ELectronics EXpress

IEICE Electronics Express, Vol.4, No.14, 448–454

Operation scheduling for the synthesis of false loop free circuits

Shih-Hsu Huang[a)] **and Chun-Hua Cheng**

Department of Electronic Engineering,

Chung Yuan Christian University, Chung Li, Taiwan, R.O.C.

a) *shhuang@cycu.edu.tw*

Abstract: If resource constraints are specified, the false loop free circuit must be built during the scheduling phase. Although the previous approach guarantees to have a false loop free circuit mapping, it does not attempt to minimize the number of control steps. In this paper, we present an effective approach to find a scheduled code, which not only guarantees to have a false loop free circuit mapping but also to minimize the number of control steps. Experimental results show that our approach achieves good results in terms of the number of control steps.

Keywords: electronic design automation, high-level synthesis

Classification: Science and engineering for electronics

References

[1] L. Stok, "False Loops through Resource Sharing," *in the Proc. of Int. Conf. on Computer-Aided Design*, pp. 345–348, 1992.

[2] S. H. Huang, T. Y. Liu, Y. C. Hsu, and Y. J. Oyang, "Synthesis of False Loop Free Circuits," *in the Proc. of Asia and South Pacific Design Automation Conf.*, pp. 55–60, 1995.

[3] S. Davidson, D. Landskov, B. D. Shriver, and P. W. Mallet, "Some Experiments in Local Microcode Compaction for Horizontal Machines," *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 460–477, 1981.

[4] P. Faraboschi, J. A. Fisher, and C. Young, "Instruction Scheduling for Instruction Level Parallel Processors," *Proc. of the IEEE*, vol. 89, no. 11, pp. 1638–1659, 2001.

[5] C. Chen and M. Sarrafzadeh, "Power-Manageable Scheduling Technique for Control Dominated High-Level Synthesis," *Proc. of IEEE Design, Automation, and Test in Europe Conference and Exhibition*, pp. 1016–1020, 2002.

1 Introduction

A false path of a combinational circuit is defined as a path that will never be traversed under any combination of input values. A false loop is a special case of false path, where the start point and the end point of the false path are the same. False loops mainly occur in design style where the constraints

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

allow chaining of many data path operations, especially in the designs that consist of many control steps or consist of many parts executed under exclusive conditions. Because most timing analyzers and delay calculators cannot handle false loops, it is therefore desirable to generate a circuit that contains no false loops.

The HIS system [1] is the first behavior synthesis system to tackle the false loop problem. It solves the problem during the allocation phase [1]. But it has a drawback that in many cases a false loop free circuit cannot be built under given resource constraints. Additional resources have to be added in order to build a false loop free circuit. Different from HIS system, Huang et al. [2] propose an approach to solve the problem during the scheduling phase. A resource allocation graph is defined to model the circuit configuration, such as operation chaining and resource sharing, at the higher abstraction. The proposed approach is to incrementally construct an acyclic resource allocation graph, which corresponds to a false loop free circuit mapping under the specified resource constraints. As a result, a scheduled code, which guarantees to have a false loop free circuit mapping under the given resource constraints, is obtained.

Although [2] solves the false loop problem, the approach is a greedy algorithm itself. It is obvious that different directed resource allocation graphs have different circuit mappings. Since the acyclic resource allocation graph in [2] is constructed greedily, in many cases the obtained scheduled code still can be further improved. The improvements can be evaluated in terms of the number of control steps. Moreover, the improvements cannot be achieved during the allocation phase, since the scheduled code is already fixed.

In this paper, we present an effective approach to find a scheduled code, which not only guarantees to have a false loop free circuit mapping under the given resource constraints, but also to minimize the number of control steps. Our approach can be easily incorporated into any of existing list scheduling like algorithms [3, 4, 5].

2 PRELIMINARIES

To detect false loops during the scheduling process, a resource allocation graph is defined to model the circuit configuration. Let FU(o) denote the functional unit to which operation o is assigned.

DEFINITION 1: A resource allocation graph G(V,E) is defined as a directed graph, where: (1) each vertex in V represents a functional unit defined in the resource constraints; and (2) a directed edge e connecting from vertex $FU(o_i)$ to vertex $FU(o_j)$, if operation o_i and operation o_j are chained in the same control step.

The following theorem states that we can detect false loops on a resource allocation graph.

THEOREM 1: There is a false loop in the final hardware, if and only if there is a cycle in the corresponding resource allocation graph.

Proof: The proof is given in [2]. **Q.E.D.**

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

Based on Theorem 1, an approach that guarantees to generate a false loop free schedule is proposed in [2]. The core algorithm is a list scheduling like algorithm. The approach is to maintain an acyclic resource allocation graph during scheduling. Initially, the edge set in the resource allocation graph is empty. Whenever two operations o_i and o_j are chained in the same control step, a directed edge from FU(o_i) to FU(o_j) forms in the resource allocation graph. Note that the directed edges in the acyclic resource allocation graph define a topological ordering of the functional units. At each control step, the functional units are chosen in the sequence of their topological ordering. Although this approach ensures to have a false loop free schedule, it does not attempt to minimize the number of control steps. Therefore, this approach is called greedy false loop free scheduling.

3 THE PROPOSED APPROACH

Because the bindings of functional units are limited by their topological ordering, the greedy false loop free scheduling may introduce extra control steps. To minimize the number of control steps, we propose the concept of tentative edge. Then, the greedy false loop free scheduling is iteratively invoked with the tentative edges pre-defined in the resource allocation graph.

A. The Motivation

Let's use the data flow graph in Figure 1 as an example. Suppose that the resource constraints are two adders (i.e., add1 and add2) and one subtractor (i.e., sub1). By applying list scheduling, the scheduled code is obtained as shown in Figure 1 (a). Operations 1, 2, and 3 are scheduled in control step 1, and operations 4, 5, and 6 are scheduled in control step 2. We can easily prove that the scheduled code is not a false loop free schedule. Since operation 3 uses the outputs of operations 1 and 2 at control step 1, we have directed edges FU(1) $\rightarrow$ FU(3) and FU(2) $\rightarrow$ FU(3) in the resource allocation graph. Note that FU(3) is the subtractor sub1. Since operations 1 and 2 are scheduled at the same control step, they must be assigned to different adders. Hence, we have directed edges add1 $\rightarrow$ sub1 and add2 $\rightarrow$ sub1. However, because operations 4 and 6 are chained at control step 2, a directed edge from FU(4) (i.e., sub1) to FU(6) (i.e., the adder executes operation 6) must be added to the resource allocation graph. Consequently, there is a cycle (i.e., false loop) between sub1 and FU(6). In other words, there is no binding, which is false loop free, for this schedule. Figure 1 (b) is a corresponding resource allocation graph for the scheduled code, where FU(1) and FU(5) are add1, FU(2) and FU(6) are add2, and FU(3) and FU(4) are sub1. We can find that a cycle between add2 and sub1 forms in this resource allocation graph.

The objective of greedy false loop free scheduling [2] is to find a schedule in which the corresponding resource allocation graph is acyclic. Let's use the data flow graph in Figure 1 (a) as an example. The scheduling starts with control step 1. Operations 1, 2, and 3 are assigned to add1, add2, and sub1, respectively. Edges add1 $\rightarrow$ sub1 and add2 $\rightarrow$ sub1 are added into the

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

resource allocation graph. Next, we move to control step 2. Operations 4 and 5 are assigned to sub1 and add1, respectively. Then, we move to operation 6. We cannot assign operation 6 to the functional unit add2, because a cycle between sub1 and add2 will be formed. There is no binding available for operation 6. Therefore, we cannot schedule operation 6 in this control step, even though add2 has not been sealed. After we complete the task of false loop free scheduling, we obtain the schedule code as shown in Figure 1 (c). In order to avoid false loops, operation 6 is scheduled at control step 3. The resource allocation graph is shown in Figure 1 (d). If compared with the scheduled code without false loop avoidance as shown in Figure 1 (a), the false loop free scheduling introduces an extra control step.

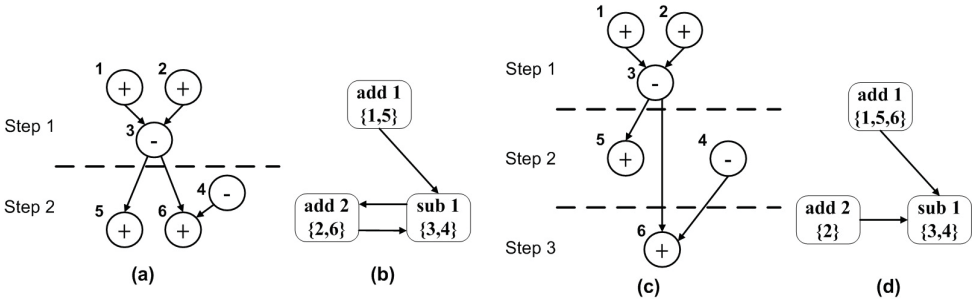


Fig. 1. An example.

B. Tentative Edges

In this paper, we propose the concept of tentative edge for further optimization.

DEFINITION 2: During the process of greedy false loop free scheduling, a directed edge is not allowed to add into the resource allocation graph if and only if it will form a cycle in the resource allocation graph. We define such a directed edge is a tentative edge, which is forbidden to add into the resource allocation graph due to false loop avoidance.

Let's use Figure 1 (c) and Figure 1 (d) to illustrate the definition. There is no binding for operation 6 at control step 2, because the edge sub1 → add2 is not allowed. Otherwise, a cycle will form in the resource allocation graph. Note that the dependency of operations 4 and 6 belongs to the chaining type sub1 → add2. If the directed edge sub1 → add2 is allowed, operations 4 and 6 can be chained together at control step 2. Therefore, we say that directed edge sub1 → add2 is a tentative edge.

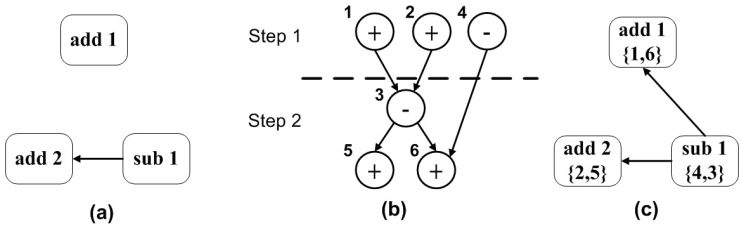
C. The Algorithm

Our basic idea is to perform the greedy false loop free scheduling again on the same control/data flow graph by using the tentative edge as a design constraint. Under the prerequisite of the tentative edge, the greedy false loop free scheduling constructs a new resource allocation graph and hence obtains a new scheduled code. The new scheduled code will be accepted

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

if the number of control steps is reduced. The iteration repeats until the number of control steps cannot be further reduced or no new tentative edge appears.

Let's use Figure 1 for illustration. According to the experience of greedy false loop free scheduling as shown in Figure 1 (c), we find that the directed edge sub1 → add2 is a tentative edge. Therefore, by using the tentative edge as the prerequisite, we perform greedy false loop free scheduling one more time. Before scheduling, the tentative edge sub1 → add2 is added into the resource allocation graph. As a result, we have an initial resource allocation graph as shown in Figure 2 (a). Based on this resource allocation graph, the scheduler starts from control step 1. Firstly, operation 1 is scheduled and functional unit add1 is sealed. Then, operation 2 is scheduled and functional unit add2 is sealed. Although operation 3 is ready, it cannot be chained with operation 2 because of the tentative edge sub1 → add2. Otherwise, a cycle between sub1 and add2 will form. Therefore, operation 4 is scheduled and functional unit sub1 is sealed. Next, the scheduler moves to control step 2. Operation 3 is scheduled and hence functional unit sub1 is sealed. Then, operation 5 is scheduled and chained with operation 4. In order to save the number of interconnections, the directed edge sub1 → add2 is used. Therefore, functional unit add2 is sealed. Finally, operation 6 is scheduled and chained with operation 4. Functional unit add1 is sealed. A new directed edge sub1 → add1 is added into the resource allocation graph. The final scheduled code, which only takes two control steps, is shown in Figure 2 (b). The corresponding resource allocation graph, which is false loop free, is shown in Figure 2 (c).




```
Procedure OurApproach()
{PE0 = Ø; S0 = FalseLoopFreeScheduling(PE0); no = 0;
Repeat
BestS = Sno; BestPE = PEno; no = no + 1; Sno = BestS;
For each tentative edge e in TE(BestS) do
 {TempPE = BestPE ∪{e};
  TempS = FalseLoopFreeScheduling(TempPE);
  if (cost(TempS) < cost(Sno)) then
    {Sno = TempS; PEno = TempPE;}}
until (BestS=Sno);
return(BestS);}
```

(d)

Fig. 2. The proposed algorithm.

Figure 2 (d) shows the pseudo code of the proposed algorithm. The notation *no* denotes the number of repeat-until iterations executed. Since a

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

tentative edge is added into the set of pre-defined edges per repeat-until iteration, the notation no also means the number of pre-defined edges. The notation S_{no} denotes the best schedule currently in the no iteration. The notation PE_{no} denotes the pre-defined edges for the S_{no}. The notation TE is a function, which returns the set of tentative edges of a false loop free schedule. The notation *cost* is a function, which returns the cost of a schedule code. The notation *BestS* is the best schedule up to now. The notation *BestPE* denotes the set of pre-defined edges for the schedule *BestS*. The details of the algorithm are elaborated as the below.

During the scheduling of $S0$, we may have some tentative edges. Each tentative edge e in $TE(S0)$ is sequentially evaluated. Based on the evaluations, the best schedule S1 is chosen among the false loop free schedules obtained in the for-loop iterations. The PE_1 is the tentative edge which results in $S1$. If the cost of $S1$ is fewer than the cost of $S0$, the repeat-until iteration is executed one more time. Then, the tentative edges in $TE(S1)$ are evaluated sequentially. The repeat-until iteration proceeds until the scheduled code cannot be further improved.

Let's use the data flow graph in Figure 1 (a) as an example. Firstly, we perform greedy false loop free scheduling without any pre-defined edge. Note that PE_0 is $\emptyset$. The $S0$ is obtained as shown in Figure 1 (c). The set $TE(S0)$ is { sub1 $\rightarrow$ add2 }. Therefore, we try to use the tentative edge *sub1* $\rightarrow$ *add2* as a pre-defined edge. The *TempPE* becomes {sub1$\rightarrow$add2}. As a result, $S1$ are obtained as the scheduled code as shown in Figure 2 (b). The cost of $S1$ is fewer than the cost of $S0$ in terms of the number of control steps. Because the set $TE(S1)$ is $\emptyset$, we cannot further improve the false loop free schedule $S1$. Therefore, the scheduled code, as shown in Figure 2 (b), is returned.

4 Experimental Results

Five benchmark circuits are used to test the effectiveness of our approach. Benchmark circuits *Ellip*, *LPC*, *QRS*, and *Filter* are adopted from [2]. Benchmark circuit *Example* denote the example used in this paper. Table I tabulates the experimental results with the resource constraints on the number of adders (*#adds*), the number of subtracters (*#subs*), the number of ALUs (*#alus*), and the number of multipliers (*#muls*). The column *Without* gives the scheduling result without false loop avoidance, including the number of control steps (*#steps*), and if it contains a false loop which cannot be eliminated under the resource constraints or not (*floop*). The column *[2]* denotes the greedy false loop free scheduling [2]. The column *Ours* denotes our proposed approach.

From Table I, we find that: benchmark circuits *Ellip*, *LPC*, *Filter*, and *Example* contain false loops if false loop avoidance is not considered. If only using greedy false loop free scheduling [2], benchmark circuits *LPC*, *Filter*, and *Example* need an extra control step. However, by applying our proposed approach, false loop free circuit mapping can be built without introducing an extra control step.

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007

Table I. Experimental results on benchmark circuits.

Circuit	Resource constraints				Without		[2]	Ours
	#adds	#subs	#alus	#muls	#steps	floop	#steps	#steps
Ellip	3	0	0	1	8	yes	8	8
LPC	0	0	2	2	11	yes	12	11
QRS	1	1	0	0	37	no	37	37
Filter	3	0	0	1	11	yes	12	11
Example	2	1	0	0	2	yes	3	2

5 Conclusions

In this paper, we present an effective approach to find a scheduled code, which guarantees to have a false loop free circuit mapping under the given resource constraints, with the objective to minimize the number of control steps. Benchmark data shows that false loop free circuits can be built without introducing extra control steps.

Acknowledgments

This work was supported in part by the National Science Council of Taiwan, R.O.C., under grant number NSC 93-2220-E-033-001.

DOI: 10.1587/elex.4.448
Received May 01, 2007
Accepted June 25, 2007
Published July 25, 2007