

# **A Fast Incremental Clock Skew Scheduling Algorithm for Slack Optimization**

Kui Wang, Hao Fang, Hu Xu, Xu Cheng  
Microprocessor research and development  
center of Peking University, Beijing

# What is clock skew scheduling

- Take the clock arrival time to each flip-flop as a manageable resource
- It can promote many kinds of optimizations
- But usually for timing
- It is friendly to ASIC flow
  - We design synthesizable processor cores and SoCs
  - So we are interested.

# CSS for timing

- Period optimization
  - Achieve the minimum clock period
  - Stop when critical paths construct a loop
- Slack optimization
  - Enlarge the slacks of critical paths, wherever possible
  - Balance the slacks of critical paths and their neighbors
- Utilizing period opt. algorithms for slack opt.
  - We presented the approach at DAC'06
  - We ignore the hold time constrains, **safely**

# Today's Topic: Speed up CSS

- How?
  - Faster graph-theoretic CSS algorithms
  - Partial delay extraction
- Why?
  - Shorter turn-around time
  - Integration with front-end tools

# Previous CSS algorithms

- Binary search for the minimum clock period
  - Takahashi, IEICE'06,  $O(jm+j^2n)$
  - $j$ : number of arcs in a shortest trail
- Incrementally decrease the clock period
  - Burns, ICCAD'03,  $O(n^2m)$
  - Albrecht, DATE'06,  $O(nm+n^2\log n)$
- Partial delay extraction
  - Albrecht, DATE'06

## We want:

- A linear-time CSS algorithm
  - Like Takahashi's algorithm
  - Solves the period optimization problem
- Allows partial delay extraction
  - Like Albrecht's algorithm
  - Linear-time for delay extraction
- Easy to extend to slack optimization
  - Still linear-time

# The key of our approach

- A CSS algorithm whose iteration count has a constant upper-bound!
  - The time complexity of each iteration is linear
  - So the whole algorithm is linear-time
- The constant upper-bound also resulted in an efficient partial delay extraction strategy
  - A linear-time delay extraction is performed after each iteration

# The key of our algorithm

- We use discrete values (integers)
  - Ease the design and analysis of our algorithm
  - The error is no larger than one delay unit
  - The error of path delays caused by inaccurate RC estimation is usually tens of picoseconds in pre-route designs (our design experience)
  - Remember CSS is before CTS, no detailed route information is available.



# Problem formulation with integers

- Period optimization
  - Find a minimal clock period under which
  - some 1-slack and 0-zero slack arcs construct a loop
- Slack optimization
  - For each node, the minimum incoming slack and outgoing slack are all large enough, or
  - their difference is no larger than one

# The basic idea of CSS

- Slack passing or timing borrow
- $\text{slack} = T_{\text{period}} - T_{\text{path}} - L_{\text{start}} + L_{\text{end}}$



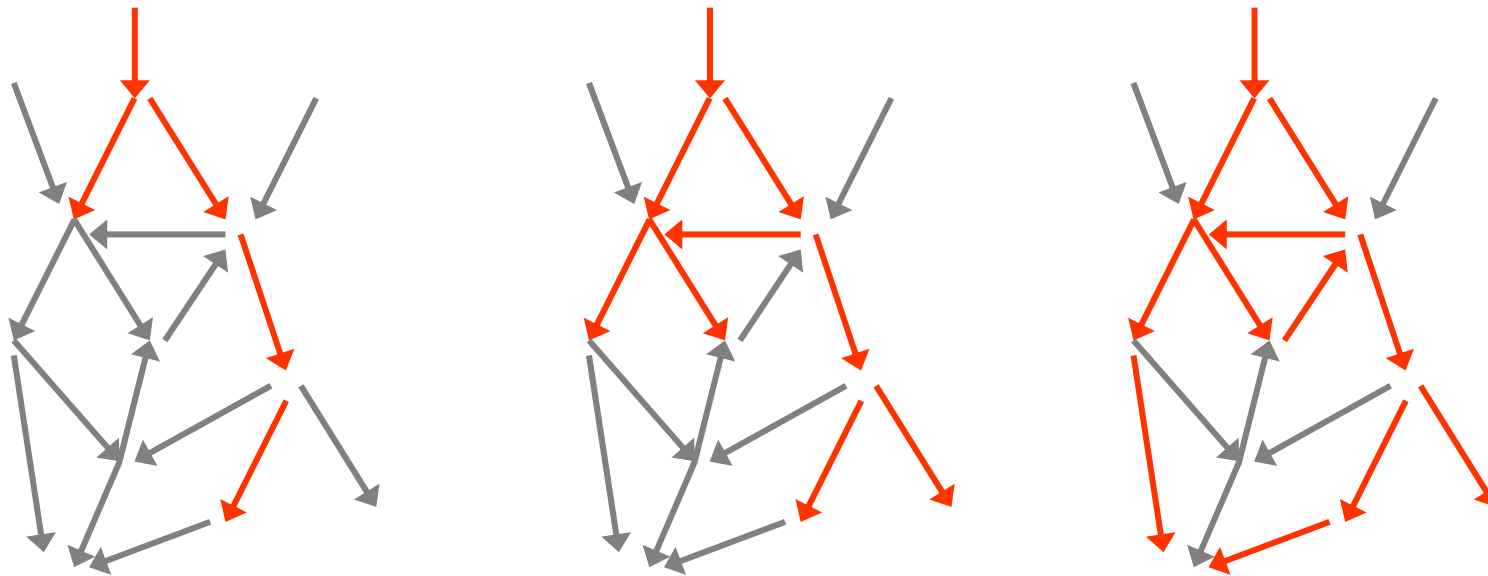
$$\begin{aligned}\text{slack}(u,v) &= -1 \\ \text{slack}(v,w) &= 1\end{aligned}$$



$$\begin{aligned}\text{slack}(u,v) &= 0 \\ \text{slack}(v,w) &= 0\end{aligned}$$

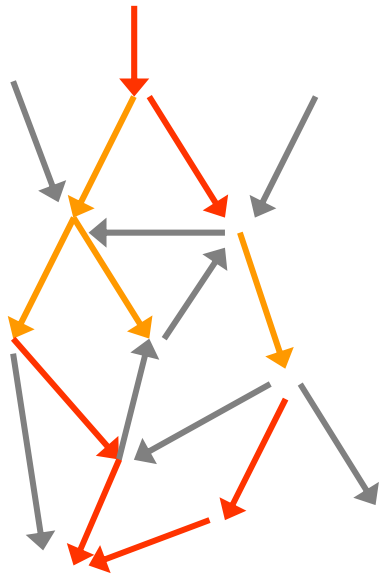
# The basic idea of CSS

- Let the DAG with critical arcs grow!
  - Add new arcs to the DAG to reduce its average arc delay
  - The slacks of the new-added arcs are “borrowed”
  - Until it is no longer a directed acyclic graph



## How does our DAG look like

- Both 1-slack arcs and 0-slack arcs are critical arcs



Maximum level=4

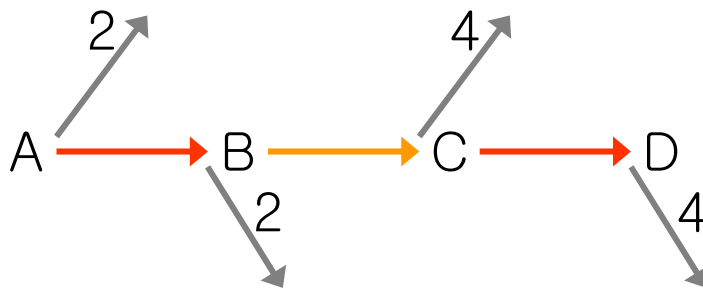
- Maximum level: the maximum number of 0-slack arcs in a path from source to sink

# Overview of our algorithm

- Step1: In reversed topological order, calculate the budget for the increment of the clock latency
- Step2: In topological order, increase the clock latencies to change 0-slack into 1-slack
- Step3: If there is no 0-slack arc, decrease the clock period, changing 1-slack into 0-slack
- Iterate over these steps until a cycle of critical arcs are discovered.

## Step1: the budget for increasing latency: $\Theta$

- Constraints from two kinds of arcs
  - Critical arc  $(u,v)$ :  $\Theta'$ 
    - $\text{budget}(u)=\text{budget}(v)$  if  $\text{slack}(u,v)=1$
    - $\text{budget}(u)=\text{budget}(v)-1$  if  $\text{slack}(u,v)=0$  Make it 1-slack arc!
  - Non-critical arc:  $\Theta_B$ 
    - As long as it will not be a 0-slack arc.
    - 1-slack is OK
  - $\Theta'$  is decided by  $\Theta_B$  of some downstream node



$$\theta(A)=0$$

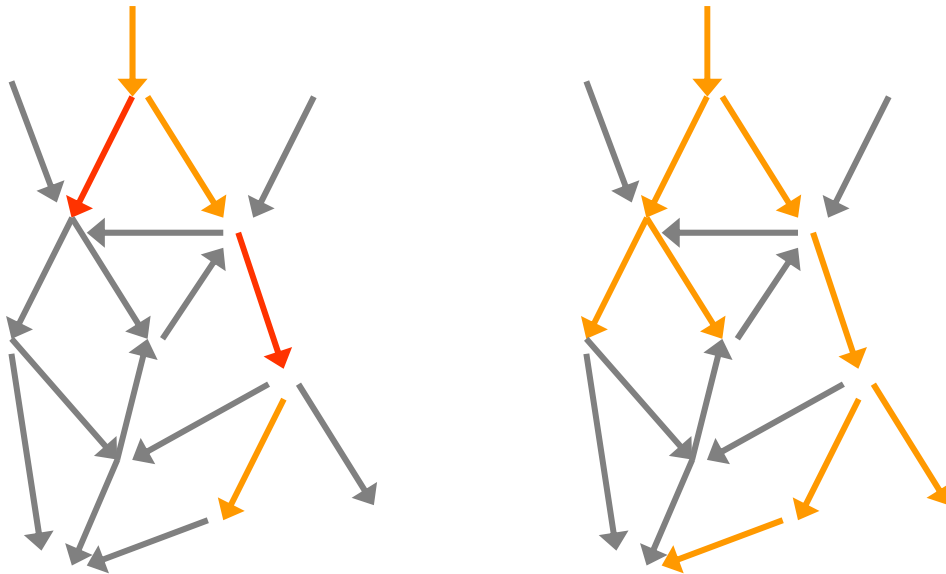
$$\theta(B)=1$$

$$\theta(C)=2$$

$$\theta(D)=3$$

## Step2: increasing the latencies

- If the budget allows it, increase the clock latency of  $v$ , to turn the 0-slack arcs ending at  $v$  to 1-slack
- A node is live if at least one of its upstream arcs is 0-slack. Dead nodes need no visit in this step.



## Why is this algorithm faster?

- Each leaf live node “grows” or dies!
- If budget is enough to eliminate 0-slack arcs, it dies
- If budget is not enough, some non-critical outgoing arc will be 1-slack and be added to the DAG.
- At each iteration, multiple arcs are added to DAG!
- In Burns’ algorithm and Albrecht’s algorithm, only one arc for each iteration



# Complexity Analysis

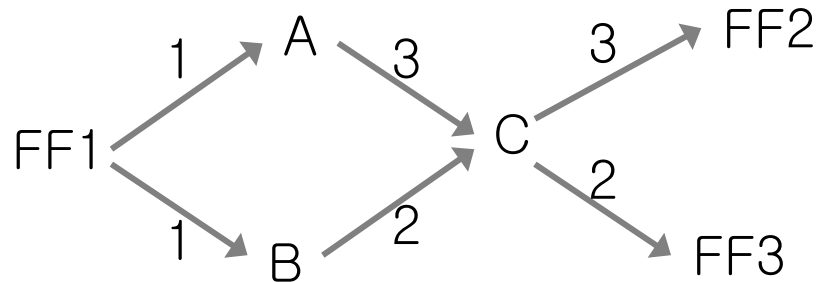
- If budgets are all enough, 0-slacks are eliminated at one iteration
- If budgets are not enough, at least each new-added arc will “counteract” one 0-slack arc
- The maximum level decreases at each iteration
- $k$ : the upper bound for the maximum level
  - In real circuits, it can be looked as a constant
- After at most  $k$  iterations, period is decreased by 1
- Clock period also has a constant upper bound
  - So the total iteration count has a constant upper bound

# Partial delay extraction

- Our recent work, today just go through the basic ideas
- All the algorithm need to known is  $\theta_B$  (and which non-critical arc decides  $\theta_B$ )
- $\theta_B$  is decided by the longest non-critical path
  - Or we can name it as “the next longest path”
  - Or “the longest arc of the arcs we have not known”

## Two passes to compute $\Theta_B$

- $D_{fs}$  : maximum propagation delay from Q pins of flip-flops to a combinational cell
  - A pass in topological order
- $D_{te}$  : maximum propagation delay from a combination cell to D pins of flip-flops, **following a non-critical path** ( $D_{fs} + D_{te} < \text{threshold}$ )
  - A pass in reversed topological order

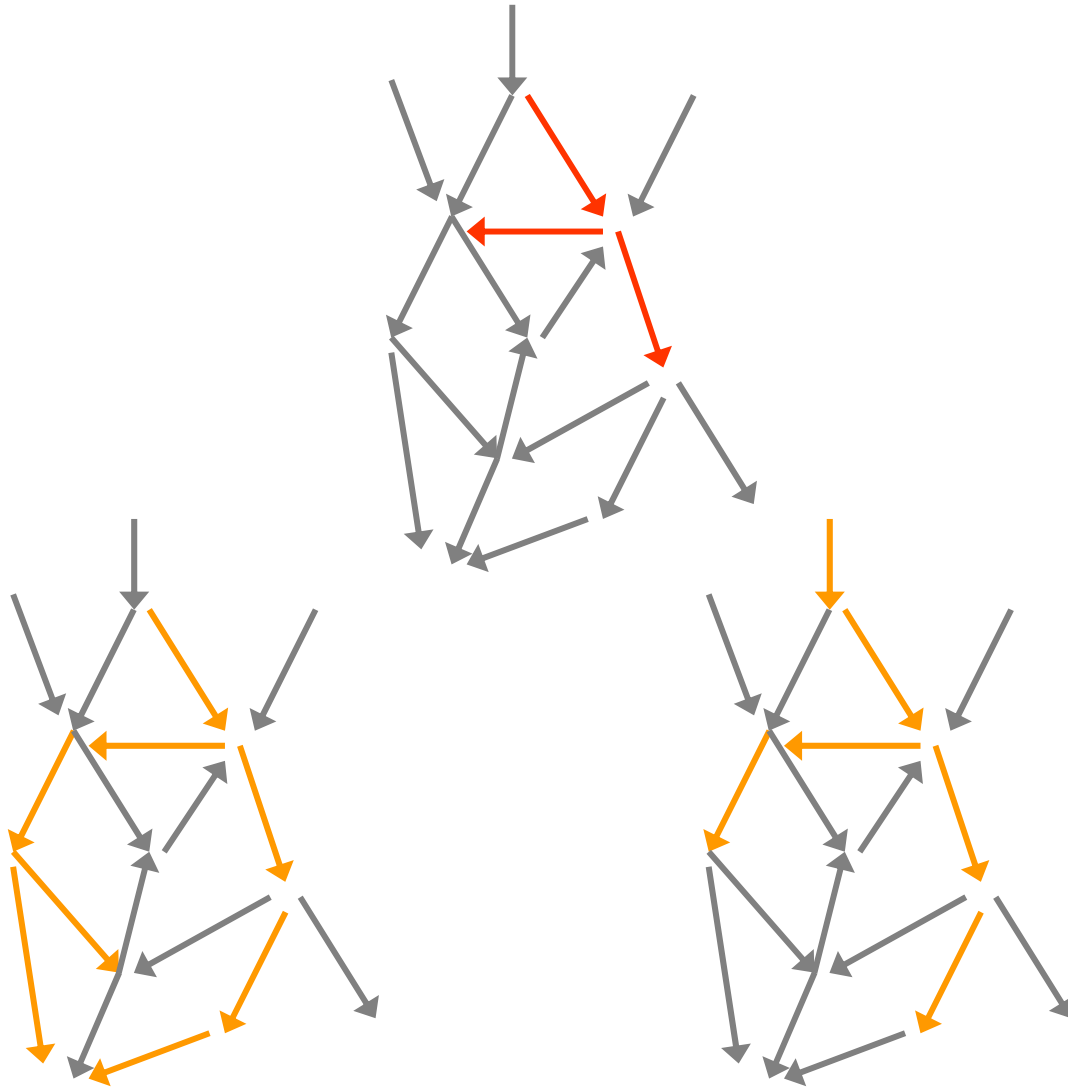


Threshold=7  
 $D_{fs}(C)=4$   
 $D_{te}(C)=2$   
 $D_{te}(FF1)=6$   
 $\Theta_B(FF1)=7-6=1$

# Linear time for delay extraction

- Each pass is linear-time
- Total iteration count has constant upper bound
- Actually, updating  $\theta_B$  does not need two FULL passes
- Only a few of nodes change their latencies , only a few gates need to change their  $D_{fs}$  and  $D_{te}$

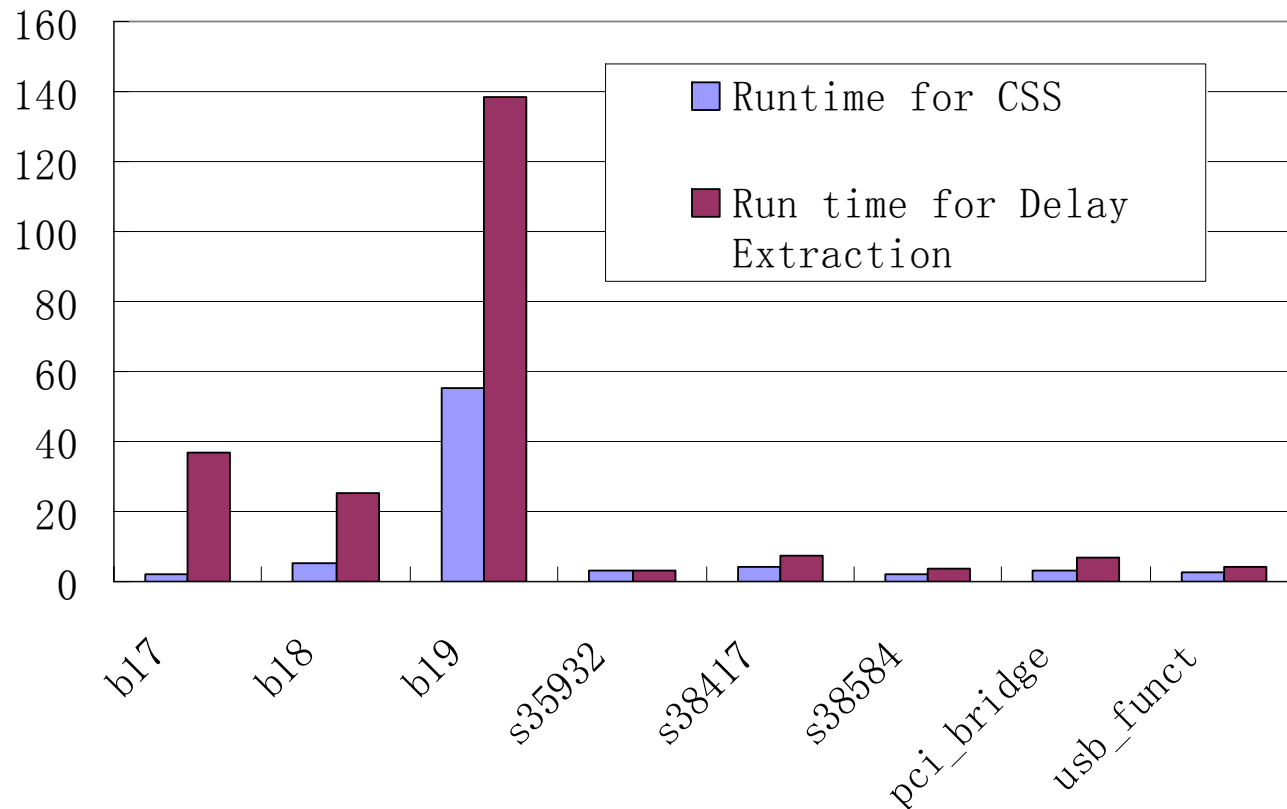
# Let the DAG grow at both directions



Make less  
change  
to clock  
latencies!

# Experimental results

- We can now reduce the total runtime of CSS to minutes, for the biggest block in IWLS2005, b19
- The synthesis process takes one hour for b19



**Thank You!**