MILP-based Scheduling for Clock Latency Minimization in High-level Synthesis

Keisuke Inoue[1] and Mineo Kaneko[2]
[1]Department of Global Information and Management, Kanazawa Technical College
2-270 Hisayasu, Kanazawa, Ishikawa 921-8601, Japan
[2]School of Information Science, Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Nomi, Ishikawa 923-1211, Japan
E-mail : [1]k-inoue@neptune.kanazawa-it.ac.jp

Abstract: The hardware cost of the realization of a clock skew schedule tends to increase due to additional delay elements. Therefore, there is a demand to reduce clock latencies at early design stage. This paper discusses a novel design problem in high-level synthesis, to minimize clock latencies with scheduling, since scheduling highly affects the required clock latencies. Two variations of the clock latency minimization problem are formulated as mixed integer linear programs. Experimental results show that the proposed approach can reduce, the maximum clock latency, and the sum of clock latencies, over the conventional design, in practical time.

Keywords—**MILP, Clock skew, High-Level Synthesis**

1. Introduction

In clock synchronous circuit design, the clock signal is globally distributed to registers via a clock network. The clock latency of a register is the difference between the clock arrival times at the register and at a reference register, and the clock skew is the difference of clock latencies between two registers. A clock skew schedule is a set of clock latencies, and clock skew scheduling (CSS) is a task of deciding a clock skew schedule. CSS views clock skews as a manageable resource rather than a liability for circuit performance improvement [1]. Recently, CSS has been studied in high-level synthesis (HLS) [4], since the earlier we are in the design flow, more is the amount of improvement possible.

A clock skew schedule can be implemented by inserting delay elements into the clock line. The amount of delay elements increases roughly in proportion to the amount of clock latencies. It causes area and power overheads, and is vulnerable to the physical conditions such as process variation, which would results in timing violation. Therefore, reducing clock latencies is important in CSS-aware design.

In this paper, we point out that scheduling, a primary task in HLS, significantly affects the amount of clock latencies. Specifically, we tackle two variations of the clock latency minimization problem: (1) To minimize the maximum clock latency; and (2) to minimize the sum of clock latencies. We propose mixed integer programming (MILP)-based approaches for these problems. Experimental results using benchmark circuits show that the proposed approaches can reduce the maximum clock latency, and the sum of clock latencies over the conventional design, without degrading the operating clock period.

The remainder of the paper proceeds as follows: Section 2 covers the preliminaries. Section 3 describe our motivation, and formulate our problems. In Section 4, we propose MILPs to tackle the clock latency minimization problems. Experimental results are shown in Section 5. Section 6 concludes the paper, and presents some future works.

2. Preliminaries

2.1 Definitions and Assumptions

- We assume that an input processing algorithm is given in the form of data-flow graph (DFG), in which a vertex represents an operation, and an arc represents a data-dependence.
- Scheduling is a task of assigning operations to time-slots (clock-cycles (CCs)), such that the precedence and resource constraints are met. Let σ_i be a CC, such that the output data of o_i (denoted as a_i) is written in a register at the rising clock-edge of CC$(\sigma_i + 1)$. We call σ_i the schedule of operation o_i. We regard the primary input as the output of an imaginary operation scheduled in CC0.
- Functional unit (FU) binding is a task of assigning operations to FUs, so that two operations with overlapping lifetimes (execution time intervals) are not assigned to the same FU (the lifetime constraint).
- Register binding is a task of assigning data to registers, so that the lifetime constraint is met in a way similar to FU binding. The lifetime of a data is a time-interval during which the data must be stored in a register.
- CSS is a task of assigning real values to registers. Let τ_i be the clock latency of a register to which a_i is assigned. We call τ_i the clock latency of a_i.
- After performing scheduling, FU binding, and register binding, a register-transfer-level datapath is generated for each combination of an operation and its one input data. We call it a *local datapath*, then the overall datapath can be regarded as a set of local datapaths.
- Clock skew schedules are realized by inserting delay elements for individual registers.

2.2 Timing Constraints of Local Datapath

Let T_i be the actual time at which data a_i is written in a register (T_i is referred to as the register write-time of a_i). Then, the following equation holds.

$$T_i = \sigma_i \cdot P + \tau_i, \tag{1}$$

where P is the clock period.

For each combination of operation o_j and an input data a_i, the register write-time of a_j, T_j, must be later than the completion time of executing o_j. Therefore, the following *setup constraint* must be met:

$$T_i + D_j + T_{setup} \leq T_j, \tag{2}$$

where D_j is the longest-path delay time of an FU which executes o_j, and T_{setup} is the setup time of a register.

In general, several data can share a register. If a_i and another data a_k share the same register, and a_k overwrites a_i, T_j must be earlier than the arrival time of the overwriting effect. Therefore, the following *hold constraint* must be met:

$$T_j \leq T_k + d_j - T_{hold}, \tag{3}$$

where d_j is the shortest-path delay time of an FU to which o_j is assigned, and T_{hold} is the hold time of a register.

For each data-pair a_i and a_j that is assigned to the same register, their clock latencies become same.

$$\tau_i = \tau_j \tag{4}$$

The permissible range of the clock latency must be restricted to a certain range as follows.

$$0 \leq \tau_i \leq T_{\max}, \tag{5}$$

where $T_{\max}$ is a user-specified positive constant.

3. Problem Statement

Suppose that we are given (1) a DFG ; (2) a set of FUs with delay information; (3) a set of registers $\mathcal{R}$; (4) an FU binding result such that the lifetime constraint is met; (5) a register binding result such that the lifetime constraint is met; (6) the maximum-permissible clock latency $T_{\max}$, $0 \leq {}^{\forall}\tau_i \leq T_{\max}$; (7) the maximum permissible schedule length $L_{\max}$, $0 \leq {}^{\forall}\sigma_i \leq L_{\max}$; and (8) the clock period P.

Then, the HLS problems to be solved in this paper are described as follows:
Problem 1: Perform scheduling and CSS, such that the maximum clock latency is minimized.
Problem 2: Perform scheduling and CSS, such that the sum of clock latencies is minimized.

4. MILP-Based Approach

To solve our optimization problems, we propose MILP-based approaches. Although MILP is NP-hard, it can be solved by an MILP solver (e.g., [9]).

4.1 Temporal Order Constraint

MILP would be time-consuming for a large circuit. In this section, we introduce an additional design constraint for FU and register, namely *the temporal order constraint*, to deal with a large circuit. The temporal order constraint for an FU is that for an operation-pair o_{i_1} and o_{i_2} assigned to the FU, if $\sigma_{i_1} < \sigma_{i_2}$ holds in the original FU binding result, then $\sigma_{i_1} < \sigma_{i_2}$ must be kept for the FU during scheduling. We represent this temporal order for o_{i_1} and o_{i_2} as $o_{i_1} \rightarrowtail o_{i_2}$. iThe temporal order constraint for a register is defined in a similar way. Intuitively, the temporal order constraint tries to reduce the solution space by removing the degree of freedom in the execution order (the storing order) in an FU (a register), since the possible number of these orders exponentially increase in the worst case.

4.2 Variables Definition

- σ_i: An integer variable for every operation o_i, which represents the schedule of o_i.
- τ_i^A: A real-valued variable for every data a_i, which represents the clock latency of a register to which a_i is assigned.
- τ_j^R: A real-valued variable for every register R$_j$, which represents the clock latency of R$_j$.
- $\tau_{\max}$: A real-valued variable which represents the maximum clock latency.

4.3 MILP Constraints

For each arc (o_{i_1}, o_{i_2}) in DFG, the precedence constraint must be met.

$$\sigma_{i_1} + exec(o_{i_2}) \leq \sigma_{i_2}, \tag{6}$$

where $exec(o_{i_2})$ is the execution CCs of o_{i_2}.

For every pair of operations o_{i_1} and o_{i_2} assigned to the same FU such that the temporal order $o_{i_1} \rightarrowtail o_{i_2}$ holds, σ_{i_2} must be no smaller than σ_{i_1} plus $exec(o_{i_2})$ to meet both the lifetime constraint, and the temporal order constraint for FUs

$$\sigma_{i_1} + exec(o_{i_2}) \leq \sigma_{i_2} \tag{7}$$

The lifetime constraint for registers must be met. Differently from the lifetime length of an operation, the lifetime length of a data can be changed during scheduling. Since the end of the lifetime of a data a is determined by the schedule of an operation o which references a lastly, the operation whose output data overwrites a must be scheduled later than o. For data a_i such that there is a data a_ℓ which overwrites a_i (the temporal order $a_i \rightarrowtail a_\ell$ holds), σ_ℓ must be no smaller than σ_k, where o_k is an operation which references a_i.

$$\sigma_k \leq \sigma_\ell \tag{8}$$

Note that the resource constraints for FUs and registers are always met since we are given FU and register binding solutions.

For every operation o_{i_2} and its input data a_{i_1}, the setup constraint and the hold constraint must be met. Therefore, we have the following constraints.

$$\sigma_{i_1} \cdot P + \tau_{i_1}^A + D_{i_2} + T_{setup} \leq \sigma_{i_2} \cdot P + \tau_{i_2}^A, \tag{9}$$
$$\sigma_{i_2} \cdot P + \tau_{i_2}^A \leq \sigma_{i_3} \cdot P + \tau_{i_3}^A + d_{i_2} - T_{hold}, \tag{10}$$

where a_{i_3} is a data which overwrites a_{i_1}.

For every data a_i, if a_i is assigned to R$_j$, we have the following constraint.

$$\tau_i^A = \tau_j^R \tag{11}$$

The range of every τ_j^R and σ_i must be restricted.

$$0 \leq \quad \tau_j^R \quad \leq T_{\max}, \tag{12}$$
$$0 \leq \quad \sigma_i \quad \leq L_{\max}. \tag{13}$$

4.4 Objectives

For Problem 1, we have to prepare the following constraint:

$$\tau_j^R \leq \tau_{\max}. \tag{14}$$

The objective is to minimize the maximum clock latency:

$$\text{Minimize}: \ \tau_{\max}. \tag{15}$$

For Problem 2, the objective is to minimize the sum of clock latencies (note that (14) is not required in this case):

$$\text{Minimize}: \sum_{R_j \in \mathcal{R}} \tau_j^R. \tag{16}$$

5. Experimental Results

We have implemented our proposed MILP in C, ran on a PC equipped with 2.67 GHz Intel^{R} Core^{TM} i5 processor, and tested it on a set of benchmarks. Each MILP was solved by the commercial ILP solver IBM ILOG CPLEX ver. 11.0.0 [9]. As benchmarks, we used the Jaumann Wave digital Filter (JWF), the fifth-order Elliptic Wave digital Filter (EWF), and the 16-point Fast Fourier Transform (16-FFT). We used the FUs: ALU (addition/subtraction, longest-/shortest-delay times = 6.0/3.0) and MUL (multiplication, longest-/shortest-delay times = 15.0/8.0). These delay times include T_{setup} and T_{hold}.

To obtain FU binding solution, we use a temporal scheduling solution obtained by a heuristic scheduling algorithm, namely *list-scheduling algorithm* [10] under the assumption that the clock period is 8.0. Then, the execution CCs of an addition and a multiplication become 1 and 2, respectively. $L_{\max}$ is obtained by this scheduling solution. An FU binding solution and $|\mathcal{R}|$ is obtained by the left-edge algorithm [10] based on this scheduling solution. A register binding solution and P is obtained by the MILP of CSS-aware register binding for clock period minimization [7]. $T_{\max}$ is set as P.

For comparative experiments, we have tested the following design.

List: Scheduling solution is obtained by list-scheduling, and never changed.

Proposed: Scheduling solution is optimized by our MILP.

Table 1 shows the results for Problems 1 and 2. The column "DFG (#op)" represents the name of DFG with the number of operations, #op. In the column "FUs," the parenthesis (x, y) represents that the resource constraint includes x ALUs and y MULs. The column "L' " represents the schedule length obtained by list-scheduling after reducing the clock period. That is, the execution CCs of an operation is changed (e.g., the execution CCs of MUL is 2 in the initial scheduling with $P = 8.0$, but changed to 3 if $P = 6.75$). The value "L' " means that if we allow to extend the schedule length to "L'," we can obtain zero-skew design. The column "time[s]" represents the consumed CPU time (seconds) for running CPLEX.

For almost all the cases, the proposed approaches were able to reduce the maximum clock latency, and the sum of clock latencies compared to design 'List.' For 16-FFT with (5,2), clock latencies were completely removed. It means that although there is a zero-skew schedule with $L_{\max}$, list-scheduling was not able to find it. Therefore, CSS-aware scheduling is necessary. We can see that there is a trade-off between the maximum schedule length and the amount of clock latencies. The development of simultaneous optimization of them requires further investigation.

6. Conclusion and Future Work

In this paper, we have proposed a scheduling approach to clock latency minimization in high-level synthesis (HLS). We have proposed mixed integer linear programming-based approaches. Experiments confirmed the effectiveness of the proposed approaches. Development of a comprehensive design framework considering HLS, clock skew scheduling, and clock tree synthesis is left as an interesting future work.

References

[1] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 37–42, January 1997.

[2] J.L. Neves and E.G. Friedman, "Optimal clock skew scheduling tolerant to process variations," *Proc. IEEE/ACM Design Automation Conference (DAC)*, pp. 623–628, November 1996.

[3] P. Vuillod, L. Benini, A. Bogliolo, and G. De Micheli, "Clock-skew optimization for peak current reduction," *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 265–270, August 1996.

[4] T. Obata and M. Kaneko, "Control signal skew scheduling for RT level datapath synthesis," *Proc. IEEE Mid-West Symposium on Circuits and Systems (MWSCAS)*, vol. 2, pp. 1087–1090, August 2005.

[5] S.-H. Huang, C.-H. Cheng, Y.-T. Nieh, and W.-C. Yu, "Register binding for clock period minimization," *Proc. IEEE/ACM DAC*, pp. 439–444, July 2006.

[6] M. Ni and S.O. Memik, "Early planning for clock skew scheduling during register binding," *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 429–434, November 2007.

[7] S.-H. Huang and C.-H. Cheng, "Timing driven power gating in high-level synthesis," *Proc. IEEE/ACM ASP-DAC*, pp. 173–178, January 2009.

[8] K. Inoue and M. Kaneko, "Early planning for RT-level delay insertion during clock skew-aware register binding," *Proc. IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 154–159, October 2011.

[9] IBM ILOG CPLEX, http://www.ilog.com/

[10] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, New York: McGraw Hill, 1994.

Table 1. Experimental results

circuit (#op)	FUs[*]	$\lvert\mathcal{R}\rvert$	P	$L_{\max}$	L'	**Problem 1:** Max. clock latency			**Problem 2:** Sum of clock latencies		
						List	Proposed		List	Proposed	
						$\tau_{\max}$	$\tau_{\max}$	time[s]	$\sum \tau_i$	$\sum \tau_i$	time[s]
JWF (22)	(2,1)	7	6.75	11	15	1.5	1.4 (-7%)	0.00	3.8	3.2 (-16%)	0.00
	(2,2)	8	6.75	10	12	1.5	1.4 (-7%)	0.00	5.3	4.0 (-25%)	0.00
	(3,2)	8	6.75	10	12	3.8	1.4 (-63%)	0.00	16.9	3.4 (-80%)	0.00
EWF (42)	(2,3)	13	6.75	19	22	2.3	2.0 (-15%)	0.01	10.5	8.8 (-16%)	0.01
	(3,3)	11	7.00	17	21	2.0	2.0 (-0%)	0.00	13.0	11.0 (-15%)	0.00
	(3,4)	11	7.00	17	20	2.0	2.0 (-0%)	0.01	11.0	8.0 (-27%)	0.01
16-FFT (97)	(5,2)	20	7.20	20	29	2.4	0.0 (-100%)	0.01	24.0	0.0 (-100%)	0.01
	(7,5)	18	7.33	11	15	2.0	1.2 (-40%)	0.00	14.0	4.0 (-71%)	0.01
	(10,5)	18	7.20	11	15	1.8	0.6 (-67%)	0.01	18.6	1.2 (-94%)	0.01

[*](#ALU, #MUL)