Spec-based flip-flop and latch planning

Manch Hon

Problem Statement

This paper addresses “fast” and “intelligent” placement of “clocked repeaters”

Why do we need it?

Scaling trend implies

- Frequency increases
- Metal distance per clock decreasing faster than repeater-repeater distance
- More clocked repeaters
- Intel study predicts 25% repeaters clocked at 45nm node

Need automation to optimize clocked repeaters placement

Previous Approaches

Requirements:

- Respect latency constraints set a priori
- Handles multi-pin nets
- Fast
- Simple to code

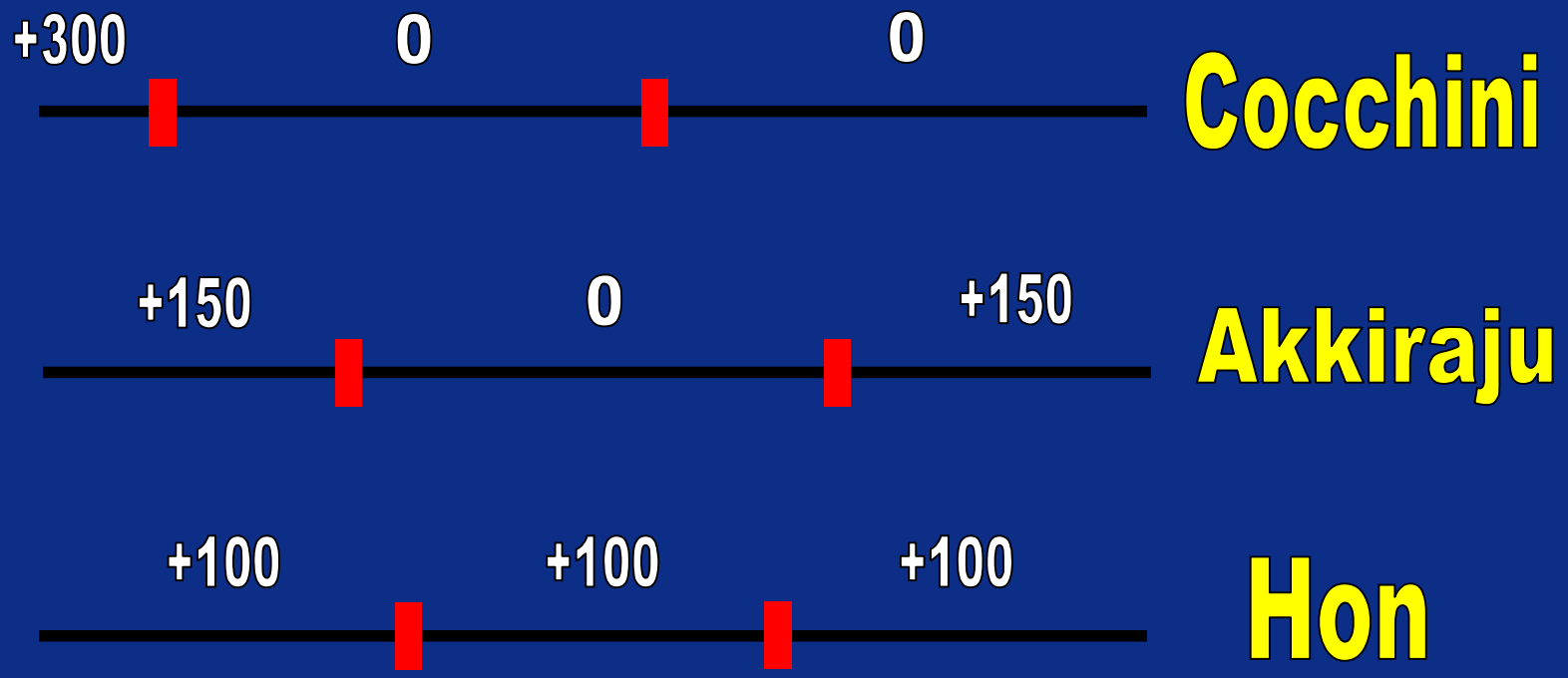
Two approaches stand out (both from Intel):

1. Cocchini (2002)
2. Akkiraju and Mohan (2003)

Differs in the way they handle margins

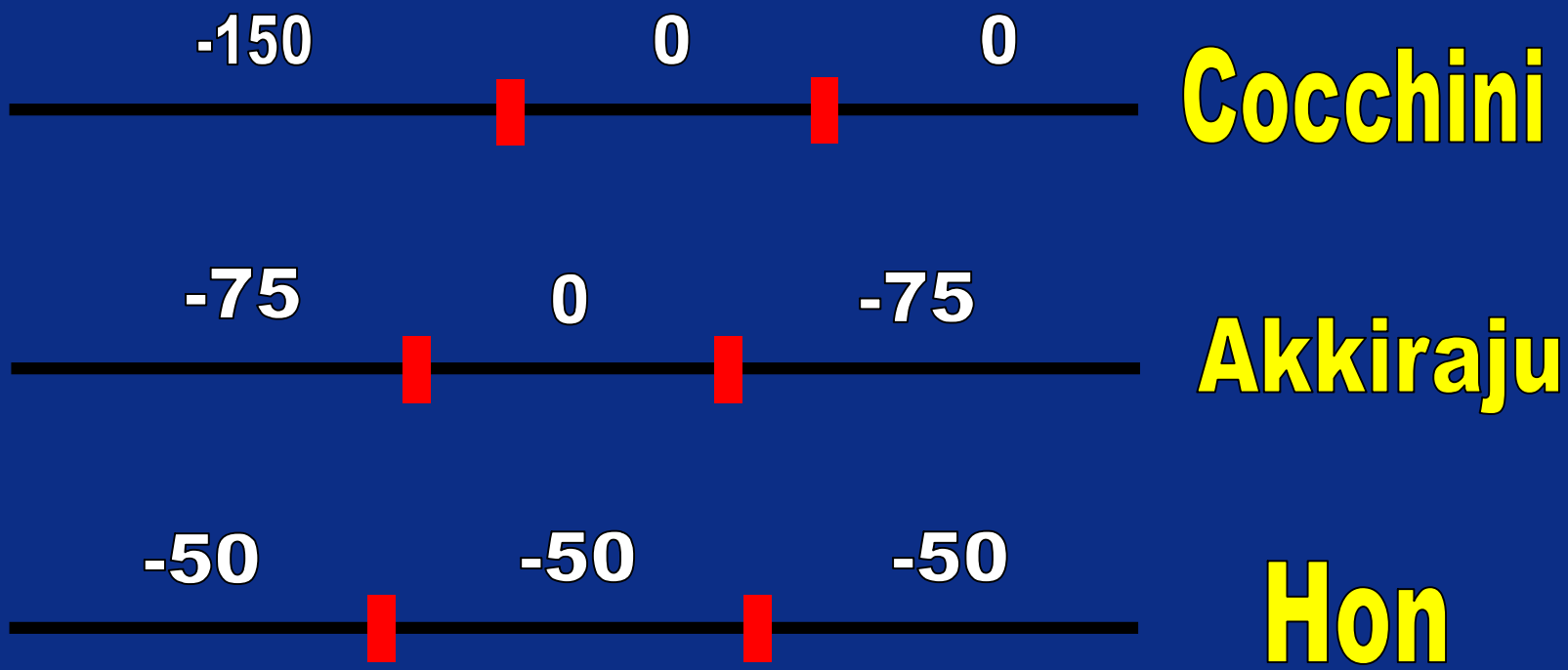
Previous Approaches (cont)

Positive Margin Case (Clk = 400ps)



Previous Approaches (cont)

Negative Margin Case (Clk = 250ps)



Our Contribution

This paper addresses “fast” and “intelligent” placement of “clocked repeaters”

“Fast”: O(n log n) runtime

“Intelligent”: even out margins

- Timing easier to fix
- Frequency target easier to raise

“clocked repeaters”: flip-flops and 2-phase symmetric latches

Observation: Minimizing period leads to more even margin.

Retiming!!

Problem Setup

Tree T

- Inteconnect modeled as trees
- Nodes – driver, receivers, candidate locations for clocked repeaters
- Edges – wires

Latency requirement l(v)

- One for each receiver
- Non-negative Integer
- Number of flops from driver to v

Problem Setup (cont)

Margin requirement m(v)

- One for each receiver
- Signal must arrive m(v) units of time before clk fall edge

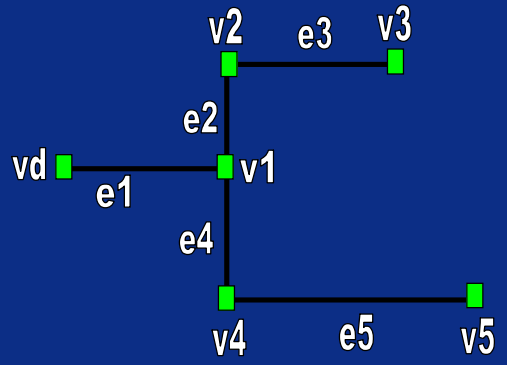
Wire Delay

- Proportional to $\sqrt{}$(R*C) because of buffering
- Experiments measure delay optimally-buffered wires at various lengths on different metal layers
- Curve-fit to a*$\sqrt{}$(R*C) + b

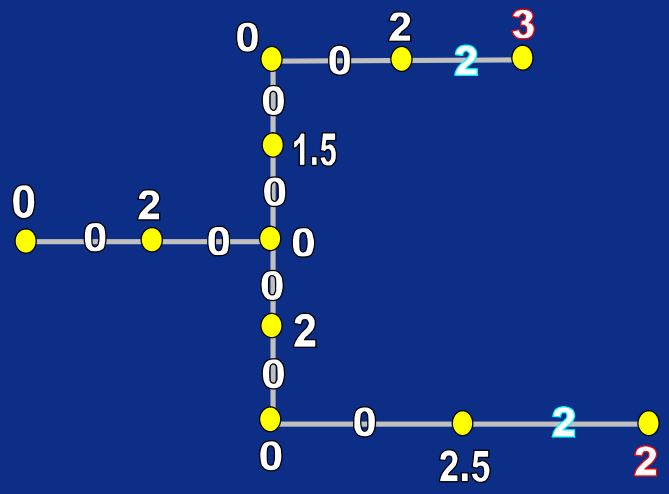
Cell Delay

- Modeled as one single constant number d_{cell}

Graph Translation



d(e1)	2	m(v3)	3
d(e2)	1.5	m(v5)	2
d(e3)	2		
d(e4)	2	l(v3)	2
d(e5)	2.5	l(v5)	2



Flop Insertion Algorithm Outline

- Works off translated graph
- Binary search over clock periods
- For each clock period c, tests its feasibility by calculating the retiming label r(v)
- # flops on edge uv = r(v) – r(u) + w(uv)
- Traverse from driver down, add node weight to counter
- Whenever counter exceeds c, increment r and reset counter
- c is feasible if
  - Calculated period <= c on retimed circuit
  - r(v) – r(u) + w(uv) >= 0 for all edge uv
  - r(driver) = r(receiver) for all receiver

Flop Insertion Algorithm Details

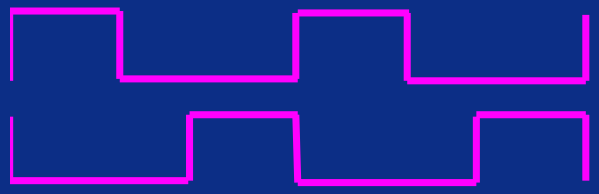
```
Tree_Flop_Insert(T,ε)
/* Tree Flip-Flop Insertion
 * Input:  Tree T(V,E), rel error ε
 * Output:  Retiming label r*/
1    d_max ← max_{v∈V} d(v)
2    c_hi ← max_{v_r} d(v_d ⇝ v_r)
3    c_lo ← max_{v_r} {d(v_d ⇝ v_r)/(l(v_r) + 1)}
4    r ← 0
5    while c_hi − c_lo > ε * d_max {
6        c ← (c_hi + c_lo)/2
7        if Tree_Flop_Clk_Feas(T,r,c) = true
8        then c_hi ← c
9        else c_lo ← c
10   }
11   return r
```

Flop Insertion Algorithm Details (cont)

```
Tree_Flop_Clk_Feas(T,r,c)
/* Input:  Tree T(V,E), label r, period c
 * Output:  Feasibility of c */
1    for each v ∈ V  r(v) ← 0
2    for v ∈ V in topological order {
3        Compute δ(v)
4        if δ(v) > c
5            then r(v) ← r(parent(v)) + 1
6                 δ(v) ← d(v)
7            else r(v) ← r(parent(v))
8    }
9    Compute retimed edge weights
10   if max_{v∈V} δ(v) > c
11       then no feasible retiming
12       else the current r is legal
```

Latch Insertion Algorithm Outline

- Two-phase symmetric clocking scheme



- Properly timed if latches hold same value even when gates/wires have zero delay
- Clock feasibility can be checked in linear time by modified formula of Ishii, Leiserson and Papaefthymiou
- Binary search over clock periods

Experiment setup

- 3 algorithms compared:
  - SBFIA (Akkiraju & Mohan)
  - RFLOP
  - LATCH
- Test on block from Xeon microprocessor, 90nm design
- 1769 multi-cycle nets, fanouts 1 to 34
- Candidate nodes inserted every 100 micron
- Relative error set to 0.001
- Run on Xeon machine with 4x2.8GHz/512KB/8GB

Experiment Result

Algorithm	Flops	Runtime (seconds)
SFBIA	14989	3104.38
RFLOP	14907	11.54
LATCH	15028.5	11.92

- RFLOP and LATCH much more efficient than SFBIA
- RFLOP uses fewer flops than SFBIA

Experiment Result (cont)

Algorithm	# flops	Avg Stage Spread (ps)
1-fanout (941)		
SBFIA	5766	175.96
RFLOP	5766	80.31
2-fanout (456)		
SBFIA	4968	253.46
RFLOP	4956	152.98
3-fanout (187)		
SBFIA	1877	256.91
RFLOP	1876	181.92
4-fanout to 6-fanout (145)		
SBFIA	1703	281.46
RFLOP	1671	200.73
7-fanout+ (40)		
SBFIA	675	319.72
RFLOP	638	224.37

Stage Spread = difference between max and min flop-flop delay

Experiment (cont)

Algorithm	Negative Slack			Avg Freq Gain
	Total	Worst	Nets	
1-fanout (941)				
SBFIA	-38242	-213	707	0
RFLOP	-253.2	-16.6	192	19.36%
LATCH	0	0	0	29.67%
2-fanout (456)				
SBFIA	-17860	-220	413	0
RFLOP	-69.1	-6.3	59	17.13%
LATCH	0	0	0	25.53%
3-fanout (187)				
SBFIA	-7704	-181	168	0
RFLOP	-278.6	-15	44	17.45%
LATCH	0	0	0	26.48%

Algorithm	Negative Slack			Avg Freq Gain
	Total	Worst	Nets	
4-fanout to 6-fanout (145)				
SBFIA	-7757	-190	142	0
RFLOP	-335.8	-15.8	60	21.66%
LATCH	0	0	0	29.27%
7-fanout + (40)				
SBFIA	-3312	-169	40	0
RFLOP	-119.4	-13.2	22	32.36%
LATCH	0	0	0	39.58%