Fast algorithms for retiming large digital circuits

by

Naresh Maheshwari

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Major Professor: Sachin Sapatnekar

Iowa State University

Ames, Iowa

1998

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Naresh Maheshwari

has met the dissertation requirements of Iowa State University

---

Committee Member

Committee Member

Committee Member

Committee Member

Major Professor

For the Major Program

For the Graduate College

Dedicated to my Grandparents

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation for my advisor and mentor, Professor Sachin Sapatnekar. He offered me enthusiastic support and guidance, and provided valuable suggestions and encouragement. I also thank Professors L. F. Chao, M. Hassoun, S. C. Kothari, A Tyagi, R. L. Geiger, G. M. Prabhu and D. Fernandez-Baca for their advice and comments.

I would like to acknowledge many useful discussions with Professor B. Vinnkota at University of Minnesota; Professors S. M. Reddy and I. Pomeranz at University of Iowa; Dr. R. Rudell, Dr. N. Shenoy, P. Zepter and J. C. Madre at Synopsys; and Dr. C. Visweswariah, Dr. L. Stok, A. Mets, Anthony Drumm and Brian Wilson at IBM. I also thank Professors Anshul Kumar, M. Balakrishnan and Shashi Kumar of Indian Institute of Technology, Delhi for introducing me to the field of Electronic Design Automation.

Many thanks to fellow graduate students for making my stay pleasant and fruitful: Daksh Lehther, Govindaraju Venkatesh, Sabita Pilli, Jatan C. Shah, Huibo Hou, Yanbin Jiang, Min Zhao, Kishore V. Kasamsetty, Joon-Yun Kim, Jeff Echtenkamp, Prasoonkumar Surti, Lakshmikant Bhupathi, Navin Venkata, Naveen Bohra, Tapas Ray, Madhusudhan R. Midhe, Jian-Feng Shi, Ankur Gupta, Ajay Jani, Sridharan Sucheendran, Ashutosh Joharapurkar, Mugdha Kulkarni, Seshadri Anantharama, Siamak Mortezapou, Sudha Nagavarapu, Raghuram Madabushi, Jayesh Sidhiwala, Sunil Mishra, Vara Varavithya, Manoj Sahu, Saqib Malik, Tara John, Sunita Dash, Sudeer Vemulapalli, Jatin Upadhyay, Chung Yen Chang, Russ Meier, Lavanya Apsani, Seema, Sucheendran Sridharan, Sudnya Shroff, Jiang Hu, Kaushik Gala and Junsoo Lee. Special thanks to Kaushik Gala for proof-reading this thesis, and to Vara Varavithya for helping me with various formalities of thesis submmition.

Thanks to Andrew, Jerry, Idarto, Chimai and other "sysadmin" people for keeping the computers up and running. I am grateful to National Science Foundation, Lucent Technologies and Design Automation Conference for funding parts of my research; IEEE and Iowa State University for providing financial support to attend conferences; and IBM for providing me with the opportunity to work two summers.

I feel indebted to my aunts, uncles and cousins in US for their love and emotional support and for many wonderful vacations; and to my parents for having faith in me.

ABSTRACT

The increasing complexity of VLSI systems and shrinking time to market requirments demand good optimization tools capable of handling large circuits. Retiming is a powerful transformation that preserves functionality, and can be used to optimize sequential circuits for a wide range of objective functions by judiciously relocating the memory elements. Leiserson and Saxe, who introduced the concept, presented algorithms for period optimization (minperiod retiming) and area optimization (minarea retiming). The ASTRA algorithm proposed an alternative view of retiming using the equivalence between retiming and clock skew optimization.

The first part of this thesis defines the relationship between the Leiserson-Saxe and the ASTRA approaches and utilizes it for efficient minarea retiming of large circuits. The new algorithm, Minaret, uses the same linear program formulation as the Leiserson-Saxe approach. The underlying philosophy of the ASTRA approach is incorporated to reduce the number of variables and constraints in this linear program. This allows minarea retiming of circuits with over 56,000 gates in under fifteen minutes.

The movement of flip-flops in control logic changes the state encoding of finite state machines, requiring the preservation of initial (reset) states. In the next part of this work the problem of minimizing the number of flip-flops in control logic subject to a specified clock period and with the guarantee of an equivalent initial state, is formulated as a mixed integer linear program. Bounds on the retiming variables are used to guarantee an equivalent initial state in the retimed circuit. These bounds lead to a simple method for calculating an equivalent initial state for the retimed circuit.

The transparent nature of level sensitive latches enables level-clocked circuits to operate faster and require less area. However, this transparency makes the operation of level-clocked circuits very complex, and optimization of level-clocked circuits is a difficult task. This thesis also presents efficient algorithms for retiming large level-clocked circuits. The relationship between retiming and clock skew optimization for level-clocked circuits is defined and utilized to develop efficient retiming algorithms for period and area optimization. Using these algorithms a circuit with 56,000 gates could be retimed for minimum period in under twenty seconds and for minimum area in under 1.5 hours.

1 INTRODUCTION

With the advances in integrated circuit (IC) technology, more than 10 million devices can be manufactured on a single chip today. Because of this increase in the complexity, Very Large Scale Integration (VLSI) circuit designs require sophisticated Electronic Design Automation (EDA) tools capable of handling large circuits. Due to the increase in complexity and reduced time to market, designers cannot rely on their intuition to design fast, low power sequential circuits with minimum area. Thus circuit optimization tools are indispensable for designers, and much work needs to be done to develop good computer-aided design (CAD) tools. Most of the traditional circuit optimization techniques operate on combinational subcircuits extracted from sequential designs. Thus they have limited capabilities for optimization and true sequential optimization techniques are needed. This work develops CAD tools for optimizing large sequential circuits.

Retiming is a powerful transformation that has great potential for sequential circuit optimization. It is the concept of moving storage devices across computation nodes to improve performance without changing the input-output behavior, and can operate at gate level netlists or higher abstractions (e.g. data flow graphs, communication graphs, processor schedules).

At the circuit level these storage devices are called registers which can be either edge-triggered flip-flops (or FF's) or level sensitive latches (or latches), and the computation nodes are combinational gates. Retiming moves registers across gates without changing the number of registers in any cycle or on any path from the primary inputs to the primary outputs. This preserves the input-output latency of the circuit. Since retiming does not directly affect the combinational part of the circuit the circuit behavior remains unchanged. However since retiming can change the boundaries of combinational logic, it has the potential to affect the results of combinational synthesis as well.

1.1 Types of Retiming

Retiming can be performed to improve the circuit behavior with respect to different objective functions. Some of these objective functions are discussed below.

Clock Period The simplest objective function used in retiming is minimization of the clock period. Since the clock period in an edge-triggered circuit is given by the maximum combinational delay, registers can be relocated to reduce the clock period. For the

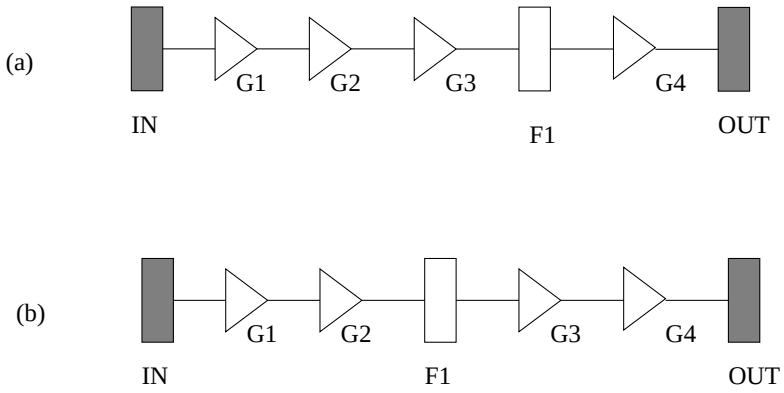

Figure 1.1 Effect of retiming on clock period

circuit shown in Figure 1.1 (a), with unit delay gates, the clock period is 3.0 time units. If we relocate register L1 from the output of gate G3 to its input, we get the circuit in Figure 1.1 (b), with a clock period of 2.0 units. Thus relocating registers can reduce the clock period of a circuit, and retiming can be used to minimize the clock period. Retiming to minimize the clock period is termed **minperiod** retiming. Notice that the input-output behavior is not changed by retiming since in both cases the output is produced after 2 clock cycles. Retiming a circuit to achieve a given target clock period is a special case of this problem.

Area Since retiming does not affect the combinational part of the circuit, the area overhead of the combinational part remains constant. Retiming can, however, affect the overall area of the circuit since it can alter the number of registers in the circuit.

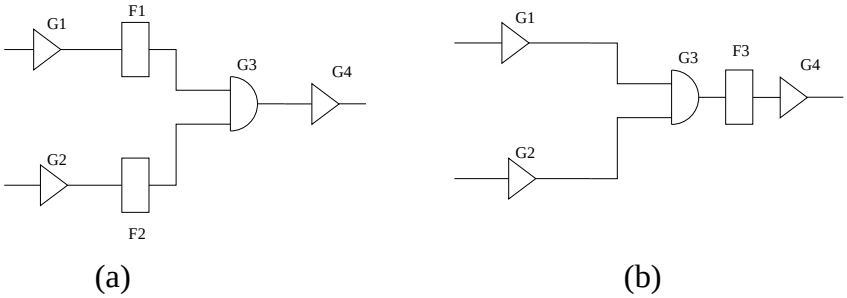

Figure 1.2 Effect of retiming on number of registers

Two circuits can have the same input-output behavior and clock period, but require different number of registers. To illustrate this consider the circuits in Figure 1.2 which are equivalent under the retiming transformation. The circuit in Figure 1.2 (a) requires two registers while that in Figure 1.2 (b) requires only one register.

Retiming can therefore be used to minimize the number of registers in the circuit. This can be done without any constraint on the clock period of the resulting circuit, or subject

to a target clock period. The former is called unconstrained minarea retiming while the latter is called constrained minarea retiming or simply **minarea** retiming.

Power The power dissipated in a circuit depends on the product of switching activity and the load capacitance at the output of a gate, summed over all gates. Since registers can filter out glitches, relocation of registers will affect the switching activity at gate outputs. In addition relocating registers also changes the load capacitance seen by gates. Thus retiming can change the power requirements of a circuit, and can be used for reducing the power dissipation in sequential circuits by placing registers on interconnections with high switching activity and high capacitive loads.

Testability Since retiming relocates registers, it changes the state encoding in sequential circuits, thus affecting the test generation time, and the number of redundant faults. The repositioning of registers also affects the length of scan chains, required for partial or full scan designs. Retiming can, therefore, be used to improve the testability of sequential circuits.

Quality of Logic Optimization Most logic optimization techniques operate on combinational logic within register boundaries. Hence changing these register boundaries (by retiming the registers) affects the quality of results obtained by logic optimization.

Most of these objective functions have been used for retiming different kinds of circuits. A brief survey of publications describing these research activities is presented in Section 1.2.

Algorithms for retiming a circuit must address the specific requirements of a circuit, and the clocking discipline used. Four major classes of circuits are described below.

Edge-triggered Circuits containing only edge-triggered FF's are called edge-triggered circuits. In an edge-triggered circuit the clock period is given simply by the largest combinational delay. The first publications on retiming addressed this class of circuits.

Level-clocked Circuits using level-sensitive latches are called level-clocked circuits. Latches are transparent during the period when the clock waveform is active. This transparent nature of latches give level-clocked circuits the potential to operate at a faster clock period, and require less area than the corresponding edge-triggered circuits. Unfortunately this also complicates the analysis of level-clocked circuits and hence finding an optimal retiming can be computationaly expensive.

Control Logic Since control logic consists of Finite State Machines (FSM's), the registers in the circuit are associated with the FSM states. Retiming changes the locations of these registers and hence the state encoding of the FSM. Thus issues regarding safe replaceability become important. In circuits that have a meaningful initial state, it is important to find a retimed circuit with an equivalent initial state. Not all otherwise

valid retimings of a circuit will have equivalent initial states. Consider the circuit in Figure 1.3 (a). If we wish to move FF A and B across gate G1 (to FF C in Figure 1.3 (b)), we need to find a initial value of FF C which is equivalent to the initial values of FF A and B. If FF A and B have conflicting values, no such equivalent initial value at FF C exists. Thus additional constraints have to be imposed to ensure the presence of an equivalent initial state when retiming control logic.

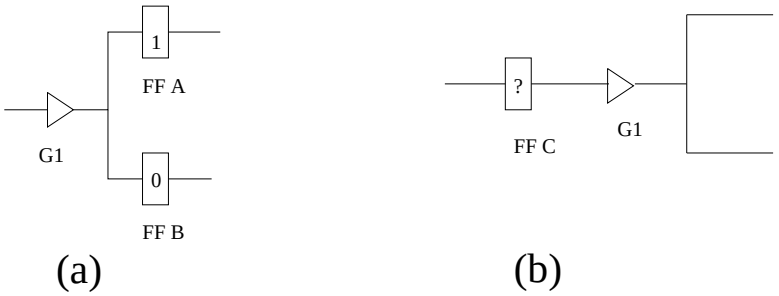

Figure 1.3 Equivalent initial states in reverse retiming.

FPGA's Field Programmable Gate Arrays (FPGA's) present some different requirements. In LUT-based FPGA's the amount of logic is dependent on the number of inputs and not on the complexity of the logic. Further since FPGA's have limited resources with memory elements at fixed locations extra constraints are placed on the movement of memory elements during retiming.

1.2 Research on Retiming

Since retiming was introduced by Leiserson and Saxe [58, 59] there has been significant amount of research has been done on retiming both in the academia [10, 21, 28, 53, 92, 93, 103, 104, 111, 112, 121, 40] and in the industry (e.g, IBM, Synopsys and Philips). In this section we present a brief literature survey of retiming related research. A good survey of retiming research is also provided in [113]. A good introduction to retiming can also be found in Section 9.3.1 of [86].

Edge-triggered Circuits Leiserson and Saxe introduced algorithms for minperiod and minarea retiming of edge-triggered circuits[58]. The circuit is represented by a graph and polynomial time algorithms are presented without any experimental implementations. The minperiod retiming problem is solved by performing a binary search for the best clock period. The feasibility of a given clock period is checked by a Bellman-Ford like relaxation algorithm. The minarea problem is formulated as a Linear Program (LP). This LP is the dual of a mincost network flow problem, and thus can be solved efficiently. Details of this approach, which we call the "LS approach", are provided in [59] and described briefly in Section 2.1.

Shenoy and Rudell presented an efficient and clever implementation of the LS algorithms in [115]. Their main contributions include reducing the memory requirements from $O(|G|^2)$ to $O(|G|)$, where $|G|$ is the number of gates in the circuit, and the use of back pointers to speed up the feasibility check during the binary search for minimum clock period. A technique for reducing the number of constraints in the minarea LP was presented in [1].

The ASTRA algorithm [21, 22, 109] exploited the retiming-skew equivalence for fast minperiod retiming. ASTRA first finds a minimum period achievable by skew optimization, and then translates these skews into retiming. Circuits with 20,000 gates could be retimed in 2 minutes.

Improved Timing Models Both the LS approach and the ASTRA approach assume the gate delays to be fixed and all FF delays to be equal. Since these are approximations, much effort has been spent in improving the delay models. Delay models that incorporate clock skews, register delays, etc. are presented in [83, 124, 126, 127]. DelaY [54, 56] provides a mixed integer linear program (MILP) formulation for a model that has delays associated with interconnects. Constraints are created from each interconnect to every other interconnect, unlike the traditional LS retiming approach that formulates constraints between gates.

The work in [51] presents retiming under variable delays, while [118] presents retiming under variable topology. The work in [57] presents techniques for handling multi-cycle paths and multiple period clocks for minperiod retiming.

Level-Clocked Circuits A signal that flows through a latch during its transparent phase can initiate the computation of the next combinational stage before the beginning of the next clock cycle; this phenomenon is called *cycle stealing*. Due to cycle stealing, level-sensitive circuits have a potential to operate faster, and require less area. Algorithms to retime single phase level-clocked circuits are presented in [114]. Algorithms based on the LS model for retiming multi-phase level-clocked circuits were presented in [8, 9, 44, 68, 70, 105, 106, 107]. TIM [96] is a comprehensive timing analysis and optimization CAD tool for level-clocked circuits that is available in public domain. TIM was used to empirically compare edge-triggered and level-sensitive circuits in [95]. In [46] a polynomial-time algorithm is presented for pipelining two-phase, level-clocked circuits under a bounded delay model.

Retiming with Equivalent Initial States Traditional retiming algorithms do not pay any regard to initial states or power-on states of circuits and are not very useful for control logic. Control logic usually have s meaningful initial states and any useful retiming must also find a new initial state for the retimed circuit that is equivalent to the initial state of the original circuit.

A method for minperiod retiming with equivalent initial states was presented in [130] and uses only forward retimings. In some cases this approach may require modifications in the circuit before it can be used. An efficient technique for these modifications is presented in [122]. Reversed retiming [30, 128] uses minimum number of reverse (backward) retiming moves and does not require any modifications in the circuit.

Low Power Retiming can alter the amount of switching that takes place in a circuit, and can, therefore, affect the power consumption of a circuit. The change in the fanout capacitance due to the motion of FF's further affects the power consumption. A mechanism for reducing power by retiming [87], places FF's on interconnects with high switching activity. In [55], algorithms to reduce power by retiming only one phase in a two-phase circuit are presented. The advantage of retiming only one phase is that it preserves the testability of the circuit. A similar approach is taken in [120] to reduce power in DSP designs.

Testing Retiming can be used both to improve testability of a circuit, and as an aid to automatic test generation. In the former case the retimed circuit is actually implemented, while in the later case the retimed circuit is used just by the test generator and the original circuit is implemented. Some work has been performed to characterize the effect of FF relocation on the redundancy of faults [18, 23, 135, 136]. In [26], it was shown that retiming preserves testability with respect to a single stuck-at-fault test set by adding a prefix sequence of a pre-determined number of arbitrary input vectors. Retiming may convert sequential redundancies into a combinational redundancies which are easier to identify, thus improving testability. Retiming can also be used for reducing test lengths in scan based designs [41, 42, 49], for improving built-in self test (BIST) [50, 60, 62] and for pseudo-exhaustive testing [61].

Pipelining and Architectural retiming Since retiming preserves the input-output behavior of the circuit, the number of registers on any path from a primary input (PI) to a primary output (PO) does not change during retiming. A path here refers to a signal flow through zero or more registers and not to a purely combinational path. Thus the minimum period possible under retiming is restricted by a *critical cycle or IO-path*. A critical cycle is a maximum average delay cycle, i.e., a cycle for which the total delay divided by the number of registers is maximum. A critical IO-path is similarly a maximum average delay path from any PI to any PO.

Pipelining is a technique that increases the latency, i.e., the number of latches on a PI to PO path, of a circuit in order to reduce the clock period [101]. Since pipelining can change the latency, the minimum clock period achievable by pipelining is restricted only by critical cycles and <u>not</u> by the critical IO-paths. Pipelining can be achieved by adding one or more registers to all PI's (or PO's) and then retiming the circuit. Thus pipelining

has a potential of achieving lower periods than retiming, by changing the latency of the circuit [46, 82, 101, 116, 125].

Architectural retiming [36, 37, 39, 40] modifies the combinational part of a circuit to increase the number of registers on a critical cycle or path without increasing the perceived latency. Thus architectural retiming unlike retiming and pipelining is not limited by a critical cycle in period reduction, however, it changes the circuit structure, and is difficult to automate.

Verification Issues Research has also been performed on validating the replacement of a controller circuit by a retimed version [47, 102, 123]. The work in [123] shows that while an accurate logic simulation may distinguish a retimed circuit from the original circuit, a conservative three-valued simulator cannot do so. Techniques for verification of retimed circuits are presented in [35, 85, 104]. The work in [43] uses an ATPG based approach for verifying retimed circuits.

Other Applications Retiming has been used during the technology mapping step in FPGA synthesis [15, 16, 91, 129, 133], to improve circuit partitioning [66], for scheduling in high level synthesis [12, 132] and in multiprocessor scheduling [11]. Other approaches for retiming for system level throughput optimization include [89, 134]. Retiming has been combined with other logic synthesis techniques in [4, 63, 65, 80, 81, 90, 99]. Retiming has also been used extensively in DSP applications [27, 34, 97, 98, 100]. The work in [84] presents techniques to handle enable registers. Other work on retiming include [13, 19, 20, 25, 29, 32, 33, 48, 64, 69, 88, 94, 117, 119, 137].

1.3 Organization of this Thesis

This thesis focuses on the issue of efficient retiming of large circuits. Efficient retiming algorithms capable of handling large edge-triggered and level-clocked circuits are presented for delay and area optimization. Parts of this research have been published in [72, 73, 74, 75, 76, 77, 78, 79]. The remainder of the thesis is organized as follows

Background In this chapter we briefly describe the minperiod and minarea retiming methods given by Leiserson and Saxe in [59], and the ASTRA approach to minperiod retiming in [109]. The minimum clock periodis obtained by perfoming a binary search on the clock period. The minarea retiming problem is formulated as a Linear Program (LP).

Minarea Retiming Since minperiod retiming may significantly increase the number of FF's in the circuit, minarea retiming is a important problem. However, traditional algorithms have had a high computational expense, which has limited its use. In Chapter 3 we present an efficient algorithm for delay constrained minimum area retiming of large circuits with edge-triggered FF's. This algorithm is called Minaret and it performs minarea

retiming through an amalgamation of the Leiserson-Saxe approach and the ASTRA approach. The minarea retiming problem is formulated as an LP and the ASTRA approach is used to find tight bounds on the retiming variables. These bounds then help us reduce both the number of variables and the number of constraints in the problem without any loss in accuracy. By spending a small amount of additional CPU time on the ASTRA runs, this method leads to significant reductions in the total execution time of the minarea retiming problem. The reduction in the problem size also reduces the memory requirements, thus enabling retiming of large circuits.

Retiming Control Logic In control logic the initial state of a circuit is an integral part of the behavior. Hence any retimed circuit must have an equivalent initial state in order to have the same behavior as the original circuit. In Chapter 4 we present an algorithm for minarea retiming with a guarantee of equivalent initial states, we call this problem the minarea initial state retiming problem.

There are two basic problems in minarea initial state retiming: firstly to ensure equivalent initial state, and secondly to correctly model the conditional sharing of FF's at the outputs of a gate. We guarantee an equivalent initial state by allowing only those backward moves that have an equivalent initial value. This is achieved by enforcing a bound on the retiming variables.

Unfortunately the unconditional sharing model of [59] is not valid for minarea initial state retiming, where FF's have initial states associated with them, and hence these initial states need to be taken into account in modeling FF sharing. We present a new 0/1 MILP formulation for modeling this conditional latch sharing of FF's at the output of a gate.

Retiming Level Clocked Circuits Level-clocked circuits have the potential to operate faster and require less area. However due to the transparent nature of latches the timing analysis and hence optimization of level-clocked circuits is a hard problem. Although polynomial time algorithms for retiming level-clocked circuits are known, they can not handle large circuits. In Chapter 5 we first present the equivalence between retiming and skew optimization for level-clocked circuits, then we utilize this relation for efficient minperiod and minarea retiming of large level-clocked circuits.

We use a two phase solution for the minperiod retiming problem for general multi-phase clock schedules. In Phase A we solve a clock skew optimization problem efficiently, to obtain the relocation needed for each latch, in order to achieve the optimal clock period under a given schedule. In Phase B the latches are relocated across gates to achieve this target clock period. Since latches can absorb some skew (equal to the active period of the clock), we can stop relocating latches as soon as the skew is small enough to be completely absorbed.

In minarea retiming of level-clocked circuits, the number of constraints is very high since we may have constraints through multiple latches. We use the retiming-skew relation to obtain bounds on the retiming variables. These bounds are then used to reduce the number of variables, and the number of constraints. We also use additional pruning techniques to further reduce the number of constraints, and for their efficient generation.

Conclusion In this chapter we conclude this thesis and present a number of open problems which need to be solved before retiming can become widely accepted. We also present some ideas and thoughts on these problems. The problems disscuesed include limtation on design styles, verification issues, combining retiming with logic synthesis and improved delay models for retiming.

2 BACKGROUND

We now briefly describe the LS approach, details of which can be found in [58, 59]. We will then describe relationship between clock skew and retiming, and the ASTRA approach [21, 22, 109].

2.1 The Leiserson-Saxe Approach

2.1.1 Notation

A sequential circuit can be represented by a directed graph $G(V, E, d, w)$, where each vertex v corresponds to a gate, and a directed edge e_{uv} represents a connection from the output of gate u to the input of gate v, through zero or more registers. Each edge has a weight $w(e_{uv})$, which is the number of registers between the output of gate u and the input of gate v. Each vertex has a fixed delay $d(v)$, that does not change during the retiming process. A special vertex, the host vertex, is introduced in the graph, with edges from the host vertex to all primary inputs of the circuit, and edges from all primary outputs to the host vertex.

A retiming is a labeling of the vertices $r : V \rightarrow Z$, where Z is the set of integers. The weight of an edge e_{uv} after retiming, denoted by $w_r(e_{uv})$ is given by

$$w_r(e_{uv}) = r(v) + w(e_{uv}) - r(u) \tag{2.1}$$

The retiming label $r(v)$ for a vertex v represents the number of registers that have been moved from its outputs to its inputs. Retiming can also be viewed as an assignment of a lag $r(v)$ to every vertex v in the circuit. One may define the weight of any path p originating at vertex u and terminating at vertex v (represented as $u \stackrel{p}{\rightarrow} v$), $w(p)$, as the sum of the weights on the edges on p, and its delay $d(p)$ as the sum of the weights of the vertices on p. A path with $w(p) = 0$ corresponds to a purely combinational path with no registers on it; therefore, the clock period can be calculated as

$$c = \max_{\forall\, p | w(p)=0} \{d(p)\} \tag{2.2}$$

Another important concept used in the Leiserson-Saxe approach is that of the W and D

matrices that are defined as follows:

$$W(u,v) = \min_{\forall\, p:u \overset{p}{\rightsquigarrow} v} \{w(p)\} \tag{2.3}$$

$$D(u,v) = \max_{\forall\, p:u \overset{p}{\rightsquigarrow} v \text{ and } w(p)=W(u,v)} \{d(p)\} \tag{2.4}$$

The matrices are defined for all pairs of vertices (u, v) such that there exists a path $p : u \overset{p}{\rightsquigarrow} v$ that does not include the host vertex. $W(u, v)$ denotes the minimum latency, in clock cycles, for the data flowing from u to v and $D(u, v)$ gives the maximum delay from u to v for the minimum latency.

2.1.2 The Minperiod Retiming Algorithm

The minimum period obtainable under retiming is found by performing a binary search over all possible clock periods. At each step in the binary search, an attempt is made to retime the circuit for the current value of the clock period. The smallest period for which retiming succeeds is returned as the best clock period.

The following $O(|V||E|)$-time algorithm is used for obtaining a retiming for a given clock period.

```
Algorithm FEAS
Given a synchronous circuit G = ⟨V, E, d, w⟩, and a desired clock period c, return
a retiming r of G such that the clock period of the retimed circuit Φ(G_r) ≥ c.
{
  1.  For each vertex v ∈ V, set r(v) ← 0.
  2.  Repeat the following |V| − 1 times
      2.1 Compute graph G_r with existing values of r.
      2.2 Run Algorithm CP on the graph G_r to determine Δ(v) for each vertex v ∈ V.
      2.3 For each v such that Δ(v) > c, set r(v) ← r(v) + 1.
  3.  Run Algorithm CP on the circuit G_r.  If Φ(G_r) > c, then no
      feasible retiming exists.  Otherwise, r is the desired retiming.
}

Algorithm CP
This algorithm computes the clock period Φ(G) for a synchronous circuit
G = ⟨V, E, d, w⟩.
{
  1.  Let G_0 be the subgraph of G with contains precisely those edges e
      with register count w(e) = 0.
  2.  Perform a topological sort on G_0, totally ordering its vertices
      so that if there is an edge from vertex u to vertex v in G_0,
```

then u precedes v in the total order.

3. Go through the vertices in the order defined by the topological sort. On visiting each vertex v, compute the quantity $\Delta(v)$ as follows:
   a. If there is no incoming edge to v, set $\Delta(v) \leftarrow d(v)$.
   b. Otherwise, set $\Delta(v) \leftarrow d(v) + max\{\Delta(u) : u \xrightarrow[e]{} v$ and $w(e) = 0$.
4. The clock period $\Phi(G)$ is $\max_{v \in V} \Delta(v)$.

}

2.1.3 The Minarea Retiming Algorithm

The minarea retiming problem for a target period P can be formulated as the following LP:

$$\begin{array}{lcll} & \text{minimize } \sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & & (2.5) \\ \text{subject to} & r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E & \\ & r(u) - r(v) \leq W(u,v) - 1 & \forall D(u,v) > P & \\ & -\infty \leq r(u) \leq \infty & \forall u \in (V \cup M) & \end{array}$$

where $FI(v)$ and $FO(v)$ represent the fanin and fanout sets of the gate v.

The significance of the objective function and the constraints is as follows (the reader is referred to [59] for details).

- The objective function represents the number of registers added to the retimed circuit in relation to the original circuit.
- The first constraint ensures that the weight e_{uv} of each edge (i.e., the number of registers between the output of gate u and the input of gate v) after retiming is nonnegative. We will refer to these constraints as *circuit constraints*.
- The second constraint ensures that after retiming, each path whose delay is larger than the clock period has at least one register on it. These constraints, being dependent on the clock period, are often referred to as *period constraints*.

It is pointed out in [59] that the dual of this problem is an instance of a minimum cost network flow problem. Hence the LP can be solved efficiently by solving this dual.

2.1.4 A More Accurate Area Model

The cost function in the LP's of Equation (2.5) assumes that each FF has exactly one fanout. However, in practice a FF can have multiple fanouts, allowing the FF's on different fanout edges of a gate to be shared. This sharing must be taken into account for an accurate area model. For example, consider gate A in Figure 2.1 with three fanouts B, C, and D having

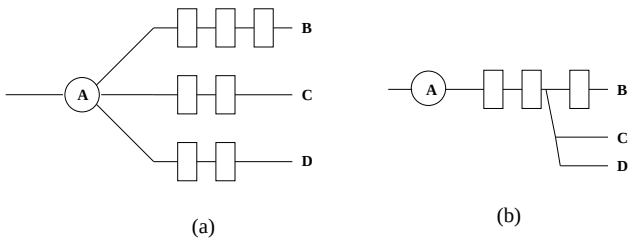

Figure 2.1 Unconditional register sharing at multiple fanouts.

three, two and two FF's respectively. The LP in Equation (2.5) will model the total number of FF's as seven as shown in Figure 2.1(a). However the FF's can be merged or shared as shown in Figure 2.1(b) resulting in a total cost of only three FF's.

To model the maximal FF sharing[1] the work in [59] introduces a mirror vertex m_i for each gate i that has more than one fanout, as shown in Figure 2.2. Further details of this maximal latch sharing model can be found in [111]. Every edge e_{ij}, in addition to having a weight $w(e_{ij})$, now also has a width $\beta(e_{ij})$. In Figure 2.2, the edge weights are shown above the edges while the edge widths are shown below the edges. Consider a gate u with k fanouts to gates $v_j,\ j = 1 \cdots k$. To model the maximum sharing of FF's, an extra edge is added from each fanout gate v_j to the mirror vertex, m_u, with weight $w(e_{v_j m_u}) = w(max_u) - w(e_{uv_j})$, where $w(max_u) = \max_{\forall i \in FO(u)}(w(e_{ui}))$ is the maximum weight on any fanout edge of gate u. Each of the edges from the gate i to its fanouts j, and from the fanouts to the mirror vertex has a width of $1/k$, i.e.,

$$\beta(e_{uv_j}) = 1/k \text{ and } \beta(e_{v_j m_u}) = 1/k \quad \text{for } j = 1 \cdots k.$$

The original LP in Equation (2.5) is modified to include the effect of register sharing as follows:

$$\begin{aligned} \min \quad & \sum_{v \in (V \cup M)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] \\ \text{subject to} \quad & r(u) - r(v) \leq w(e_{uv}) && \forall e_{uv} \in E \\ & r(u) - r(v) \leq W(u,v) - 1 && \forall D(u,v) > P \\ & r(j) - r(m_i) \leq w(max_i) - w(e_{jm_i}) && \forall (m_i) \in M \text{ and } \forall j \in FO(i) \\ & -\infty \leq r(u) \leq \infty && \forall u \in (V \cup M) \end{aligned} \tag{2.6}$$

where $M = \{m_v | v \in V \text{ and } |FO(v)| > 1\}$ is the set of all the mirror vertices, and additional constraints due to the mirror vertices are called the mirror constraints. For simplicity we can rewrite the above LP as follows

$$\min \quad \sum_{v \in (V \cup M)} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right]$$

[1] This model is valid only for unconditional sharing of FF's. In Section 4.3 we will present a model for conditional sharing of FF's.

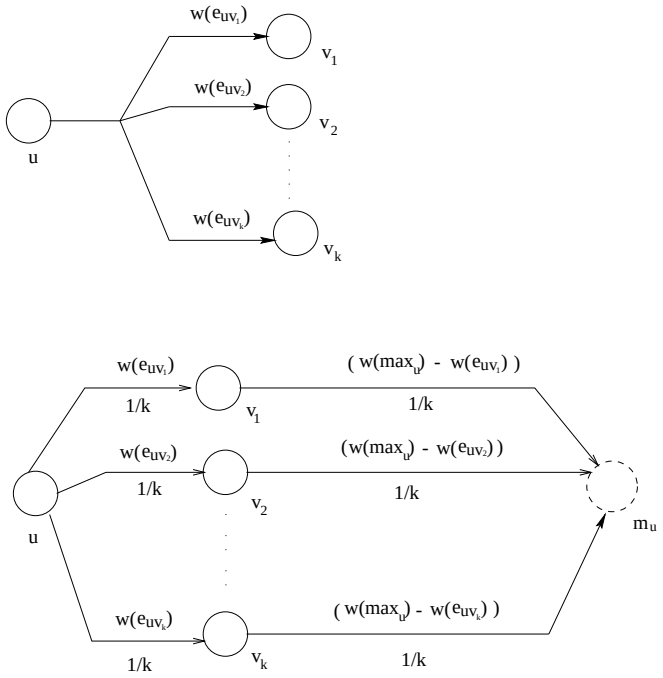

Figure 2.2 Model for maximum register sharing at multiple fanouts.

$$\begin{aligned} \text{subject to} \quad & r(u) - r(v) \leq c_{uv} \quad \forall (u,v) \in C \\ & -\infty \leq r(u) \leq \infty \quad \forall u \ \in (V \cup M) \end{aligned} \tag{2.7}$$

where $C = C_p \cup C_c \cup C_m$ is the constraint set of the LP in Equation (2.6), and includes the period constraint set (C_p), the circuit constraint set (C_c) and the mirror constraint set (C_m). A constraint (i,j) in the constraint set C is of the form

$$\begin{array}{rrcll} & r(i) - r(j) & \leq & c_{ij} & \forall (i,j) \in C \\ \text{where} & c_{ij} & = & w(e_{ij}) & \forall (i,j) \in C_c, \text{ i.e., } e_{ij} \in E \\ & c_{ij} & = & W(i,j) - 1 & \forall (i,j) \in C_p, \text{ i.e., } D(i,j) > P \\ & c_{ij} & = & w(max_i) - w(e_{jm_i}) & \forall (i,j) \in C_m, \text{ i.e., } m_i \in M \text{ and } \forall j \in FO(i) \end{array} \tag{2.8}$$

The objective function of the LP in Equation (2.7) now denotes the increase in the number of FF's assuming maximal sharing of FF's at the output of all gates. The weights on all paths from gate u to its mirror vertex m_u are the same before retiming, i.e., $w(e_{uv_i}) + w(e_{v_i m_u}) = w(max_i)$ $1 \leq i \leq k$, and therefore the weights on all paths from gate u to its mirror vertex m_u must be equal after retiming. Since the mirror vertex m_u is a sink in the graph, the register count on one of the edge from the fanout nodes to m_u will be zero, i.e., $\exists i \ | w(e_{v_i,m_u}) = 0$. Thus the weight on all paths from gate u to mirror vertex m_u after retiming will be $w_r(max_u) = \max_{\forall j \in FO(u)}(w_r(e_{uj}))$. Since there are k paths, each with width $1/k$, the total cost of all paths will be $w_r(max_u)$ as desired. Like the LP in Equation (2.5) the LP in Equation (2.7) is also the dual of a minimum cost network flow problem.

We now present an alternate view of this model. The change in cost function due to adding or removing FF's from the fanout junction of gate u is modeled by two retiming variables: one for the gate, $r(u)$ and other for the mirror vertex, $r(m_u)$. Any change in the cost function due to FF's moving across the multi-fanout gate itself are modeled by $r(u)$, while any change due to FF motion across its fanout gates v_i, $1 \leq i \leq k$ is modeled by the mirror variable $r(m_u)$.

The change in the number of FF's in the circuit, under maximal sharing obtained by retiming a gate u by one unit can be calculated as follows. The decrease in the cost function obtained by removing a FF from each of the fanouts of a gate is one unit, even for multiple fanout gates since the FF's on all the fanouts were shared. The increase in the cost function from adding a FF to all the inputs of a gate u is equal to the number of fanins of u that have only one fanout, since any FF added to a fanin j of gate u that has more than one fanout ($|FO(j)| > 1$) is already modeled by the mirror variable of that fanin gate m_j. Thus the cost contribution of any single fanout gate u is given by $(|FI'(u)| - 1) \cdot r(u)$, while that of a multi-fanout gate is given by $(|FI'(u)| - 1) \cdot r(u) + r(m_u)$, where $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u)\ AND\ |FO(v)| = 1\}$.

2.2 The ASTRA Approach

2.2.1 The Relationship Between Clock Skew and Retiming

In a sequential VLSI circuit, due to differences in interconnect delays on the clock distribution network, clock signals do not arrive at all of the FF's at the same time. Thus, there is *skew* between the clock arrival times at different FF's. In a single-phase clocked circuit, in the case where there is no clock skew, the designer must ensure that each input-output path of a combinational circuit block has a delay that is less than the clock period. In the presence of skew, however, the relation grows more complex, as one must compensate for this effect in ensuring that the combinational blocks meet the timing requirements.

The basis of the ASTRA approach is the equivalence between clock skew and retiming, as illustrated by the following example. Let us first consider the use of intentional clock skews for improving the circuit performance. In Figure 2.3, assume the delays of the inverters to be 1.0 unit each. The delays of the first and second combinational blocks are 3.0 and 1.0 units, respectively, and therefore, the fastest allowable clock has a period of 3.0 units. However, if a skew of +1.0 unit is applied to the clock line to FF L1, as shown in Figure 2.4, the circuit can run with a clock period of 2.0 units. This approach was formalized in the work by Fishburn [31], where the clock skew optimization problem was formulated as a linear program (LP) that may be solved to find the optimal clock period.

However, it is easy to see that for the given circuit, the period can also be minimized to 2.0 units by *retiming*, i.e., by relocating the FF L1 to the left across the inverter G3. This results in both the combinational blocks having delays of 2.0 units each as seen in Figure 2.5.

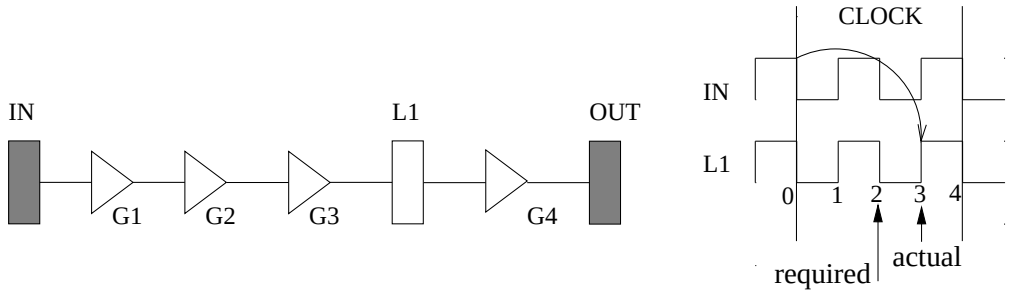

Figure 2.3 An example circuit.

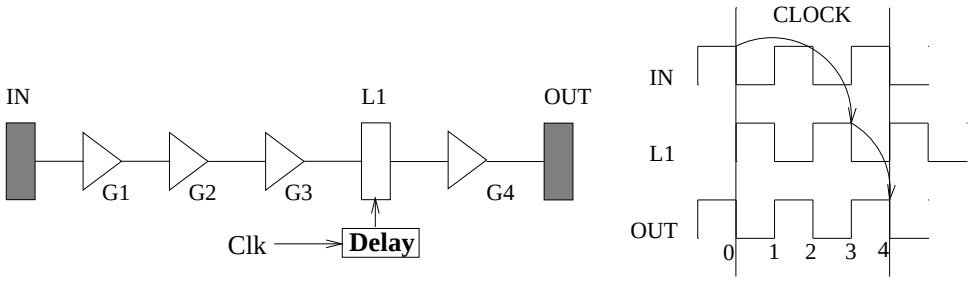

Figure 2.4 Using clock skew to reduce clock period.

This leads us to conclude that in each case, one unit of time is borrowed by the first combinational block from the second; the manner in which cycle-borrowing occurs may either be by the vehicle of clock skew or via retiming.

2.2.2 Minperiod Retiming Algorithm

The details of the ASTRA algorithm for minperiod retiming are provided in [21, 109]; a brief description is presented here for completeness. The relationship between skew and retiming motivates the following two-phase solution to the retiming problem:

Phase A: The clock skew optimization problem is solved to find the optimal value of the skew at each FF, with the objective of minimizing the clock period, or to satisfy a given

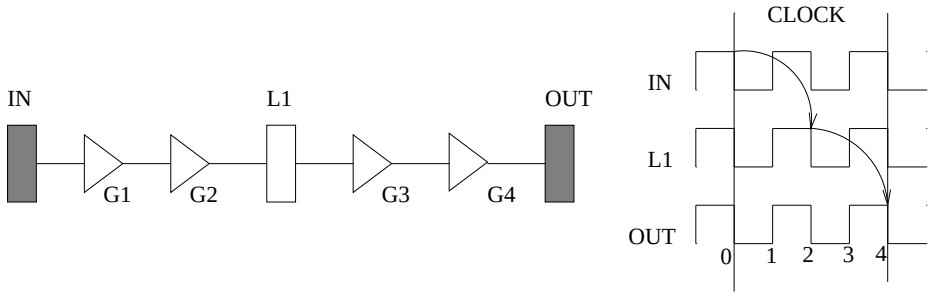

Figure 2.5 Retiming for clock period optimization.

(feasible) clock period. This involves the (possibly repeated) application of the Bellman-Ford algorithm [17] on a constraint graph [109].

Phase B: The skew solution is translated to retiming and some FF's are relocated across gates in an attempt to set the values of all skews to be as close to zero as possible. We attempt to move each positive skew FF opposite to the direction of signal propagation, and each negative skew FF in the direction of signal propagation to reduce the magnitude of its skew. A formal rationalization is provided in [109], but the example in Figure 2.5 should suffice to explain the intuition.

After Phase B, any skews that could not be set exactly to zero are forced to zero. This could cause the clock period to increase from Phase A; however, it is shown that this increase will be no greater than the maximum gate delay. Note, however, that this is not necessarily suboptimal since the minimum clock period using skews may not be achievable using retiming, since retiming allows cycle-borrowing only in discrete amounts (corresponding to gate delays), while skew is a continuous cycle-borrowing optimization [31].

3 RETIMING FOR MINIMUM AREA

3.1 Introduction

For digital design the interesting problem is of delay constrained area optimization, and constrained minimum area retiming is one way to solve this problem. However, the high computational expense of this optimization has limited its use. In this chapter, we approach the problem of constrained minarea retiming for circuits with edge-triggered FF's through an amalgamation of the Leiserson-Saxe approach and the ASTRA approach. By utilizing the merits of both approaches we develop an efficient algorithm for constrained minarea retiming which is also capable of handling very large circuits. The basic idea of the approach is to use the ASTRA approach to find tight bounds on the retiming variables. These bounds help us reduce both the number of variables and the number of constraints in the problem without any loss in accuracy. By spending a small amount of additional CPU time on the ASTRA runs, this method leads to significant reductions in the total execution time of the minarea retiming problem. The reduction in the problem size also reduces the memory requirements, thus enabling retiming of large circuits.

The chapter is organized as follows. In Section 3.2, we show the relationship between these two, and utilize it to efficiently solve the minarea retiming problem. Section 3.3 describes our minarea retiming algorithm. Experimental results are presented in Section 3.4 followed by concluding remarks in Section 3.5.

3.2 Reducing the Problem Size

In practical circuits, it is found that the number of period constraints is phenomenally large. For a circuit with n gates the number of period constraints is $O(n^2)$. However, it is also true that a large fraction of these constraints are redundant as they are implied by some of the other constraints. Any algorithm with pretensions to practicality must use techniques for pruning these redundant constraints. Note that the exactness of the solution is not sacrificed in doing so, since none of the essential constraints are removed. Our approach is to find tight bounds on the variable values, and to use these bounds to avoid generating the redundant constraints. By appropriate application of these bounds, we expect not only to prune the constraint set but also to reduce the number of variables. In this way, we simplify the problem and enable the

LP to be solved more efficiently. We are also able to generate this set of reduced constraints efficiently.

3.2.1 The Concept of Restricted Mobility

A modification of the procedure used in ASTRA can be used to identify how far FF's may possibly be moved. For the circuit in Figure 3.1, to achieve the minimum clock period of 4.0 units, one must move one copy of FF B to the output of gate G4. The possible locations for FF's along the other path to FF C are at the input to gate G8, or at the output of gate G8, or the inputs of gates (G9,G10) or the outputs of gates (G9,G10); no other locations are permissible

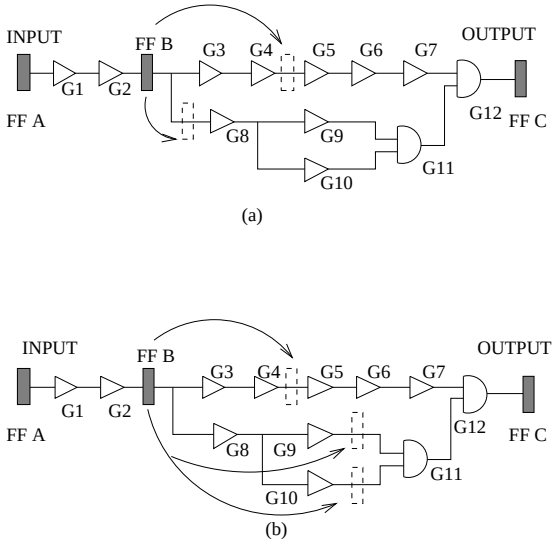

Figure 3.1 Possible FF locations after retiming.

Therefore, it can be seen that the FF's cannot be sent to *just any* location in the circuit; rather, there is a restricted range of locations into which each FF may be moved, and the mobility of each FF is restricted. This restricted mobility may be used to reduce the search space, and hence the number of constraints.

This range of motion of FF's can be derived from the skews calculated by the Bellman-Ford procedure (which calculates the minimum allowable skew value at each FF) [109], and the corresponding slacks in the constraint graph. The idea in this chapter is that the skew values can be used to reduce the search space for the minarea retiming algorithm using restricted mobility. This is seen to translate to a smaller LP.

We will now show the relation between the Leiserson-Saxe approach and the ASTRA approach, and how a modified version of ASTRA can be used to derive bounds on the r variables in the Leiserson-Saxe method. Next, we show how these bounds can be used to prune the number of constraints in minarea Leiserson-Saxe retiming. Finally, we present an example to

illustrate the method.

3.2.2 Deriving Bounds for the r Variables

The concept of restricted mobility is related to the "nearest" and "farthest" location that any FF can occupy under the target clock period. This is relatively easy to map on to the clock skew optimization problem. To understand this, we provide a brief review of the clock skew optimization problem. Given a pair of FF's, i and j, if the maximum delay of any purely combinational path connecting them is D_{ij}, then the following long-path constraint must hold:

$$x_i + D_{ij} \leq x_j + P \tag{3.1}$$

where x_i and x_j are the clock skews at FF's i and j, respectively, and P is the target clock period. For a specified clock period, this may be written as a difference constraint [17] as follows:

$$x_j - x_i \geq P - D_{ij} \tag{3.2}$$

Note that the right hand side of the above equation is a constant, since the clock period is a specified value. For a given circuit, one may build a set of difference constraints with one such constraint for every pair of FF's that have a purely combinational path connecting them, and these difference constraints may be represented by a constraint graph. The Bellman-Ford algorithm may be applied to this graph to find the longest path in the graph. The final value associated with each vertex provides the required skew at that vertex and gives one possible set of skews that can achieve the clock period P. Note that this is not the only allowable set of skews, since slacks [109] in the arcs of the constraint graph can lead to other allowable solutions. Therefore, the first order of business is to determine bounds on the allowable skews at each FF.

ASTRA initializes all skews to 0 to achieve the minimum range of skews. To obtain the bounding skews we need to initialize all skews to $-\infty$. Now when the the Bellman-Ford algorithm [17] is applied to the constraint graph for a specified clock period, the as-late-as-possible[1] (ALAP) skews are calculated for the network. The as-soon-as-possible (ASAP) skews can be obtained by running the Bellman-Ford algorithm on the transpose of this constraint graph [17] (i.e., a graph with the same vertex set as the original graph, but with the edge directions reversed).

These ASAP/ALAP skews can be translated to ASAP/ALAP locations for FF's. These locations can be used to obtain bounds on the retiming variables of the Leiserson-Saxe approach, r, associated with the gates in the circuit as illustrated by the following example. Here we use the terms "ASAP locations" to refer to the case when all FF's are as close to the primary

[1] The calculation of ASAP and ALAP times is a technique that is routinely used in scheduling in high-level synthesis; see, for example, [86] .

input as possible. Similarly the set of ALAP locations has all FF's as close to the primary output as possible. For ASAP locations any available slacks are used to avoid moving a FF in the direction of signal flow, while for ALAP locations they are used to avoid FF motion against the direction of signal flow.

Example: For the circuit in Figure 3.1, the locations for the FF's in the retimed circuit corresponding to the ASAP and ALAP skew solutions are shown in Figure 3.1 (a) and (b), respectively. This implies that during retiming, no FF will move across gates G1, G2, G5, G6, G7, G11 and G12; one FF each will move from the input to the output of gates G3 and G4, and either 0 or 1 FF will move from the input to the output of gates G8, G9 and G10. Referring to Section 2.1 for the definition of the r variables, this implies that one may set the following bounds on the r variables.
(1) $r(u) = 0$ for $u \in \{G1, G2, G5, G6, G7, G11, G12\}$
(2) $r(u) = -1$ for $u \in \{G3, G4\}$, and
(3) $-1 \leq r(u) \leq 0$ for $u \in \{G8, G9, G10\}$. Λ

As explained in [109], FF's that have positive skews are moved in the direction opposite to the signal flow direction, and FF's with negative skews are relocated in the direction of signal flow (see Section 2.2 for a brief explanation). The procedure for finding the ASAP and ALAP locations proceeds along the same lines as in [109], with a few variations described below. During this procedure, we also generate the bounds on the r variables.

When we consider the ASAP locations for the retimed FF's, the aim is to push the FF's as far as possible in a direction opposite to the direction of signal propagation. Therefore, each positive skew FF is moved as far as possible in the direction opposite to the signal flow, and each negative skew FF is moved as little as possible in the direction of signal flow. Therefore,

(1) for a FF with positive skew s that is being moved across a single-fanout gate p against the direction of signal propagation, the skew value after the relocation at input i of p is set to $s - \text{delay}(p)$. If this value is non-positive, then the ASAP location has been found. For gates with multiple fanouts, $s = \min_{\text{all outputs}}(s_i)$, where s_i is the skew of the FF at the i^{th} output, as shown in Figure 3.2(a).

(2) for a FF with negative skew s that is being moved across a single-fanin gate p in the direction of signal propagation, the skew value after the relocation at output i of p is set to $s + \text{delay}(p) + \text{slack(i)}$, where slack(i) is the slack associated with the output i. This slack is defined as the amount by which the delay at output i may be increased before it becomes the critical output of p; by definition, the critical output has a slack of 0. If the new skew is nonnegative, then the ASAP location has been found. For gates with multiple fanins, $s = \max_{\text{all inputs}}(s_i)$, where s_i is the effective skew of the FF at the i^{th} output, as shown in Figure 3.2(b).

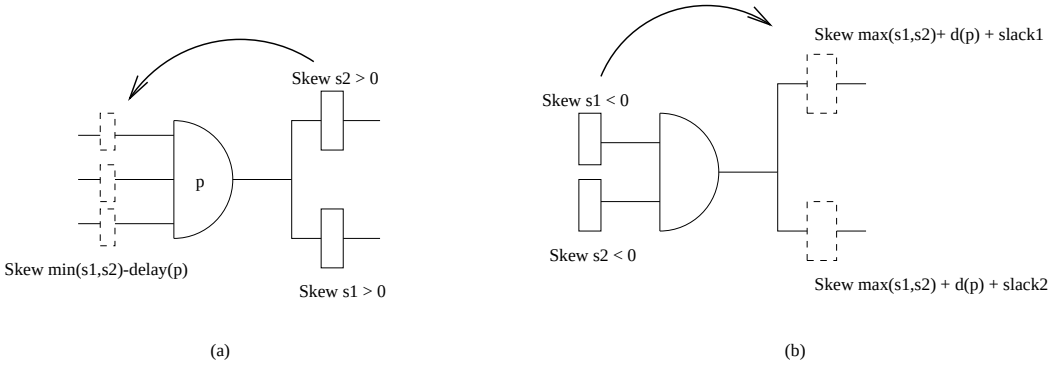

Figure 3.2 Effective skews at FF's after ASAP retiming across a gate.

The ALAP locations can be found similarly with positive skew FF's being moved as little as possible in the direction opposite to the signal flow direction, and negative skew FF's being moved as much as possible in the signal flow direction.

While moving the FF's to ASAP and ALAP locations, subject to the specified clock period P, we count the number of FF's that traverse each gate; these lead us to upper and lower bounds, respectively, on the r variables for each gate. A FF moving from the inputs to the output of a gate decrements the count by one, while one moving from the output to the inputs increments it by one.

For the ASAP case, we move FF's as far as possible against the direction of signal propagation. In other words, we relocate the largest number of FF's possible from the output to the inputs of a gate. By the definition of the r variables, this gives us an upper bound on r for the gates.

Similarly, the ALAP times are used to relocate the largest number of FF's that can move from the inputs of a gate towards its output, and this gives us a lower bound on the r values for the gates in the circuit. Therefore, this procedure provides upper and lower bounds on the r variable corresponding to each gate y of the form.

$$L_y \leq r(y) \leq U_y \tag{3.3}$$

We will refer to L_y as the lower bound for gate y and to U_y as the upper bound of gate y. Like the ASAP and ALAP retimings, these bounds are with reference to a fixed host vertex, i.e., $L_H = U_H = 0$. If $U_u = L_u = k_u$ we say that gate u is If $U_y = L_y = k_y$ we say that gate y is *fixed* or immobile since $r(y) = k_y$ is not really a variable any more. On the other hand if $U_y \neq L_y$ we say that gate y is *flexible* or mobile. Thus we can reduce the variable set V of the Leiserson-Saxe model to $V' \subseteq V$, the variable set of Minaret where

$$V' = \{v \in V | U_v \neq L_v\} \tag{3.4}$$

Bounds on the mirror vertices, introduced to model the maximal latch sharing can be obtained directly from the bounds on fanout gates (as given by Theorem 1). The mirror

variable set M is also reduced to $M' \subseteq M$ the mirror variable set of Minaret where

$$M' = \{m \in M | U_m \neq L_m\} \tag{3.5}$$

Theorem 1 *The bounds on the r value of a mirror vertex m_i of gate i in Figure 2.2 can easily be derived from the bounds on the fanout gates and are given by*

$$\begin{aligned} U_{m_i} &= \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i) \\ L_{m_i} &= \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i) \end{aligned} \tag{3.6}$$

Proof: After optimal retiming the weight on at least one of the edges to the mirror vertex (see Figure 2.1) will be zero [59] hence

$$\begin{aligned} \min_{\forall j \in FO(i)} (w_r(e_{jm_i})) &= 0 \\ \min_{\forall j \in FO(i)} (w(e_{jm_i}) + r(m_i) - r(j)) &= 0 \\ \text{i.e., } \min_{\forall j \in FO(i)} (w(e_{jm_i}) - r(j)) &= -r(m_i) \\ \text{i.e., } r(m_i) &= \max_{\forall j \in FO(i)} (r(j) - w(e_{jm_i})) \\ \text{Since } r(j) &\leq U_j \ \forall j \in FO(i) \\ \text{we have } \max_{\forall j \in FO(i)} (r(j) - w(e_{jm_i})) &\leq \max_{\forall j \in FO(i)} (U_j - w(e_{jm_i})) \\ \text{Therefore } r(m_i) &\leq \max_{\forall j \in FO(i)} (U_j - w(e_{jm_i})) \end{aligned}$$

Thus the upper bound is

$$U_{m_i} = \max_{\forall j \in FO(i)} (U_j + w(e_{ij})) - w(max_i)$$

After retiming all edge weights including edges to mirror vertices must be nonnegative, that is

$$\begin{aligned} w_r(e_{jm_i}) &\geq 0 \quad \forall j \in FO(i) \\ \text{or } w(e_{jm_i}) + r(m_i) - r(j) &\geq 0 \quad \forall j \in FO(i) \\ \text{i.e., } r(m_i) &\geq r(j) - w(e_{jm_i}) \quad \forall j \in FO(i) \\ \text{i.e., } r(m_i) &\geq r(j) + w(e_{ij}) - w(max_i) \quad \forall j \in FO(i) \\ \text{or } r(m_i) &\geq \max_{\forall j \in FO(i)} (r(j) + w(e_{ij})) - w(max_i) \end{aligned}$$

Therefore the lower bound is

$$L_{m_i} = \max_{\forall j \in FO(i)} (L_j + w(e_{ij})) - w(max_i)$$

3.2.3 Eliminating Unnecessary Constraints

In this section, we illustrate how the addition of bounds (derived previously) to the LP of Equation (2.7) in Section 2.1.4 may be used to reduce the constraint set by dropping redundant constraints. It can be seen from the bounds on $r(i)$ and $r(j)$ in Equation (3.3) that $r(i)-r(j) \leq U_i - L_j$. Therefore, if $U_i - L_j \leq c_{ij}$ then $r(i) - r(j) \leq c_{ij}$ is also true, and the constraint (i, j) can be dropped. Thus the Leiserson-Saxe constraint set C can be reduced to the Minaret constraint set $C' \subseteq C$ where

$$C' = \{(i,j) \in C |\ U_i - L_j > c_{ij}\} \tag{3.7}$$

Notice that constraints associated with fixed or immobile gates can be treated as bounds and need not be included in C'. Like the Leiserson-Saxe constraints, the Minaret constraints also consists of circuit, period and mirror constraints, i.e., $C' = C'_c \cup C'_p \cup C'_m$, where C'_c is the reduced circuit constraint set, C'_p is the reduced period constraint set, and C'_m is the reduced mirror constraint set.

3.2.4 Reduced Linear Program

We use the Equations (3.4), (3.5) and (3.7) to reduce the LP in Equation (2.7) to the following LP in Minaret

$$\min \sum_{v \in \{V' \cup M'\}} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] \tag{3.8}$$

$$\begin{aligned} \text{subject to} \quad & r(u) - r(v) \leq c_{uv} \ \ \forall (u,v) \in C' \\ & L_u \leq r(u) \leq U_u \ \ \forall u \ \in (V' \cup M') \end{aligned}$$

3.2.5 An Example

The following example illustrates the method and shows how the number of constraints can be reduced using our approach.

Consider the circuit example shown in Figure 3.3. As in the previous examples, we make the assumption that the gates have unit delays. We consider two possible clock periods of 2 units and 3 units in this example.

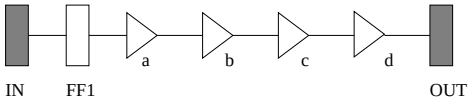

Figure 3.3 Example illustrating the approach.

3.2.5.1 When $P = 2$ units

For a clock period of two units, the list of constraints generated by the approach in [115] is listed below.

$$\begin{aligned}
\underline{\text{Circuit constraints}} \quad & r(h) - r(a) \leq 1 \\
& r(a) - r(b) \leq 0 \\
& r(b) - r(c) \leq 0 \\
& r(c) - r(d) \leq 0 \\
& r(d) - r(h) \leq 0 \\
\underline{\text{Period constraints}} \quad & r(h) - r(c) \leq 0 \\
& r(a) - r(c) \leq -1 \\
& r(b) - r(d) \leq -1
\end{aligned}$$

Note that
(a) the delay associated with the host node is zero, and
(b) the value of $r(h)$ is set to zero as a reference, so that it is not really a variable.
Therefore, this is a problem with four variables and eight linear constraints (of which three act as simple bounds).

In our approach, for a clock period of 2, we first find the bounding skews. The FF's at the input and output may not be moved, and therefore, the only movable FF is FF1, which is assigned a skew of -2 units. The correctness of this skew value is easy to verify since the only feasible location of FF1 under $c = 2$ is two delay units to the right of its current location. Therefore, we find that by using the concept of restricted mobility,

$$\begin{aligned}
-1 \leq r(a) \leq -1 \quad & \Rightarrow r(a) = -1 \\
-1 \leq r(b) \leq -1 \quad & \Rightarrow r(b) = -1 \\
0 \leq r(c) \leq 0 \quad & \Rightarrow r(c) = 0 \\
0 \leq r(d) \leq 0 \quad & \Rightarrow r(d) = 0
\end{aligned}$$

Since all nodes are fixed, and all the constraints can be dropped, all of the constraints and variables have been eliminated!

3.2.5.2 When $P = 3$ units

With the clock period is set to 3 units, the list of constraints is

$$\begin{array}{lll} \underline{\text{Circuit constraints}} & r(h) - r(a) & \leq 1 \\ & r(a) - r(b) & \leq 0 \\ & r(b) - r(c) & \leq 0 \\ & r(c) - r(d) & \leq 0 \\ & r(d) - r(h) & \leq 0 \\ \underline{\text{Period constraints}} & r(h) - r(d) & \leq 0 \\ & r(a) - r(d) & \leq -1, \end{array}$$

As before, $r(h) = 0$ is set as a reference, giving a problem with four variables (as before) and seven linear constraints (of which three act as simple bounds).

Under our approach, the relocated FF can reside either at the input of gate b, the output of gate b, or the output of gate c. Therefore, we have

$$\begin{array}{rl} -1 \leq r(a) \leq -1 & \Rightarrow r(a) = -1 \\ -1 \leq r(b) \leq 0 & \\ -1 \leq r(c) \leq 0 & \\ 0 \leq r(d) \leq 0 & \Rightarrow r(d) = 0 \end{array}$$

Using these bounds we drop all constraints but

$$r(b) - r(c) \leq 0$$

Therefore, we have reduced the problem complexity to two variables, each with fixed upper and lower bounds and one linear constraint. (Note that upper/lower bound constraints are typically much easier to handle in LP's than general linear constraints; in fact, in many cases, upper and lower bounds are actually helpful in solving the LP.)

3.3 Minarea Retiming Using Minaret

The ideas described so far have been encapsulated in Minaret (MINimum Area RETiming), a minimum area retiming algorithm for large sequential circuits. Minaret consists of three phases of finding the bounds, generating the LP and solving it. Each of these is described in detail in this section.

3.3.1 Deriving Bounds on the r Variables

As described in Section 3.2.2 the bounds are derived by finding the ASAP and ALAP locations of the FF's, using a modified form of ASTRA. An efficient method for calculating

all FF-to-FF delays (D_{ij}'s) required by ASTRA, presented in [108], is used in Minaret. If the initial locations of FF's satisfy the target clock period all lower bounds must be nonpositive and all upper bounds must be nonnegative (i.e., $L_i \leq 0$ and $U_i \geq 0 \ \ \forall\, i$), since $r(i) = 0 \,\forall\, i$ is a feasible solution. However, if the target clock period is smaller than the initial period, we may be forced to move a FF from the inputs of a gate to its outputs to obtain any feasible (including the ASAP) locations. Thus it is possible to have a negative upper bound. Similarly it is possible to have a positive lower bound if the target clock period is smaller than the initial period. The bounds on the mirror vertices for all gates with more than one fanout are derived from the circuit graph using Theorem 1.

3.3.2 Generating the Linear Program

Using the alternative description of the maximal FF sharing in Section 2.1.4 the objective function coefficients are obtained by inspection of the circuit, without explicitly adding the mirror vertices. The circuit and the mirror constraints in C' are obtained from direct inspection of the circuit graph using Equation (3.7). Because the bounds on the mirror vertices can also be obtained directly from the bounds on the gate vertices, we do not need to explicitly add the mirror vertices to the circuit graph. Since every multi-fanout gate has a mirror vertex, this gives us important savings in terms of the space and time requirements. We now describe how to obtain the period constraints in C'.

For large circuits (with tens of thousand gates) $O(|V|^2)$ memory required by the Leiserson and Saxe method of generating period constraints [59] is not practical, therefore, we use the method from [115], which requires only $O(|V|)$ memory. We take advantage of the bounds obtained in Section 3.3.1 to modify this method to run faster, generating only the reduced constraint set C'.

The algorithm in [115] uses a combination of the Dijkstra's algorithm and the Bellman-Ford algorithm. The algorithm works by generating one (s^{th}) row of the W and the D matrix at a time. An ordered pair $(w(e_{ij}), -d(i))$, denoted by (a_i, b_i), is associated with each edge e_{ij} and is used to compute the shortest distance from vertex a source vertex s. A heap is maintained for each distinct value of a_i and is indexed by this value. Until all heaps are empty, we extract the node u at the top of the minimum index heap using the function pop-min(heap index). The fanouts of u are added to the appropriate heaps if their a_u or b_u values are updated (Bellman-Ford relaxation). At the end of this procedure $D(s,u) = -b_u$ and $W(s,u) = a_u$.

Note that to satisfy a clock period P, all we have to do is to ensure that each path whose delay is greater than P has at least one FF on it. The number of FF's on any path is monotonic with the path length because negative edge weights are not allowed. Due to the monotonicity of edge weights, if we ensure at least one FF on any sub-path, we are assured to have at least one FF on all paths containing this sub-path. This strategy can be used to prune the number of constraints generated as well as the gates examined.

Adding a period constraint from s to u is one way to ensure at least one FF on all paths from s to u. This observation presented by Leiserson-Saxe was used in [115] to prune the constraint set. The idea was to add a period edge to only the vertex v, reachable from s, that satisfies the following:

$$D(s,v) > P \text{ and } D(s,u) \leq P \; \forall \; u \text{ on } s \rightsquigarrow FI(v) \tag{3.9}$$

where $s \rightsquigarrow FI(v)$ is a path from s to a fanin of v. Thus if the period constraint is added, the fanouts of u need not be relaxed. Similarly if the bounds on the r variables guarantee us at least one FF on any sub-path, we need not process any path containing this sub-path.

At the end of the ASTRA run for obtaining the lower bounds all FF's are in the ALAP locations. If the delay of all the gates is not the same, it is possible that retimed circuit obtained by ASTRA with FF's in the ALAP locations may have some purely combinational paths with delays that are greater than the target clock period P. However in practice, most of the other paths satisfy the target clock period. We will use this observation to further speed up the constraint generation process.

Consider a fixed gate a in the circuit at the end of the ALAP run. Now if none of the combinational paths starting at this gate violate the clock period, we have $W_{ALAP}(a,i) \geq 1$ if $D(a,i) > P \quad \forall i$. Since $W_{ALAP}(a,i) = W(a,i) + L_i - L_a$ we have $L_a - L_i \leq W(a,i) - 1$, or $L_a - L_i \leq c_{a,i}$. Since gate a is fixed $U_a = L_a$, we obtain $U_a - L_i \leq c_{a,i} \;\; \forall i \;\in V$, which is guaranteed to be true, and hence all constraints starting from fixed gate a are redundant, and we do not need to generate them. Thus we must generate period constraints only from those fixed gates which have at least one purely combinational path starting from it with delay more than the clock period. Let us call this set V''.

The pseudo code presented bellow explains how we use the bounds on the r variables to generate the reduced constraint set C' efficiently.

```
P = target clock period;
L_i ≤ r(i) ≤ U_i  ∀i ∈ V;
S_k= the k^th heap;
L_min = min(L_i)  ∀i ∈ V;
∀s ∈ V' ∪ V''
  {
    s = current vertex;
    ∀v ∈ V, a_v = ∞ and b_v = 0;
    S_0 = {s}, a_s = 0,   and   b_s = −d(s);
    k = current register weight;
    do {
       k = min{p | S_p ≠ ∅};
       if(k ≥ U_s − L_min + 1) break;
```

```
    u = pop-min(S_k) ;
    if(U_s − L_u ≤ k − 1) continue;
    if(−b_u > P)
    add a period edge c(s, u) with weight a_u − 1
    else {
        ∀v ∈ FO(u) {
        if( k − U_s + L_v < 1)
            if((a_v, b_v) > (a_u + s_{u,v}, b_v − d(v)))
            heap-insert(S_{a(u)+s_{u,v}}, v);
            }
        }
  } while(∃ p | S_p ≠ ∅)
}
```

3.3.3 Solving the Linear Program

Like Equation (2.7), the LP in Equation (3.8) is also a dual of a minimum cost network flow problem. We found that it could be solved very efficiently using the network simplex algorithm from [5]. The network simplex method is a graph based adaptation of the LP simplex method which exploits the network structure to achieve very good efficiency. The upper and lower bounds on the r variables provide a initial feasible spanning tree. This tree has two levels only, with the host node as the root and all other nodes as leaves. To prevent cycling we construct the initial basis to be strongly feasible by using the appropriate bound (upper or lower) to connect a node to the root (host node). It is easy to maintain strongly feasible trees during the simplex operations, and details are given in [5].

Using the first eligible arc pivot rule with a wraparound arc list from [3] (page 417) gave us significant improvements in the run time. The dual variables (r variables) are directly available from the min cost flow solution. We could solve problems with more than 57,000 variables and 3.6 Million constraints in about 2.5 minutes.

3.4 Experimental Results

We now present area minimization results on circuits in the ISCAS89 [7] benchmark suite, subject to a given clock period. We assume that all gates have a unit delay, although we emphasize that the algorithm is applicable when gates have non-unit delays. The target clock period is set to be the minimum achievable clock period for the circuit under retiming and is calculated using ASTRA. Therefore the results show the smallest number of FF's for the best clock period for all circuits. Since we did not have access to large circuits ($> 20,000$ gates) we

created some large circuits (myex1 through myex5) by combining circuits from the ISCAS89 benchmark suite.

We present the results in two tables. Table 3.1 presents measures of the quality of minimum area retiming in the circuits. For each circuit, the number of gates $|G|$, the target clock period P, the final number of FF's in the circuit from both ASTRA and Minaret, and the CPU time in seconds T_{exec} of Minaret are shown. Also shown are two metrics on the circuits: F_{fx}, the percentage of gates found to be fixed and M_{avg}, the average mobility, i.e., the average value of $(U_y - L_y)$ over all gates in the circuit. Since $U_y - L_y$ gives the range in possible values (or mobility) of $r(y)$, M_{avg} is a measure of the average mobility in the circuit. The number of FF's both in ASTRA and Minaret are obtained under the more accurate area model of Section 2.1.4, after taking into account the maximum sharing of FF's at all nodes (including primary inputs) in the circuit. The execution times are in seconds on a DEC AXP system 3000/900 workstation, and include the time spent in getting the bounds, generating the LP and solving it.

For most ISCAS89 circuits M_{avg} was less than unity and the average over all ISCAS89 circuits was about 0.7. The percentage of fixed nodes F_{fx} varied from being as high as 95% to being below 1% (for s38417). We observed that circuits that have a small critical part (perhaps a cycle in the retiming graph) with most gates being off the critical paths in the timing graph, result in high values of M_{avg}. We note that these circuits are not very well suited for retiming since the small critical parts of the circuit restrict the rest of the circuit from achieving better clock periods. The CPU time T_{exec} depends on the the number of gates in the circuit $|G|$, the average mobility M_{avg} and F_{fx}.

In [115] the circuit s38584 needed 38 hours of CPU time, while Minaret could retime it in about one minute. We point out, though, that such a comparison is not entirely fair since (a) the results are generated on different platforms and (b) the circuits used in [115] are modified ISCAS89 benchmarks and have a much smaller number of gates. For example s38584 has 7882 gates in [115] while it has 19,253 gates in this work.

In Table 3.2, we compare the size of the LP for Minaret and the original problem by presenting the number of variables and constraints for both methods. The number of variables include both the gate and mirror variables. The number of constraints for Minaret includes the upper and lower bounds, while that for the original method are obtained by using the pruning strategy suggested in [59], and implemented in [115]. The reduction in the number of constraints in Minaret depends on the average mobility M_{avg} and F_{fx}. However, since the original constraints are generated after some pruning them self, the reduction is affected by other factors as well. Table 3.2 also presents the breakup of the CPU time (in seconds) in terms of the time spent in using ASTRA to arrive at the bounds for the r variables (T_b), the time spent in generating the LP of Equation (3.8) (T_g) and the time needed to solve this LP by the network simplex method (T_s).

Table 3.1 Minarea Retiming Using Minaret

Circuit	$\|G\|$	P	# FFs		F_{fx}	M_{avg}	T_{exec}
			ASTRA	Minaret			
s27	10	6.0	3	3	86.67%	0.13	0.00s
s208.1	104	10.0	27	9	66.09%	0.36	0.01s
s298	119	6.0	36	22	34.38%	0.75	0.01s
s382	158	7.0	33	23	32.93%	0.67	0.02s
s386	159	11.0	6	6	90.75%	0.09	0.01s
s344	160	14.0	22	19	11.11%	1.24	0.04s
s349	161	14.0	22	19	11.05%	1.24	0.04s
s444	181	7.0	49	28	31.05%	0.69	0.03s
s526n	194	6.0	41	30	42.36%	0.59	0.02s
s510	211	11.0	8	7	42.62%	0.62	0.07s
s420.1	218	12.0	57	17	60.76%	0.41	0.03s
s635	286	66.0	35	35	68.17%	0.32	0.04s
s641	379	74.0	19	19	70.55%	0.29	0.05s
s713	393	74.0	19	19	67.85%	0.32	0.08s
s967	394	12.0	41	35	12.24%	0.88	0.22s
s953	395	13.0	44	27	8.99%	0.93	0.26s
s838.1	446	16.0	117	33	53.01%	0.51	0.14s
s938	446	16.0	117	33	53.01%	0.51	0.14s
s1196	529	24.0	18	18	81.33%	0.19	0.03s
s1238	508	22.0	18	18	81.53%	0.19	0.03s
s1269	569	19.0	111	84	54.77%	0.49	0.11s
s1494	647	16.0	20	7	93.47%	0.07	0.05s
s1488	653	16.0	17	7	95.44%	0.05	0.05s
s1423	657	53.0	76	76	28.13%	0.83	0.59s
s1512	780	23.0	84	70	18.55%	0.99	1.05s
s3271	1,572	15.0	306	168	49.38%	0.81	0.25s
prolog	1,601	13.0	358	122	49.77%	0.55	0.27s
s3384	1,685	27.0	438	167	14.31%	3.15	2.44s
s3330	1,789	14.0	331	110	63.46%	0.39	0.22s
s4863	2,342	30.0	201	138	28.46%	0.97	5.24s
s5378	2,779	21.0	555	173	36.12%	0.85	1.28s
s6669	3,080	26.0	719	305	40.02%	0.76	2.20s
s9234.1	3,270	38.0	205	134	14.62%	1.55	6.18s
s13207.1	7,791	51.0	629	446	21.49%	2.96	10.38s
s15850.1	9,617	63.0	571	525	24.15%	1.52	38.81s
s35932	16,065	27.0	1,729	1,729	55.27%	0.54	7.56s
s38584.1	19,253	48.0	1,428	1,427	14.22%	2.13	65.07s
s38417	21,370	32.0	1,616	1,370	0.88%	4.35	146.92s
myex1	25,717	42.0	5,146	2,293	4.75%	2.51	169.10s
myex2	28,946	45.0	5,655	2,022	8.73%	2.26	160.47s
myex3	35,353	35.0	8,052	3,279	5.22%	2.65	489.52s
myex4	40,661	35.0	11,591	2,803	1.80%	4.12	421.50s
myex5	56,751	47.0	11,488	3,378	4.95%	3.98	799.64s

Table 3.2 Reduction in Number of Variables and Constraints in Minaret

Circuit	# Variables			# Constraints			T_b	T_c	T_s
	Minaret	Original	F_{red}	Minaret	Original	F_{red}			
s27	5	20	75.00%	10	35	71.43%	0.00s	0.00s	0.00s
s208.1	54	144	62.50%	239	540	55.74%	0.00s	0.00s	0.00s
s298	117	163	28.22%	628	1,471	57.31%	0.01s	0.00s	0.00s
s382	157	217	27.65%	1,005	2,146	46.83%	0.01s	0.01s	0.00s
s386	22	200	89.00%	73	2,903	97.94%	0.01s	0.01s	0.00s
s344	201	221	9.05%	1,722	2,117	18.66%	0.01s	0.03s	0.00s
s349	203	223	8.97%	1,581	1,847	14.40%	0.01s	0.03s	0.00s
s444	177	256	30.86%	1,430	3,121	54.18%	0.01s	0.02s	0.00s
s526n	167	258	35.27%	1,097	4,674	76.53%	0.01s	0.01s	0.00s
s510	183	311	41.16%	2,303	7,331	68.59%	0.01s	0.06s	0.00s
s420.1	123	296	58.45%	553	609	9.19%	0.01s	0.02s	0.00s
s635	157	416	62.26%	478	1,283	37.26%	0.02s	0.02s	0.00s
s641	158	496	68.15%	521	1476	64.70%	0.02s	0.03s	0.00s
s713	191	532	64.10%	663	2,373	72.06%	0.02s	0.07s	0.00s
s967	527	583	9.61%	9,223	13,929	33.79%	0.02s	0.18s	0.02s
s953	554	593	6.58%	10,918	12,585	13.24%	0.02s	0.22s	0.03s
s838.1	296	600	50.67%	1,235	2,482	50.24%	0.03s	0.11s	0.00s
s938	296	600	50.67%	1,235	2,484	50.24%	0.03s	0.11s	0.00s
s1196	184	713	74.19%	570	1,686	66.19%	0.02s	0.01s	0.00s
s1238	182	702	74.07%	565	1,781	68.28%	0.02s	0.01s	0.00s
s1269	371	765	51.50%	1,363	20,250	93.27%	0.03s	0.08s	0.00s
s1494	50	751	93.34%	247	32,215	99.24%	0.02s	0.03s	0.00s
s1488	37	757	95.11%	154	33,277	99.54%	0.03s	0.03s	0.00s
s1423	647	860	24.77%	2,359	16,266	85.50%	0.05s	0.54s	0.01s
s1512	823	983	16.28%	24,331	52,346	53.52%	0.04s	0.98s	0.03s
s3271	1,079	2,038	47.06%	5,492	43,506	87.38%	0.07s	0.16s	0.02s
prolog	1,039	1,992	47.84%	5,304	37,319	85.79%	0.08s	0.17s	0.02s
s3384	1,870	2,166	13.67%	47,916	49,487	3.17%	0.12s	2.07s	0.25s
s3330	858	2,212	61.21%	4,595	30,409	84.84%	0.08s	0.12s	0.01s
s4863	2,170	2,995	27.55%	92,873	597,323	84.45%	0.11s	4.80s	0.34s
s5378	2,385	3,664	34.91%	19,170	168,530	88.63%	0.11s	1.04s	0.13s
s6669	2,539	4,100	38.07%	20,041	341,750	94.14%	0.17s	1.89s	0.14s
s9234.1	3,366	3,893	13.54%	54,610	137,962	60.42%	0.18s	5.65s	0.35s
s13207.1	7,303	9,180	20.45%	38,630	491,561	92.14%	0.63s	8.74s	1.01s
s15850.1	8,740	11,332	22.87%	38,318	1,046,108	96.34%	0.99s	36.32s	1.50s
s35932	10,306	21,716	52.54%	53,087	389,647	86.38%	1.13s	4.55s	1.88s
s38584.1	20,486	23,390	12.42%	97,268	11,450,472	99.18%	2.23s	54.40s	8.44s
s38417	25,731	25,923	0.74%	1,507,162	1,628,544	7.45%	2.93s	91.60s	52.38s
myex1	31,476	32,922	4.39%	812,872	3,275,567	75.18%	3.41s	146.34s	19.36s
myex2	31,704	34,493	8.09%	398,697	17,185,252	97.68%	4.26s	131.83s	24.38s
myex3	42,604	44,812	4.93%	5,693,689	16,978,788	66.46%	5.01s	403.84s	80.66s
myex4	48,415	49,214	1.62%	2,635,127	8,186,340	67.81%	6.47s	311.51s	103.52s
myex5	57,488	60,241	4.57%	3,600,681	24,316,717	85.20%	10.33s	637.84s	151.47s

T_b depends on the number of gates and FF's in the circuit, $|G|$ and the average mobility, M_{avg}, of the circuit. Phase A of ASTRA is dependent on the number of gates for obtaining the FF to FF delays and on the number of FF's for the Bellman-Ford runs. The CPU time taken for phase B of ASTRA depends on M_{avg}, since M_{avg} gives a measure of how many retiming (or movement of FF's across gates) are performed in phase B.

T_g is most strongly influenced by the number of flexible gates, i.e., $(1 - F_{fx}) \cdot |G|$, which is equal to the number of rows of W and D matrices we need to generate. It is also influenced by M_{avg} in that it determines the number of gates processed for each row of W and D matrices. T_s depends on the size of the LP in terms of the number of variables and constraints.

3.5 Conclusion

In this chapter we presented a fast algorithm for minarea retiming of large circuits. The contributions of this chapter are twofold. First, it reconciles the Leiserson-Saxe algorithm with the ASTRA algorithm and shows the relation between these two. Second, it utilizes this relationship to good purpose by modifying the ASTRA algorithm to make available information from the skew-retiming equivalence that is of great benefit in solving the minarea retiming problem under the Leiserson-Saxe framework.

Experimental results on benchmark circuits in the ISCAS89 benchmark suite have been presented, and the procedure is seen to give good benefits. The number of variables and constraints where dramatically reduced in most cases. The entire ISCAS89 benchmark suite could be retimed in minutes. This chapter shows that it is possible to perform minarea retiming on large circuits in a reasonable amount of time.

Even though the average mobility M_{avg} is high and the fraction of fixed gates F_{fx} is low for the large circuits we created, we are still able to retime them in a reasonable amount of time. Because of the various pruning techniques used in Minaret, the number of constraints in practical circuits grows at a far slower rate than $O(|G|^2)$.

Minaret also has a reduced memory requirement since a significant number of constraints are not stored. We found that for large circuits having constraints in millions, the memory requirement becomes a bottleneck. The reduction in the number of constraints also reduces both the problem generation and the problem solution time.

To the best of our knowledge, no other retiming algorithm incorporates pruning methods to reduce the number of variables. This reduction in the number of variables significantly reduces the problem generation time. Notice that due to the presence of mirror vertices, the number of variables can be up to twice the number of gates in the circuit. Hence the reductions in the number of variables and constraints provided by Minaret are important to retime large circuits.

4 RETIMING CONTROL LOGIC

4.1 Introduction

A major problem associated with the application of retiming to control logic, is the preservation of the initial (reset) state of a circuit, which is determined by the initial values of the registers in the circuits. In the synthesis of control logic, the initial state of the circuit is an integral part of its behavior therefore, it is necessary to find an equivalent initial state for the retimed circuit. An initial state in the retimed circuit is equivalent to that in the original circuit if for any input sequence applied to both the circuits, with the original circuit started in the initial state and the retimed circuit started in the equivalent initial state, the same sequence of outputs is produced [30].

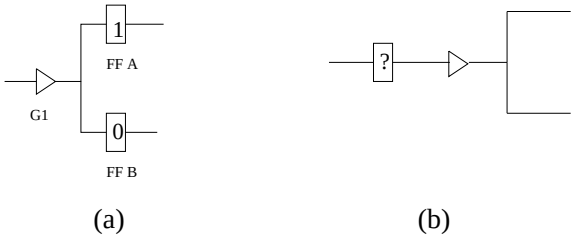

Figure 4.1 (a) Original circuit. (b) Retimed circuit

It is not always possible to find an equivalent initial state for the retimed circuit without modifying it. For example, consider the circuit in Figure 4.1 taken from [130]. If the initial value of FF's A and B are 1 and 0, respectively, then the retimed circuit cannot be initialized to have the same behavior as the original circuit since an equivalent initial value of FF C in the retimed circuit cannot be found. Techniques for finding a retiming with an equivalent initial state were proposed in [30, 130].

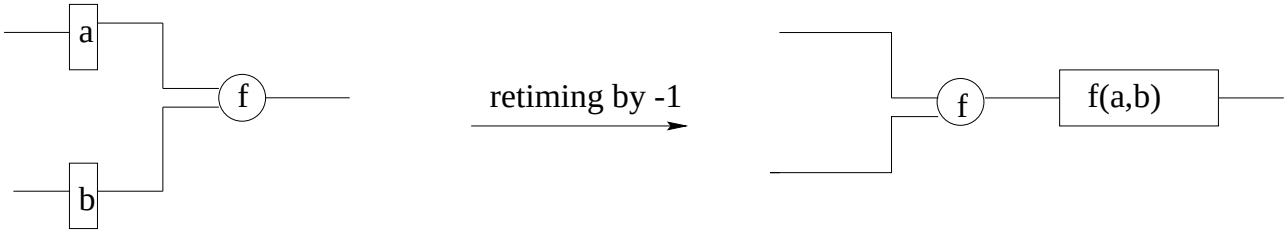

Figure 4.2 Forward retiming of a combinational logic node

As shown in Figure 4.2, an equivalent initial state can always be found for forward motion of FF's (referred to as "negative retimings" using the notation of Leiserson and Saxe). Thus, one way to ensure that an equivalent initial state can always be found, is to permit only forward retiming moves. This concept was used by Touati and Brayton in [130] to compute initial states of retimed circuits. In their approach, FF's may be removed from all the primary inputs and inserted at all the primary outputs, corresponding to a motion across the host node (defined in Section 2.1). The problem is then reduced to determining the initial values for the FF's inserted at the primary inputs. If k FF's are inserted at the primary inputs then a sequence of k input values is required. This sequence is obtained from the state transition diagram extracted from the circuit. Such a sequence exists if the initial state is reachable from any other state in k transitions; otherwise the circuit has to be modified by incorporating additional logic to obtain such a sequence. This logic increases the circuit area and may also increase the minimum achievable clock period [30].

Permitting only forward moves is too restrictive because some backward moves have equivalent initial states. For example, if in the circuit in Figure 4.1, both FF A and B have the same initial value, then the backward move across gate G1 is possible while maintaining equivalent initial state. Hence another retiming requiring some backward moves may exist, that enables one to find an equivalent initial state without any modifications to the circuit, even for circuits where the method of [130] required circuit modifications. Reverse retiming [30, 128] finds this retiming by disallowing FF moves across the primary outputs and by minimizing their backward motion.

For digital circuit design, the most useful objective function is that of constrained minarea retiming. However none of the above methods considers the area penalty during retiming to achieve the target clock period since they perform minperiod retiming rather than minarea retiming. The standard minarea algorithms, e.g., [59, 115] or Minaret pay no regard to the initial states, and while they have applications in datapaths where the initial state is unimportant, they cannot be used to optimize control logic since an equivalent initial state is not guaranteed to exist in the retimed circuit.

We believe that this thesis is the first to target the problem of minarea retiming for control logic guaranteeing equivalent initial states. As in [130], we use the phrase *retiming an initial state* to mean finding a retiming (with a initial state) such that the original circuit and the retimed circuit have the same behavior when started in their respective initial states. We use the term *minarea initial state retiming* to refer to retiming an initial state with minimum number of FF's.

In this chapter we use bounds on the retiming variables to allow backward motion of FF's only if an equivalent initial value exits. Therefore, any retiming thus obtained will have an equivalent initial state. There may be multiple sets of these bounds, and all of them must be explored to obtain an optimal minarea initial state retiming. However the number of FF's

obtained by standard minarea retiming can be used as a lower bound to prune this exploration.

This chapter also provides a new formulation that takes into account the initial value of the FF's while modeling the sharing of FF's at the output of a multi-fanout gate. The method presented here is applicable for retiming of any circuit which has more than one type of memory elements (e.g., FF's with load enables) such that memory elements of different types can not be merged together.

The rest of the chapter is organized as follows: In Section 4.2 we present an method to ensure the existence of an equivalent initial state, followed by the FF sharing model in Section 4.3. We present experimental results in Section 4.4 and conclude the chapter in Section 4.5.

4.2 Ensuring Equivalent Initial States

The requirement of initial state equivalence imposes restrictions in addition to those in the conventional minarea retiming problem. Thus the number of FF's obtained by the conventional minarea retiming is a lower bound on the number of FF's obtainable by a minarea equivalent state retiming. We call this lower bound Γ.

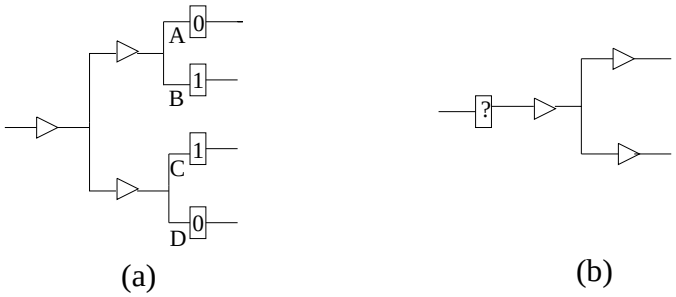

Figure 4.3 Conventional minarea retiming

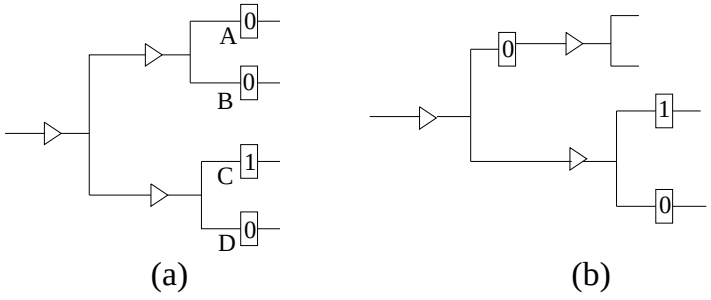

Figure 4.4 Example of variation in the number of FF's with initial state

However it is not always possible to achieve this lower bound. As an example, consider the circuit shown in Figure 4.3(a), and the conventional minarea retiming (without regard to initial state) shown in Figure 4.3(b) requiring only one FF. If the initial value for FF A through D is as shown, then the retimed circuit in Figure 4.3(b) does not have an equivalent initial

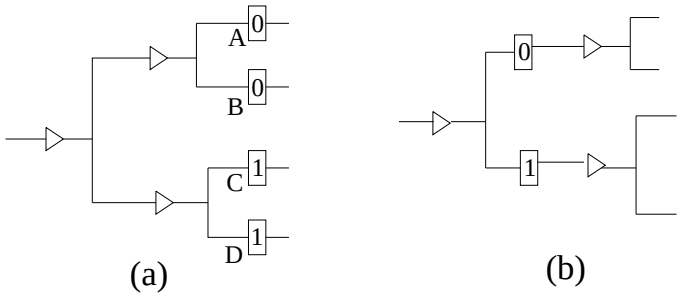

Figure 4.5 Another example of variation in the number of FF's with initial state

state. Further, there is no possible retiming with one FF that has an equivalent initial state. However, if the initial value for each FF in the circuit is 0 and FF D is 0, then the retiming obtained in Figure 4.3(b) is feasible in terms of initial state equivalence. Therefore, depending on the initial state, it may or may not be possible to achieve the lower bound Γ.

Additionally, the optimal number of FF's with equivalent initial state depends on the initial state of the original circuit. As shown in Figure 4.4, if initial value of FF's are {A=0, B=0, C=1, D=0} then the minimum number of FF's possible is 3. It can readily be verified that if the initial value of FF's are {A=0, B=0, C=1, D=1} then the minimum number of FF's is two, as shown in Figure 4.5.

In this chapter, we will attempt to find the minarea equivalent state retiming for a given circuit topology, and a given set of initial values. To ensure the existence of an equivalent initial state in the retimed circuit we allow only those retiming moves that have an equivalent initial state. This includes all forward retimings except across host node ($r(H) = 0$), and backward retiming moves with equivalent initial states. We forbid retiming across the host vertex because it requires a sequence of initial values for the primary inputs to be obtained from the state transition diagram extracted from the circuit and, in addition could require modifications to the original circuit [130].

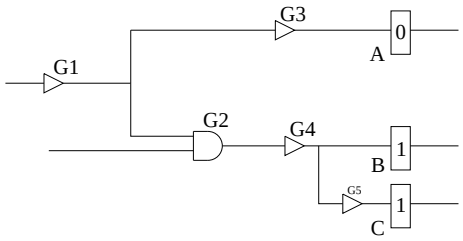

Figure 4.6 An example circuit where lower bound Γ is achievable

As an example, consider the circuit in Figure 4.6 and its retimed version in Figure 4.7. If the initial values of FF's are {A=0, B=1, C=1}, then there is no equivalent set of initial values for the retimed FF's D and E in Figure 4.7. Figure 4.8 presents a alternative retiming of the original circuit requiring the same number of FF's, but in this case an initial value

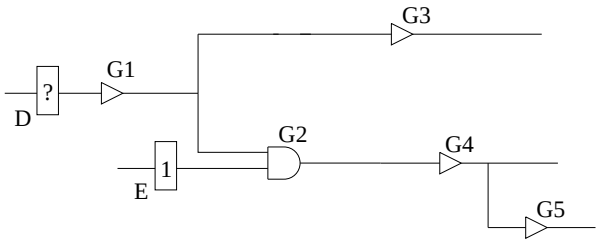

Figure 4.7 A minarea retiming without equivalent initial state

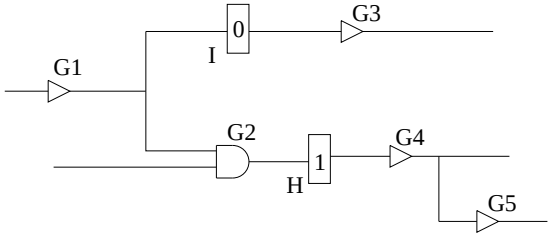

Figure 4.8 A minarea retiming with equivalent initial state

of 0 at FF I and 1 at FF H is equivalent to the original initial state. Note that performing conventional minarea retiming without regard to initial states could result in either of these retimings. Therefore, even if the lower bound Γ is achievable, there is no guarantee that an algorithm that ignores the initial state will find it.

To understand why some retimings have equivalent initial states and others do not, we observe that the fundamental reason for being unable to achieve an equivalent initial state for retiming is the presence of conflicting values at the fanouts of a gate. For example, in Figure 4.6 if we try to move the FF's in the original circuit backwards to obtain the retiming in Figure 4.7 we get FF's at the output of gate G1 with values 1 and 0, which cannot be moved to the input of gate G1 while maintaining an equivalent initial state. We refer to this situation as a *conflict*, and it is the reason why the lower bound Γ is not always achievable.

To see how we perform initial state retiming in the presence of conflicts, consider the circuit of Figure 4.6. If we do not allow any backward motion of FF's across gate G1, than we can be assured that every retiming has an equivalent initial state. Backward motion across G1 can be prohibited by forcing an upper bound of 0 on the r variable of gate G1. It can be seen that any retiming with $r(G1) \leq 0$ and $r(H) = 0$ has an initial state equivalent to the initial state of the original circuit.

Thus one way to ensure that any obtained retiming has an equivalent initial state, is to update the upper bounds U_v in the LP of Equation (3.8) so that conflicting FF's at the fanouts of a gate are never retimed to its inputs. This will ensure a valid equivalent state in the retimed circuit. This new upper bound on gate v that ensures a valid equivalent state is called B_v. Since we want a retiming that has an equivalent state *and* satisfies the target clock period we need to enforce $r(v) \leq B_v$ and $r(v) \leq U_v$. If we define *justification upper bound* as $J_v = \min(B_v, U_v)$,

then we only need to ensure $r(v) \leq J_v$. A set of such justification upper bounds denoted by $\Lambda^i = \{J_u^i | \ \forall u \in V\}$. Since forward retiming moves always have equivalent initial values, the lower bounds from Equation (3.8) for conventional minarea retiming are still valid for minarea initial state retiming. Thus we obtain the following modified LP

$$\begin{aligned} & \text{minimize } \textstyle\sum_{v \in V} [(|FI(v)| - |FO(v)|) \cdot r(v)] & & (4.1) \\ \text{subject to} \quad & r(u) - r(v) \leq c_{uv} & \forall c(u,v) \in C \\ & L_u \leq r(u) \leq J_u & \forall u \ \in V \end{aligned}$$

Any solution to this LP will have an initial state that is equivalent to the initial state in the original circuit and will satisfy the target clock period. The techniques of Section 3.2.4 can be applied to further reduce the size of the LP in Equation (4.1).

4.2.1 Obtaining the Justification Bounds

We will now describe a method for obtaining these new justification upper bounds J_v for a gate v. The procedure consists of two steps: a justification step, where an equivalent initial state is found, and a bound computation step, where the bounds J_v on each gate under that equivalent initial state are calculated.

With every FF we associate a three valued (1,0,X) logic. We define *compatibility* as follows: a logic value of 0 is compatible with both 0 and X, but logic values 0 and 1 are not compatible with each other[1]. A gate can only be retimed if it has FF's with compatible logic values at all of its fanouts. A gate is retimed in the backward direction by removing a FF from each of its fanouts, and adding one to each of its inputs. A gate is called output-ready if it has a FF on each of its fanouts and the logic value on each such FF is compatible with the values on the others. The procedure maintains a list of gates that can be retimed. A gate is taken from the list and retimed, and the list is updated. As the gates are retimed, a procedure similar to the one in Section 3.2.2 is used to compute the bounds. The upper bounds, J_v are obtained by moving FF's as far backwards, as possible without violating the period constraints. The count of the FF's moved across any gate gives its upper bound on the r variable of the gate.

Each time FF's are moved from the outputs of a gate to its inputs, we must assign logic values to the new FF's added at the inputs. These logic values must be equivalent to the original value at the output of the gate in order to obtain a initial state retiming. This assignment, in general, may not be unique and is similar to the phase of justification in the process of automatic test pattern generation [2]. We classify these output ready gates into the following two categories.

Unique Justification If there is only a unique mapping of the logic value at the output to the logic values at the inputs, then we do not have to make any choices. These justifications

[1] For circuits with multiple types of memory elements that cannot be combined, compatibility can be defined similarly.

are maintained in a unique justification queue, U. The following cases are examples of unique justifications:

- A single input gate such as inverter or buffer.
- A logic value of X at the output: in this case all inputs can be assigned a logic value of X.
- A logic value of 0 at the output of an OR (NAND) gate: in this case we assign all inputs to logic value 0 (1).
- A logic value of 1 at the output of an AND (NOR) gate: in this case we assign all inputs to logic value 1 (0).

Nonunique Justification If there are multiple mappings possible for the logic value at the output to the logic values at the inputs, then we must make a choice or a decision in this case. These decisions are maintained in a decision queue, D. Since the solution to the LP in Equation (4.1) depends on the set Λ^i which in turn depend on these decision we make here, we may have to revisit these decisions. The following cases are examples of non-unique justifications:

- A logic value of 1 at the output of an OR (NAND) gate: in this case we assign any one input to logic value 1 (0) and the rest to logic value X.
- A logic value of 0 at the output of an AND (NOR) gate: in this case we assign any one input to logic value 0 (1) and the rest to logic value X.

4.2.2 Searching for the Optimal Solution

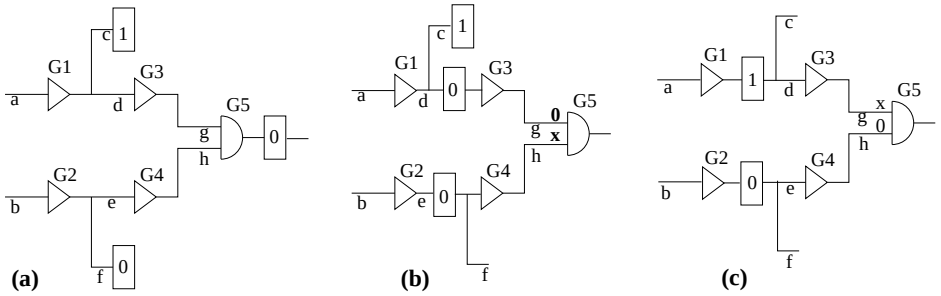

Figure 4.9 Effect of justification of on the number of FF

Different justification decisions may lead to a different number of FF's obtained after minarea retiming. As an example, consider the circuit shown in Figure 4.9(a), with a FF with value 0 at the output of an AND gate, leading to two possible choices shown in Figures 4.9(b) and 4.9(c). The corresponding decisions lead to retimed circuits with three and two FF's, respectively.

Under nonunique justifications, a number of different allowable justifications are possible. Let us define a set of one such possible justification as Δ^i. Each such Δ^i will give us a set (one for each gate) of justification upper bounds $\Lambda^i = \{J_v^i | \ \forall v \in \ V\}$ that is used to solve the minarea LP. If the number of FF's so obtained is not equal to the minarea lower bound Γ, we must backtrack and obtain another set of justifications Δ^j that leads to a different Λ^j. This process is repeated until we either achieve the minarea lower bound Γ, or no more justifications exist. Since a complete exploration will be computationally expensive, one may halt the exploration of the search space at any time and take the best solution obtained so far.

Thus the process of minarea initial state retiming can be given by the following pseudocode. The procedure returns the minarea retiming with an equivalent initial state.

```
1     Obtain minarea lower bound Γ
2     j = 0;
3     Best = ∞;
4     while (true)
5     {
6       while (U ≠ ∅ OR D ≠ ∅)
7       {
8         if (U ≠ ∅) do_unique_justification
9         if (D ≠ ∅) = do_decision_justification
10      }
11        /* This gives us a justification set Δ^j.  */
12        /* which corresponds to a set of justification upper bounds Λ^j.  */
13      Obj = lp_solve(Λ^j); /* solve LP in Equation (4.1) */
14      If(Obj == Γ) return(Obj); /* lower bound obtained */
15      If(Best > Obj) Best = Obj; /* store best result */
16      B = backtrack(Δ^j);
17      If(B == Infeasible) return(Best); /* all justifications explored */
18    }
```

The function `backtrack` changes the last decision that has a yet unexplored choice, and is similar to one used in automatic test pattern generation (see for example [52]). The period constraints need be generated only once during the entire procedure since they do not depend on the justification process. This is helpful since the period constraint generation is a very computationally intensive process.

The theoretical upper bound on the number of possible justification sets Δ^i's in the worst case is $|FF| \cdot \prod_{\forall v \in V} |FI(v)|$, where $|FI(v)|$ is the number of fanins of the gate v, and $|FF|$ is the number of FF's in the original circuit. This upper bound is due to the fact that in the worst case, each FF in the circuit may move across every gate and every such move may require

a decision. This bound is clearly exponential and thus the problem of finding all possible justification sets for a general circuit is NP-hard, as in the case of the justification phase of automatic test pattern generation [52].

However in practical circuits the number of feasible justifications will be much less than this theoretical upper bound due to the following reasons

- As shown in Section 3.4 the mobility of FF's is very limited in practice and hence all FF's cannot move across all gates as assumed by the theoretical bound above.
- Due to conflicts at the gate fanouts the FF's may not be able to move towards the inputs of that gate, and this further restricts the mobility of the FF's.
- Some FF's moving across gates have unique justifications.
- Every time a decision is made in case of a nonunique justification, all fanins but one are assigned logic value X. This logical X moves backward through unique justification until it is forced to a 0 or 1.
- As soon as the lower bound of Γ is achieved we do not need any more justification sets. In our experimental results we founds that in many circuits this lower bound is achieved in the first few iterations.
- Only backward moves need justification, while forward moves have a unique mapping of logic values and hence do not add to the number of Δ^i's.

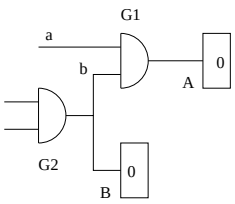

Figure 4.10 An example of a pruning technique

The number of justification sets can be further reduced by pruning suboptimal Λ^i's. Consider the circuit in Figure 4.10 with the logic values of FF A and FF B equal to 0. Since the output of the AND gate G1 is at logic value 0 we have two possible mappings for the equivalent values at the inputs a and b. However the choice of setting input a to X and input b to 0 is better than the choice of $a = 0$ and $b = X$. This is because in the presence of FF B with logic value 0, the X on input b will be forced to a 0, effectively setting the choice to $a = 0$ and $b = 0$. This is suboptimal to the choice of $a = X$ and $b = 0$, since X on input a can move further back than a 0.

4.3 Conditional FF sharing

The LP of Equation (2.7) in Section 2.1.4 assumes that a FF can be combined with any other FF, and hence is not applicable to minarea initial state retiming where FF's have logic values associated with them, and a FF with logic value 1 can not be shared with one that has logic value 0. For example, consider the circuit in Figure 2.1(a) with initial state values as shown in Figure 4.11(a). With these initial values, the sharing given by the mirror vertex model of Section 2.1.4 is shown in Figure 2.1(b); this is not valid for the given initial values. Instead, the maximal sharing is as shown in Figure 4.11(b) and requires a total of six FF's. The reason is that only two FF's, shown in the dashed box in Figure 4.11(a), can be shared. The situation is further complicated by the fact that two FF's can be shared only if the FF's at their fanins (if any) are also shared. For example, consider the circuit in Figure 4.11(b), the FF's on output C and D cannot be shared, although both have an initial state value of 1, because their fanins are not shared.

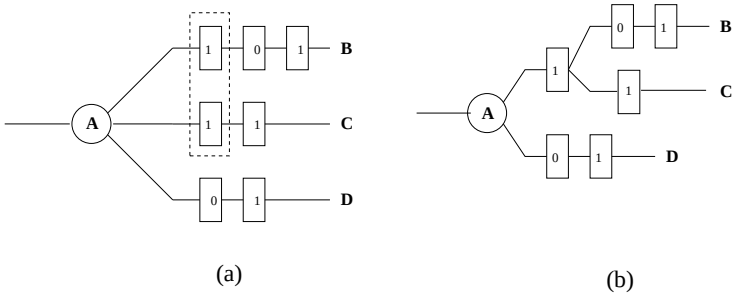

Figure 4.11 Conditional register sharing at multiple fanouts

Thus we need a way to model the conditional sharing when FF's have initial values associated with them. This conditional sharing is also required for circuits having more than one type of FF's that can not be shared with each other. We will now present the modifications required to model the conditional sharing by a 0/1-MILP formulation. This modification is used for all gates with conflicts at their fanouts, and for all other gates the simpler model of Section 2.1.4 is used. This combination keeps the number of integer variables within a small fraction of the total number of variables. We will first present the model and then illustrate it through an example.

The justification process of Section 4.2 determines the logic values of all FF's that can possibly be retimed backwards to arrive at the fanout of a given gate. There is a sequence of these "possible" FF's that may arrive at every fanout of every gate, and possibly be retimed across the gate, or remain at the gate output; the final retiming may contain only a subsequence of this possible sequence. The logic values of these possible FF's at the fanouts of a gate u are represented by a table T_u with $|FO(u)|$ rows as shown in Figure 4.12. Since a maximum of J_v FF's can be moved backwards across gate v to its fanins, and $w(e_{uv})$ FF's already exist between gate u and gate v, the maximum number of FF's possible between gate u and gate v

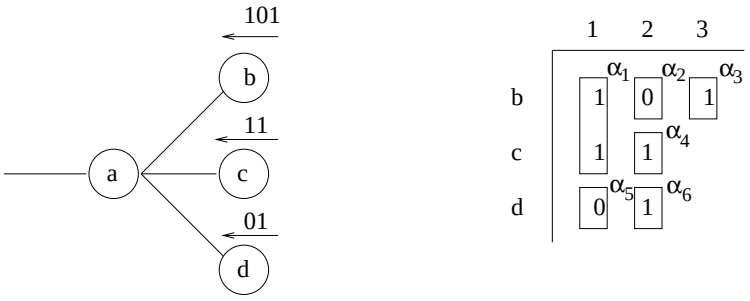

Figure 4.12 An example of FF sharing

is $J_v + w(e_{uv})$. Therefore each row, $v \in FO(u)$, has $J_v + w(e_{uv})$ entries, each of which is either a 0 or a 1.

The value in the v^{th} row and k^{th} column of the table is denoted by $T_u(v,k)$. We define a sharing class[2] S_i to contain a set of values that can be shared, and represent the set of sharing classes for the fanouts of gate u by N_u. Two values $T_u(p,q)$ and $T_u(r,s)$ can be shared (i.e., belong to the same sharing class) only if $q = s$ and $T_u(p,i) = T_u(r,i)$ for $i = 0, \cdots, s-1$. A function $class(T_u(v,k))$ gives the index of the sharing class for entry (v,k) in table T_u, e.g., $S_{class(T_u(v,k))}$ is the sharing class containing the k^{th} FF between gate u and its fanout v (counting from u). All the FF's in a sharing class can be shared with each other, and hence require only one physical FF. Each sharing class S_i is represented in the MILP by a variable $\alpha_i \in \{0,1\}$. If $\alpha_i = 1$ in the optimal solution of the MILP, then the FF's of sharing class S_i share a physical FF and the sharing class S_i is said to be active. FF's moved forward across gate u to its fanouts can be shared unconditionally and will be handled later.

To ensure that the k^{th} FF retimed across gate v activates its own sharing class variable $\alpha_{class(T_u(v,k))}$, we require that the variable $\alpha_{class(T_u(v,k))}$ be active before the variable $\alpha_{class(T_u(v,k+1))}$. This is achieved by adding the following constraint

$$\alpha_{class(T_u(v,k))} \geq \alpha_{class(T_u(v,k+1))} \quad \forall v \in FO(u) \text{ and } 1 \leq k \leq J_v + w(e_{uv}) - 1$$

For every multi-fanout gate u we also define a integer variable $\rho_u \leq 0$, which models the forward retiming. This is required because unlike backward retiming, FF's introduced at the fanouts by forward retiming across gate u can be unconditionally shared since all of them have the same logic value. Thus the α variables model the backward retiming and ρ models the forward retimings. Notice that this is different from the unconditional sharing model of Section 2.1.4 where the mirror variable $r(m_u)$ modeled FF's moved by both forward and backward retimings since all FF's could be unconditionally shared. Requiring the α variables to be nonnegative, $\alpha_i \geq 0$ ensures that they model only forward (positive) retiming moves, while the condition $\rho_u \leq 0$, ensures that ρ_u models only backward (negative) retiming moves.

[2]Sharing classes for circuits with different types of FF's can be defined similarly.

4.3.1 Modifications in the Objective Function

To model the conditional sharing represented by the sharing classes, the objective function term for a gate u that has a conflict at its fanouts is modified to

$$(|FI'(u)| - 1) \cdot r(u) + \rho_u + \sum_{i \in N_u} \alpha_i \tag{4.2}$$

This expression counts the number of FF's that settle at the output of gate u after retiming and the significance of each term is as follows:

- The first term $(|FI'(u)| - 1) \cdot r(u)$ in Equation (4.2) models the increase in the number of FF's when gate u is retimed by one unit, and is similar to the model in [59]. As earlier $FI'(u)$ is the set of fanins that have only a single output, i.e., $FI'(u) = \{v | v \in FI(u)\ AND\ |FO(v)| = 1\}$. It assumes a shared cost of one at the fanouts of gate u for any set of FF's retimed in either direction across gate u. In forward retiming, all FF's inserted at the outputs of a gate have the same logic values, and therefore the shared cost at fanouts of gate u in forward retiming is one. Since a gate can be retimed backwards only if all FF's at its output have the same logic values, the shared cost at the outputs before retiming is also one, as modeled by this term. The bound $r(u) \leq J_u$, ensures that no set of FF's, with shared cost greater than one, is ever retimed backwards across gate u.

- The second term $\rho_u \leq 0$ is a correction factor applied to correctly model the situation in which a set of FF's moves forward across gate u and all its fanouts. It is active only during forward retiming steps, and models the number of FF's moved across the fanout junction of gate u by forward retiming. Since a negative value of ρ_u denotes forward retiming, it reflects a cost saving in the objective function.

- As mentioned earlier $\alpha_i = 1$ implies that the sharing class S_i is active, therefore $\sum_{\forall\ i \in N_u} \alpha_i$ denotes the number of active sharing classes at the fanouts of gate u. Since each active sharing class requires one FF, the number of active sharing classes is also the number of physical FF's required at the fanouts of gate u. The minimization of the objective function will force the maximal sharing at the outputs of gate u. The first FF in a sharing class S_i that arrives at the fanout junction activates the sharing class variable α_i, incurring a cost of one in the objective function. The remaining FF's in that sharing class can then arrive without incurring any extra cost in the objective function.

4.3.2 Additional Constraints

The number of FF's between gate u and its fanout v is given by $w_r(e_{uv}) = w(e_{uv}) + r(v) - r(u)$. The cost of the FF's between u and v is given by $\sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))}$, out of which $r(u)$ FF's are removed by backward retiming across gate u and $-\rho_u$ FF's are removed by

forward (negative) retiming across the fanouts. The conditional sharing of FF's is automatically modeled by the sharing of the α variables amongst the fanouts. Since the cost of FF's should be same as the number of actual FF's, we get

$$w(e_{uv}) + r(v) - r(u) = \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \; - r(u) + \rho_u \;\; \forall\, v \in FO(u) \tag{4.3}$$

which can be rewritten as

$$w(e_{uv}) + r(v) = \rho_u + \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \;\; \forall\, v \in FO(u) \tag{4.4}$$

Since the right hand side of Equation (4.4) is being minimized in the objective function, we can relax the equality to the following inequality

$$w(e_{uv}) + r(v) \leq \rho_u + \sum_{k=1}^{J_v + w(e_{uv})} \alpha_{class(T_u(v,k))} \;\; \forall\, v \in FO(u) \tag{4.5}$$

4.3.3 An Example

Consider the circuit with the sharing classes in its table of logic values, as shown in Figure 4.12. The MILP for this circuit is

$$Minimize : -r(b) - r(c) - r(d) + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_6 + \rho_a$$

$$\text{subject to} \quad r(b) \leq \alpha_1 + \alpha_2 + \alpha_3 + \rho_a$$

$$r(c) \leq \alpha_1 + \alpha_4 + \rho_a$$

$$r(d) \leq \alpha_5 + \alpha_6 + \rho_a$$

$$\alpha_1 \geq \alpha_2 \geq \alpha_3$$

$$\alpha_1 \geq \alpha_4 \; ; \; \alpha_5 \geq \alpha_6$$

$$\rho_a \leq 0 \; ; \; \alpha_i \in \{0, 1\} \;\; \forall i$$

Backward Retiming: Suppose we want to model the sharing for $r(a) = 0$, $r(b) = 3$, $r(c) = 1$ and $r(d) = 2$. Then the optimal objective function value of the above LP is -1, which gives the correct increase in the number of FF's from the original circuit in Figure 4.13(a) to the retimed circuit in Figure 4.13(b).

Forward retiming: Now suppose we want to model the sharing for $r(a) = -2$, $r(b) = -2$, $r(c) = -1$ and $r(d) = -1$. Then the optimal objective function value is 3, which is the increase in the number of FF's from the original circuit in Figure 4.14(a) to the retimed circuit in Figure 4.14(b). As can be seen one FF is shared for the edges e_{ac} and e_{ad} even though they where not in the same sharing class. This is possible because the FF's moved forward to the outputs of gate a hence they all have same logic value without regard to the sharing class which are defined for backward movements. Thus these FF's can be shared and our formulation correctly models the cost.

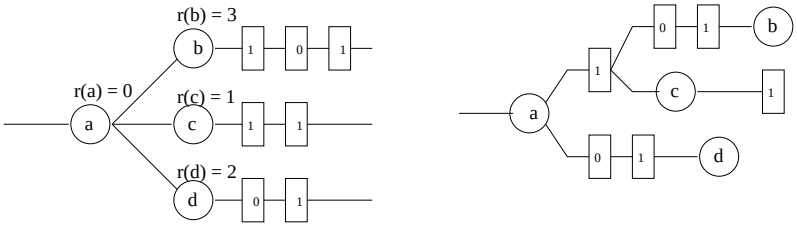

Figure 4.13 Example of positive retiming

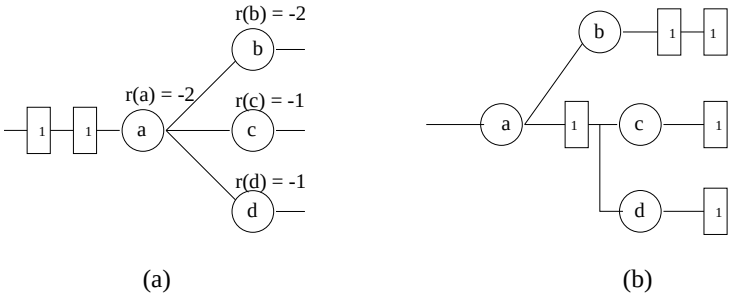

Figure 4.14 Example of negative retiming

4.3.4 FF Sharing with Don't Cares

Since every nonunique justification decision generates an X, the actual problem of FF sharing is to find the optimal sharing between 0, 1 and X, FF's. The logic value X can be shared with either 0 or 1 and hence presents additional modeling problems. Particularly difficult are the cases when X's are followed by 0's and 1's in a sequence, since the choice of merging X with 0 or 1 influences sharing of the remaining sequence.

As an example consider the following values at the output of a gate.

0	1
1	X
1	0

The X can only be shared with 0 and hence should be converted to 0. However it is not always possible to uniquely determine the possible values a X may take. Consider the following values at the output of a gate.

Here the X may be converted to either 0 or 1. If all 6 FF's arrive, then it is beneficial to merge X with 0; however, if only the first X and 1 arrive then X should be merged with 1. The values in the table show the FF's that can potentially arrive at this junction by reverse retiming, but not all of these FF's are required to arrive at the junction. In fact, due to the constraints and the objective function any combination of these FF's may arrive, and this makes it difficult to model the sharing in presence of X's. To avoid this problem, we convert all X's to either

0	1
X	1
1	0

0 or 1. If the X can be shared with only 0 (or a 1) then it is converted to a 0 (or a 1). For optimal solution both possibilities need to be explored, however, in our implementation we use a random assignment to convert X's to 0 or 1.

4.4 Experimental Results

We implemented a initial state minarea retiming based on the presentation in this chapter. Since obtaining an optimal solution requires complete exploration of the problem, it implies generating all the possible justification sets Δ_i's, and solving the corresponding LP's. Since this is a NP-hard problem, it is not likely to be computationally feasible. Hence, we implement a justification algorithm that makes random choices whenever there is a non-unique justification. The LP is then solved for the corresponding Λ_i. If the lower bound Γ is not achieved, then we perform another random decision based justification. This process is repeated until either the lower bound is reached or a user specified number of iterations are performed, and the best solution found is reported. Although it may seem arbitrary to use random decisions, our experimental results show that the algorithm gives us good engineering solutions that are close to the (possibly unachievable) lower bound. Other possible stopping criteria could be (a) having the best result obtained so far be within a given percentage of the lower bound, or (b) obtaining no improvements in the best solution for a given number of iterations, etc. If there are no gates with conflicts, then the LP is the dual of a network flow mincost flow problem, and is solved using a network simplex algorithm of Section 3.3.3. If, however, we have to solve the general MILP we use the public domain MILP solver, *lp_solve* [6].

We present results on the ISCAS89 [7] benchmark suite in Table 4.1. For each circuit, we show the number of gates $|G|$, the target clock period P, and the lower bound on the number of FF's obtained by Minaret Γ. We also show the minimum number of FF's obtained with equivalent initial state and the execution time T_{exec} (in hours:minutes:seeconds) for all the tasks including solving the LP for all iteration on a HP 9000/777 C110 workstation. In the absence of initial state values for the benchmark circuits, we present results for four different initial state assignment. Case A has all FF's initialed to 0, while case B has all initialized to 1, case C and D are for random state assignments. As can be seen from the results, for many circuits the lower bound is achieved in a small number of iterations for almost any initial state. In fact, in almost all of these cases the lower bound Γ is obtained in the first iteration itself. For some circuits the lower bound was not reached. This, however, does not imply that the solution obtained is not optimal since the lower bound is not always achievable with equivalent initial state. For these circuits, we report the best solution obtained in 50 (5 for s15850.1) iterations.

Table 4.1 Minarea Initial State Retiming

Circuit	$\|G\|$	P	Γ	A		B		C		D	
				#FF	T_{exec}	#FF	T_{exec}	#FF	T_{exec}	#FF	T_{exec}
s27	11	6.0	3	3	0.01s	3	0.00s	3	0.01s	3	0.00s
s208.1	105	10.0	8	8	0.02s	8	0.02s	8	0.03s	8	0.03
s298	120	6.0	22	22	0.40s	22	0.44s	22	0.44s	22	0.44s
s382	159	7.0	23	23	2.59s	23	4.34s	23	4.35s	23	4.45s
s386	169	11.0	6	6	0.04s	6	0.03s	6	0.04s	6	0.03s
s344	161	14.0	19	19	1.77s	19	1.79s	19	1.77s	19	1.82s
s349	162	14.0	19	19	1.62s	19	1.62s	19	1.69s	19	1.62s
s526n	195	6.0	30	30	0.95s	30	0.95s	30	2.75s	30	0.97s
s510	212	11.0	7	7	0.12s	7	0.12s	7	012s	7	0.12s
s420.1	219	12.0	17	17	0.07s	17	0.06s	17	0.07s	17	0.06s
s641	380	74.0	19	19	0.11s	19	0.43s	19	0.44s	19	0.43s
s713	394	74	19	19	0.18s	19	0.68s	19	0.68s	19	0.69s
s967	395	12.0	35	35	28.52s	35	27.21s	35	28.05s	35	27.27s
s938	447	16.0	33	33	1.45s	33	1.53s	33	1.49s	33	1.53s
s1196	530	24.0	18	18	0.08s	18	0.07s	18	0.08s	18	0.17s
s1238	5.09	22.0	18	18	0.08s	18	0.07s	18	0.56s	18	0.08s
s1423	658	53.0	76	76	8.77s	76	9.23s	76	8.89s	76	9.31s
s1488	654	16.0	7	7	0.11s	7	0.11s	7	0.12s	7	0.11s
s1494	648	16.0	7	7	0.13s	7	0.12s	7	0.13s	7	0.12s
s3330	1790	14.0	110	110	0.58s	110	0.56s	110	0.59s	110	0.56s
s5378	2780	21.0	173	173	3:18s	173	3:19s	173	3:18s	173	3:17s
s9234.1	3271	38.0	134	134	21:18s	134	21:19s	134	23:47s	134	21:15s
s635	287	66.0	35	42	22.6s	42	22.38s	35	1.08s	39	22.5s
s953	396	13.0	27	32	32:02s	32	27:35s	32	31:30s	32	27:2s
s1269	570	19.0	84	84	0.26s	85	1:33s	85	1:31s	85	1:4s
s1512	781	23.0	70	71	1:51:19s	72	1:52s 1s	72	1:56:23s	70	1:38s
s3271	1573	15.0	168	169	16:46s	173	16:5s	170	16:40s	173	16:29s
prolog	1602	13.0	122	124	16:40s	125	16:42s	125	16:39s	125	16:29s
s3384	1686	27.0	167	168	55:42s	169	1h3:18s	169	1:2:56s	169	51:3s
s15850.1	9618	63.0	525	544	3:9:56s	540	4:2:36s	542	3:57:7s	544	3:59:5s

The execution time of our method is considerably higher than the run times for conventional minarea reported for Minaret, since here we need to solve possibly multiple MILP's, unlike Minaret which needs to solve only one mincost flow problem. However note that in most cases where the lower bound is achieved the execution times are comparable to those of Minaret. In the circuits where the lower bound Γ is not achieved the solution reported by our algorithm is very close to Γ and therefore corresponds to a good engineering solution. Since the optimal number of FF's in a circuit depend on the initial state of the original circuit, some variation in the number of FF's and execution time is obtained for different initial states. For `s635, s1269 and s1512` the lower bound was seen to be achieved for only some initial states.

4.5 Conclusion

We have presented a method to obtain minarea retiming of control logic subject to a given target clock period and an equivalent initial state. Any minarea retiming algorithm, that does not consider initial states will, in general, not give a solution with a valid equivalent state and hence cannot be used for control logic, where initial states are important. Our method, on the other hand, will always result in a retimed circuit with an equivalent initial state, i.e., the retimed circuit starting in the equivalent initial state will have the same behavior as the original circuit starting in its given initial state. Unlike conventional minarea retiming algorithms our approach can be used for performance constrained, area optimization of control circuits. This approach also has applications in minarea retiming of circuits that contain different types of memory elements that can not be shared with each other, e.g. load enable registers.

We provide a simple way to incorporate the constraints for ensuring that the resulting retiming has an equivalent initial state. This is achieved by imposing upper bounds on the retiming variables so that any retiming respecting those bounds will have an equivalent initial state. This equivalent state can easily be found after the retiming by using the information stored from the justification phase. The technique also utilizes a new approach that incorporates conditional FF sharing, since the idea of mirror vertices used by Leiserson and Saxe to model unconditional FF sharing [59] cannot be extended to the initial state retiming problem. The solution approach searches the justification space for the initial states and for each possible justification, solves an LP. The exploration of the justification space can be stopped by the user at any time, and it was seen that for all circuits tested, good engineering solutions that were close to a (possibly unachievable) lower bound were found by the technique after a small amount of exploration.

Minarea initial state retiming can also be performed by extending the approach in [30]. In this method conventional minarea retiming is performed first. If a conflict occurs at a gate while moving the FF's to obtain this retiming, then an appropriate bound is placed on the retiming variable of this gate, and the minarea retiming problem is solved again. This procedure is repeated until no more conflicts are obtained. Thus the final circuit has an equivalent initial state although it may require more FF's than the conventional minarea, since the extra bounds placed on the retiming variables to ensure equivalent initial states can increase the number of FF's in the optimal solution. This method can be seen as a "dual" of our approach, since it starts from the lower bound and tries to achieve feasibility (equivalent initial state), while in our approach we start with a feasible solution and try to achieve optimality. However in this approach the initial state value on the FF's that can be possibly retimed to the fanouts of a gate is not known before solving the minarea LP. Hence this approach will not be able to model the conditional sharing, making the solution suboptimal.

The work in [122] showed that backward retiming with equivalent initial states such as the one in Figure 4.1 can always be obtained if the reset signal is expressed explicitly. This,

however, requires the addition of a multiplexer before the FF and thus changes the path delays in the circuit. This may cause the clock period of the circuit to increase and is therefore not considered here.

5 RETIMING LEVEL-CLOCKED CIRCUITS

5.1 Introduction

The memory elements in a circuit can be either edge-triggered, called *flip-flops (FF's)* or level-sensitive, called *latches*. Unlike a FF, a latch is transparent during the active period of the clock. Even though the transparent nature of latches makes design and analysis of level-clocked circuits (circuits with level-sensitive latches) very complex, they are widely used for high performance designs because they offer more flexibility both in terms of the minimum clock period achievable and the minimum number of memory elements required. Optimizing level-clocked circuits is therefore a complex but important task, and there is a acute need of good automation tools. Several efforts have been made to retime circuits with level-sensitive latches based on the Leiserson-Saxe approach, e.g., [96, 70]. Although these algorithms have polynomial time complexity, their high space and time requirement makes them incapable of handling circuits with even a few thousand gates, and the only published results are on circuits with less than 400 gates. Our goal in this chapter is to able to retime circuits with tens of thousands of gates in reasonable time, and we present results on circuits with up to 56,000 gates.

For edge-triggered circuits (circuits with edge-triggered FF's) the delays through all combinational logic paths must be less than the clock period, hence we must enforce timing constraints only between FF's connected by a purely combinational path. For level-clocked circuits the delay through a combinational logic path can be longer than one clock cycle, as long as it is compensated by shorter path delays in the subsequent cycles. To ensure that the extra delay is compensated we must enforce timing constraints between a latch and every other latch reachable from it (possibly through multiple latches). Consider a linear N stage pipeline with $N+1$ memory elements $(m_0, m_1 \ldots m_N)$. If these memory elements are edge-triggered FF's, then we need only N timing constraints (m_i - m_{i+1}, $0 \leq i \leq N$). However, if these memory elements are level sensitive latches, then we would need $N \cdot (N+1)/2$ timing constraints (m_i - m_j $\forall j > i$ and $0 \leq i \leq N$). In presence of feedback paths, timing analysis of level-clocked circuits becomes even more complex.

These traditional methods [96, 70] solve the minperiod retiming problem by performing a binary search over all possible clock periods. At each step of this binary search, the feasibility of achieving the clock period by retiming is checked by solving a single source shortest path

problem using the Bellman-Ford algorithm on a constraint graph. This constraint graph consists of $|G|$ vertices and edges between every pair of vertices (where $|G|$ is the number of gates in the circuit), and is obtained by solving a all-pairs shortest path problem on the original circuit graph. This graph has to be reconstructed for every binary search point, because as shown in [70, Section VI-A], unlike edge-triggered circuits, critical paths in level-clocked circuits can be different for different clock periods. Therefore the methods in [96] and [70] have $O(|G|^2)$ space requirement and high (although polynomial) time complexity. This complexity of retiming level-clocked circuits arises due to the transparent nature of latches, which forces us to consider constraints on paths going through multiple latches.

In this chapter we present a minperiod retiming algorithm that is capable of retiming very large multi-phase circuits with general clock schedules. This is achieved by introducing the concept of *Global Departure Time (GDT)* to map the minperiod retiming problem to a skew optimization problem and thus solving it much like the simpler problem of retiming edge-triggered circuit using the approach of [109]. In each step of the binary search we solve the single source shortest path problem on a much smaller constraint graph with only $|\Psi|$ vertices, where $|\Psi|$ is the number of latches in the circuit. This constraint graph contains edges only between latches that have a purely combinational path between them, and therefore is much smaller and sparse as compared to the constraint graph in traditional methods. Unlike the traditional methods that reconstruct the constraint graph for every binary search point, we perform a simple reweighting of the edges. Once the minimum period is obtained, the latches are relocated to obtain this minimum period.

The minarea retiming problem can be formulated as a linear program (LP) [59]. This LP is generated by solving an all-pair shortest path problem, and has $|G|$ variables and almost $\frac{|G|^2}{2}$ constraints. This LP can be solved efficiently by solving its min-cost flow dual [59]. For edge-triggered circuits, the work in [115] presented an efficient technique for pruning the number of constraints which also had the beneficial effect of reducing the computation involved in generating these constraints. This was achieved by utilizing the observation that in edge-triggered circuits, if a subpath satisfies the timing constraints, then any path containing this subpath will also satisfy the timing constraints (unfortunately this is not true for level-clocked circuits due to the transparent nature of latches). Section 3.3 builts on the idea and adds efficient techniques to obtain bounds on the variables of the LP for edge-triggered circuits. These bounds were used to further reduce the size of the LP and the time required to generate it.

The concept of GDT presented in this chapter makes it possible for us to apply similar techniques to generate bounds on the variables in the minarea LP for level-clocked circuits, and to use it to reduce the size of this LP. However, due to the transparent nature of latches, unlike edge-triggered circuits, the techniques of [115] and Minaret cannot be used to reduce the time required to generate the minarea LP in level-clocked circuits. This presents a major

hurdle in retiming large level-clocked circuits for minimum area, because in the absence of any efficiency-improving techniques, the minarea LP can not be generated in any reasonable time. In this chapter we present new techniques for pruning the minarea LP for level-clocked circuits, and reducing the time required to generate it. Using the techniques presented in this chapter, the entire ISCAS-89 benchmark suite could be retimed for minimum period in seconds, and for minimum area in minutes.

The remainder of this chapter is organized as follows. In Section 5.2, we present some background material, after which in Section 5.3, we discuss a relation between retiming and clock skew optimization for level-clocked circuits. This relation is then utilized for efficient minimum period and minimum area retiming in Section 5.4 and Section 5.5 respectively. Experimental results are presented in Section 5.6, followed by concluding remarks in Section 5.7.

5.2 Background

Consider the simple circuit in Figure 5.1 with unit delay gates and a single-phase clocking scheme with 50% duty cycle. In this thesis we will assume that the data signals are available at the primary inputs at the falling edge of the clock, and must arrive at the primary outputs before the falling edge. For any latch that is not a primary input or primary output, the data may depart at any time during the active period of the clock. Under this assumption a data signal in this circuit gets exactly two clock periods to reach the primary output from the primary input.

A clock period of 2.0 units is not feasible for the circuit in Figure 5.1. This is because as shown in the figure the actual arrival time (3.0 units) is one time unit later than the required arrival time (2.0 units). Hence the minimum clock period at which this circuit can operate without any modifications is 3.0 units. However, a clock period of 2.0 units can be achieved by moving the latch L1 across the gate G3. Notice that this is not the only possible location of memory element L1 that can achieve the clock period of 2.0 units; placing latch L1 at the output of gate G1 also achieves the same clock period as shown in Figure 5.3. This is possible because of the transparent nature of the latches which allows the data signal to depart from the latch at any time during the active period of the clock.

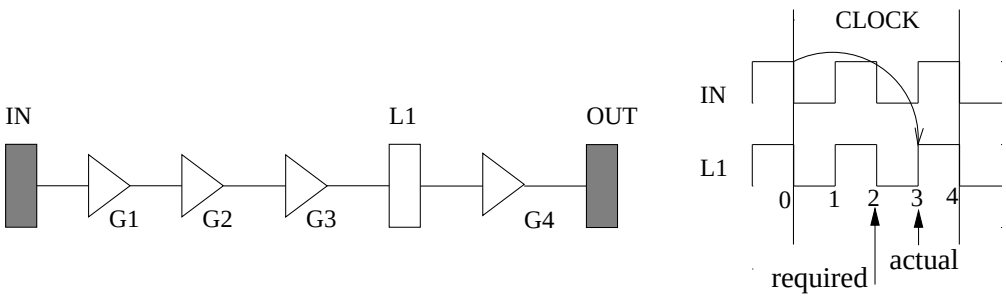

Figure 5.1 An example circuit.

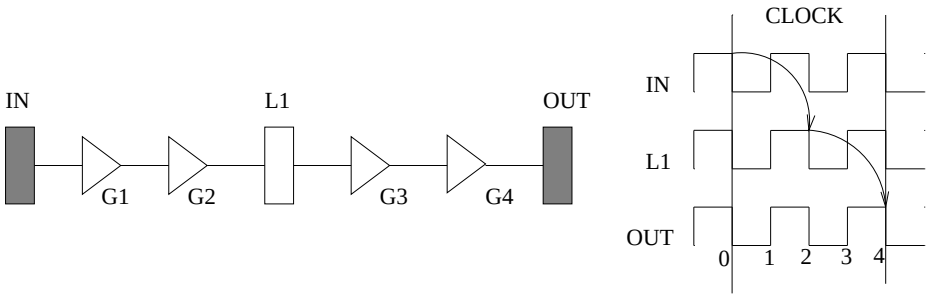

Figure 5.2 Retiming for clock period optimization.

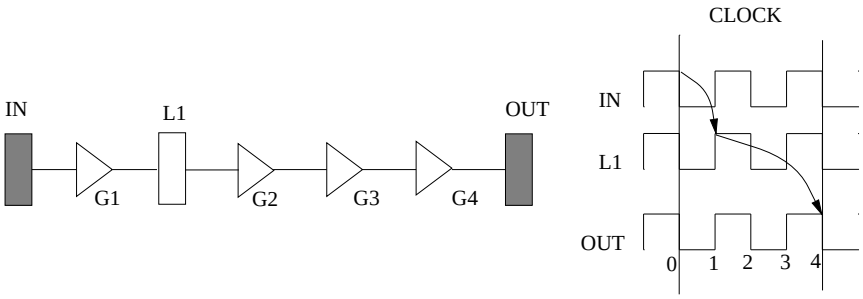

Figure 5.3 Alternate retiming for clock period optimization.

We use the term *right* to denote the direction of the signal flow and *left* to indicate the direction against the signal flow. Thus retiming a latch by moving it to the right across a gate implies removing a latch from each of the fanins of that gate and adding one to all of the fanouts of that gate. Similarly retiming a latch left across a gate implies removing a latch from each of its fanouts and adding one to each of the fanins. The set of latches in the given circuit is denoted by Ψ.

5.2.1 Clock Model

In this chapter, we have adopted the clock model of Sakallah, Mudge and Olukotun [107], and we describe it here for completeness. A k-phase clock is a set of k periodic signals, $\Phi = \{\phi_1 \ldots \phi_k\}$ where ϕ_i is referred to as phase i of the clock Φ. All of the ϕ_i's have the same clock period T_Φ, and each phase i has an active interval of duration T_{ϕ_i} and a passive interval of duration $(T_\Phi - T_{\phi_i})$. Each latch $i \in \Psi$ is clocked by exactly one phase of the clock Φ, which is denoted by $p(i)$. The latches controlled by a clock phase are enabled during the active interval and disabled during the passive interval. When the clock period, T_Φ, is changed, the active intervals of each phase are scaled proportionately. The term "clocking scheme" is used to indicate the relative ratios and duty cycles of the individual phases. Thus a clocking scheme together with a clock period T_Φ, define a "clock schedule" Φ.

Associated with each phase i is a *local time zone*, shown in Figure 5.4, such that the passive interval starts at time 0, the enabling edge occurs at time $(T_\Phi - T_{\phi_i})$, and the latching edge

occurs at time T_Φ. Phases are ordered so that $e_1 \leq e_2 \ldots \leq e_{k-1} \leq e_k = T_\Phi$, and are numbered modulo-k, i.e., $\phi_{k+1} = \phi_1$ and $\phi_{1-1} = \phi_k$. There is also a global time reference and e_i denotes the time when the phase ϕ_i ends, relative to this global time reference.

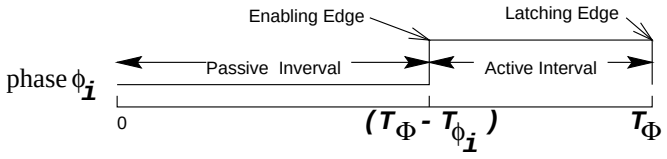

Figure 5.4 Phase i of a k-phase clock (all times in local time zone).

A phase shift operator $E_{i,j}$, shown in Figure 5.5, is defined as follows:

$$E_{i,j} = \begin{cases} (e_j - e_i) & \text{for } i < j \\ (T_\Phi + e_j - e_i) & \text{for } i \geq j \end{cases} \tag{5.1}$$

Note that $E_{i,j}$ takes on positive values, and when subtracted from a time point in the current time zone of ϕ_i, it changes the frame of reference to the next local time zone of ϕ_j.

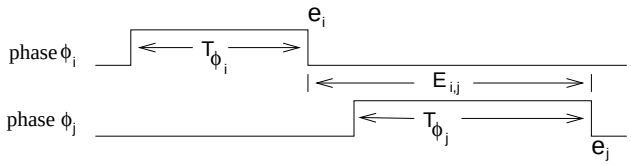

Figure 5.5 The phase shift operator.

5.2.2 Timing Constraints for Level-Clocked Circuits

We now enumerate the set of timing constraints, that dictate the correct operation of a level-clocked circuits. We neglect to consider latch setup and hold times here, since they can be incorporated easily by including the setup times in the path delays and the hold time in the clock periods.

Each latch i also has an associated latest arrival time A_i, and a latest departure time D_i, in its local time zone. Due to the transparent nature of the latches, a signal can depart from a latch i any time during the active interval of the phase $p(i)$, i.e.,

$$T_\Phi - T_{\phi_{p(i)}} \leq D_i \leq T_\Phi$$

However, a signal cannot depart from a latch before it has arrived at that latch, i.e.,

$$A_i \leq D_i$$

The arrival time at a latch j of a signal departing from another latch i connected by a purely combinational path (denoted as $i \hookrightarrow j$) of delay d_{ij} must satisfy the following relation

$$D_i + d_{ij} - E_{p(i),p(j)}) \leq A_j$$

Combining the above relations we can obtain the timing constraints for properly clocking of level clocked circuits, considering only long path constraints[1] as

$$\begin{aligned} D_i + d_{ij} - E_{p(i),p(j)} &\leq D_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{p(i)} \leq D_i &\leq T_\Phi \quad \forall\, i \in \Psi \end{aligned} \tag{5.2}$$

5.3 Relation Between Retiming and Skew

Clock skew at any latch is defined as the time by which the clock is delayed in arriving at the latch, with respect to a fixed reference (the arrival time of the clock at the primary inputs). Clock skews have traditionally been considered to be a liability and various techniques to get a skew-free clocking network have been proposed [131, 14, 24]. An alternative approach views clock skews as a manageable resource rather than a liability, and intentionally introduces skews to improve the performance of the circuit [31]. Consider the circuit in Figure 5.1 where the clock period of 2.0 units is not feasible since the actual arrival time (3.0 units) is one time unit after the required arrival time (2.0 units). However, as shown in Figure 5.6 if a skew of +1.0 unit is applied to the clock at latch L1, the required arrival time at latch L1 becomes 3.0 time units, and the data is properly clocked at latch L1. The circuit can now run with a clock period of 2.0 units. Thus clock skews can be used to improve the performance of a circuit.

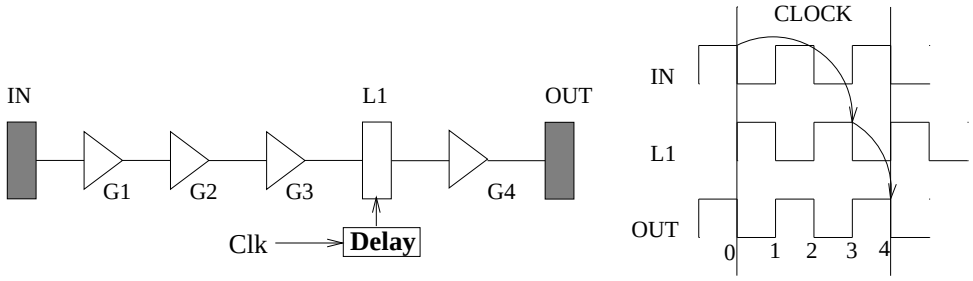

Figure 5.6 Using clock skew to reduce clock period.

To derive timing constraints in presence of skews we now augment the Sakallah-Mudge-Olukotun model with our own notation. We associate a skew S_i with every latch $i \in \Psi$. Note that the skew values here are not physical skews to be applied to the final circuit, but conceptual ideas that will eventually help us to achieve a retiming solution. Therefore no restrictions are placed on the value of S_i, i.e. $-\infty \leq S_i \leq \infty$.

We define a *latch shift* operator $L_{i,j}$, shown in Figure 5.7, much like the phase shift operator. This operator converts time from the local time zone of latch i to the local time zone of latch

[1] We do not consider short path constraints here, and rely on Theorem 1 in [70], which assures us that for valid clock schedules [70], there will be no short path violations. In this thesis we consider only valid clock schedules.

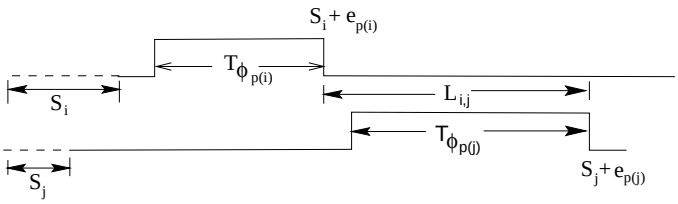

Figure 5.7 The latch shift operator.

j, taking into account their skews. It is defined as

$$L_{i,j} = \begin{cases} (S_j + e_{p(j)}) - (S_i + e_{p(i)}) & \text{for } i < j \\ T_\Phi + (S_j + e_{p(j)}) - (S_i + e_{p(i)}) & \text{for } i \geq j \end{cases}$$

which can be rewritten in terms of the phase shift operator as

$$L_{i,j} = (S_j - S_i) + E_{p(i),p(j)} \tag{5.3}$$

In presence of skews at latches, the timing constraints in relation (5.2), must be modified by using the latch shift operator instead of the phase shift operator. Thus the timing constraints for a level clocked circuit to be properly clocked by a clock schedule Φ, in presence of skews are

$$\begin{aligned} D_i + d_{ij} - L_{i,j} &\leq D_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{p(i)} \leq D_i &\leq T_\Phi \quad \forall\, i \in \Psi \end{aligned}$$

These timing constraints can be rewritten as

$$\begin{aligned} (S_i + D_i) + d_{ij} - E_{p(i),p(j)} &\leq (S_j + D_j) \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ T_\Phi - T_{p(i)} &\leq D_i \leq T_\Phi \quad \forall\, i \in \Psi \\ -\infty &\leq S_i \leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

To make the discussion simpler we subtract T_Φ from both sides of the first relation, and substitute

$$X_i = (S_i + D_i - T_\Phi) \tag{5.4}$$

We refer to X_i as the *Global Departure Time* (GDT). This gives us

$$\begin{aligned} X_i + d_{ij} - E_{p(i),P(j)} &\leq X_j \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ -\infty \leq X_i &\leq \infty \quad \forall\, i \in \Psi \end{aligned}$$

These can be written as the following set of difference constraints.

$$\begin{aligned} X_i - X_j &\leq E_{p(i),p(j)} - d_{ij} \quad \forall\, i \hookrightarrow j \mid i,j \in \Psi \\ -\infty \leq X_i &\leq \infty \quad \forall\, i \in \Psi \end{aligned} \tag{5.5}$$

As shown earlier, both skew and retiming can modify the circuit in Figure 5.1 to operate at a faster clock period of 2.0 units. In fact, both achieve this objective by the same basic principle of borrowing time from one cycle and lending it to another. Therefore retiming and skew optimization can be considered to be related to each other. A formal presentation of this relationship is given in [109], for edge triggered FF's. A FF can be conceptualized as a level sensitive latch with a very small active interval.

The physical meaning of GDT is as follows. If we can apply arbitrary skews at latches, we can adjust the skew, S_i, of a latch so as to force $D_i = T_\Phi$, which is same as a negative edge triggered FF. Since $X_i = S_i + D_i - T_\Phi$, setting $D_i = T_\Phi$ gives $S_i = X_i$. Hence, we can look at X_i for latches in the same way as we look at skews for FF's.

The difference constraint between GDT values of two latches given in relation (5.5) is similar to the difference constraints between skews at FF's in [109]. Therefore we suggest a relation between retiming and GDT values of level-sensitive latches, similar to the one given in [109] between retiming and skews for edge triggered FF's. This relation will allow us to develop efficient techniques for retiming level-clocked circuits. We now state a theorem similar to Theorem 1 in [109]; the proof of this theorem is similar to the one given in [109].

Theorem 2 *In a level-clocked circuit, retiming a latch by moving it to the right [left] across a gate with delay d_1 is equivalent to increasing [decreasing] its GDT by d_1.*

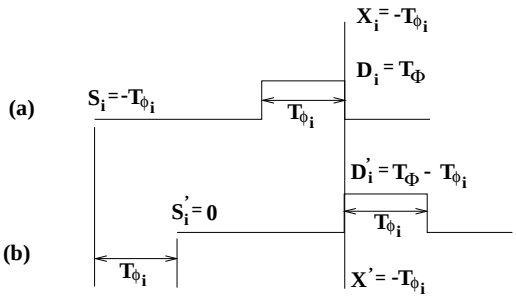

Figure 5.8 The ability of a latch to absorb some skew.

Note that in reality, we are not compelled to set $D_i = T_\Phi$, and that we can reduce D_i by as much as T_{ϕ_i} and increase S_i by the same amount, while keeping X_i constant. Since only GDT's (X_i's) appear in the timing constraint of relation (5.5), keeping them constant keeps the clock period constant. Consider Figure 5.8 (a) where $S_i = -T_{\phi_i}$ and $D_i = T_\Phi$, thus $X_i = -T_{\phi_i}$. We can increase the skew to zero ($S'_i = 0$), without changing the GDT as shown in Figure 5.8 (b), by reducing the departure time by the same amount $D'_i = T_\Phi - T_{\phi_i}$, leaving the GDT unchanged ($X'_i = X_i = -T_{\phi_i}$). Therefore, we can absorb a skew of up to $-T_{\phi_i}$ in the D_i without violating the long path constraint. Thus a GDT value between $-T_{\phi_i}$ and 0 is allowed and this range will be referred to as the allowable range. If different phases have different active interval then this allowable GDT range of a latch will depend on its phase. Therefore

in our model, level-sensitive latches can be conceptualized as FF's that have the capacity to absorb some skew.

At this time, we also note the relation between the GDT, X_i, of a latch i and the corresponding minimum magnitude skew, S_i:

$$S_i = \begin{cases} X_i & \text{if } X_i \geq 0 \\ 0 & \text{if } -T_{\phi_i} \leq X_i \leq 0 \\ X_i + T_{\phi_i} & \text{if } -T_{\phi_i} > X_i \end{cases} \tag{5.6}$$

5.4 Minimum Period Retiming

Given a circuit and a clocking scheme, minimum period (minperiod) retiming finds the minimum possible clock period T_Φ, for which there exists a retimed circuit that will be properly clocked by Φ (the clock schedule for the given clocking scheme and clock period T_Φ), and the retiming that makes this clock schedule possible. As mentioned in Section 5.1, the traditional techniques of [96, 70] are unable to handle large level-clocked circuits in a reasonable time. We utilize the relationship between GDT and retiming presented in Section 5.3 to map the problem of retiming level-clocked circuits for minimum period to the simpler problem of retiming edge-triggered circuits for minimum period as solved in [109]. This mapping motivates the following two-phase method of retiming for minimum clock period under a given clocking scheme.

Phase A: Find the minimum clock period and a set of GDT values that will achieve this period.

Phase B: Relocate latches across gates to get all the GDT values to be within allowable range.

As mentioned later in Section 5.4.1, in Phase A of this method we construct a small and sparse graph only once, unlike the traditional methods [96, 70] which construct multiple large and dense graphs. In Phase B we perform fast local transforms to obtain the retiming solution. Therefore using this two phase method we can retime large level-clocked circuits very efficiently.

As in [109] it must be noted that since gate delays take on discrete values, it cannot be guaranteed that the GDT at every latch can be reduced to be within the allowable range through retiming operations. After the GDT values have been reduced as much as possible, the retimed circuit may be implemented either by applying the requisite (remaining) skews at a latch (to get the optimal clock period achievable by skew optimization), or by setting all skews to zero to get a clock period that is, as will be shown in Section 5.4.4, no more than a fixed bound above the optimal period with skews. Note, however, that this is not necessarily suboptimal since the minimum clock period using skews may not be achievable using retiming alone, since retiming allows cycle-borrowing only in discrete amounts (corresponding to gate delays), while skew is a continuous cycle-borrowing optimization [31]. As will be shown in

Section 5.4.4, if the maximum gate delay is less than the least T_{ϕ_i}, we can always achieve the optimal skew optimization period by retiming alone.

We first describe the two phases of minperiod retiming, followed by the special case of retiming a circuit for a given clock period. We then present the bound on the difference between the optimal skew optimization period and the clock period obtained by our method.

5.4.1 Phase A: Clock Period Optimization

The problem of minimizing the clock period, T_Φ, for a given clocking scheme can be represented as the following linear program:

$$\begin{aligned} &\text{minimize} && T_\Phi \\ &\text{subject to} && X_i - X_j \leq E_{p(i),p(j)} - d_{ij} \quad i \hookrightarrow j \mid i,j \in \Psi \end{aligned} \tag{5.7}$$

Solving the above linear program we obtain the minimum clock period and the GDT's correspond to it. Our strategy is to transform the GDT solution to a retiming solution to achieve the minimum clock period.

For a given circuit, d_{ij} is constant and for a given clock schedule that has a fixed T_Φ, $E_{p(i),p(j)}$ is constant. Therefore, the constraint matrix reduces to a system of difference constraints. A feasible solution to the constraint matrix exists if the corresponding constraint graph contains no positive cycles [17]. Thus we can solve the, linear program by performing a binary search on the clock period T_Φ. The minimum clock period corresponds to the smallest value of T_Φ at which no positive cycle exists.

The constraint graph has a vertex for each latch in the circuit and one for the host node representing the primary inputs and outputs. There is a edge (i,j) from vertex i to vertex j if there is a purely combinational path from latch i to latch j. The weight on this edge is a function of the clock period T_Φ and is given by $d_{ij} - E_{p(i),p(j)}$. The Bellman Ford algorithm [17] is applied as in [109] using the same speedup techniques which provide a fast implementation. The GDT's at all primary inputs and primary outputs are assumed to be zero. The values of d_{ij}'s are obtained efficiently by using the method in [108].

Notice that number of variables in this constraint graph is equal to the number of latches $|\Psi|$ in the circuit, and the constraints are only between latches with a purely combinational path between them. Therefore this constraint graph is much smaller and sparse as compared to the traditional constraint graphs of [96, 70], which have one variable for every gate and constraints to all reachable gates. Further unlike the traditional methods of [96, 70], which need to construct the larger and denser constraint graph for every binary search point (by solving an all-pair shortest path problem), our constraint graph needs to be constructed only once. At each point in the binary search the constraint graph can be obtained by a simple reweighting of the graph edges. Therefore the complexity of this binary search is much less than that of the traditional methods.

This optimal clock period with skews is called P_s, and it is same as the maximum delay-to-register ratio of [93]. Both are lower bounds on clock period obtainable via retiming. However, instead of using it, just as a lower bound (as in [93]), we use it to obtain the amount by which each latch is required to move in order to satisfy the clock period. This amount is then used to obtain a retiming solution as described next.

5.4.2 Phase B: GDT Minimization

In Phase B, we reduce the GDT values obtained in Phase A by applying retiming transformations using Theorem 2. This procedure relocates the latches with nonzero GDT's across logic gates, while maintaining the optimal clock period previously found. Because of the freedom provided to D_i by the active interval of clock phase $p(i)$ (which allows D_i to be set to any value between $T_\Phi - T_{\phi_{p(i)}}$ and T_Φ), $S_i = 0$ can be achieved if $-T_{\phi_{p(i)}} \leq X_i \leq 0$. If S_i cannot be set to zero, we try to bring X_i as close to 0 or $-T_{\phi_{p(i)}}$ as possible so as to minimize the magnitude of the final skew S_i.

Although the method for FF relocation presented in [109] can be modified to work for latches, we present a equivalent yet conceptually simpler method of GDT minimization by latch relocation. A gate can be retimed in forward [backward] direction if it has latches on all of its inputs [outputs], this retiming will result in removing one latch from all its inputs [outputs] and adding one latch to all its outputs [inputs].

We maintain two sets one for the gates that are to be forward retimed and one for the gates that are to be backward retimed. The forward retiming set F is initialized to contain all gates that have at least one latch on all their inputs. Similarly the backward retiming set B is initialized with gates that have at least one latch on all their outputs. We than process these sets by taking a gate from the set and retiming it, if the skew on the latches can be reduced by this retiming. After every retiming the sets are updated. The pseudo code for this is given below as the function `retime()` below:

```
retime()
{
  F = {v|v ∈ V and w(e_uv) ≥ 1 ∀u ∈ FI(v)} /* initialize F */
  B = {v|v ∈ V and w(e_vu) ≥ 1 ∀u ∈ FO(v)} /* initialize B */
  while(∃ u ∈ F) do forward_retime(u, F);
  while(∃ u ∈ B) do backward_retime(u, B);
}
```

The functions `forward_retime(gate,set)` and `backward_retime(gate,set)` retime the gate if needed, and update the respective sets, their pseudo code is given below.

`forward_retime(`v, F`)`

```
{
  F ← F − v; /* remove gate v from F */
  X_i = maximum GDT at the inputs of gate v;
  X'_i = X_i + d(v);
  S_i = GDT_to_skew(X_i);
  S'_i = GDT_to_skew(X'_i);
  if (|S'_i| < |S_i|) do
  { /* retime gate v */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) − 1};
        /* delete a latch from each input */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) + 1};
        /* add a latch with GDT = X'_i on all outputs */
    for ∀u ∈ FO(v) do
        if (w(e_wu) ≥ 1 ∀w ∈ FI(u)) do F ← F ∪ u; /* update F */
  }
}

backward_retime(v, B)
{
  B ← B − v; /* remove gate v from B */
  X_i = minimum GDT at the outputs of gate v;
  X'_i = X_i − d(v);
  S_i = GDT_to_skew(X_i);
  S'_i = GDT_to_skew(X'_i);
  if (|S'_i| < |S_i|) do
  { /* retime gate v */
    for ∀u ∈ FO(v) do {w(e_vu) ← w(e_vu) − 1};
        /* delete a latch from each output */
    for ∀u ∈ FI(v) do {w(e_uv) ← w(e_uv) + 1};
        /* add a latch with GDT = X'_i on all inputs */
    for ∀u ∈ FI(v) do
        if (w(e_uw) ≥ 1 ∀w ∈ FO(u)) do B ← B ∪ u; /* update B */
  }
}
```

The function `GDT_to_skew(GDT)` converts a GDT value to the corresponding minimum magnitude skew using relation (5.6). For forward retiming of a gate v the effective GDT before retiming, X_i is given by the maximum GDT at its inputs, while the effective GDT after

retiming X_i' is given by $X_i' = X_i + d(v)$. For backward retiming the effective GDT before retiming X_i is given by the minimum GDT at its outputs, and the GDT after retiming X_i' is given by $X_i' = X_i - d(u)$. In either case the gate is retimed only if magnitude of the effective skew after retiming S_i' is less than the magnitude of the effective skew before retiming S_i. As mentioned earlier a gate v is forward [backward] retimed by removing a latch from each of its inputs [outputs] and adding a latch with GDT X_i' to all its outputs [inputs]. If after forward [backward] retiming a gate v, any of its fanout [fanin] gate w now has at least one latch on all its fanins [fanouts], then we add gate w to the forward [backward] set F [B].

Retiming a latch forward across a gate u affects the edge weights on only its own fanouts and not the edge weights on fanouts of any other gate. Therefore forward retiming a gate u cannot enable the backward retiming of any other gate that could not be previously retimed in the backward direction. Since we forward retime a gate u only if the effective skew magnitude reduces by this retiming, and not if it remains the same, a gate u cannot be backward retimed after it has been forward retimed once (even though it may have latches on all its fanouts), because this backward retiming will increase the skew magnitude. Therefore a gate can never be retimed in both the forward and backward direction. Thus forward retimings have no effect on backward retimings and both types of retimings can be carried out independently. Due to this reason we do have to process the forward set again after it has been processed once.

5.4.3 Retiming for a Target Period

Retiming a circuit for a given target clock period is a special case of the minperiod retiming problem. In this problem we are given a circuit and a clock schedule Φ that has a fixed T_Φ. If the given clock schedule is feasible, the method should return a retimed circuit that is properly clocked. If the clock schedule is not feasible the method should indicate so. For this problem we do not need to perform the binary search in Phase A. The constraint graph is constructed as earlier and the Bellman-Ford algorithm is applied on it to obtain the set of required GDT's. If the Bellman-Ford algorithm detects a positive cycle the clock scheme is not feasible, and is reported as such, otherwise Phase B is performed.

Due to the flexibility in the non-critical part of the circuit, and the transparent nature of the latches, retiming for a given clock period is not unique, and different retimed circuits can be obtained all of which satisfy the target clock period. As an example for the circuit in Figure 5.1, two different retimings are shown in Figure 5.2 and Figure 5.3 for the same target clock period of 2.0 units. Our objective in minperiod retiming is to find one of these possible solutions efficiently, with as few retiming moves as possible. As in [109], we initialize the GDT's to 0 in the Bellman-Ford algorithm, and take advantage of the slacks to minimize the number of moves. For minperiod retiming of the circuit in Figure 5.1, our method will generate the circuit in Figure 5.2, since it requires less latch motions than the circuit in Figure 5.3.

5.4.3.1 The ALAP and ASAP Retimings

Out of the set of all possible retimings for the given clock scheme, two are of particular interest. We can obtain a retiming such that all latches move as far as possible to the left. This is called "as soon as possible (ASAP)" retiming. Similarly, the retiming that moves all the latches as far as possible to the right is referred to as the "as late as possible (ALAP)" retiming. Both ASAP and ALAP retiming assume no latch is moved across the host node (H). As in Section 3.3 these ASAP and the ALAP locations can be seen as the extreme locations of latches in the circuit for the given clock scheme, and will be utilized, in Section 5.5 for efficient minarea retiming. For the circuit in Figure 5.1 the ALAP and ASAP retimings are shown in Figure 5.2 and Figure 5.3 respectively. As in Section 3.2.2 these ASAP and ALAP retimings can be obtained by modifying the minperiod retiming algorithm.

Unlike retiming for a given period, in ALAP retiming, our objective is to move the latches to the right, as much as possible. For this we initialize all GDT's to $-\infty$, before applying the longest path Bellman-Ford algorithm to the constraint graph. In Phase B we use the allowable range of GDT's to move a latch to the right as much as possible, i.e., if the new GDT after moving a latch to the right is still within the allowable range, we move the latch to the right. Notice that this is done even if the original GDT was within the allowable range.

In ASAP retiming we obtain the GDT's by running the Bellman-Ford algorithm on the transpose[17] of the original constraint graph (i.e., a graph with the same vertex set as the original graph, but with the edge directions reversed) with all latches initialized to $-\infty$. Since all the edge directions where reversed the longest path values for all latches must undergo a sign reversal to obtain the correct GDT values.

5.4.4 A Bound on the Clock Period of the Retimed Circuit

Theorem 3 *At the end of the retiming procedure in Phase B, the magnitude of skew at each latch i, is no more than*

$$\gamma_{p(i)} = \max\left(0, \frac{M - T_{\phi_{p(i)}}}{2}\right) \tag{5.8}$$

where M is the maximum delay of any gate in the circuit.

Proof: We have two cases

Case A : $\underline{M \leq T_{\phi_{p(i)}}}$ If the maximum gate delay is less than the active duration of the clock, we need to prove that at the end of Phase B, all latches will have zero skew. We will prove this by contradiction, assuming that a latch i has nonzero skew S_i at the end of Phase B. We have two sub cases.

Case 1: $\underline{S_i > 0}$ In this case the GDT of latch i is $X_i = S_i$. The new GDT of the latch after it is moved left across a gate of delay d_1 is given by $X_i' = X_i - d_1$.

Since $d_1 \leq M \leq T_{\phi_{p(i)}}$ we have $X_i' \geq -T_{\phi_{p(i)}}$, thus the effective skew S_i' after this possible move is either zero (if $T_{\phi_{p(i)}} \leq X_i' \leq 0$), or $|S_i'| < |S_i|$. In either case the latch i can be moved left across the gate and have its skew reduced. This contradicts the assumption that Phase B is complete.

Case 2: $\underline{S_i < 0}$ If $S_i < 0$ then the GDT of latch i is negative, i.e., $X_i = S_i - T_{\phi_{p(i)}}$, and the proof is similar to case 1.

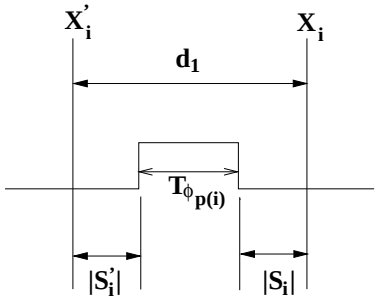

Figure 5.9 Worst-case situation for remaining skew.

Case B: $\underline{M > T_{\phi_{p(i)}}}$ If the maximum gate delay is more than the active duration of the clock, we need to prove that for any latch i, at the end of Phase B the skew magnitude is less than $\gamma_{p(i)}$. Phase B is complete only when for every latch i we have $|S_i'| \geq |S_i|$, where S_i is the current skew and S_i' is the skew after a possible move across a gate with delay d_1. As shown in Figure 5.9 the largest possible final skew magnitude corresponds to the situation when $|S_i| = |S_i'|$. In this case we have $d_1 = 2 \cdot |S_i| + T_{\phi_{p(i)}}$ and hence $|S_i| \leq \frac{d_1 - T_{\phi_{p(i)}}}{2}$. Since M is the maximum gate delay this implies that $|S_i| \leq \frac{M - T_{\phi_{p(i)}}}{2}$.

Theorem 4 *If, in a k phase circuit at the end of the retiming procedure all skews are set to zero, then the final clock period (P_r) satisfies the following condition*

$$P_r \leq P_s + \sum_{i=1}^{k} \max(0, M - T_{\phi_i})$$

where P_s is the optimal clock period with skews found in Phase A, and M is the maximum delay of any gate in the circuit.

Proof : Each difference constraint for the optimal clock period (with skews) is of the form

$$X_i - X_j \leq E_{p(i),p(j)} - d_{ij}.$$

Theorem 3 guarantees us that at the end of Phase B $|X_i|$ and $|X_j|$ are within $\gamma_{p(i)}$ and $\gamma_{p(j)}$ of their optimal values respectively. Therefore the actual value of $X_i - X_j$ after Phase B must lie within $(\gamma_{p(i)} + \gamma_{p(j)})$ of the required value of $X_i - X_j$ in Phase A.

This implies that the inequality that defines the difference constraint can be maintain by increasing $E_{p(i),p(j)}$ by no more than $(\gamma_{p(i)} + \gamma_{p(j)})$. Since each $E_{i,j}$ increases by no more than $(\gamma_{p(i)} + \gamma_{p(i)})$, the clock period $T_\Phi = \sum_{i=1}^{k} E_{p(i),p(j)}$ will increase by no more than $\sum_{i=1}^{k}(2 \cdot \gamma_i)$ or $\sum_{i=1}^{k} \max(0, M - T_{\phi_i})$.

5.5 Minimum Area Retiming

Although the minperiod retiming algorithms can achieve significant improvement in the clock period, they pay no regard to the number of latches in the circuit. As a result minperiod retiming can significantly increase the number of latches in the circuit, and hence the circuit area and power. To contain this increase, we perform constrained minarea retiming. Performing a constrained minarea retiming with the target period set to the period obtained by minperiod retiming, will give us the fastest circuit with least area overhead.

The minarea retiming problem can be modeled as a LP [59]. Unfortunately, under general clock schedules with unequal phases, the minarea retiming problem must be modeled as a general integer linear program of the type given in [67], while restricting the clock scheme to a symmetric multi-phase clock enables us to model the minarea retiming problem as an efficiently solvable LP (dual of min-cost flow problem) [96]. Therefore in this thesis we will consider only symmetric clock schemes. As the LP presented in [96] has almost $\frac{|G|^2}{2}$ constraints for a circuit with $|G|$ gates, minarea retiming of large circuits is not feasible. In this section we present an efficient method for minarea retiming of large level-clocked circuits. Our approach is to improve the efficiency of minarea retiming by
(a) reducing the size of the LP,
(b) generating this LP faster, and
(c) solving the LP efficiently.
Reducing the size of the LP reduces the space requirement of minarea retiming making it feasible for large circuits. Efficient techniques for generating the LP are essential to retime large circuits in reasonable times. Lastly since the size of even the reduced LP will be significant, highly efficient algorithms for solving it are imperative.

In this section we first present the LP formulation of minarea retiming. We then reduce the size of this LP, both in terms of number of variables and constraints, without sacrificing any optimality. Finally we present efficient techniques for generating and solving this LP.

5.5.1 The Minarea Linear Program

The minarea retiming LP for level-clocked circuits is similar to the LP for edge-triggered circuits given in Equation (2.7). The decision variables of this LP are the r variables of the gates, and the objective function represents the number of latches added to the retimed circuit in relation to the original circuit. Since the latches at the output of a gate can be combined

or shared, we use the mirror vertex model of Section 2.1.4 to take into account maximal latch sharing.

Like the minarea LP of Equation (2.7), the minarea LP for level-clocked circuit also contains circuit constraints, period constraints and mirror constraints. The circuit and mirror constraints are defined in the same way as in Section 2.1.4. Since the timing constraints in level-clocked circuits are different than those in edge-triggered circuit, the period constraints are derived as follows

For a k phase symmetric clock we have $T_{\phi_i} = T_\phi \ \forall i = 1 \cdots k$ and $\pi = \frac{T_\Phi}{k}$. For a level-clocked circuit to be properly clocked the delay on any path should be less than the time available, i.e.,

$$d(p) \leq \quad (w_r(p) + 1) \cdot \pi + T_\phi \quad \forall\, p : u \rightarrowtail v \tag{5.9}$$

This constraint can be rewritten after substitution of Equation (2.1) as

$$r(u) - r(v) \quad \leq w(p) - \tfrac{d(p)}{\pi} + 1 + \tfrac{T_\phi}{\pi} \quad \forall\, p : u \rightarrowtail v \tag{5.10}$$

Clearly if there are multiple paths from u to v, only the tightest constraint (one with minimum right hand side) is irredundant. We denote the minimum value of $\left[w(p) - \frac{d(p)}{\pi}\right]$ over all paths from u to v by $\delta(u, v)$, i.e.,

$$\delta(u, v) = \min_{\forall p: u \rightarrowtail v} \left(w(p) - \frac{d(p)}{\pi} \right) \tag{5.11}$$

Let us define $\Delta(u, v)$ as

$$\Delta(u, v) = \left\lfloor \delta(u, v) + \frac{T_\phi}{\pi} + 1 \right\rfloor \tag{5.12}$$

Since the retiming variables $r(u)$ and $r(v)$ are integers, we can rewrite relation (5.10) as the period constraints

$$r(u) - r(v) \quad \leq \Delta(u, v) \quad \forall\, p : u \rightarrowtail v \tag{5.13}$$

We now have the constrained minarea retiming LP as:

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V \cup M} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] & \\ \text{subject to} & r(u) - r(v) \leq c_{uv} & \forall (u, v) \in C \\ & -\infty \leq r(u) \leq \infty & \forall u \ \in (V \cup M) \end{array} \tag{5.14}$$

where $C = C_p \cup C_c \cup C_m$ is the constraint set of the LP, and includes the period constraint set (C_p), the circuit constraint set (C_c) and the mirror constraint set (C_m). A constraint (i, j) in the constraint set C is

$$\begin{array}{lrcll} & r(i) - r(j) & \leq & c_{ij} & \forall (i, j) \in C \\ \text{where} & c_{ij} & = & w(e_{ij}) & \forall (i, j) \in C_c, \text{ i.e., } e_{ij} \in E \\ & c_{ij} & = & \Delta(i, j) & \forall (i, j) \in C_p, \text{ i.e., } i \in V \text{ and } j \in V \\ & c_{ij} & = & w(max_i) - w(e_{jm_i}) & \forall (i, j) \in C_m, \text{ i.e., } m_i \in M \text{ and } \forall j \in FO(i) \end{array} \tag{5.15}$$

5.5.2 Reducing the Linear Program

To reduce the space requirements of minarea LP, it is imperative that we have some techniques to prune the constraints as they are generated, rather than after all the constraints have been generated. In this section we will take advantage of the relationship between retiming and GDT presented in Section 5.3 to reduce the size of the LP by using bounds on the r variables.

As in Section 3.2.2, the ALAP and ASAP retimings described in Section 5.4.3.1 give us bounds on the r variables, of the form

$$L_u \leq r(u) \leq U_u \ \ \forall v \in V \tag{5.16}$$

These bounds give us a reduced variable set $V' \subseteq V$ as

$$V' = \{v \in V | U_v \neq L_v\} \tag{5.17}$$

We use Theorem 1 to obtain bounds on the mirror variables and thus obtain the reduced mirror variable set $M' = \{v \in M | U_v \neq L_v\}$.

Example: For the circuit in Figure 5.1, the ASAP location for the latch $L1$ is at the output of gate $G1$ as shown in Figure 5.3. The number of latches moved across each gate in arriving at this ASAP location, and hence the upper bounds are: $U_{G1} = 0$, $U_{G2} = 1$, $U_{G3} = 1$, and $U_{G4} = 0$. The ALAP location of latch $L1$ as shown in Figure 5.2, is at the output of gate $G2$. The number of latches moved across each gate in arriving at this ALAP location, and hence the lower bounds are: $L_{G1} = 0$, $L_{G2} = 0$, $L_{G3} = 1$, and $L_{G4} = 0$, i.e.

$$\begin{aligned} V &= \{G1, G2, G3, G4\} \\ V' &= \{G2\} \end{aligned}$$

The presence of the bounds obtained in Equation (5.16) makes a large number of constraints redundant, i.e., these constraints are implied by the bounds. We now present a rule to identify these redundant constraints.

Rule 5 *Any constraint (i, j) of the form $r(i) - r(j) \leq c_{ij}$ is redundant in the presence of the bounds of Equation (5.16) and can be dropped if $U_i - L_j \leq c_{ij}$.*

Proof : It can be seen from the bounds on $r(i)$ and $r(j)$ in Equation (5.16) that

$$r(i) - r(j) \leq U_i - L_j$$

Therefore, if $U_i - L_j \leq c_{ij}$ then $r(i) - r(j) \leq c_{ij}$ must also be true. Thus any constraint (i, j) is redundant and can be dropped if $U_i - L_j \leq c_{ij}$.

To obtain the reduced constraint set $C' \subseteq C$ we accept only those constraints from C that are not dropped by the application of Rule 5[2]. Thus

$$C' = \{(i, j) \in C | U_i - L_j > c_{ij}\} \tag{5.18}$$

[2]Some additional techniques to prune redundant period constraints are presented later in Section 5.5.3.3

Thus the original LP in Equation (5.14) is reduced to a much smaller yet equivalent LP given below

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V' \cup M'} \left[\left(\sum_{\forall j \in FI(v)} \beta(e_{jv}) - \sum_{\forall j \in FO(v)} \beta(e_{vj}) \right) \cdot r(v) \right] & (5.19) \\ \text{subject to} & r(u) - r(v) \leq c_{uv} & \forall (u,v) \ \in C' \\ & L_u \leq r(u) \leq U_u & \forall u \ \in V' \end{array}$$

5.5.3 Generating the Reduced Linear Program

A major portion of the effort in retiming a level-clocked circuit for minimum area is spent in generating the period constraints set C'_p. We now describe efficient techniques for generating this set C'_p. The generation of period constraints requires computation of δ values for all-pairs of gates in the circuit. However if the ALAP retiming satisfies the target clock period[3], then we need to compute δ values only from flexible gates, as stated in the following theorem.

Theorem 6 *If the ALAP retiming satisfies the target clock period, any period constraint from a fixed node a (i.e., $U_a = L_a$) is redundant in the presence of the bounds of Equation (5.16) and need not be generated.*

Proof : Since ALAP positions are feasible solutions the following holds for all constraints in C_p.

$$L_i - L_j \quad \leq \quad \Delta(i,j) \ \ \forall \ (i,j) \in C_p \tag{5.20}$$

Consider any period constraint $(a,b) \in C_p$ from a fixed gate a, to any other gate b of the form $r(a) - r(b) \leq \Delta(a,b)$. By relation (5.20) $L_a - L_b \leq \Delta(a,b)$, and by Equation (5.16) $r(a) - r(b) \leq U_a - L_b$. Because gate a is fixed $U_a = L_a$, therefore $r(a) - r(b) \leq L_a - L_b \leq \Delta(a,b)$. Thus the constraint (a,b) is redundant and can be dropped. Since this is true for any period constraint from gate a, we do not need to generate any period constraint from a fixed gate, as they will all be redundant.

5.5.3.1 Computing the δ Values

The δ values can be obtained by re-weighting each edge e_{ij} with $w'(e_{ij}) = \left[w(e_{ij}) - \frac{d(i)}{\pi} \right]$ and computing all-pair shortest paths. We use Johnson's algorithm [17] which has $O(|V|)$ memory requirement, since $O(|V|^2)$ memory is not practical for large circuits with tens of

[3]Notice that if the maximum gate delay in the circuit is more than the active period T_ϕ, it is possible for ALAP retiming to violate the target clock period even if the target clock period is feasible by retiming alone. This is because the method of finding ALAP retiming converts a (continuous) skew optimization solution to a (discrete) retiming solution. This, however, does not imply that these ALAP bounds are wrong, but merely that they are not tight enough. In level clocked circuits, due to the flexibility offered by the transparent nature of latches it is very unlikely that the ALAP retiming will violate the target clock period, and in our experiments no ALAP retiming violated the target clock period.

thousand gates. Johnson's algorithm first re-weights all edges to ensure nonnegative edge weights. The shortest paths between all pair of gates are then computed by running Dijkstra's algorithm for each gate as source.

Let us consider a particular run of Dijkstra's algorithm with gate a as the source, and let b be a gate to which the shortest path $\delta(a,b)$ has been obtained. Let c be any other gate in the circuit, reachable from gate a.

$$\begin{aligned} \text{By definition,} \quad r(a) - r(b) &\leq U_a - L_b \\ \text{If} \quad U_a - L_b &\leq \delta(a,b), \\ \text{then} \quad r(a) - r(b) &\leq \delta(a,b). \end{aligned} \tag{5.21}$$

From relation (5.10) and (5.11)

$$r(b) - r(c) \leq \delta(b,c) + \frac{T_\phi}{\pi} + 1,$$

which when combined with relation (5.21) gives

$$r(a) - r(c) \leq \delta(a,b) + \delta(b,c) + \frac{T_\phi}{\pi} + 1 \tag{5.22}$$

If the shortest path from gate a to gate c does not go through gate b, then $\delta(a,b) + \delta(b,c) \geq \delta(a,c)$ and we do not need to process the fanouts of gate b to obtain $\delta(a,c)$. On the other hand, if the shortest path from gate a to gate c is indeed through gate b then $\delta(a,b) + \delta(b,c) = \delta(a,c)$ and relation (5.22) is same as the period constraint $r(a) - r(c) \leq \Delta(a,c)$. If $U_a - L_b \leq \delta(a,b)$ then this period constraint is redundant. In either case we need not process the fanouts of gate b. Since this is true for any c, reachable from gate a, and we are interested only in gates reachable from gate a, we get the following rule.

Rule 7 *If during the shortest path calculations from source a using the Dijkstra's algorithm, for any gate b we have $U_a - L_b \leq \delta(a,b)$, we do not need to process the fanouts of gate b.*

We take advantage of the bounds on r variables to speed up the computations, by applying Theorem 6 to compute δ values only from the flexible gates, and using Rule 7 to reduce the computation for the δ values actually computed. We found that this significantly improved the time taken to generate the period constraints.

5.5.3.2 Reusing δ Computations

We now describe how to reuse some of the computations performed in obtaining the δ values to further speed up the generation of period constraints. The idea is motivated by the fact that in most practical circuits (e.g., ISCAS-89) a high percentage of gates are single-fanout gates. Consider one such single-fanout gate a with fanout b. For the gate a, the shortest paths to all other gates must be via gate b, which implies that $\delta(a,c) = w'(e_{ab}) + \delta(b,c)$. Therefore

we can obtain the shortest paths from gate a by simply adding $w'(e_{a,b})$ to the shortest paths from gate b. Thus if we somehow ensure that shortest paths from gate b are obtained just before those from gate a, we will save one complete execution of Dijkstra's algorithm for gate a as source. We call this approach "chaining" and the set of gates for which only one set of δ computations is performed as "chains". We now define a simple chaining technique that stores δ values from only one source, hence we call it "1-chaining".

For 1-chaining a graph $G(V, E)$ we preprocess it to get a set of chains $\Omega = \{\omega^1, \omega^2 \ldots \omega^{|\Omega|}\}$ such that every vertex in the graph is included in exactly one chain, i.e.,

$$\omega^i \cap \omega^j = \emptyset \ \ \forall i \neq j \text{ and } \omega^1 \cup \omega^2 \cup \ldots \omega^{|\Omega|} = V$$

Each chain ω^i itself is a ordered list (of size $|\omega^i|$) of vertices in the graph, i.e., $\omega^i =< \omega^i_1, \omega^i_2 \ldots \omega^i_{|\omega^i|} >$. Thus ω^i_j is the j^{th} gate in the i^{th} chain. The first gate ω^i_1 in a chain ω^i is called its head, and all gates in a chain except the head must have only one fanout, i.e., $|FO(\omega^i_j)| = 1 \ \forall i$ and $\forall \ j > 1$. The gates in a chain are ordered such that any fanin of a gate appears after it in the chain, i.e.,

$$e_{\omega^i_{j+1}\omega^i_j} \in E \ \ \forall i \text{ and } 1 \leq j < |\omega^i|$$

We only need to obtain the δ values from the gates that are at the head of a chain, i.e., we only need to compute the values $\delta(\omega^i_1, u) \ \forall u \in V$ and $1 \leq i \leq |\Omega|$. For all other gates the δ values can be obtained by adding the re-weighted edge weight to the δ values from its fanouts, i.e.,

$$\delta(\omega^i_{j+1}, u) = \ \ \delta(\omega^i_j, u) + w'(e_{\omega^i_{j+1}\omega^i_j}) \quad \forall u \in V \text{ and } 1 \leq j < |\omega^i| \text{ and } 1 \leq i \leq |\Omega|$$

Notice that for a gate that is not at the head of any chain we obtain the δ values by a simple addition, instead of a full run of Dijkstra's algorithm. Since we need to run Dijkstra's algorithm only for gates at the head of a chain we need to perform only $|\Omega|$ single-source shortest path computations ($|\Omega| \leq |V|$). Thus our goal in obtaining these chain is to reduce there number, i.e., minimize $|\Omega|$. In the worst case where every gate in the circuit has more than one fanout, each chain contains only one gate, and $|\Omega| = |V|$, then we need to perform the complete Johnson's algorithm. The idea of chaining can be further generalized. Conceptually there are two extremes of chaining:

- No information about the δ values is stored, e.g., repeated single-source shortest paths algorithms like Johnson's algorithm with $O(|V|)$ memory requirements.
- All information about the δ values is stored, e.g., direct all-pairs shortest path algorithms like Floyd Warshall algorithm [17] with $O(|V|^2)$ memory requirements.

The 1-chaining described above is an intermediate method in which we save δ values from only one source. Conceptually we can define k-chaining as a method that stores δ values from k

appropriately chosen sources. This k-chaining in general will require $O(k \cdot |V|)$ memory and careful selection of the k sources, and is not considered in this thesis.

We now describe a simple preprocessing technique to obtain 1-chaining. This preprocessing step first assigns a label to each gate which indicates the number of gates that can reuse its δ computation. All the gates have their labels initialized to 0. These labels are updated by a relaxation step, in which every single-fanout gate relaxes the label of its fanout gate by increasing it (to its own label plus one). Since multiple-fanout gates can not reuse δ computations of their fanout gates in 1-chaining, they do not relax the labels of their fanout gates. This relaxation process is finite because we cannot have cycles containing only single-fanout gates. The chains are then formed by initializing a queue with all multiple-fanout gates. Every gate in this queue starts a new chain. For the gate at the end of the chain, we process the fanin gates, adding the single-fanout gate with the highest label (amongst the fanins) to the chain; all other gates in the fanin are added to the queue. The fanins of the gate now at the end of the chain are processed similarly, until no more gates can be added to this chain. This procedure is repeated until the queue is empty.

We found that, on an average we could reduce the time spent in generating the period constraints by about 50% using the simple 1-chaining technique described above. The time spent in preprocessing to obtain 1-chaining is very small, making it a useful procedure even if only a small number of gates have single fanout. As a side note, Rule 7 must be modified for use with chaining to ensure it is holds for all gates that reuse the δ computation.

5.5.3.3 Additional Constraint Pruning Techniques

We now present some more techniques to remove redundant period constraints. Consider three gates a, b and c, such that gate b lies on the path from gate a to gate c.

If gate b is a fanin of gate c then we have

$$\begin{aligned} C1: &\quad r(a) - r(b) \leq \Delta(a,b) \\ C2: &\quad r(b) - r(c) \leq w(e_{bc}) \\ C3: &\quad r(a) - r(c) \leq \Delta(a,c) \end{aligned}$$

If $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then constraint $C3$ is redundant and can be dropped. This leads us to the following rule

Rule 8 *If b and c are two gates reachable from gate a, such that gate b is a fanin of gate c and $\Delta(a,b) + w(e_{bc}) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.*

Since we generate the period constraints from one gate (say gate a) at a time, both $\Delta(a,b)$ and $\Delta(a,c)$ are available in the same iteration. Further because gate b is a fanin of gate c the value $w(e_{bc})$ is available directly from the circuit graph. Therefore Rule 8 can be efficiently

applied to drop redundant period constraints as they are generated. This reduces the space (memory) requirement of the period constraints.

If gate b is a fanout of gate a then we have

$$\begin{aligned} C4: \quad & r(a) - r(b) \leq \quad w(e_{ab}) \\ C5: \quad & r(b) - r(c) \leq \quad \Delta(b,c) \\ C6: \quad & r(a) - r(c) \leq \quad \Delta(a,c) \end{aligned}$$

If $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then constraint $C6$ is redundant and can be dropped. This leads us to the following rule.

Rule 9 *If gate b is a fanout of gate a and gate c is some gate reachable from gate b, then if $w(e_{ab}) + \Delta(b,c) \leq \Delta(a,c)$ then the period constraint from gate a to c is redundant and can be dropped.*

To apply Rule 9 we require the value of $\Delta(b,c)$ and $\Delta(a,c)$. Since we generate period constraints from one gate at a time, the period constraints to a gate (c) from two different sources (gate a and b) cannot be efficiently accessed. Thus it would appear that Rule 9 cannot be efficiently applied. However because of the reuse of δ computation described in Section 5.5.3.2, Rule 9 can be efficiently applied if gate a has only one fanout (gate b). This is possible because $\Delta(a,c)$ is derived from $\Delta(b,c)$, and hence both are available when the period constraint from a to c is being generated. Thus we can drop redundant period constraints from gate a as they are generated.

Rule 5 is valid only in presence of the bounds and it prunes the constraint sets because the information in these bounds make some constraints redundant. Rule 8 and Rule 9 on the other hand do not depend on bounds and, they prune the period constraint set because of the discrete nature of the Δ values. Rule 8 and Rule 9, can be generalized to include implication by more than two constraints; these generalized rules will, however, be computationally expensive to apply.

5.5.4 Solving the Linear Program

Like Equation (3.8), the LP in Equation (5.19) is also the dual of a min-cost flow problem, and we use the network simplex algorithm described in Section 3.3.3 to solve the dual. Using this method we could solve a mincost flow problems with 70,000 variables and 8.2 million constraints in about 9 minutes.

5.6 Experimental Results

We performed retiming on the complete ISCAS-89 benchmark suite, but present results only on the larger circuits. Due to unavailability of large circuits, we combine circuits from

Table 5.1 Quality of Retiming for Single Phase Circuits

Circuit	$\|G\|$	Period				# Latches				CPU time	
		P_i	P_s	P_r	R_{period}	$\|\Psi_i\|$	$\|\Psi_p\|$	$\|\Psi_a\|$	R_{area}	T_{period}	T_{area}
s3384	1,685	84.0	38.5	38.5	54.2%	183	326	164	10.4%	0.23s	2.34s
s4863	2,342	116.0	59.0	59.0	49.1%	104	254	114	-9.6%	0.22s	4.94s
s5378	2,779	48.0	48.0	48.0	0.0%	179	263	143	20.1%	0.21s	2.99s
s6669	3,080	118.7	49.0	49.0	58.7%	239	472	278	-16.3%	0.56s	4.09s
s13207.1	7,791	127.0	120.0	120.0	5.5%	627	890	446	28.9%	1.09s	13.94s
s15850.1	9,617	187.0	147.0	147.0	21.4%	527	869	533	-1.1%	1.84s	38.26s
s35932	16,065	77.0	71.0	71.0	7.8%	1728	2076	1795	-3.9%	2.81s	63.21s
s38584.1	19,253	125.0	118.0	118.0	5.6%	1426	3298	1427	-0.1%	4.10s	1:49.76s
s38417	21,370	68.7	56.0	56.0	18.5%	1564	2436	1360	13.0%	4.20s	5:28.60s
myex1	28,946	256.0	216.0	216.0	15.6%	1953	4332	1958	-0.3%	8.75s	5:32.08s
myex2	40,661	104.0	97.0	97.0	6.7%	2990	6197	2763	7.6%	9.28s	23:14.83s
myex3	56,751	137.0	119.0	119.0	13.1%	4718	8918	4533	3.9%	14.24s	1:2:22.48s

Table 5.2 Quality of Retiming for Two Phase Circuits

Circuit	$\|G\|$	Period				# Latches				CPU time	
		P_i	P_s	P_r	R_{period}	$\|\Psi_i\|$	$\|\Psi_p\|$	$\|\Psi_a\|$	R_{area}	T_{period}	T_{area}
s3384	1,685	126.0	38.5	38.5	69.4%	366	638	337	7.9%	0.40s	2.56s
s4863	2,342	117.0	59.0	59.0	49.6%	208	473	234	-12.5%	0.29s	5.36s
s5378	2,779	48.0	48.0	48.0	0.0%	358	480	286	20.1%	0.29s	3.22s
s6669	3,080	178.0	49.0	49.0	72.5%	478	960	542	-13.4%	0.76s	6.17s
s13207.1	7,791	127.0	120.0	120.0	5.5%	1,254	1,795	890	29.0%	1.48s	18.61s
s15850.1	9,617	187.0	147.0	147.0	21.4%	1,054	1,777	1,041	1.2%	2.53s	45.82s
s35932	16,065	77.0	71.0	71.0	7.8%	3,456	4,144	3,523	-1.9%	3.98s	67.26s
s38584.1	19,253	125.0	118.0	118.0	5.6%	2,852	7,558	2,852	0.0%	5.02s	1:57.52s
s38417	21,370	103.0	56.0	56.0	45.6%	3,128	4,938	2,766	11.6%	30.45s	6:26.99s
myex1	28,946	256.0	216.0	216.0	15.6%	3,906	9,065	3,891	0.4%	10.08s	6:37.48s
myex2	40,661	128.0	97.0	97.0	24.2%	5,980	13,820	5,551	7.2%	11.25s	31:16.52s
myex3	56,751	137.0	119.0	119.0	13.1%	9,436	17,019	9,041	4.2%	17.46s	1:19:43.07s

the ISCAS-89 benchmark suite to obtain circuits (myex1 through myex3) with up to 56,000 gates. We present results for both minarea and minperiod retiming on single phase and two phase circuits. These results are for a duty cycle and phase ratio of 50%. In absence of delay information in the ISCAS-89 circuits, we assign random delay values (between 1.0 and 20.0 units) to each gate. As in [96] we convert the edge-triggered circuits in ISCAS-89 benchmark to a k phase level-clocked circuits by replacing each FF by k latches.

Table 5.1 and Table 5.2 present the quality of retiming for single phase and two phase circuits respectively. For each circuit the number of gates $|G|$, the initial clock period P_i, the optimal clock period with skews at end of Phase A P_s, the final clock period after retiming P_r, and the percentage decrease in clock period $R_{period} = \frac{P_i - P_r}{P_i}$ are shown. Retiming is able to achieve the same clock period as skew optimization in all the cases. This is possible due to the transparent nature of the latches and underscores the usefulness of retiming level-clocked circuits. Retiming is also able to achieve significant reduction in the clock period, on an average the clock period is reduced 21.5% for single phase circuits, and 27.52% for two-phase circuits.

Table 5.3 Reduction in the Size of LP for Single Phase Circuits

Circuit	G_{fx}	F_{avg}	# Variables			# Constraints		
			Minaret-L	Original	$R_{variables}$	Minaret-L	Original	$R_{constraints}$
s3384	9.18%	2.59	1,988	2,166	8.22%	33,103	761,365	95.65%
s4863	17.28%	1.21	2,497	2,995	16.63%	32,880	5,481,911	99.40%
s5378	25.50%	1.09	2,728	3,664	25.55%	17,121	4,595,645	99.63%
s6669	26.38%	0.98	3,089	4,100	24.66%	14,267	1,923,524	99.25%
s13207.1	19.97%	3.00	7,449	9,180	18.86%	45,563	22,908,799	99.80%
s15850.1	23.46%	1.88	8,813	11,332	22.23%	64,283	39,493,334	99.84%
s35932	8.43%	2.66	20,071	21716	7.58%	145,978	130,080,328	99.89%
s38584.1	14.21%	2.20	20,501	23,390	12.35%	118,771	293,482,797	99.96%
s38417	1.51%	4.66	25567	25,923	1.37%	1,289,378	149,492,588	99.14%
myex1	13.27%	2.32	30,287	34,417	12.00%	142,525	504,055,977	99.97%
myex2	4.25%	4.34	47,409	49,214	3.67%	1,608,132	819,701,299	99.80%
myex3	1.36%	5.19	69,546	70,414	1.23%	3,608,210	1,624,913,333	99.78%

We also show the area cost in terms of number of latches in the initial circuit $|\Psi_i|$, the circuit after minperiod retiming $|\Psi_p|$, and the circuit after constrained minarea retiming with P_r as the target period $|\Psi_a|$. The percentage decrease in number of latches from the initial circuit is given by $R_{area} = \frac{|\Psi_i|-|\Psi_a|}{|\Psi_i|}$. For almost all circuits, minarea retiming reduces the number of latches $|\Psi_a|$ in the circuit by a factor of two to three as compared to minperiod retiming $|\Psi_p|$, even though both retime the circuit for the same clock period P_r. This shows the importance of minarea retiming.

The execution time in seconds on a DEC AXP system 3000/900 workstation, with 256M RAM is shown for both minarea retiming T_{area} and minperiod retiming T_{period}, and highlight the efficiency of our techniques. The minperiod retiming presented here is more efficient than the one for edge-triggered circuits in [109] because it uses the simpler procedure presented in Section 5.4.2 for Phase B latch relocation. The CPU time for minarea timing T_{area} was heavily dominated ($> 90\%$ for large circuits) by the time required to generate the LP, this emphasizes the importance of our use of efficiency-enhancing techniques (chaining, Rule 7, and Theorem 6) while generating the LP.

As can be seen from the results, retiming (minperiod + minarea) can obtain significant reduction in the clock period with no or little area overhead. For example in two phase clocking, for most circuits (except `s4863, s6669` and `s35932`) the clock period is reduced with no area overhead; in fact the area is also reduced (in some cases significantly, e.g., 29% for `s13207.1`). For the other circuits the area overhead is small compared to the gain in clock speed, e.g., for `s6669` a 13.4% area overhead can reduce the clock period by 72.5%.

Table 5.3 provides a closer look at the reduction in the size of LP for minarea retiming for single phase circuits, while Table 5.4 has results for two phase circuits. The size of the LP is shown in terms of the number of variables and the number of constraints. Original represents

Table 5.4 Reduction in the Size of LP for Two Phase Circuits

Circuit	G_{fx}	F_{avg}	# Variables			# Constraints		
			Minaret	Original	$R_{variables}$	Minaret	Original	$R_{constraints}$
s3384	8.15%	5.22	2,006	2,166	7.39%	55,980	761,365	92.65%
s4863	10.51%	2.30	2,706	2,995	9.65%	72,451	5,481,911	98.68%
s5378	19.32%	2.23	2,970	3,664	18.94%	31,765	4,595,645	99.31%
s6669	10.04%	1.92	3,735	4,100	8.90%	20,841	1,923,524	98.92%
s13207.1	17.57%	6.25	7,656	9,180	16.60%	55,395	22,908,799	99.76%
s15850.1	21.60%	3.81	9,013	11,332	20.46%	69,142	39,493,334	99.83%
s35932	7.27%	5.07	20,264	21,716	6.69%	189,068	130,080,328	99.85%
s38584.1	13.78%	4.39	20,590	23,390	11.97%	127,488	293,482,797	99.96%
s38417	0.87%	9.43	25,735	25,923	0.73%	2,446,798	149,492,588	98.36%
myex1	12.63%	4.70	30,489	34,417	11.41%	154,603	504,055,977	99.97%
myex2	1.52%	8.72	48,560	49,214	1.33%	3,638,182	819,701,299	99.56%
myex3	0.67%	10.41	70,000	70,414	0.59%	8,207,036	1,624,913,333	99.50%

the traditional LP of Equation (5.14) used in [96] while Minaret-L represents the reduced LP of Equation (5.19). $R_{variables}$ and $R_{constraints}$ give the percentage reduction in the number of variables and constraints respectively, due to the pruning techniques presented in this chapter. Also presented are two metrics on the circuits: G_{fx} the number of gates found to be fixed and F_{avg} the average flexibility, i.e., the average values of $(U_y - L_y)$ over all gates in the circuits. The number of variables include both gate and mirror variables and hence the reduction in variables can be different from G_{fx} which does not include mirror vertices. High G_{fx} and low F_{avg} indicates less mobility or flexibility in the circuit, yielding higher percentage reduction in the number of constraints, and faster minarea retiming. It can be seen that up to three orders of magnitude reduction is obtained in the number of constraints by using Minaret-L, e.g., for one phase circuit myex3 the number of constraints reduce from about 1.6 billion to only 3.6 million. The number of unpruned constraints grow at the rate of $O(|G|^2)$ and our pruning techniques reduce this rate of growth significantly. Although the bounds on the r variables help significantly in reducing the CPU time for minarea retiming, the time spent in obtaining these bounds is a insignificant fraction (less than half a percent) of the total CPU time for minarea retiming. Amongst single phase and two phase circuits the single phase circuits have less flexibility, and a much smaller LP than two phase circuits.

5.7 Conclusion

Efficient algorithms for both minperiod and minarea retiming of large level-clocked circuits have been presented. The entire ISCAS-89 benchmark suite could be retimed in minutes. The chief reason for the efficiency of this minperiod algorithm is that it uses the retiming skew relation to map the problem of retiming level-clocked circuits to the much simpler problem of retiming edge-triggered circuit. This enabled us to greatly speed up the process of performing

binary search for the optimal clock period. This is possible because we create a small and sparse constraint graph, only once rather than in each step of the binary search as done by traditional methods [96, 70]. The second phase of minperiod retiming is fast because latches do not have to be moved across a large numbers of gates during retiming.

The minarea retiming algorithm is made practical for large circuits by utilizing the retiming-skew relation, and several other pruning techniques (Rule 5, Rule 8 and Rule 9) to reduce the LP in Equation (5.14) to a much smaller LP in Equation (5.19), without sacrificing any optimality. A reduction of two to three orders of magnitudes in the number of constraints is obtained for most circuits. The use of Theorem 6, Rule 7, and chaining, greatly speed up the period constraint generation making the overall algorithm very efficient.

In summary, the contributions of this chapter, which applies retiming-skew relation for fast minarea and minperiod retiming for level-clocked circuits are the following:

- It handles level sensitive latches like edge triggered FF's, thus avoiding a complicated formulation that is forced to handle critical path propagation over several latches. This also avoids the need of generating the constraint graph for every point in the binary search, which is necessitated by the fact that critical paths change with the clock period [70].

- It provides a conceptually simpler technique than [109] for reducing the GDT's in Phase B of minperiod retiming which can also be applied to edge-triggered circuits.

- It provides efficient techniques for generating and pruning the minarea LP.

- It shows that retiming can optimize large level-clocked circuits for high performance with little or no area overhead.

The algorithms presented in this chapter can also be used to solve the interesting problem of optimizing edge-triggered circuits which allow some skew (less than a given maximum skew magnitude) at the FF's. Some design methodologies may allow a small amount of skew at the FF's. The method presented in this chapter can take advantage of this skew to yield better optimization.

6 CONCLUSION

6.1 Conclusion

In this thesis we presented efficient techniques for delay and area optimization of sequential circuits via retiming. Retiming relocates the memory elements in a circuit without changing its behavior. Our techniques can handle large circuits (with tens of thousands of gates) using either edge-triggered FF's or level-sensitive latches.

In Chapter 3 we presented the Minaret algorithm, which solves the problem of constrained minarea retiming for circuits with edge-triggered FF's through an amalgamation of the Leiserson-Saxe approach and the ASTRA approach. By utilizing the merits of both approaches an efficient algorithm for constrained minarea retiming, capable of handling very large circuits, has been developed. The basic idea is to use the ASTRA approach to find tight bounds on the retiming variables. These bounds then helped us reduce both the number of variables and the number of constraints in the problem without any loss in accuracy. On an average Minaret obtained a 30% reduction in the number of variables and an 80% reduction in the number of constraints. Minaret could retime a circuit with more than 56,000 gates in under 15 minutes. In contrast the best results published before ours [115] take about 39 hours for a 8,000 gate circuit.

In Chapter 4 we addressed the problem of minarea retiming with a guarantee of equivalent initial states, and called it minarea initial state retiming. In control logic the initial state of a circuit is an integral part of the behavior, and hence any retiming must also generate an equivalent initial state for the retimed circuit in order for it to have the same behavior as the original circuit. The presence of an equivalent initial state was guaranteed by adding "justification upper bounds" on the retiming variables. These bounds also helped in obtaining the equivalent initial state by a simple method. To obtain an accurate estimate of the number of FF's it is essential to correctly model the sharing of FF's with reset values at the output of a gate. We presented a 0/1 MILP formulation to model this conditional FF sharing. This model is also useful for performing minarea retiming of circuits that contain more than one kind of FF's, such that different kinds of FF's cannot be shared with each other. Although the formulation requires us to solve an MILP, our experimental results showed that practical size circuits can be handled in reasonable time. This is achieved by ensuring that the number of integer variables in the LP is small.

In Chapter 5 we presented the equivalence between retiming and skew optimization for level-clocked circuits. We also showed that by using the concept of Global Departure Time (GDT), one can treat latches like FF's with the ability to absorb some skew, where the GDT for a latch corresponds to skews in the case of a FF. This equivalence was then utilized to achieve fast retiming for large multi-phase level-clocked circuits.

We presented delay and area optimization results on the entire ISCAS benchmark suite for both single phase and two phase clocks. We obtained an average 27.52% improvement in the clock period, while also reducing the area by 4.48% on an average. In many cases the clock period is reduced significantly without any area overhead, while in other cases, the area overhead is small as compared to the gain in clock speed, e.g. for `s6669` 13.4% area increase could reduce the clock period by 72.5%.

The advantages of utilizing the retiming-skew equivalence are more significant in level-clocked circuits than in edge-triggered circuits. The algorithms presented are very efficient, and are able to retime a circuit with more than 56,000 gates in about 15 seconds for minimum period and 1.5 hours for minimum area. In contrast the only published results [54] are for circuits with less than 400 gates. The reduction in the number of constraints for level-clocked circuits was as much as three orders of magnitude, and the constraints for a 56,000 gate circuit were reduced from 1.6 billion to 3.6 million.

6.2 Directions for Further Research

Although a significant amount of research has been performed on retiming, some key issues need to be addressed before retiming is widely accepted by the design community. We now present some of these issues, and our thoughts on them.

6.2.1 Restriction on Design Styles

The traditional retiming methods impose severe design style restrictions on the circuits they can handle. Many of these styles are very popular in high performance designs, and these restrictions need to be relaxed before retiming can be applied to a large section of designs. Some of these restrictions are

Gated clocks Many low power designs contain gated clocks. A gated clock can be modeled by a MUX at the latch input with a feed back loop. This will enable retiming to treat gated clock latches as ordinary latches, but this may result in a structure that is not recognizable as a gated clock after retiming, and hence may not be desired by the designers. If the gated clock latches are marked as latches that cannot be moved, then the gated clock structure is preserved; however, optimality may be sacrificed. A better approach can be obtained by developing the techniques in [45].

Multi-cycle paths Traditional retiming techniques do not handle designs containing paths on which data is allowed to propagate for more than one clock cycle. Multi-cycle paths can be handled for specified-period retiming by using either the techniques of [57] or [109]. However, performing minarea retiming on these circuits is a much harder problem.

Registers with logic Most retiming algorithms assume that all registers have only one data input and one data output. Many design libraries contain registers with some logic, e.g, AOI latches. This requires retiming techniques to handle library binding issues, such as finding logic in the fanin cone of the register to be merged with it. Minimum area retiming becomes more complex since the combinational part of the circuit is changed by register relocation under this scenario.

Mix of register types Designs containing more then one type of register are difficult to retime. In minperiod retiming, different types of registers can be handled by a post-processing phase. However, incorporating different types of registers in minarea retiming is harder. One method is to formulate the minarea LP assuming that all registers are of the same type and then use a post-processing step to ensure that registers of different types are not merged or shared together; however, the solution so obtained would not be optimal. The MILP formulation in Section 4.3 can be used for minarea retiming of circuits with different register types, although at a higher computational cost.

Don't Care timing assertions Some design contains paths for which timing is not important, i.e, with don't care timing assertions (e.g. scan chain for testing or clock network paths). These paths should not limit the clock period of the retimed circuit, but valid retimings must consider these paths when performing register moves. While the ASTRA approach can probably be modified to handle these paths during minperiod retiming, handling these paths in minarea retiming appears to be harder problem.

6.2.2 Verification

One of the main road-blocks in the use of retiming is the problem of verifying the correctness of retimed circuits. Retiming changes the number and location of memory elements in the circuit, hence for FSM's it changes both the encoding of the states and the number of states. Sequential verification is therefore required to verify retimed circuits. Unfortunately sequential verification is a hard problem, however, it is hoped that verifying circuits that are just retimed versions of each other is possible and preliminary efforts in this direction include [104, 85].

Enough information about the register movement during retiming can be produced by a retiming tool to possibly enable Boolean equivalency checks on the combinational parts even though latch boundaries have changed. However this method would not yield a true independent verification, which is the goal of the verification process. One simple sanity check

on retiming tools is to perform a structural verification by verifying that the number of latches on all cycles are the same [113].

6.2.3 Position in Design Flow

Retiming is a very general transform that can be applied at various levels of abstraction. Retiming can be used during high-level synthesis to improveschedules, and has been used at the data flow graph levels for digital signal processing applications. Gate level retiming can be used at almost any stage during logic synthesis, and finding the best point in the design flow at which to perform retiming is a problem for further research. The ability to verify retimed circuits is a major factor in determining the place of retiming in the design flow.

Retiming can be applied early during the technology independent stage, since the fixed delay model used by retiming is better suited for this stage. It could also be applied near the end of the synthesis process because the sequential nature of retiming makes it hard for the designer to recorgnize the retimed circuit.

6.2.4 Improved Delay Models

As mentioned earlier, one of the the main drawbacks of the current retiming algorithms is that they assume constant gate delays. Even in research that uses a more general delay model such as [54, 127], the delay of a gate does not depend on the number of its fanouts. Although retiming does not change the topology of a circuit, the sharing of FF's at the output of a gate can change the number of fanouts for that gate.

For the circuits shown in Figure 6.1, let the delay of a gate or a FF be equal to the number of its fanouts. Different FF placements can lead to different delay distributions as shown in Figure 6.1, e.g, {5,5,5,5}, {4,4,4,4}, {5,5,5,3}. Thus any retiming algorithm using a fanout dependent delay model must also explore these delay distribution options.

Iterating an appropriately modified ASTRA may be able to obtain good approximate solutions to this problem. An exact solution will, however, most likely require constraints between all pairs of edges as in [54]. This would make the method incapable of handling large circuits. Modifications in the Minaret approach may be able to prune this constraint set.

Retiming techniques with better delay models can be combined with transistor sizing [71, 110] for area and delay optimization. While one way is to iterate between retiming and sizing, a more integrated approach is likely to provide better results.

6.2.5 Retiming and Logic Synthesis

Retiming is a simple yet powerful sequential transform, which operates over the complete sequential circuits, unlike most other logic optimization techniques, which operate only on combinational sub-circuits. The sequential nature of retiming makes it possible to improve

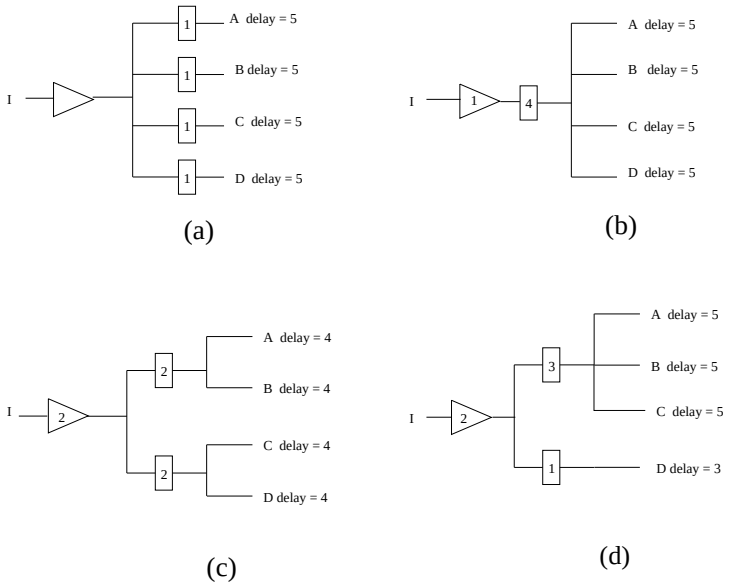

Figure 6.1 Different path delays in a fanout dependent delay model.

the quality of results obtained by subsequent combinational logic optimization. In general, there are multiple valid retiming solutions that may have different effects on subsequent logic optimization. The retiming transform can possibly be modified to give a different yet optimal solution every time so that retiming's effect on other logic optimization transforms is better explored. However, this approach will make application of incremental logic optimization difficult.

Most logic optimization transforms are rather localized and heuristic based combinational transforms. Retiming, on the other hand, is a global sequential transform which is optimal under its assumptions. One way to use retiming is to add it to the "bag of tricks" used by a logic optimization tool as in [33]. Another way is to use retiming in a more systematic way at predefined points in the process.

Preliminary efforts at iterating between standard retiming and combinational synthesis have not resulted in significant improvements [38]. We believe that a better way to combine retiming with other logic optimization techniques is to retime a circuit so as to give the largest possible combinational sub-circuits. Our approach is to perform a modified minarea retiming called mincut retiming, where the objective is to minimize the number of edges that contain one or more FF's on them. The justification for this objective function is that, when converting a sequential circuit to combinational sub-circuits, every edge with at least one FF on it is cut by adding a primary output and a primary input. Hence the objective should be to minimize these cuts, i.e., edges with one or more FF on them, and not the total number of FF's in the

circuit. This problem can be formulated as the following MILP

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} z(v) & \\ \text{subject to} & w(e_{uv}) + r(v) - r(u) \leq F \cdot z(u) & \forall e_{uv} \in E \\ & r(u) - r(v) \leq w(e_{uv}) & \forall e_{uv} \in E \\ & z(v) \in \{0, 1\} & \forall v \in V \end{array} \tag{6.1}$$

where $z(u)$ is a 0/1 integer variable, if there is at least one FF at the fanout of gate u then $z(u) = 1$, otherwise $z(u) = 0$. F is a large constant such that no edge can have more than F number of FF's after retiming, i.e, $F = |FF|$. This MILP can be modified to include peripheral retiming [81] by removing the circuit constraints corresponding to the peripheral edges.

The optimal solution of this MILP is used to retime the circuit, which is then converted to combinational sub-circuits. Each of these sub-circuits are then separately optimized, and the full circuit recreated by recombining these sub-circuits. The sequential circuit thus obtained is again retimed for the desired clock period.

BIBLIOGRAPHY

[1] A. van der Werf, J. L. Van Meerbergen, E. H. L. Aarts, W. F. J. Verhaegh, and P.E.R. Lippens. Efficient timing constraint derivation for optimally retiming high speed processing units. In *International Symposium on High Level Synthesis*, pages 48–53, 1994.

[2] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. W. H. Freeman and Company, New York, NY, 1990.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[4] A. Balakrishnan and S. Chakradhar. Retiming with logic duplication transformation: Theory and an application to partial scan. In *Proceedings of the International Conference on VLSI Design*, pages 296–302, 1996.

[5] M. S. Bazaraa, J. J. Javis, and H.D. Sherali. *Linear Programming and Network Flows*. John Wiley, New York, NY, 1977.

[6] M.R.C.M. Berkelaar. *LP_SOLVE User's Manual*. Eindhoven University of Technology, Eindhoven, The Netherlands, 1992.

[7] F. Brglez, D.Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1929–1934, 1989.

[8] T. M. Burks and K. A. Sakallah. Optimization of critical paths in circuits with level-sensitive latches. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 468–473, 1994.

[9] T. M. Burks, K. A. Sakallah, and T. N. Mudge. Critical paths in circuits with level-sensitive latches. *IEEE Transactions on VLSI Systems*, 3(2):273–291, June 1995.

[10] L. F. Chao. *Scheduling and Behavioral Transformations for Parallel Systems*. PhD thesis, Princeton University, Princeton, NJ, 1993.

[11] L. F. Chao and E. H.-M. Sha. Unfolding and retiming data-flow DSP programs for RISC multiprocessor scheduling. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 565–568, 1992.

[12] L. F. Chao and E. H.-M. Sha. Efficient retiming and unfolding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 421–424, 1993.

[13] L. F. Chao and E. H.-M. Sha. Retiming and clock skew for synchronous systems. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1.283–1.286, 1994.

[14] T. H. Chao, Y. C. Hsu, and J. M. Ho. Zero-skew clock net routing. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 518–523, 1992.

[15] Y.P. Chen and D. F. Wong. On retiming for FPGA logic module minimization. In *Proceedings of the IEEE International Conference on Computer Design*, pages 394 –397, 1994.

[16] J. Cong and C. Wu. An improved algorithm for performance optimal technology mapping with retiming in LUT-based FPGA design. In *Proceedings of the IEEE International Conference on Computer Design*, pages 572–578, 1996.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, NY, 1990.

[18] D. K. Das and B. B. Bhattacharya. Does retiming affect redundancy in sequential circuits Γ In *Proceedings of the International Conference on VLSI Design*, pages 260–263, 1996.

[19] G. De Micheli and T. Klein. Algorithms for synchronous logic synthesis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 756–761, 1989.

[20] T. C. Denk and K. K. Parhi. A unified framework for characterizing retiming and scheduling solutions. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 568–571, 1997.

[21] R. B. Deokar. Clock period minimization using skew optimization and retiming. Master's thesis, Iowa State University, Ames IA, 1994.

[22] R. B. Deokar and S. S. Sapatnekar. A fresh look at retiming via clock skew optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 310–315, 1995.

[23] S. Dey and S. Chakradhar. Retiming sequential circuits to enhance testability. In *Proceedings of the IEEE VLSI Test Symposium*, pages 28–33, 1994.

[24] M. Edahiro. An efficient zero-skew routing algorithm. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 375–380, 1994.

[25] D. Eisenbiegler, R. Kumar, and C. Blumenrohr. A constructive approach towards correctness of synthesis- application within retiming. In *Proceedings of the European Design and Test Conference*, pages 427–431, 1997.

[26] A. El-Maleh, T. Marchok, J. Rajski, and W. Maly. On test set preservation of retimed circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 176–182, 1995.

[27] P. Duncan *et al.* HI-PASS: A computer-aided synthesis system for maximally parallel digital signal processing ASICS. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages V–605–608, 1992.

[28] G. Even. *Design of VLSI circuits using retiming.* PhD thesis, Israel Insistitute of Technology, Haifia, Israel, 1994.

[29] G. Even and A. Litman. Overcomming chip-to-chip delays and clock skew. In *Proceeding of International Conference on Application Specific Systems*, pages 199–208, 1996.

[30] G. Even, I. Y. Spillinger, and L. Stok. Retiming revisited and reversed. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(3):348–357, March 1996.

[31] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, July 1990.

[32] E. G. Friedman. The application of localized clock distribution design to improving the performance of retimed circuits. In *Proceedings of IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS)*, 1992.

[33] G. De Micheli. Synchronous logic synthesis: Algorithms for cycle time minimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):63–73, January 1991.

[34] C. H. Gebotys. Throughput optimized architectural synthesis. *IEEE Transactions on VLSI Systems*, 1:254–261, 1993.

[35] M. Genoe, L. Claesen, and H. De Man. A parallel method for functional verification of medium and high throughput DSP synthesis. In *Proceedings of the IEEE International Conference on Computer Design*, pages 460–463, 1994.

[36] S. Hassoun and C. Ebeling. Architectural retiming: An overview. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 27–38, 1996.

[37] S. Hassoun and C. Ebeling. Architectural retiming: Pipelining latency-constrained circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 708–713, 1996.

[38] S. Hassoun and C. Ebeling. Experiments in the iterative application of resynthesis and retiming. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 164–169, 1997.

[39] S. Hassoun and C. Ebeling. An overview of prediction-based architectural retiming. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 39–48, 1997.

[40] Soha Hassoun. *Architectural Retiming: A Technique for Optimizing Latency-Constrained Circuits*. PhD thesis, University of Washington, Seattle, WA 1997.

[41] Y. Higami, S. Kajihara, and K. Kinoshita. Test sequence compaction by reduced scan shift and retiming. In *Proceedings of the IEEE Asian Test Symposium*, pages 169–175, 1995.

[42] Y. Higami, S. Kajihara, and K. Kinoshita. Partially parallel scan chain for test length reduction by using retiming technique. In *Proceedings of the IEEE Asian Test Symposium*, pages 94–99, 1996.

[43] S. Y. Huang, K. T. Cheng, and K. C. Chen. On verifying the correctness of retimed circuits. In *Proceedings of the Great Lake Symposium on VLSI*, pages 277–280, 1996.

[44] A. Ishii, C. E. Leiserson, and M. C. Papaefthymiou. Optimizing two-phase, level-clocked circuitry. In *Advanced Research in VLSI and Parallel Systems: Proceedings of the 1992 Brown/MIT Conference*, pages 246–264, 1992.

[45] A. T. Ishii. Retiming gated-clocks and precharged circuit structures. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 300–307, 1993.

[46] A. T. Ishii and M. C. Papaefthymiou. Efficient pipelining of level-clocked circuits with min-max propagation delays. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 39–51, 1996.

[47] B. Iyer and M. Ciesielski. Metamorphosis: State assignment by retiming and re-encoding. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 614–617, 1996.

[48] D. Kagaris and S. Tragoudas. Partial scan with retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 249–254, 1993.

[49] D. Kagaris and S. Tragoudas. Retiming algorithm with application to VLSI testability. In *Proceedings of the Great Lake Symposium on VLSI*, pages 216–221, 1994.

[50] D. Kagaris, S. Tragoudas, and D. Bhatia. Pseudo-exhaustive BIST for sequential circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 523–527, 1993.

[51] I. Karkowski and R.H.J.M. Otten. Retiming synchronous circuitry with imprecise delays. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 322–326, 1995.

[52] S. Kundu, L. Huisman, I. Nair, V. Iyengar, and L. Reddy. A small test generator for large designs. In *Proceedings of the IEEE International Test Conference*, pages 30–40, 1992.

[53] K. N. Lalgudi. *Architecural-Level Design of High-Performance, Energey-Efficient VLSI Systems*. PhD thesis, Yale University, New Haven, CT, 1996.

[54] K. N. Lalgudi and M. Papaefthymiou. DELAY: An efficient tool for retiming with realistic delay modeling. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 304–309, 1995.

[55] K. N. Lalgudi and M. Papaefthymiou. Fixed-phase retiming for low power. In *Proceedings of the International Symposium of Low Power Electronic Devices*, pages 259–264, 1996.

[56] K. N. Lalgudi and M. C. Papaefthymiou. Efficient retiming under a general delay model. In *Advanced Research in VLSI : the 1995 MIT/UNC-Chapel Hill Conference*, 1995.

[57] C. Legl, P. Vanbekbergen, and A. Wang. Retiming of edge-triggered circuits with multiple clocks and load enables. In *Workshop Notes, International Workshop on Logic Synthesis*, 1997.

[58] C. Leiserson, F. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. In *Proceedings of the 3rd Caltech Conference on VLSI*, pages 87–116, 1983.

[59] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[60] S. Lejmi, B. Kaminska, and B. Ayari. Retiming for BIST-sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1740–1742, 1995.

[61] S. Lejmi, B. Kaminska, and B. Ayari. Retiming, resynthesis, and partitioning for the pseudo-exhaustive testing of sequential circuits. In *Proceedings of the IEEE VLSI Test Symposium*, pages 434–439, 1995.

[62] S. Lejmi, B. Kaminska, and B. Ayari. Synthesis and retiming for the pseudo-exhaustive BIST of synchronous sequential circuits. In *Proceedings of the IEEE International Test Conference*, pages 683–692, 1995.

[63] S. Lejmi, B. Kaminska, and E. Wagneur. Resynthesis and retiming of synchronous sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1674–1677, 1993.

[64] S. Lejmi, B. Kaminska, and E. Wagneur. Retiming for the global optimization of synchronous sequential circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 398–401, 1994.

[65] B. Lin. Restructuring of synchronous logic circuits. In *Proceedings of the European Design Automation Conference*, pages 205–209, 1993.

[66] L.-T. Liu, M. Shih, N.-C. Chou, C.-K. Cheng, and W. Ku. Performance-driven partitioning using retiming and replication. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 296–299, 1993.

[67] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. Technical Report UW-CSE-91-10-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1991.

[68] B. Lockyear and C. Ebeling. Optimal retiming of multi-phase level-clocked circuits. In *Advanced Research in VLSI and Parallel Systems: Procedings of the 1992 Brown/MIT Conference*, pages 265–280, 1992.

[69] B. Lockyear and C. Ebeling. The practical application of retiming to the design of high-performance systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 288–295, 1993.

[70] B. Lockyear and C. Ebeling. Optimal retiming of level-clocked circuits using symmetric clock schedules. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1097–1109, September 1994.

[71] N. Maheshwari and S. S. Sapatnekar. Gate size optimization for row-based layouts. In *Proceedings of the 38th Midwest Symposium on Circuits and Systems*, pages 777–800, 1995.

[72] N. Maheshwari and S. S. Sapatnekar. A practical algorithm for retiming level-clocked circuits. In *Proceedings of the IEEE International Conference on Computer Design*, pages 440–445, 1996.

[73] N. Maheshwari and S. S. Sapatnekar. An improved algorithm for minimum-area retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 2–7, 1997.

[74] N. Maheshwari and S. S. Sapatnekar. Minimum area retiming with equivalent initial states. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 216–219, 1997.

[75] N. Maheshwari and S. S. Sapatnekar. Retiming level-clocked circuits for latch count minimization. In *Workshop Notes, International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 135–140, 1997.

[76] N. Maheshwari and S. S. Sapatnekar. Efficient minarea retiming for large level-clocked circuits. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 840–845, 1998.

[77] N. Maheshwari and S. S. Sapatnekar. Efficient retiming of large circuits. *IEEE Transactions on VLSI Systems*, pages 74–83, March 1998.

[78] N. Maheshwari and S. S. Sapatnekar. Optimizing large multi-phase level-clocked circuits. *submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1998.

[79] N. Maheshwari and S. S. Sapatnekar. Retiming control logic. *submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1998.

[80] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. In *Proceedings of the 23rd Anual Hawaii International Conference on System Sciences*, pages 397–406, 1990.

[81] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimizing sequential networks with combinational techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(1):74–84, January 1991.

[82] S. Malik, K. J. Singh, R. K. Brayton, and A. Sangiovanni-Vincentelli. Performance optimization of pipelined circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 410–413, 1990.

[83] H.-G. Martin. Retiming by combination of relocation and clock delay adjustment. In *Proceedings of the European Design Automation Conference*, pages 384–389, 1993.

[84] H.-G. Martin. Retiming for circuits with enable registers. In *Proceedings of EUROMICRO-22*, pages 275–280, 1996.

[85] A. Mets. *Formal verification of sequential circuits using implicit state enumeration.* Master's thesis, Eindhoven University of Technology, Netherlands, 1994.

[86] G. De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, New York, NY, 1994.

[87] J. Monteiro, S. Devadas, and A. Ghosh. Retiming sequential circuits for low power. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 398–402, 1993.

[88] M. Moonen, I.K. Proudler, J. G. McWhirter, and G. Hekstra. On the formal derivation of a systolic array for recursive least square estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages II–477–480, 1994.

[89] A. Munzner and G Hemme. Converting combinational circuits into pipelined data paths. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 368–371, 1991.

[90] P. Pan. Continuous retiming: Algorithms and applications. In *Proceedings of the IEEE International Conference on Computer Design*, 1997.

[91] P. Pan and C.L. Liu. Optimal clock period technology mapping for FPGA circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 720–725, 1995.

[92] M. C. Papaefthymiou. On retiming synchronous circuitry and mixed-integer optimization. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1990.

[93] M. C. Papaefthymiou. *A Timing Analysis and Optimzation System for Level-Clocked Circuitry.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[94] M. C. Papaefthymiou. Understanding retiming through maximum average delay cycles. In *Mathematical Systems Theory*, pages 27:65–84, 1994.

[95] M. C. Papaefthymiou and K. H. Randall. Edge-triggered vs. two-phase level-clocking. In *Research on Integrated Systems: Proceedings of the 1993 Symposium*, March 1993.

[96] M. C. Papaefthymiou and K. H. Randall. TIM: A timing package for two-phase, level-clocked circuitry. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 497–502, 1993.

[97] N. L. Passos and E.H.-M. Sha. Achieving full parallelism using multidiemnsional retiming. *IEEE Transactions on Parallel and Dsitributed Systems*, 7:1150–1163, November 1996.

[98] N. L. Passos, E.H.-M. Sha, and S. C. Bass. Optimizing DSP flow graphs via schedule-based multidimensional retiming. *IEEE Transactions on Signal Processing*, 44:150–155, January 1996.

[99] M. Potkonjak and J. Rabaey. Optimizing resource utilization using transformations. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 88–91, 1991.

[100] M. Potkonjak and J. Rabaey. Fast implementation of recursive programs using transformations. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 304–308, 1992.

[101] M. Potkonjak and J. Rabaey. Pipelining: Just another transformation. In *Proceedings of the International Conference on Application Specific Array Processors*, pages 163–175, 1992.

[102] S. Qadeer, R. K. Brayton, V. Singhal, and C. Pixley. Latch redundancy removal without global reset. In *Proceedings of the IEEE International Conference on Computer Design*, pages 432–439, 1996.

[103] K. H. Randall. *Edge-triggered vs. level-clocking.* Bachelor's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1993.

[104] R. Ranjan. *Design and Implementation Verification of Finite State Systems.* PhD thesis, University of California, Berkeley, CA, 1997.

[105] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 111–117, 1990.

[106] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 552–555, 1990.

[107] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(3):322–333, March 1992.

[108] S. S. Sapatnekar. Efficient calculation of all-pair input-to-output delays in synchronous sequential circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages IV520–IV523, 1996.

[109] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming skew equivalence in a practical algorithm for retiming large circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1237–1248, October 1996.

[110] S. S. Sapatnekar and S. M. Kang. *Design Automation for Timing-Driven Layout Synthesis.* Kluwer Academic Publishers, Boston, MA, 1993.

[111] J. B. Saxe. *Decomposable Searching Problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[112] N. Shenoy. *Timing Issues in sequential circuits.* PhD thesis, University of California, Berkeley, CA, 1993.

[113] N. Shenoy. Retiming: Theory and practice. *Integration, the VLSI Journal*, 22(1):1–21, January 1997.

[114] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli. Retiming of circuits with single phase transparent latches. In *Proceedings of the IEEE International Conference on Computer Design*, pages 86–89, 1991.

[115] N. Shenoy and R. Rudell. Efficient implementation of retiming. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 226–233, 1994.

[116] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Resynthesis of multiphase pipelines. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 490–496, 1993.

[117] S. Simon, E. Bernard, M. Sauer, and J. A. Nossek. A new retiming algorithm for circuit design. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 35–38, 1994.

[118] S. Simon, R. Bucher, and J. A. Nossek. Retiming of synchronous circuits with variable topology. In *8th International Conference on VLSI Design, New Delhi, India*, pages 130–134, January 1995.

[119] S. Simon and J. Hofner. Retiming algorithms for multiplexer circuits. Technical Report TUM-LNS-TR-94-8, Institute of Network Theory and Circuit Design, Technical Univeristy of Munuch, 1994.

[120] S. Simon, C. V. Schimpfle, M. Wroblewski, and J. A. Nossek. Retiming of latches for power reduction of DSP design. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 2168–2171, 1997.

[121] V. Singhal. *Design Replacements for Sequential Circuits.* PhD thesis, Department of Computer Science and Engineering, University of California, Berkeley, March 1996.

[122] V. Singhal, Sharad Malik, and R. K. Brayton. The case for retiming with explicit reset circuitry. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 618–625, 1996.

[123] V. Singhal, C. Pixley, R. L. Rudell, and R. K. Brayton. The validity of retiming sequential circuits. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 316–321, 1995.

[124] T. Soyata and E. G. Friedman. Retiming with non-zero clock skew, variable register and interconnect delay. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 234–241, 1994.

[125] T. Soyata and E. G. Friedman. Synchronous performance and reliability improvement in pipelined ASICs. In *Proceedings of the IEEE ASIC Conference*, pages 383–390, 1994.

[126] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr. Integration of clock skew and register delays into a retiming algorithm. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1483–1486, 1993.

[127] T. Soyata, E. G. Friedman, and J. H. Mulligan, Jr. Incorporating internconnect, register and clock distribution delays into the retiming process. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(1):165–120, January 1997.

[128] I. Y. Spillinger, L. Stok, and G. Even. Improving initialization through reversed retiming. In *Proceedings of the European Design and Test Conference*, pages 150–154, 1995.

[129] H. Touati, N. V. Shenoy, and A. L. Sangiovanni-Vincentelli. Retiming for table-lookup field programmable gate arrays. In *FPGA*, pages 89–94, 1992.

[130] H. J. Touati and R. K. Brayton. Computing the initial states of retimed circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(1):157–162, January 1993.

[131] R.-S. Tsay. Exact zero skew. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 336–339, 1991.

[132] N. Wehn, J. Biesenack, T. Langmaier, M. Munch, M. Pilsl, S. Rumler, and P. Duzy. Scheduling of behavioral VHDL by retiming techniques. In *Proceedings of the European Design Automation Conference*, 1994.

[133] U. Weinmann and W. Rosenstiel. Techology mapping for sequential circuits based on retiming techniques. In *Proceedings of the European Design Automation Conference*, pages 318–323, 1993.

[134] Y.G. DeCastelo-Vide-e-Souza, M Potkonjak, and A parker. Optimal ILP-based approach for throughput optimization using simultaneous algorithm/architecure matching and retiming. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 113–118, 1995.

[135] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita. Resynthesis for sequential circuits designed with a specified initial state. In *Proceedings of the IEEE VLSI Test Symposium*, pages 152–157, 1995.

[136] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita. Synthesis for testablity by sequential redundancy removal using retiming. In *International Symposium on Fault Tolerent Computing*, pages 33–40, 1995.

[137] Y. Zhang, M. Yu, and Y. Ye. A new retiming algorithm for cycle-time minimization in synchronous logic synthesis. In *Proceedings on the 4th International Conference on Solid State and IC Technology, Beijing*, pages 631–633, 1995.

BIOGRAPHICAL SKETCH

Naresh Maheshwari received the Bachelor of Engineering degree in Electronics Engineering from Motilal Nehru Regional Engineering College, Allahabad, India in 1992; the Master of Technology degree in Computer Science and Engineering from Indian Institute of Technology, Delhi in 1994; and the Doctor of Philosophy degree in Computer Engineering from Iowa State University in 1998. He worked at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY in summer of 1996, and at IBM's AS/400 Division, Rochester, MN in summer of 1997. His research interests include high level and logic synthesis, timing analysis and optimization, and formal verification. He has published several technical papers in the area of circuit optimization, and he is a recipient of best paper award at the 1997 Design Automation Conference, and Lucent Technologies DAC Graduate Scholarship. He received a research excellence award for his doctoral work at Iowa State University.