Compatibility Path Based Binding Algorithm for Interconnect Reduction in High Level Synthesis

Taemin Kim Xun Liu
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh, NC 27695
{tkim3, xunliu}@ncsu.edu

ABSTRACT

This paper describes a register and functional unit (FU) binding algorithm in high level synthesis. Our algorithm targets the reduction of multiplexer inputs. Since multiplexers connect multiple inputs to FUs or registers, the multiplexer count is a good indicator of the interconnect complexity. Reducing the number of multiplexer inputs results in reducing interconnect cost. Specifically, our algorithm constructs a weighted and ordered compatibility graph, and binds operations that form a long path in the graph together. As a result, operations with many flow dependencies and common inputs are bound to same FU, leading to a small number of FU inputs. In addition, the operation variables generated by a single FU are assigned to the same register so that connections between FUs and registers are reduced. We have implemented our algorithm within a MATLAB to Verilog conversion tool, and applied it to a suite of benchmark programs. Our experimental results have shown that the proposed scheme achieves 11.8%, 43.6% and 58.8% multiplexer input count reduction on average over weighted bipartite matching algorithm, *k-cofamily* algorithm and left edge algorithm, respectively. To assess the impact on interconnect reduction, we have generated layouts of the circuits from our Verilog description. It is shown that our approach delivers a 10.1% reduction in total wirelength of global interconnects with minor area overhead of register and FUs in comparison to the best previously proposed scheme.

1 Introduction

With the development of semiconductor fabrication technologies, the feature size of transistors has kept decreasing at exponential rates. On the other hand, chip dimensions continue increasing. Consequently, global interconnects have become the bottleneck for improving performance of next-generation VLSI designs. Specifically, large RC characteristics of global interconnects significantly slow down the signal propagation, limiting the overall system frequency. In addition, the large capacitive load of global interconnects has become a major source of power consumption. As the impact of global interconnect continues growing with technology scaling, effective interconnect reduction schemes are urgently needed to ensure the further advance of IC designs.

Optimization techniques at early stages in a design flow are often believed to be more effective than those at later stages. As a result, interconnect reduction schemes during the high level synthesis step are preferred. Unfortunately, efficient interconnect optimization techniques in the high level synthesis are challenging to develop. In contrast to physical design, i.e., placement and routing, in which location information of circuits is known and can be used to estimate interconnects, high level synthesis only captures operations and their dependencies. Consequently, metrics that track interconnects, e.g., numbers of connections among function units or multiplexer inputs, are usually used as the objective function for minimization.

In this paper, we present a register and FU binding algorithm that targets interconnect reduction. Similar to techniques in [4, 7], we use the total input count of multiplexers to approximate the cost of interconnects. Our scheme proceeds in five steps. Specifically, given a data flow graph (DFG), compatibility graphs are generated at first for different operation types. For each compatibility graph, our scheme groups all operations into the minimal number of long paths that have as many common inputs and flow dependencies as possible. A single FU is used to perform the operations in each path. Output variables in each path are bound to the same register. Thirdly, multiple paths are concatenated to form mega-paths of multi-type operations to increase register sharing. Fourthly, lifetimes of all variables in mega-paths are analyzed to identify variables not bound to registers in the second step. Variables for primary inputs are also searched. Register binding is completed in the fifth step.

Our algorithm minimizes the total number of multiplexer inputs. In addition, it focuses on reducing the number of hardware resources, i.e., registers and FUs. In particular, the operations in a single path, when bound to a single FU and register, do not require any multiplexer that link registers and FUs in the path. As a result, the minimization of path count leads to simplified interconnect configurations and less FUs. Moreover, the construction of mega-path results in register reduction.

We have implemented our scheme into a software tool and applied it to a suite of benchmark designs. Experiments have shown that our scheme derives binding results with the fewest multiplexer input counts in comparison to previously proposed schemes. In particular, 11.8%, 43.6% and 58.8% multiplexer input count reductions are achieved over weighted bipartite matching algorithm [5], *k-cofamily* algorithm [4] and left edge algorithm [17], respectively, with a limited overhead of FUs and registers. The multiplexer input count reduction leads to short total global interconnect length. For the circuit layouts generated from our binding results, the total wirelength is reduced by 10.1%, 22.3% and 30.0% over weighted bipartite matching algorithm, *k-cofamily* algorithm and left edge algorithm, respectively.

The remainder of this paper is organized as follows. Section 2 reviews previous research on interconnect reduction in high level synthesis. Section 3 gives an example that motivates our approach. Section 4 formulates the problem of FU and register binding for interconnect reduction. Our proposed algorithm is described in Section 5. Section 6 provides the experimental results. Section 7 summarizes this paper.

2 Interconnect Reduction in High Level Synthesis

For the past two decades, a plethora of techniques have been proposed for interconnect optimization in high level synthesis. Most of these techniques can be categorized into three groups, based on how interconnects are estimated. Specifically, the interconnect reduction problem is recast as a min-cut graph partitioning problem in [19, 20, 18]. The design objective is to reduce data communication among various operation clusters, which will be bound to different FUs. The second group of schemes further improve interconnect estimation accuracy by incorporating physical design techniques, e.g., floorplanning or placement, into high level synthesis [27, 14, 26, 8, 6, 12, 13]. Since these schemes compute locations

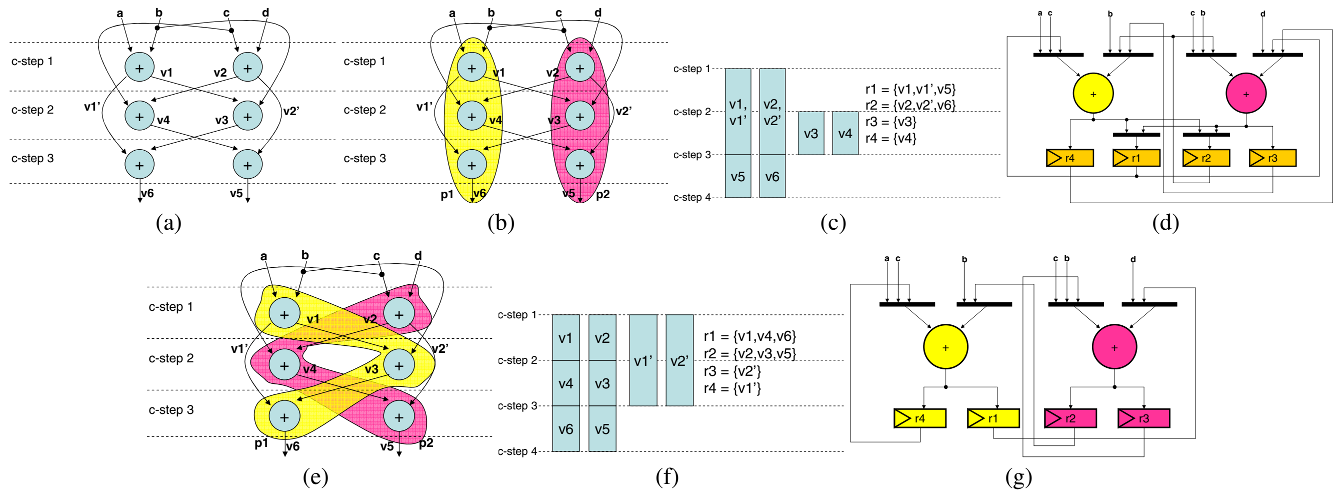

Figure 1: Motivating example (a) scheduled DFG (b) FU binding without interconnect consideration (c) register binding of variables in 1(a) by left edge algorithm (d) synthesis result of 1(b) and 1(c). (e) interconnect oriented FU binding (f) interconnect oriented register binding (g) synthesis result of 1(e) and 1(f)

of circuit blocks, they are able to derive interconnect measuring parameters such as total length and congestion, and conduct interconnect optimizations accordingly. To confine the runtimes in a reasonable range, only quick and sub-optimal physical design methods are applied, however, which degrade the effectiveness of these schemes.

The third group of interconnect reduction schemes in high level synthesis links interconnects to circuit components such as FUs, registers and multiplexers. Specifically, they bind operations with common inputs to the same FU and/or variables with common drivers[1] to the same register. Consequently, the connection among registers and FUs is simplified, leading to interconnect reduction. For register binding schemes, the work in [25, 7] assumes that registers are organized in register files whereas that in [11, 22, 23, 16, 5, 4, 3] assumes distributed registers. Similar variable-register binding schemes are extended to variable-memory binding in [1, 15, 2]. A widely adopted method in interconnect-efficient binding techniques is to use multiplexer input count to approximate the interconnect cost. Multiple multiplexer minimization heuristics are proposed including linear programming [1, 15, 22, 23], edge coloring algorithm [25], network flow algorithm [16, 4] , matching algorithm [11, 5, 7] and simulated annealing [3].

3 Motivating Example

This section gives an example that motivates our work. Figure 1(a) shows a scheduled DFG in which operations are represented as vertices. For simplicity, our example contains only addition operations. Each operation has been scheduled in a time slot, called c-step. Directed edges represent the data flow dependencies. Namely, data is sent from the starting vertex to the ending vertex of an edge. The names of variables are given next to the corresponding edges. Figure 1(b) shows a FU binding solution without interconnect consideration. The operations in the two shaded regions are bound to two adders, either of which can perform an addition in one c-step. Figure 1(c) shows the lifetime of each variable in figure 1(a), as determined by the schedule, and the register binding result by left edge algorithm (LEA). Variables $v1$ and $v1'$ are bound to the same register because they are the same output of the top-left addition operation and considered as one variable in the LEA. So are $v2$ and $v2'$. Figure 1(d) is the netlist derived from figure 1(b) and 1(c). the number of multiplexer inputs is 16. The numbers of adders and registers are two and four, respectively.

If interconnect related information is taken into consideration, the number of multiplexer inputs can be reduced. Figure 1(e) gives the interconnect oriented FU binding. Figure 1(f) shows the lifetimes of the variables in figure 1(e) and register binding . As the figure shows, the number of registers does not increase in comparison to that in figure 1(c). Figure 1(g) shows the synthesis result. The number of multiplexer inputs is eight. The numbers of adders and registers are two and four, respectively. The number of multiplexer inputs is reduced by 50% in comparison to that of 1(d) while the numbers of FUs and registers do not increase. This example reveals that it is possible to reduce multiplexer input count substantially without adding registers or FUs.

4 Preliminaries and Problem Formulation

In this section, we define some notations and formulate the problem of register and FU binding for multiplexer input count minimization.

4.1 Preliminaries

Given a scheduled DFG, two operations are compatible if they are executed in different c-steps. Compatible operations can be mapped to the same FU. The compatibility graph [10, 21] is often created to represent the compatibility among operations in which operations are represented as vertices and compatible operations are connected by edges. Our modified compatibility graph is generated as follows.

Definition 1. Weighted and ordered compatibility graph (WOCG) G(V,E) is a directed acyclic graph (DAG) with vertex set V and edge set E. Vertex set V is composed of vertices each of which represents an operation in the DFG. In addition, the operations in V have the same operation type. Edges representing the compatibility between operations compose the edge set E. The directed edge $u \rightarrow v$ is created between u and v if they are compatible, and u is scheduled earlier than v. In addition, there is a weight w_{uv} on an edge $u \rightarrow v$, representing whether there is a flow dependency between u and v and how many common inputs the two operations have.

Definition 2. A *path* in WOCG is a set of compatible operations $\{op_1, op_2, ..., op_i, ...op_j, ...op_n\}$ ordered based on their scheduled

[1]A driver of a variable is the FU what derives the variable.

times. Namely, *c-step(op_i)* $<$ *c-step(op_j)* and $i < j$, where *c-step(op)* represents the time slot where *op* is scheduled. .

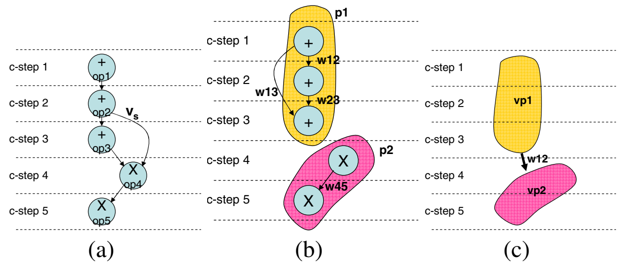

Figure 2: The procedure to construct a PCG (a) scheduled DFG (b) paths constructed (c) PCG constructed

The compatibility concept can be extended to paths. To that end, we define the lifetime of a path as follows.

Definition 3. The lifetime of a path p, denoted as L(p), is from the start execution time of the first operation in p to the end execution time of the last operation in p.

Definition 4. Path compatibility graph (PCG) G_p(V_p,E_p) is a weighted and ordered compatibility graph for paths. The vertices represent paths in WOCG. An edge is inserted between two vertices if their lifetimes do not overlap. Weights are assigned to edges to represent data flow between two paths.

Figure 2 shows the procedure to construct a PCG. Figure 2(a) is a scheduled DFG. Figure 2(b) shows paths $p1$ and $p2$ in addition type WOCG and the multiplication type WOCG, respectively. The lifetime of $p1$ is from c-step 1 to c-step 3, and that of $p2$ is from c-step 4 to c-step 5. Since the lifetimes of $p1$ and $p2$ do not overlap, $p1$ and $p2$ are compatible. Figure 2(c) represents the PCG. The vertices $vp1$ and $vp2$ represent the paths $p1$ and $p2$, respectively. The edge from $vp1$ to $vp2$ shows the compatibility between $p1$ and $p2$. The weight $w12$ is assigned to the edge. The weight computation is explained in section 5.2.

Definition 5. Let P be a path whose elements are $\{op_1, op_2, ..., op_n\}$. Suppose that operation op_i generates variable v_i. If one use-time of v_i is larger than c-step(op$_{i+1}$), v_i is a *side variable*.

The v_s in figure 2(a) is a *side variable* because it is used by $op4$ at c-step 4 which is larger than the c-step of $op3$ which is the successor of $op2$ in $p1$, the generator of v_s.

4.2 Formulation of Interconnect Oriented Binding Problem

The problem of register and FU binding for multiplexer input reduction can be formulated as follows.

PROBLEM: Given a scheduled DFG, find register and FU binding so that the numbers of FUs, registers and inputs of multiplexers among registers and FUs are minimized.

5 The Proposed Heuristic

5.1 An Overview

The problem of register binding and FU binding are correlated [11, 7]. In addition, accurate interconnect extraction requires both binding results. Consequently, simultaneous register and FU binding is crucial in reducing interconnect cost. Our algorithm binds registers and FUs in an integrated fashion. It constructs a WOCG for each FU type, and find paths in the WOCGs. Then, a FU is assigned to the operations in each path. At the same time, a register is also assigned to the variables derived from the operations in a path. Next, compatible paths are merged to form mega-paths for reduction of registers. The lifetime of each mega-path is recorded for the final register binding. Our scheme then finds *side variables* and *primary input variables*. Finally, register binding for mega-paths, side and primary input variables is performed.

5.2 Path Based Binding

In this subsection, we discuss the details of our algorithm. Figure 3 shows the overall algorithm flow, which can be divided into five steps as follows.

```
Path_FU_REG_ Bind : Path based binding algorithm
step1   1  foreach futype in FU type
        2    Create WOCG_futype for futype
        3    Put WOCG_futype into set WOCG
        4  endforeach
step2   5  foreach WOCG_futype in WOCG
        6    until There is no vertex and edge in WOCG_futype
        7      p = Find longest path in WOCG_futype
        8      Put p into path set P_futype
        9      Update WOCG_futype
       10    enduntil
       11    Assign a FU to each path in P
       12  endforeach
step3  13  Create PCG for all paths in path sets P_futype
       14  until There is no vertex and edge in PCG
       15    p = Find longest path in PCG
       16    Record start and end time of p
       17    Put p into global path set P_g
       18    Update PCG
       19  enduntil
step4  20  foreach p_i in P_g
       21    Find side variables in p_i
       22    Record start and end times of the side variables
       23  endforeach
step5  24  Find input variables in a given DFG
       25  Register binding for input, side and path variables
```

Figure 3: The proposed algorithm

Step 1 (*generation of WOCG for each operation type in a given DFG*): Step 1 is from line one to line four in figure 3. In this step, WOCG is created. Each edge in WOCG has a weight. The weight is calculated by incorporating flow dependency and the number of common inputs between operations as follows.

$$W_{ij} = \alpha * F_{ij} + NIN_{ij} + 1 \quad (1)$$

The i and j in equation (1) are the indexes of the starting vertex v_i and ending vertex v_j. F_{ij} is a boolean variable which indicates if there is a flow dependency between v_i and v_j. NIN_{ij} is the number of common inputs of v_i and v_j. α is an integer constant which represents the importance of the flow dependency. In our implementation, we set α = 2. If there is no flow dependency and common inputs, the weight is one.

The reason of using F_{ij} in the computation of W_{ij} is that the more flow dependencies in a path, the less number of inputs a FU assigned to the operations in the path has. Moreover, since the output registers are directly connected to the FU and highly probably placed near the FU, the interconnect length between the FU and output register is short. Figure 4 illustrates how flow dependency in a path affects the minimization of multiplexer input count. For simplicity, the primary inputs of the DFG is not considered. Figure 4(a) demonstrates the FU and register binding without consideration of flow dependency. Figure 4(b) is its synthesis result. The variables $\{v1, v2, v3, v7, v8\}$ and $\{v6, v4, v5\}$ are bound to register $r1$ and $r2$, respectively. The number of multiplexer inputs is eight. Figure 4(c) presents the FU binding with the incorporation of flow dependency. The consecutive operations in $p1$ and $p2$ have dependency relations. Figure 4(d) is synthesis result of figure 4(c). Variables $\{v1, v2, v6, v4, v8\}$ in path $p1$ are bound to register $r1$. The variables $\{v3, v7, v5\}$in $p2$ are bound to $r2$. The synthesis result shows that there is no multiplexer. Thus, binding with consideration of flow dependency is effective in multiplexer input reduction.

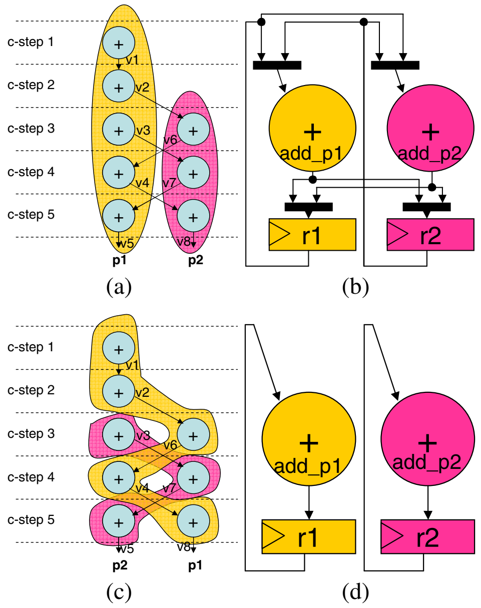
Figure 4: **Illustration of how flow dependency affect binding result**

The reason NIN_{ij} is included in W_{ij} is that the number of common inputs of two vertices could affect multiplexer input count in the synthesis result. If two operations have a common input, the common input does not require a multiplexer when both operations are bound to the same FU. Figure 5 shows two binding results for one WOCG, i.e. $\{p1, p1'\}$ in figure 5(a) and $\{p2, p2'\}$ in figure 5(c). All operations in $p1, p1', p2$ and $p2'$ have dependency relation, respectively. In case of path $p1$, there are two common inputs for the operations in the path i.e., a and b are inputs of more than one operation. On the other hand, path $p2$ has no common inputs. Figure 5(b) and 5(d) show synthesis results for $\{p1,\ p1'\}$ and $\{p2,\ p2'\}$, respectively. Adder *add_p1* and *add_p1'* are for operations in path *p1* and *p1'*, respectively. Registers *reg_p1* and *reg_p1'* are for variables in path *p1* and *p1'*. Same rule is applied to operations and variables in path *p2* and *p2'*. As can be seen, $\{p1,\ p1'\}$ is better binding with less number of multiplexer inputs.

Figure 6 shows the WOCG constructed from figure 1(a). The directed edge represents the compatibility between operations and the scheduled order. The weight on each edge is calculated by equation (1). For example, the edge from $op1$ to $op4$ means that $op1$ and $op4$ are compatible, and $op1$ is scheduled before $op4$. Since there is a flow dependency between them, $F_{14} = 1$. They have common input b, therefore $NIN_{14} = 1$. The weight w_{14} is $2*1+1+1 = 4$.

Step 2 (*FU binding step*): After generating WOCGs for all FU types, our algorithm finds paths in the WOCGs by applying the longest path algorithm [9]. After the longest path is found, edges and vertices in the path are removed from the corresponding WOCG. Edges that link vertices out of the path to those in the path are also removed. The search and removal of the longest path are repeated on the modified WOCG iteratively until there is no vertex and edge. The process is from line six to line 10 in figure 3.

Figure 7 shows the path generation process of the WOCG in figure 6. Figure 7(a) is the initial WOCG. The first longest path in the WOCG is highlighted in bold. Figure 7(b) shows the updated WOCG after the first longest path is removed with the second longest path in bold. The intuition behind the generation of the longest path is to reduce the total number of paths. Since the number of paths is equal to the number of FUs in the corresponding type

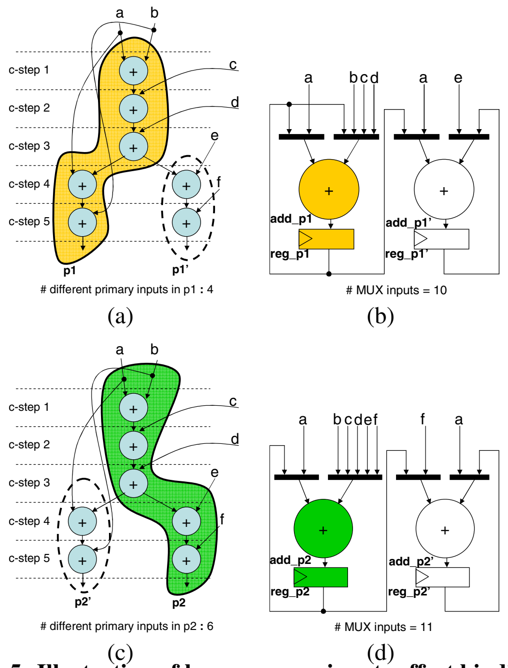
Figure 5: Illustration of how common inputs affect binding result

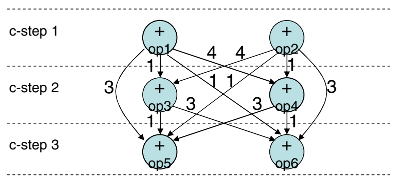
Figure 6: WOCG constructed from figure 1(a)

and directly related to the number of registers, path reduction leads to a small number of FUs and registers. In addition, the weight of an edge is monotonic to the amount of flow dependency and common inputs. Thus, finding a path with the largest weight leads to the reduction of multiplexer inputs. The operations in the path are bound to the same FU in line 11 of figure 3.

Step 3(*finding path variables step*): Step 3 is from line 13 to line 19 of figure 3. The outputs of operations in the same path in a WOCG are stored in the same register. As a result, it is crucial to reduce the number of paths for register reduction. Different from FU binding in which paths from different WOCG cannot be combined together since they contain operations with various types, these paths can be potentially combined in register binding. For path combination, the compatibility of all paths should be considered. Figure 8 illustrates the process of path combination. Figure 8(a) shows the paths for addition and multiplication types. Figure 8(b) shows the compatibility graph for paths in figure 8(a). Moreover, weights are assigned to edges of the path compatibility graph. The weights are calculated in a similar way as the weights in WOCG. However, only flow dependency between paths are considered because common inputs of different FUs do not affect multiplexer input count. Path merging is conducted by the longest path algorithm. Figure 8(c) presents the result. The outputs of operations in a merged path will be assigned to a single register. The maximum number of drivers of the register is the number of operation types in the merged path. Figure 9 shows register binding result of a single merged path in figure 8(c). The register is driven by two drivers, i.e. the adder and multiplier. The lifetimes of merged paths are recorded for the final register binding in step 5.

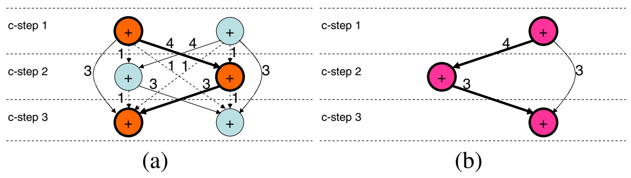

Figure 7: WOCG update process (a) the first longest path (b) updated WOCG and the second longest path

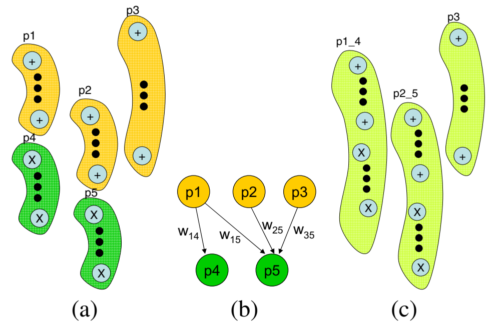

Figure 8: Construction of PCG (a) paths in a given DFG (b) path compatibility graph (c) merged paths

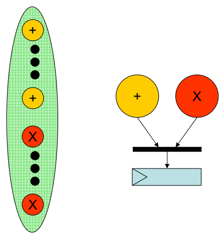

Figure 9: FU and register binding for a merged path

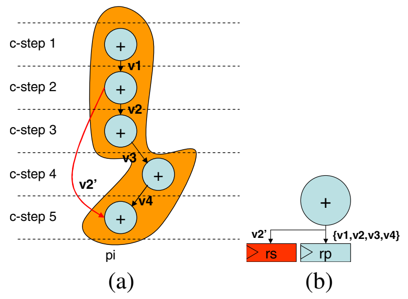

Figure 10: Side variable binding (a) path having a side variable v_2' (b) binding result

Step 4 (*finding side and input variables step*): Step 4 is from line 20 to line 24 in figure 3. Data dependencies among paths and between un-adjacent operations within paths are analyzed in line 21 to search for side variables. A register is inserted for each side variable. Figure 10 shows an example. In figure 10(a), v_2' is a side variable because it has a consumer which is not a successor of its generator within the compatibility path. Since a single register is used for all variables within the path, it cannot keep the value of $v2'$ until c-step 5. Thus, additional register is added.

In addition to registers for side variables, registers for primary input variables should also be added. The primary input variables are searched in line 24. Constants are not bound to registers. The start and end use-time are recorded for the primary input and side variables for final register binding.

Step 5 (*final register binding step*): Step 5 is in line 25. Specifically, the lifetimes of all variables from the previous steps are examined. It is possible that the lifetimes of some variables do not overlap. Registers assigned to the non-overlapping variables are merged by left edge algorithm [17].

6 Experimental Results

We have developed a software program based on our algorithm. We integrate the program in our high level synthesis tool which converts a MATLAB program to a Verilog RTL description. Our tool is implemented in C++. Benchmark programs are from [24]. They are data-oriented programs common in digital signal processing (DSP) algorithms. Since the benchmarks are coded in C originally, we convert the C codes to MATLAB descriptions. An in-house compiler is used to generate DFGs for all programs. The scheduling of DFGs is done before binding procedure.

We compared our algorithm to weighted bipartite matching algorithm [5], LEA [17] and k-cofamily algorithm [4] for multiplexer input reduction. Since LEA guarantees the minimum number of registers if there is no control dependency in a given DFG, we compared the register counts of the results from LEA and our algorithm. In addition to the number of registers, we compared the number of functional units of the results from LEA and ours. We also counted the numbers of global interconnects of every benchmark processed by all binding algorithms. The global interconnects are defined as interconnects among FUs and/or registers. Moreover, in order to verify that reducing the number of multiplexer inputs is directly related to total global wirelength reduction, we generated the layout of each benchmark circuit based on the binding results using SOC Encounter from Cadence and measured the total length of global interconnects. In case of global interconnect counts and total global wirelength, we compared our algorithm to LEA, weighted bipartite matching algorithm and k-cofamily algorithm. Table 1 presents the numbers of multiplexer inputs for all benchmark programs processed by our algorithm, LEA, weighted bipartite matching algorithm and k-cofamily algorithm. The second column to fifth column give the numbers of multiplexer inputs. Column six to nine show the normalized numbers with respect to the results of our algorithm. As the table shows, the average multiplexer input count of our algorithm is 58.8%, 43.6% and 11.8% less than those of LEA, *k-cofamily* algorithm and weighted bipartite matching algorithm, respectively. Table 2 shows the total numbers of global interconnects. It is clear that the total number of global interconnects is closely related to the number of multiplexer inputs. As the table 2 shows, the total numbers of global interconnects of circuits resulting from our algorithm are 35.2% less than that of LEA, 7.9% less than that of weighted bipartite matching algorithm and 24.1% less than that of k-cofamily algorithm on the average. Table 3 presents the total wirelength of each benchmark design for each algorithm. Our algorithm reduces total wirelength by 30.0%, 10.1% and 22.3% over LEA, weighted bipartite matching algorithm and k-cofamily algorithm, respectively. Table 4 shows the comparison of register counts between the optimal algorithm, i.e. LEA, and our algorithm. The numbers in the second and third column are the register counts. The fourth and fifth column show the normalized results. As the table presents, LEA and our algorithm produce the same results for most benchmarks. On average, our algorithm increases registers by 4%. Table 5 shows the comparison of the number of adders and multipliers from the results by LEA and our algorithm. The N_+ and N_* from the second column to the fifth column represent adder counts and multiplier counts, respec-

Benchmarks	ours	leftedge[17]	bipartite[5]	k-cofamily[4]	ours	leftedge	bipartite	k-cofamily
aircraft	1384	2901	1597	2888	1.000	2.096	1.154	2.087
chem	245	491	280	454	1.000	2.004	1.143	1.853
dir	200	267	217	229	1.000	1.335	1.085	1.145
feig_dct	679	1016	764	860	1.000	1.496	1.125	1.267
honda	93	175	101	139	1.000	1.882	1.086	1.495
mcm	180	225	186	206	1.000	1.250	1.033	1.144
pr	76	103	91	97	1.000	1.355	1.197	1.276
u5ml	410	839	427	747	1.000	2.046	1.042	1.822
wang	100	129	114	116	1.000	1.29	1.14	1.16
arai	69	97	81	83	1.000	1.406	1.174	1.203
lee	102	133	115	137	1.000	1.304	1.127	1.343
AVG					**1.000**	**1.588**	**1.118**	**1.436**

Table 1: Comparison of multiplexer input counts

Benchmarks	ours	leftedge	bipartite	k-cofamily	ours	leftedge	bipartite	k-cofamily
aircraft	5.89E4	7.60E4	6.10E4	7.55E4	1.000	1.289	1.035	1.282
chem	1.90E4	2.72E4	1.95E4	2.56E4	1.000	1.430	1.026	1.350
dir	1.07E4	1.37E4	1.11E4	1.22E4	1.000	1.282	1.037	1.144
feig_dct	3.24E4	4.20E4	3.29E4	3.60E4	1.000	1.298	1.018	1.113
honda	6.13E3	8.44E3	6.52E3	7.21E3	1.000	1.378	1.064	1.177
mcm	8.27E3	1.08E4	9.22E3	1.01E4	1.000	1.308	1.115	1.226
pr	3.58E3	4.76E3	4.13E3	4.77E3	1.000	1.330	1.154	1.331
u5ml	3.05E4	4.53E4	3.13E4	4.11E4	1.000	1.485	1.026	1.349
wang	4.11E3	5.47E3	4.50E3	4.79E3	1.000	1.331	1.096	1.166
arai	3.00E3	3.90E3	3.17E3	3.26E3	1.000	1.319	1.071	1.104
lee	4.27E3	5.81E3	5.07E3	6.20E3	1.000	1.360	1.187	1.451
AVG					**1.000**	**1.352**	**1.079**	**1.241**

Table 2: Comparison of global interconnect counts

Benchmarks	ours	leftedge	bipartite	k-cofamily	ours	leftedge	bipartite	k-cofamily
aircraft	8.78E7	1.14E8	9.41E7	1.12E8	1.000	1.298	1.071	1.270
chem	1.02E7	1.34E7	1.08E7	1.29E7	1.000	1.319	1.059	1.268
dir	6.87E6	8.38E6	7.19E6	7.54E6	1.000	1.220	1.046	1.097
feig_dct	4.02E6	5.24E6	4.19E6	4.19E6	1.000	1.303	1.042	1.041
honda	2.67E6	3.45E6	2.83E6	3.37E6	1.000	1.293	1.061	1.263
mcm	2.84E6	3.60E6	3.03E6	3.16E6	1.000	1.268	1.069	1.112
pr	1.09E6	1.48E6	1.36E6	1.46E6	1.000	1.356	1.248	1.338
u5ml	1.92E7	2.74E7	2.10E7	2.39E7	1.000	1.425	1.097	1.246
wang	1.09E6	1.33E6	1.18E6	1.29E6	1.000	1.214	1.082	1.176
arai	3.42E5	4.31E5	4.07E5	4.20E5	1.000	1.261	1.190	1.228
lee	7.02E5	9.40E5	7.82E5	1.03E6	1.000	1.339	1.113	1.464
AVG					**1.000**	**1.300**	**1.101**	**1.223**

Table 3: Comparison of the total wirelength

tively. The N_+ and N_* in the sixth and the ninth column represent the normalized results of adder and multiplier counts, with respect to the LEA, respectively. As table 5 shows, the average increases of adders and multipliers of our algorithm are 2% and 7%, respectively.

7 Conclusion

In this paper, we present a FU and register binding algorithm for interconnect reduction. Our algorithm identifies long paths in the compatibility graph generated from a DFG, and conducts FU and register binding concurrently. Our scheme targets the minimization of multiplexer inputs by analyzing the flow dependency and common inputs of operations. Experimental results show that our algorithm reduces the number of multiplexer inputs by 11.8% and the total number of global interconnects by 7.9% on average in comparison to the best previously proposed algorithms. Our scheme also achieves a total global wirelength reduction by 10.1% on average.

Benchmarks	leftedge[17]	ours	leftedge	ours
aircraft	1176	1176	1	1
chem	181	181	1	1
dir	72	72	1	1
feig_dct	94	97	1	1.03
honda	54	54	1	1
mcm	38	38	1	1
pr	15	20	1	1.33
u5ml	285	285	1	1
wang	17	19	1	1.12
arai	12	12	1	1
lee	16	16	1	1
AVG			**1**	**1.04**

Table 4: Comparison of register counts

	leftedge		ours		leftedge		ours	
Benchmarks	N_+	N_*	N_+	N_*	N_+	N_*	N_+	N_*
aircraft	52	1159	52	1159	1	1	1	1
chem	15	176	15	176	1	1	1	1
dir	8	64	8	64	1	1	1	1
feig_dct	12	8	15	8	1	1	1.25	1
honda	9	52	9	52	1	1	1	1
mcm	16	30	16	30	1	1	1	1
pr	8	14	8	14	1	1	1	1
u5ml	24	277	24	277	1	1	1	1
wang	8	8	8	10	1	1	1	1.25
arai	8	2	8	3	1	1	1	1.5
lee	8	8	8	8	1	1	1	1
AVG					**1**	**1**	**1.02**	**1.07**

Table 5: Comparison of multiplier and adder counts

8 REFERENCES

[1] I. Ahmad and C. Y. R. Chen. Post-processor for data path synthesis using multiport memories. In *Proceedings of ICCAD*, November 1991.

[2] H. A. Atat and I. Ouaiss. Register binding for fpgas with embedded memory. In *Proceedings of FCCM*, April 2004.

[3] A. Avakian and I. Ouaiss. Optimizing register binding in fpgas using simulated annealing. In *Proceedings of ReConFig*, September 2005.

[4] D. Chen and J. Cong. Register binding and port assignment for multiplexer optimization. In *Proceedings of ASPDAC*, January 2004.

[5] D. Chen, J. Cong, and Y. Fan. Low-power high-level synthesis for fpga architectures. In *Proceedings of ISLPED*, August 2003.

[6] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang. Architecture and synthesis for on-chip multicycle communication. *IEEE Transactions on CAD of integrated circuits and systems*, 2004.

[7] J. Cong, Y. Fan, and W. Jian. Platform-based resource binding using a distributed register-file microarchitecture. In *Proceedings of ICCAD*, November 2006.

[8] J. Cong, Y. Fan, X. Yang, and Z. Zhang. Architecture and synthesis for multi-cycle communication. In *Proceedings of ISPD*, April 2003.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *INTRODUCTION TO ALGORITHMS,2nd Edition*. The MIT Press, 2001.

[10] D. Gajski, A. Wu, N. Dutt, and S. Lin. *HIGH-LEVEL SYNTHESIS:Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[11] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu. Data path allocation based on bipartite weighted matching. In *Proceedings of DAC*, June 1990.

[12] A. Kaplan, P. Brisk, and R. Kastner. Data communication estimation and reduction for reconfigurable systems. In *Proceedings of DAC*, June 2003.

[13] R. Kastner, W. Gong, X. Hao, F. Brewer, A. Kaplan, P. Brisk, and M. Sarrafzadeh. Layout driven data communication optimization for high level synthesis. In *Proceedings of DATE*, March 2006.

[14] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi. Behavior-to-placed rtl synthesis with performance-driven placement. In *Proceedings of ICCAD*, November 2001.

[15] T. Kim and C. L. Liu. Utilization of multiport memories in data path synthesis. In *Proceedings of DAC*, June 1993.

[16] T. Kim and C. L. Liu. An integrated data path synthesis algorithm based on network flow method. In *Proceedings of CICC*, May 1995.

[17] F. J. Kurdahi and A. C. Parker. Real:a program for register allocation. In *Proceedings of DAC*, June 1987.

[18] X. Liu and M. C. Papaefthymiou. Design of a 20-mb/s 256-state viterbi decoder. *IEEE Transactions on VLSI Systems*, 2003.

[19] R. Mehra, L. M. Guerra, and J. M. Rabaey. Low-power architectural synthesis and the impact of exploiting locality. *Journal of VLSI Signal Processing*, 1996.

[20] R. Mehra, L. M. Guerra, and J. M. Rabaey. A partitioning scheme for optimizing interconnect power. *IEEE Journal of Solid-State Circuits*, 1997.

[21] G. D. Micheli. *SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS*. McGraw Hill, 1994.

[22] C. A. Papachristou and H.Konuk. A linear program driven scheduling and allocation method followed by an interconnect optimization algorithm. In *Proceedings of DAC*, June 1990.

[23] M. Rim, R. Jain, and R. D. Leone. Optimal allocation and binding in high-level synthesis. In *Proceedings of DAC*, June 1992.

[24] M. B. Srivastava and M. Potkonjak. Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput. *IEEE Transactions on VLSI Systems*, 1995.

[25] L. Stok. Interconnect optimization during data path allocation. In *Proceedings of EURO-DAC*, September 1990.

[26] J. Um, J. hoon Kim, and T. Kim. Layout-driven resource sharing in high-level synthesis. In *Proceedings of ICCAD*, November 2002.

[27] J.-P. Weng and A. C. Parker. 3d scheduling:high-level synthesis with floorplanning. In *Proceedings of DAC*, June 1991.