Simultaneous Scheduling and Binding for Resource Usage and Interconnect Complexity Reduction in High-Level Synthesis

Cong Hao, Jian-Mo Ni, Hui-Tong Wang and Takeshi Yoshimura
Graduate School of IPS, Waseda University, Hibikino 2-7, Wakamatsu, Kitakyushu, 808-0135, Japan

***Abstract*—This paper proposes a simultaneous scheduling and binding approach for resource and interconnect reduction in high-level synthesis. The scheme incorporates the operation scheduling into functional unit (FU) and register binding, targeting the reduction of both resource and interconnect reduction. A simplified weighted and ordered compatibility graph (SWOCG) based binding algorithm is also proposed and runs tens of times faster than the WOCG based binding algorithm. The experimental results show that our proposal achieves 4% to 15% reduction in resource usage and interconnect reduction, and also runs 5X faster compared to previous works.**

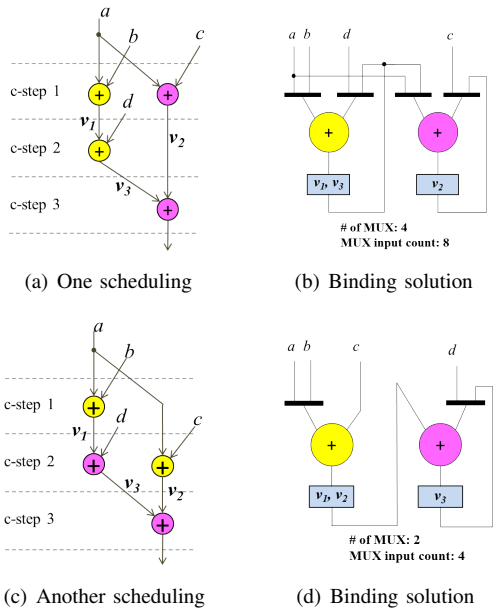
I. INTRODUCTION

With the exponential growth of the chip capacity and design scale, reducing the on-chip resources, like computing components and routable area, becomes one of the key factors in IC designs. Especially in high-level synthesis, being aware of the physical resource and interconnect information at early stages, like in resource binding or even in scheduling, are expected to be more effective than that in later stages like placement and routing. Therefore reducing the functional units (FU) and registers, as well as the global interconnects among FUs, registers and multiplexers (MUX) are important targets in scheduling and binding of high-level synthesis.

There are already some works for resource usage and interconnect optimization in high level synthesis. Some of them take accurate physical measurement by incorporating floorplan or placement into binding procedure [1] [3] [2], but with algorithm efficiency concern. Some of the works use the number of multiplexer inputs to describe the interconnect complexity since the multiplexer count directly indicates the interconnect number, and optimize the connections between FUs and registers with the objective of reducing the multiplexer input count. The work in [4] and [5] propose algorithms on the FPGA platform for register binding, but do not consider other resource bindings which also correlate with interconnects.

Kim and Liu propose FU and register binding algorithms for global interconnect reduction in [7] and [8] which greatly outperforming other works. In [7] they first build the weighted and ordered compatibility graphs (WOCG) for each type of operations, find longest paths for FU binding, and find path compatibility for register binding. This method shows high optimality but still has large room for improvement, mainly because of the large complexity of WOCG, and also because that the operation scheduling is fixed which may result in sub-optimal solutions in the binding procedure.

Consequently in this work, we propose a simultaneous operation scheduling and resource binding algorithm to minimize the resource usage and the interconnect complexity, by adopting and improving the WOCG based binding method. We describe our contributions as the following:



(a) One scheduling (b) Binding solution

(c) Another scheduling (d) Binding solution

Fig. 1: Scheduling results affect the binding solution in terms of interconnect.

1) We first propose a **Simplified Weighted and Ordered Compatibility Graph (SWOCG)**, and apply the dynamic programming on the SWOCG for resource binding, which is tens times faster than the WOCG based binding with the same solution quality. It enables us combine the binding with scheduling within reasonable execution time.
2) Moreover, we **simultaneously solve the scheduling and binding problems** instead of independent resource binding, by iteratively adjusting the operation scheduling and binding solution for optimized resource usage and interconnect complexity.

II. MOTIVATING EXAMPLE AND PROBLEM FORMULATION

A. Motivating Examples

Here we show how the scheduling results affect the binding solution in terms of resource usage and interconnect. Fig.1 shows two different scheduling results with corresponding binding solutions. Under the scheduling in Fig.1(a), the MUX input count is 8, while under the scheduling in Fig.1(b), the MUX input count is 4 and the number of MUXes is also reduced by 2. The example of how scheduling affects the

resource usage is omitted here due to the page limitation. As a result, incorporating scheduling into binding procedure for less resource usage and interconnect is highly expected.

B. Problem Formulation

We formulate the problem in this work as the following:

Given: A control-data flow graph (CDFG).

Goal: Find a scheduling for each operation, and find the FU and register binding so that the number of FUs, registers and the inputs of MUXes among registers and FUs are minimized:

$$Min : \lambda_1 \cdot FU + \lambda_2 \cdot REG + \lambda_3 \cdot MUX \tag{1}$$

III. PROPOSED ALGORITHM

A. Overall Flow

The flow of our simultaneous scheduling and binding approach is shown in Algorithm 1. We first apply the List Scheduling on given CDFG to generate the initial operation scheduling, and then perform our proposed SWOCG based binding, which is to be introduced in Section III-B. After binding, we calculate the resource usage and MUX input count, and then perform scheduling adjustment, which is to be introduced in Section III-C. This procedure is repeated until the stop criteria is met.

Algorithm 1 Simultaneous Scheduling and Binding

```
1: for each FU type in DFG do
2:     while there are vertexes in DFG do
3:         for each control-step t do
4:             for each opit on t do
5:                 Find predecessor operations of opit
6:                 for each predecessor operations opj do
7:                     record data dependencies, common primary inputs
8:                     Calculate the weights
9:                 end for
10:            end for
11:            optimize the paths
12:        end for
13:    end while
14: end for
```

B. SWOCG Based Binding Approach

1) Weighted and Ordered Compatibility Graph (WOCG): Given a scheduled CDFG, two operations are compatible if they are executed in different control steps, and can be mapped into the same FU. The modified weighted and ordered compatibility graph (WOCG) is first introduced in [7] to for better interconnection concern in binding procedure.

The WOCGs are generated for each operation type in a given CDFG. The vertices in WOCG represent the operations, and a directed edge $u \rightarrow v$ in WOCG represent the compatibility between operations u and v, and u is scheduled before v. For each edge $e = (u,v)$, a weight value w_{uv} is attached to represent if there is any data dependency or common inputs between operations u and v, which is calculated as:

$$w_{uv} = \alpha \times D_{uv} + \beta \times PI_{uv} + \gamma \times MI_{uv} + 1 \tag{2}$$

where D_{uv} is a boolean variable that indicates if there is a flow dependency between u and v, PI_{uv} is the number of common primary inputs between u and v, and MI_{uv} is the number of common FU inputs that go out from the same functional unit. Fig.2(a) is the original scheduled CDFG, and Fig.2 shows its corresponding WOCG.

On the WOCG, the longest path is calculated for FU binding, and every time a longest path is found, the vertices that belongs to the path is mapped into the same FU. In order to make it possible that every two compatible operations can be mapped to the same FU, each WOCG should be a graph that each operation is connected to all the operations that scheduled later, as shown in Fig.2(b). It results in a large WOCG graph with approximately $N^2/2$ edges and N vertices, and therefore the time complexity for building WOCG and calculating the longest path is $O(N^2)$. When the design grows large, the algorithm suffers from significant slow down.

Based on this observation, we propose simplified WOCG model, which speeds up the FU binding for tens of times.

2) Simplified WOCG: Instead of building the complete compatibility graph WOCG as shown in Fig.2(b), we generate a **Simplified WOCG (SWOCG)** with edge set E and V, only to provide the information of data dependency and common inputs between operations. The vertex set V represents all the operations of the same type in the scheduled CDFG, and the edge set E is composed as $E = E_d \cup E_{pi} \cup E_{ci} \cup E_{st}$. Each edge $e = (u,v) \in E_d$ means that there is a data dependency from u to v in CDFG; each edge $e = (u,v) \in E_{pi}$ means that the operation u and v share a common primary input, and each edge $e = (u,v) \in E_{ci}$ means that the operation u and v share a common FU input; the edges in E_{st} are connected to source S and sink T. The weight of each edge $e = (u,v)$ in SWOCG are defined as:

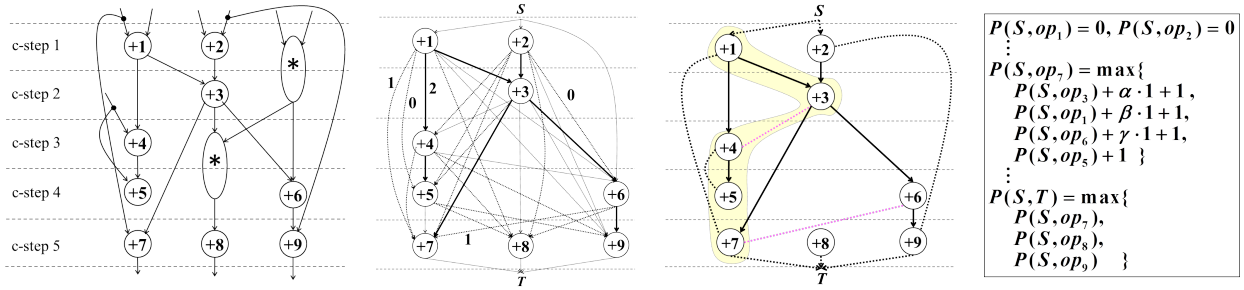
$$w_{uv} = \begin{cases} \alpha, & e = (u,v) \in E_d \ ; \\ \beta, & e = (u,v) \in E_{pi} \ ; \\ \gamma, & e = (u,v) \in E_{ci} \ . \end{cases} \tag{3}$$

where α, β and $\gamma > 1$.

Fig.2(c) shows an example of the simplified WOCG generated from the original scheduled CDFG shown in Fig.2(a). The solid edges are the original data dependencies, the doted edges as (op_1, op_7) means that there is a common primary input between operation op_1 and op_7, and the edges as (op_6, op_7) means that operations op_6 and op_7 share the same output of another operation, op_3 in this case, as shown in the scheduled CDFG in Fig.2(a).

According to our experiments, the approximate number of edges in SWOCG is no more than twice of the original number of edges in CDFG, so that the complexity of building a SWOCG is $O(N+E)$, which is about N times smaller than that of building a WOCG.

3) SWOCG Based Longest Path Calculation: In order to provide the opportunity that every two compatible operations can be mapped to the same FU, the longest path on SWOCG does not necessarily goes along the edges in SWOCG, but can go through any two operations as long as they are not scheduled in the same control step. For example in Fig.2(c), the longest path could be $\{op_1, op_3, op_4, op_5, op_7\}$, which



(a) The scheduled CDFG (b) Weighted and Ordered Compatibility Graph (WOCG) (c) Simplified WOCG (SWOCG) with one longest path (d) Example of Dynamic Programming on SWOCG

Fig. 2: The example of WOCG, simplified WOCG (SWOCG), and the idea of dynamic programming used on the SWOCG for longest path calculation. The SWOCG is much simpler than WOCG, and also the computation time for longest path is linear.

means that these five operations are mapped to the same adder, even though there is no edge between op_5 and op_7.

On SWOCG, we denote op_i^t as the i-th operation which is scheduled in control step t. $\rho(op_i, op_j)$ denotes the path length if directly goes from op_i to op_j, and $P(S, op_j)$ denotes the longest path from S to op_j. $\rho(op_i, op_j)$ is calculated as:

$$\rho(S, op_i) = \rho(op_i, T) = 0,\ (S, op_i), (op_i, T) \in SWOCG \quad (4)$$

$$\rho(op_i, op_j) = \sum_{e=(i,j)\in SWOCG} w_{ij} + 1 \quad (5)$$

and $P(S, op_j^t)$ is calculated as:

$$P(S, op_j^t) = max\|P(S, op_i^{t_0}) + \rho(op_i^{t_0}, op_j^t)\|_{t_0<t} \quad (6)$$

Our goal is to find the longest path $P(S,T)$ from S to T, and then map all the operations in the longest path to the same physical functional unit.

For simplicity, we assume that there is at least one operation in each control step, otherwise the control step should be simply deleted. We first give a lemma based on the SWOCG.

Lemma 1. The longest path from S to operation op_i^t $P(S, op_i^t)$ must goes either from an operation op_k^{t-1}, or goes from an operation $op_k^{t_0}(t_0 < t-1)$ where $e = (k,i) \in SWOCG$.

Proof: Assume that the longest path from S to op_i^t directly comes from $op_j^{t_0}(t_0 < t-1)$ where $e = (j,i) \notin SWOCG$, and an operation op_k^{t-1} exists. Then we have:

$$\begin{aligned} &P(S, op_i^t) = P(S, op_j^{t_0}) + \rho(op_j^{t_0}, op_i^t) \\ &\rho(op_j^{t_0}, op_i^t) = 1,\ since\ e = (j,i) \notin SWOCG \end{aligned} \quad (7)$$

On the other hand, if the path goes through op_k^{t-1}, the path length p from op_k^{t-1} to op_i^t is:

$$\begin{aligned} p &= P(S, op_j^{t_0}) + \rho(op_j^{t_0}, op_k^{t-1}) + \rho(op_k^{t-1}, op_i^t) \\ &\geq P(S, op_j^{t_0}) + 1 + 1 > P(S, op_i^t) \end{aligned} \quad (8)$$

p is larger than $P(S, op_i^t)$ contradicts with the assumption that $P(S, op_i^t)$ is the longest path. Assumption is wrong and thus Lemma 1 is proved. ■

Given Lemma 1, when computing $P(S, op_i^t)$, we only need to calculate $P(S, op_i^t) = P(S, op_j^{t-1}) + P(op_j^k, op_i^t)$, or $P(S, op_i^t) = P(S, op_k^{t_0}) + P(op_k^{t_0}, op_i^t)$ if $e = (k,i) \in SWOCG$. The detailed algorithm is shown in Algorithm 2.

Algorithm 2 SWOCG Based Longest Path Calucation

Require: CDFG
Ensure: The longest path $P(S,T)$ in the WOCG
for each control step t **do**
  for each op_i^t in control step t **do**
    $P(S, op_i^t) = 0$
    for each op_j^{t-1} scheduled in $t-1$ **do**
      $P(S, op_i^t) = \text{Max}\{P(S, op_i^t),\ P(S, op_j^{t-1}) + 1\}$
    end for
    for each $op_j^{t_0}$, $t_0 < t$, $e = (j,i) \in SWOCG$ **do**
      $P(S, op_i^t) = \text{Max}\{P(S, op_i^t),\ P(S, op_j^{t_0}) + \rho(op_j^{t_0}, op_i^t)\}$
    end for
  end for
end for

The Fig.2(d) shows an example of Algorithm 2. When computing the longest path for operation op_7, the path from op_5 is calculated since op_5 is scheduled one control step earlier than op_7; also, pathes op_1, op_3 and op_6 are also calculated because there are edges between them and op_7. Then the maximum value is adopted as the longest path of op_7.

C. Scheduling Adjustment

To perform quick scheduling adjustment in each iteration, we adopt the list scheduling algorithm but modify the priority of each operation as the following:

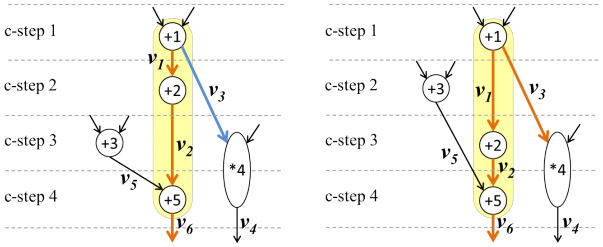
$$pr(op_i) = LP(op_i) + \gamma(op_i) \quad (9)$$

where the $LP(op_i)$ is the longest path from sink node T to operation op_i, and the $\gamma(op_i)$ is a random value varies between $[-\beta, +\beta]$. Given the priority for each operation, we follow the idea of list scheduling, that in each control step, we pick up the highest priority operations with no data dependencies, until the upper limit of resource usage is reached.

Adding the random term $\gamma(op_i)$ to the priorities of operations is a simple yet efficient way to explore resource and interconnect friendly scheduling solutions, which gives the operations more freedom for flexible scheduling results without violating the resource and latency constraint. Fig.3 explains how the random term contributes to resource usage

TABLE I: Comparison between our proposal and WOCG based binding

DFG	WOCG Binding[7]					SWOCG Binding					List Scheduling & SWOCG Binding				
	N*	N+	REG	MUX	Time	N*	N+	REG	MUX	Time	N*	N+	REG	MUX	Time
ar	2	3	11	33	16ms	2	3	11	33	2ms	2	3	11(-2)	31(-2)	6ms
ae	2	4	36	39	17ms	2	4	35(-1)	37(-2)	3ms	2	4	34(-2)	35(-4)	16ms
ad2	2	4	27	38	16ms	2	4	27	38	3ms	2	4	25(-2)	37(-1)	16ms
ellip	2	4	15	44	15ms	2	4	15	44	2ms	2	3(-1)	13(-2)	45(+1)	14ms
mpeg	2	3	23	38	21ms	2	3	23	37(-1)	3ms	2	3	22(-1)	35(-3)	20ms
fft	5	5	59	76	48ms	5	4(-1)	59	77(+1)	8ms	5	4(-1)	52(-10)	75(-1)	46ms
rand0	7	4	28	86	14ms	7	4	28	86	2ms	6(-1)	3(-1)	25(-3)	81(-5)	21ms
rand1	11	3	39	132	123ms	11	3	39	132	19ms	10(-1)	3	38(-1)	129(-3)	54ms
rand2	27	16	153	476	2624ms	27	17(+1)	153	476	37ms	26(-1)	12(-4)	153	458(-18)	371ms
AVR										36.63X	-5%	-15%	-5%	-4%	5.13X



(a) An additional register is needed for v_3

(b) v_3 can share the same register with the one used in longest path

Fig. 3: The random term in operation priority of list scheduling gives op_2 an opportunity to be scheduled after op_3 even their longest path value are the same, which makes (b) a better solution than (a).

and interconnect reduction in scheduling scheme. Suppose the longest path in this example is $\{op_1, op_2, op_3\}$, the variables within the path will be mapped to the same register, say r_1, including v_1, v_2 and v_6. Since the distances from operations op_2 and op_3 to op_5 are the same, op_2 is usually scheduled before op_3, then v_3 cannot share the same register with r_1 and another register is needed, as shown in Fig.3(a). However, in our proposed method, the random term $\gamma(op_i)$ gives op_2 an opportunity to be scheduled after op_3, like shown in Fig.3(b), which enables v_3 to be mapped into r_1 and results in less resource usage and MUX input count.

IV. Experimental Results

Our proposed algorithm is implemented in C language and run on a Linux Workstation with AMD Operon 2.6 GHz CPU and 4GB Memory. The benchmarks are from [9]. We compare our work with the previous WOCG based binding framework in [7], both in algorithm efficiency and the quality of final binding results. The parameters in the objective function Equation 1 are set as: λ_1 is 3, λ_2 is 2 and λ_3 is 1.

We first evaluate the efficiency of our proposed SWOCG based binding algorithm by comparing it to the WOCG based binding in terms of execution time, as shown in Table I. It show that our SWOCG based binding achieves 36.6X speedup compared to WOCG based binding in average, which shows a significant efficiency improvement. Especially when the benchmark grows larger, say the benchmark *rand*2 with hundreds of operations in DFG, our algorithm achieves a much higher speedup, up to 70X.

We also evaluate our simultaneous scheduling and binding scheme in Table.I. It shows that our simultaneous scheduling and binding approach for resource usage and interconnect reduction is very effective that we achieve 5% in multiplier reduction, 15% in adder reduction, 5% in register reduction and 4% in MUX input count reduction. Moreover, taking the advantage of fast SWOCG based binding, our approach does not suffer from any runtime overhead, but still runs averagely 5X faster than the original WOCG based binding algorithm with a significant resource and interconnect reduction.

V. Conclusion

In this work we propose a Simplifed WOCG based binding algorithm, and moreover a simultaneous scheduling and binding approach for resource and interconnect reduction in high-level synthesis. The experimental results show that our SWOCG based binding runs 36.6X faster than WOCG based approach, and our simultaneous scheduling and binding achieves from 4% to 15% resource and interconnect reduction compared with the previous work.

References

[1] Um, Junhyung, Jae-hoon Kim, and Taewhan Kim. "Layout-driven resource sharing in high-level synthesis." Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design. ACM, 2002.

[2] Kastner, Ryan, et al. "Layout driven data communication optimization for high level synthesis." Design, Automation and Test in Europe, 2006. DATE'06. Proceedings. Vol. 1. IEEE, 2006.

[3] Kaplan, Adam, Philip Brisk, and Ryan Kastner. "Data communication estimation and reduction for reconfigurable systems." Design Automation Conference, 2003. Proceedings. IEEE, 2003.

[4] Atat, Hassan Al, and Iyad Ouaiss. "Register binding for FPGAs with embedded memory." Field-Programmable Custom Computing Machines, 2004. FCCM 2004. 12th Annual IEEE Symposium on. IEEE, 2004.

[5] Avakian, Annie, and Iyad Ouaiss. "Optimizing register binding in FPGAs using simulated annealing." Reconfigurable Computing and FPGAs, 2005. ReConFig 2005. International Conference on. IEEE, 2005.

[6] Cong, Jason, et al. "Architecture and synthesis for multi-cycle communication." Proceedings of the 2003 international symposium on Physical design. ACM, 2003.

[7] Kim, Taemin, and Xun Liu. "Compatibility path based binding algorithm for interconnect reduction in high level synthesis." Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design. IEEE Press, 2007.

[8] Kim, Taemin, and Xun Liu. "A global interconnect reduction technique during high level synthesis." Proceedings of the 2010 Asia and South Pacific Design Automation Conference. IEEE Press, 2010.

[9] Cong, Hao, Song Chen, and Takeshi Yoshimura. "Network simplex method based Multiple Voltage Scheduling in Power-efficient High-level synthesis." Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific. IEEE, 2013.