with each other when we consider the registers that drive both ports of functional units. Thus, the refinement in [5] should help to capture the cost more accurately. There is no port assignment in both [13] and [5].

The benchmarks shown in Table 2 are from [25]. The examples used in this study are data-dominated behavioral descriptions with predominantly arithmetic operations that are commonly encountered in signal and image processing applications. The initial scheduling and functional unit binding solutions are generated by the *LOPASS* high-level synthesis system that targets

Benchmarks	*cofamily* with *pa*	*cofamily* w/o *pa*	*bipartite* w/o *pa*	*leftedge* w/o *pa*
aircraft	3579	3734	3923	5420
chem	509	522	592	782
dir	201	209	195	288
feig_dct	972	1034	1060	1389
honda	153	157	173	225
mcm	135	140	147	189
pr	60	63	60	67
steam_u4mul	316	323	337	524
u5ml_12	776	796	854	1269
wang	79	81	78	96

Table 3: Total MUX input number of different algorithms

low power designs [5]. The minimum register number shown in Table 2 is obtained by a left-edge [26] register binding algorithm, which is optimal for a comparability graph generated from non-hierarchical designs, i.e., no alternative paths and loops [9].

Table 3 shows the total MUX input results for *k-cofamily* with *pa* (port assignment), *k-cofamily* without *pa*, *bipartite* without *pa* [5], and *left-edge* without *pa*. Table 4 shows the comparison results. We can see that overall *k-cofamily+pa* is 4% better than *k-cofamily w/o pa*; 7% better than the *bipartite* algorithm *w/o pa*, and 44% better than the *left-edge* algorithm *w/o pa*. We observe that *k-cofamily+pa* performs especially well for large designs. In Table 4, if we only count the designs with more than 200 nodes, although *k-cofamily+pa* will still be 4% better than *k-cofamily w/o pa*, it will be 10% better than *bipartite*, and 55% better than the *left-edge*. The extra gain is obviously coming from register binding. The minimum cost network-flow method always gives us the optimal solution based on the assigned cost on the edges. This shows that the *k-cofamily* formulation of our work is able to capture the connection cost more accurately than the bipartite-based algorithm, especially when the search space is large for designs with large amount of nodes and edges.

Benchmarks	*cofamily* with *pa*	*cofamily* w/o *pa*	*bipartite* w/o *pa*	*leftedge* w/o *pa*
aircraft	1	1.04	1.10	1.51
chem	1	1.03	1.16	1.54
dir	1	1.04	0.97	1.43
feig_dct	1	1.06	1.09	1.43
honda	1	1.03	1.13	1.47
mcm	1	1.04	1.09	1.40
pr	1	1.05	1.00	1.12
steam_u4mul	1	1.02	1.07	1.66
u5ml_12	1	1.03	1.10	1.64
wang	1	1.03	0.99	1.22
Average	1	**1.04**	**1.07**	**1.44**

Table 4: Comparison results of different algorithms with *k-cofamily + pa* (port assignment)

Table 5 shows the upper bound of the total possible reductions of MUX inputs for all the benchmarks by port assignment. This upper bound is obtained assuming that all the registers that drive both ports of the functional units can be changed to only drive a single port through operand swapping. However, it is generally not the case due to the situations explained in Section IV such as circular operations. Nonetheless, we observe an 84% reduction of the upper bound value. This shows the effectiveness of our operand swapping method. Statistics also show that we have achieved the known optimal solutions by 65% of all the workable cases, where we either get rid of the dual-port connections or find out that it is impossible to achieve that. The actually obtained optimal solutions will most likely be larger than 65% if we can afford to compute the optimal solution through an algorithm of exponential complexity, which is forbidden because the total MUX input number for the two ports of certain functional units is very large for the large designs – around 50 for example.

Overall, the run time is fast to complete both *k*-cofamily and port assignment algorithms – within 35 seconds running on a 750 MHz SunBlade 1000 Unix box. An exception is benchmark *aircraft,* which takes slightly more than 1 hour to finish.

To get an idea of how the MUX reduction influences the final design in area, delay and power, we feed two of the RT-level designs generated by *k-cofamily+pa* and *bipartite w/o pa* to a FPGA design/architecture evaluation framework, *fpgaEva_LP* [4]. Please note that we target FPGA applications because we have access to the tool. Our MUX reduction results can be applied to ASIC designs as well. Due to the large number of LUTs (4-input look-up tables) generated, we can only test on two relatively small designs. Tables 6 and 7 show the results. *k-cofamily+pa* consistently improves area, delay and power over the *bipartite* algorithm. These results provide us with some insights into how much impact we can expect through the optimization of data path connections.

Reduction upper bound	334
Actual reduction	279
Reduction Percentage	**84%**

Table 5: Reduction percentage of the upper bound value through operand swapping

VI. Conclusions

In this paper, we presented two efficient algorithms to target the two *NP*-complete optimization problems: register binding for multiplexer reduction and port assignment for multiplexer reduction. We first formulated a *k*-cofamily-based register binding algorithm that guaranteed to maintain the minimum register number required while optimizing the multiplexers. We then further

reduced the multiplexer width through a port assignment algorithm. Experimental results show that, overall, we are 44% better than the left-edge register binding algorithm on the total usage of multiplexer inputs and 7% better than a bipartite graph-based algorithm. We observe 10% better results than the bipartite graph-based algorithm on average for large designs, which shows that our *k*-cofamily-based algorithm is able to capture connection cost more accurately for designs with large numbers of nodes and edges. After technology mapping, placement and routing for an FPGA architecture, it shows consistently positive impacts on chip area, delay and power consumption.

mcm	4-LUT No.	Delay (s)	Power (w)
bipartite w/o *pa*	7528	35.5	1.256
k-cofamily+pa	7085	34.5	1.178
Reduction	**-5.9%**	**-2.8%**	**-6.2%**

Table 6: Area, delay, and power data for *mcm*

steam_u4mul	4-LUT No.	Delay (s)	Power (w)
bipartite w/o *pa*	13732	49.9	2.500
k-cofamily+pa	13093	46	2.385
Reduction	**-4.7%**	**-7.8%**	**-4.6%**

Table 7: Area, delay, and power data for *steam_u4mul*

Acknowledgements

This work is partially supported by Altera Corporation under the California MICRO program, the NSF Grant CCR-0096383, and MARCO/DARPA Gigascale Silicon Research Center (GSRC).

References

[1] G. E. Moore, "Cramming More Components onto Integrated Circuits," Electronics Magazine, 38:114-117, 1965.

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

[3] E. Kusse and J. Rabaey, "Low-Energy Embedded FPGA Structures," Proc. of International Symposium on Low Power Electronics and Design, Aug. 1998.

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

[5] Deming Chen, Jason Cong, and Yiping Fan, "Low-Power High-Level Synthesis for FPGA Architectures," Proc. of Int. Symp. on Low Power Electronics and Design, Aug. 2003.

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

[7] D. Gajski et. al. Editors, *High-Level Synthesis – Introduction to Chip and System Design*, Kulwer Academic Publishers, 1992.

[8] L. Stok, "Data Path Synthesis," *Integration-The VLSI Journal,* Vol.18, No.1, pp.1-71, Dec. 1994, Netherlands.

[9] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., 1994.

[10] 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.

[11] P. G. Paulin, J. P. Knight, and E. F. Girczyc, "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis," 23rd IEEE Design Automation Conference, pp. 263-270, Jul. 1986.

[12] B. M. Pangrle, "Splicer: A Heuristic Approach to Connectivity Binding," 25th ACM/IEEE Design Automation Conference, pp. 536-541, Jun. 1988.

[13] C.Y. Huang, Y.S. Chen, Y.L. Lin, and Y.C. Hsu, "Data Path Allocation Based on Bipartite Weighted Matching," Proc. of the 27th Design Automation Conference, pp.499–504, 1990.

[14] Minjoong Rim, Rajiv Jain, and Renato De Leone, "Optimal Allocation and Binding in High-Level Synthesis," Proc. of DAC, pp.120-123, 1992.

[15] Taewhan 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.

[16] HW Zhu and CC Jong, "Interconnection Optimization in Data Path Allocation Using Minimal Cost Maximal Flow algorithm," *Microelectronics,* Vol.33, No.9, pp.749-59, Sept. 2002.

[17] S. Raje and R. Bergamaschi, "Generalized Resource Sharing," Proc. of ICCAD, 1997.

[18] S. Yamada, "An Optimal Block Terminal Assignment Algorithm for VLSI Data Path Allocation," *Inst. Electron. Inf. & Commun. Eng. IEICE Transactions on Fundamentals of Electronics Communications & Computer Sciences,* Vol. E80-A, No.3, pp.564-6, Mar. 1997, Japan.

[19] C.L. Liu, *Elements of Discrete Mathematics*, New York: McGraw-Hill, 1977.

[20] C. Greene and D. Kleitman, "The Structure of Sperner k-family," *J. Combinatorial Theory, Ser. A,* Vol.20, pp.41-68, 1976.

[21] R. P. Dilworth, "A Decomposition Theorem for Partially Ordered Set," *Ann. Math*, Vol.51, pp.161-166, 1950.

[22] J. Cong and C. L. Liu, "On the k-Layer Planar Subset and Topological Via Minimization Problems," *IEEE Trans. on Computer-Aided Design*, Vol. 10, pp. 972-981, August 1991.

[23] J. Edmonds and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." *J. of the ACM*, Vol. 19, No. 2, 1972.

[24] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows*, Section 10.2., Prentice Hall, 1993.

[25] M. B. Srivastava and M. Potkonjak, "Optimum and Heuristic Transformation Techniques for Simultaneous Optimization of Latency and Throughput," *IEEE Trans. on VLSI Systems*, vol.3 (1), pp.2-19, Mar. 1995.

[26] A. Hashimoto and J. Stevens, "Wire Routing by Optimizing Channel Assignment within Large Apertures", Proc. of 8th Design Automation Workshop, pp. 155-179, 1971.