Period and Glitch Reduction Via Clock Skew Scheduling, Delay Padding and GlitchLess

FPT 2009

Xiao Patrick Dong

Supervisor: Guy Lemieux

Introduction/Motivation

- Goal:
  - Reduce critical path ➔ shorter period
  - Decrease dynamic power

Introduction/Motivation

- Goal:
  - Reduce critical path → shorter period
  - Decrease dynamic power
- Approach:
  - Add clock skew at FFs → clock skew scheduling (CSS)
  - Relax CSS constraints → delay padding (DP)
  - Reduce power due to glitching → GlitchLess (GL)

Introduction/Motivation

- Goal:
  - Reduce critical path ➔ shorter period
  - Decrease dynamic power
- Approach:
  - Add clock skew at FFs ➔ clock skew scheduling (CSS)
  - Relax CSS constraints ➔ delay padding (DP)
  - Reduce power due to glitching ➔ GlitchLess (GL)
- Implementation:
  - One architectural change, to satisfy all 3 above
  - Add programmable delay elements (PDE) to clocks
    - For every FF (best QoR)
    - For every CLB (best area)

Contributions

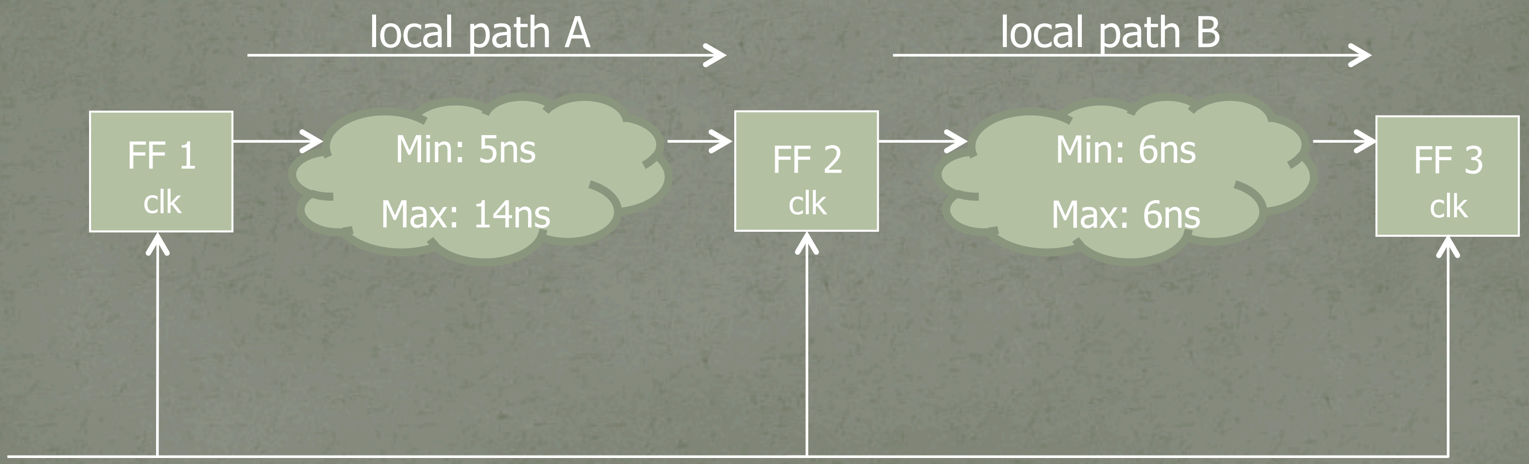
- One architectural change, to satisfy
  - CSS
  - DP
  - GlitchLess
- Delay Padding for FPGAs – first time
- Improved glitch modelling
- GlitchLess allows period increase
- Investigates period, power and area tradeoffs
- PDE sharing
- This presentation
  - Considers GlitchLess only, or CSS/DP only

Outline

- Introduction/Motivation
- Concept
- Implementation
- Results
- Conclusion
- Future Work

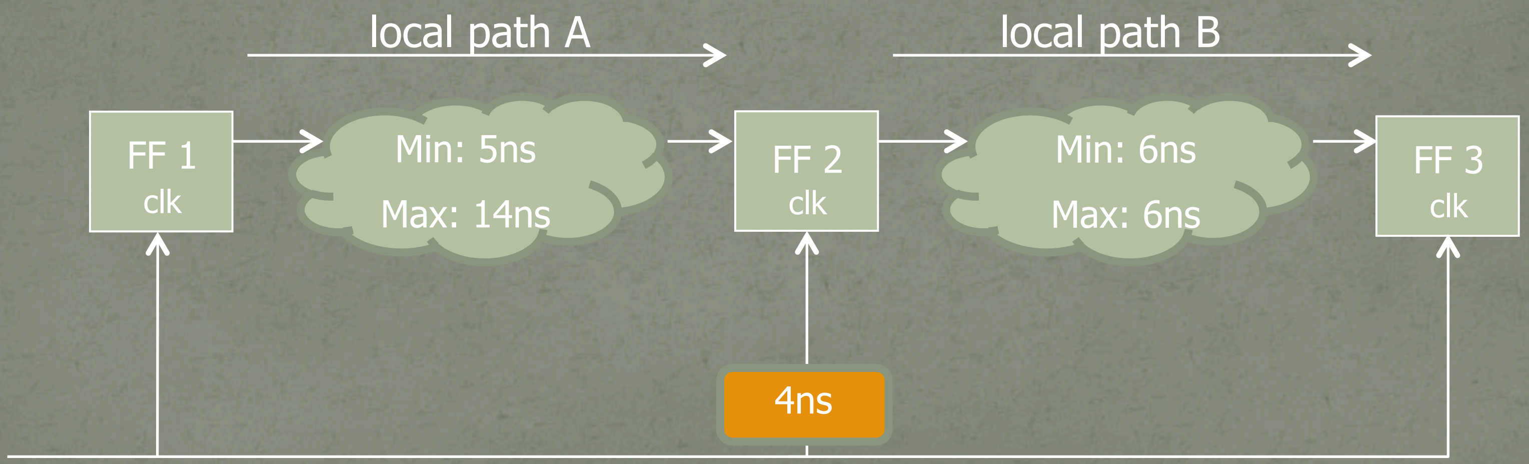
Concept – CSS

- Before:
  - 14-ns critical path delay



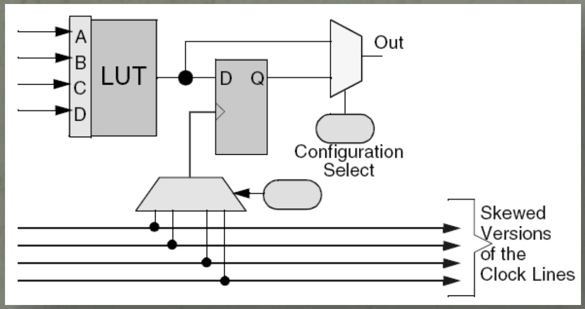
Concept – CSS

- Before:
  - 14-ns critical path delay
- After:
  - 10-ns critical delay – borrowed time



Concept – CSS

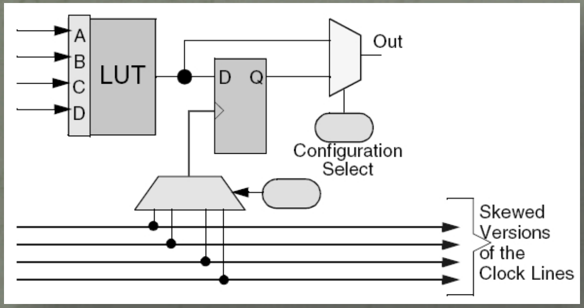
- How to implement CSS?



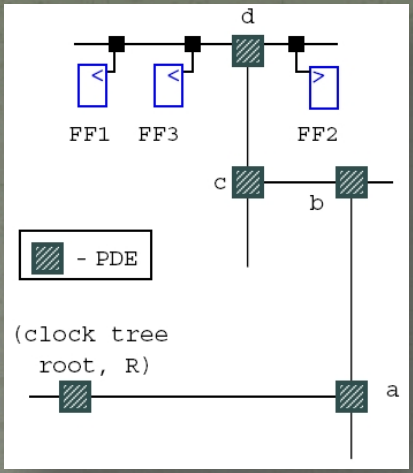
FPGA 2002: Brown

Concept – CSS

- How to implement CSS?



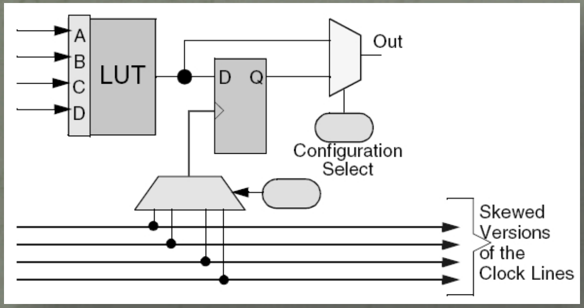
FPGA 2002: Brown



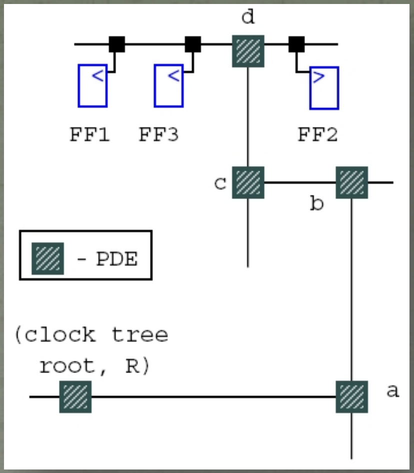
FPGA 2005: Sadowska

Concept – CSS

- How to implement CSS?



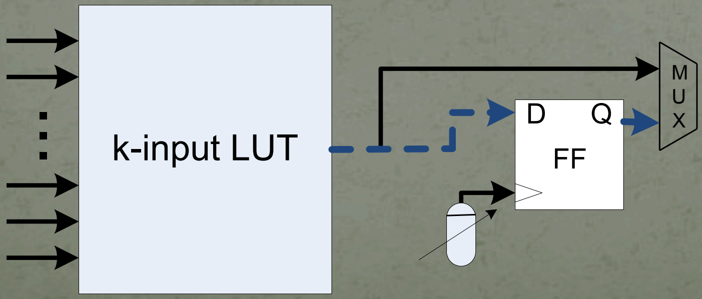
FPGA 2002: Brown



FPGA 2005: Sadowska

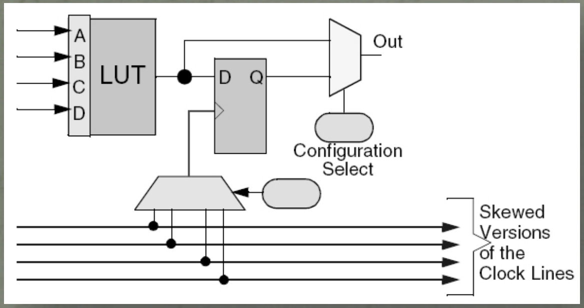
- Our 2 approaches

1 PDE for every FF

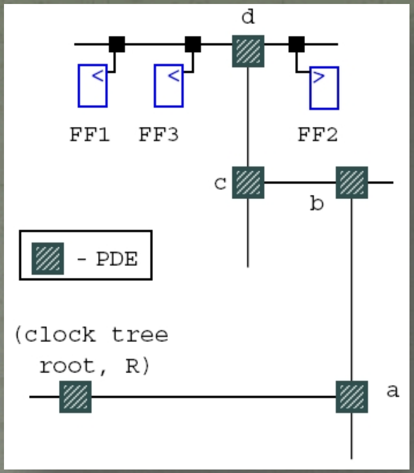


Concept – CSS

- How to implement CSS?



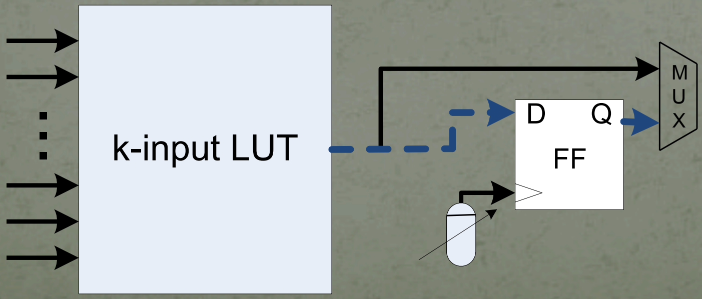
FPGA 2002: Brown

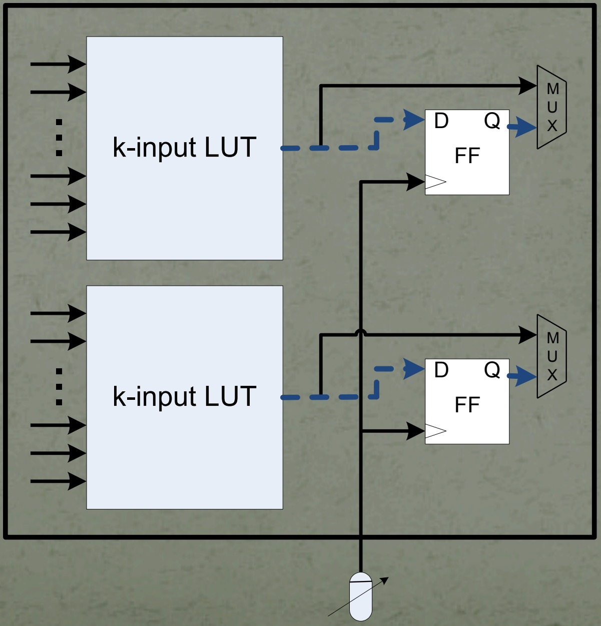


FPGA 2005: Sadowska

- Our 2 approaches

1 PDE for every FF





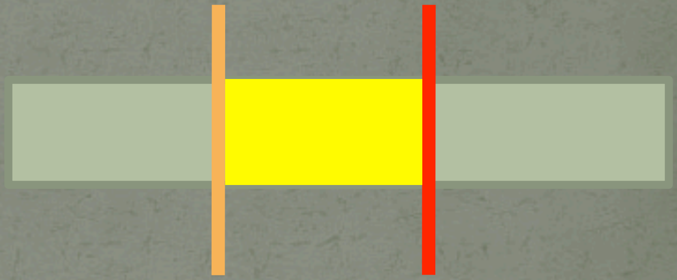
1 PDE for every CLB

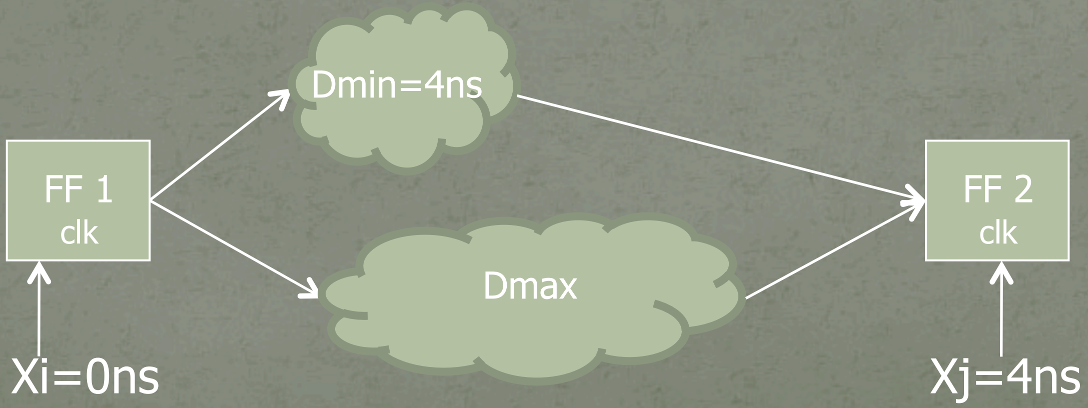
Concept - DP

- CSS constraints on permissible range of skew settings for Xi, Xj
- Increase permissible range
  - Cannot decrease Dmax
  - Increase Dmin

$$\boxed{x_j - x_i \le D_{\min} - T_{hold}}$$

$$\boxed{x_j - x_i \ge T_{setup} + D_{\max} - P}$$



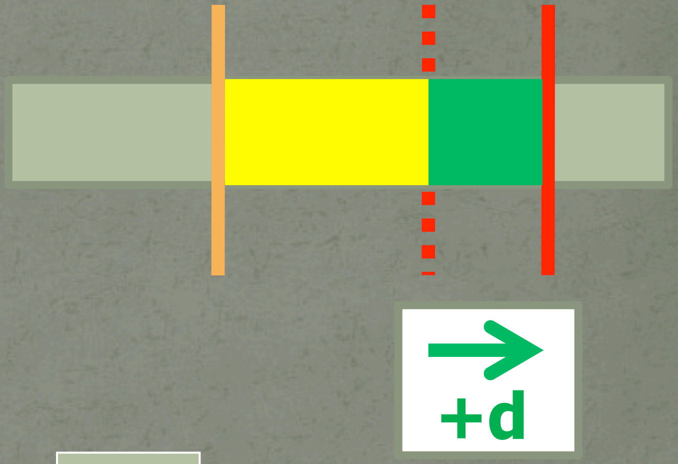


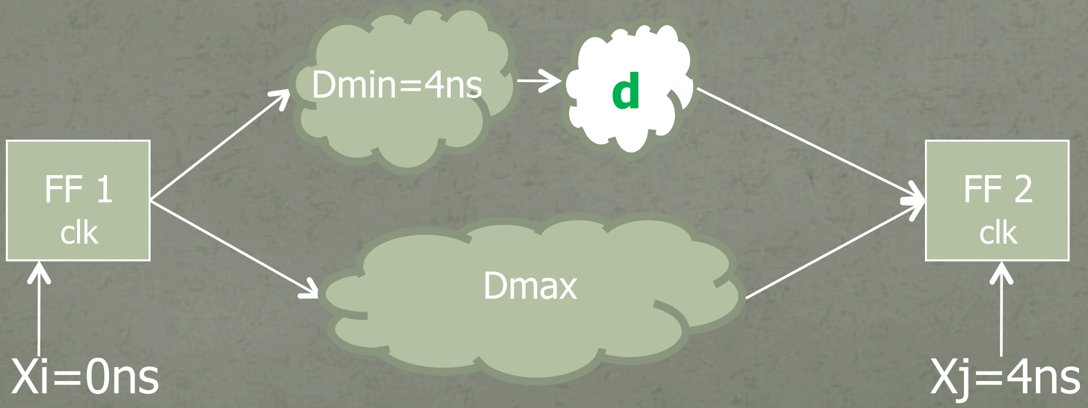
Concept - DP

- CSS constraints on permissible range of skew settings for Xi, Xj
- Increase permissible range
  - Cannot decrease Dmax
  - Increase Dmin by d

$$x_j - x_i \le D_{\min} - T_{hold} + d$$

$$x_j - x_i \ge T_{setup} + D_{\max} - P$$





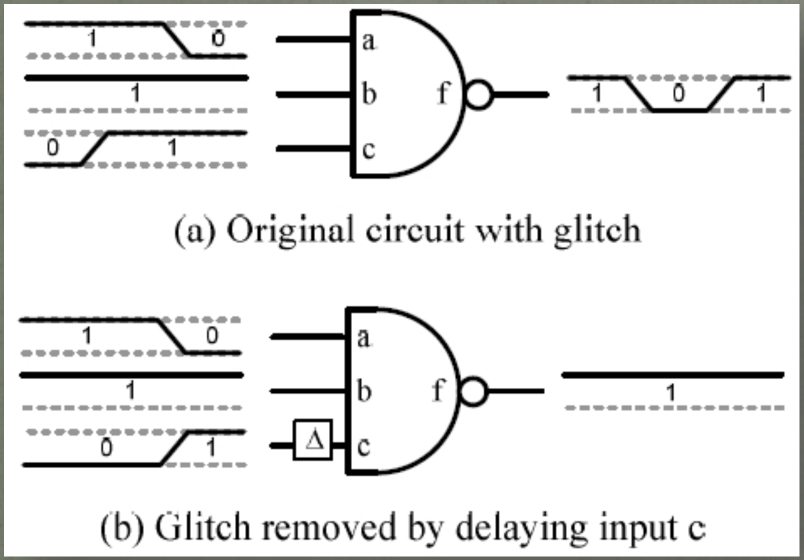
Concept – Feature Comparison

Feature		•ISCAS 1994 (Sapatnekar)	•FPGA 2002 (Brown) •FPGA 2005 (Sadowska) •FPL 2007 (Bazargan)	•ISPD 2005 (Kourtev)	•DAC 2005 (LU)	•FPT 2009 •(Our Approach)
CSS	Platform	ASIC	FPGA	ASIC	ASIC	**FPGA**
	Delays	continuous	discrete	continuous	continuous	**discrete**
	Variation modeling	✓	✓	✓	✗	✓
DP	Platform			ASIC	ASIC	**FPGA**
	Delays			continuous	continuous	**discrete**
	Variation modeling			✓	✗	✓
Algorithm		graph	graph	LP	graph	**graph**

Concept - GlitchLess

- Output fluctuate due to different input arrival

Past approach

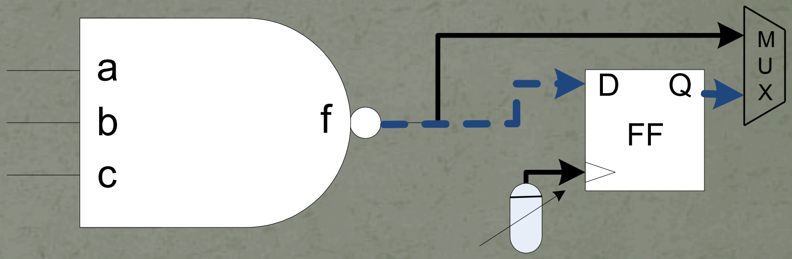


(a) Original circuit with glitch

(b) Glitch removed by delaying input c

TVLSI 2008: Lamoureux

Our approach



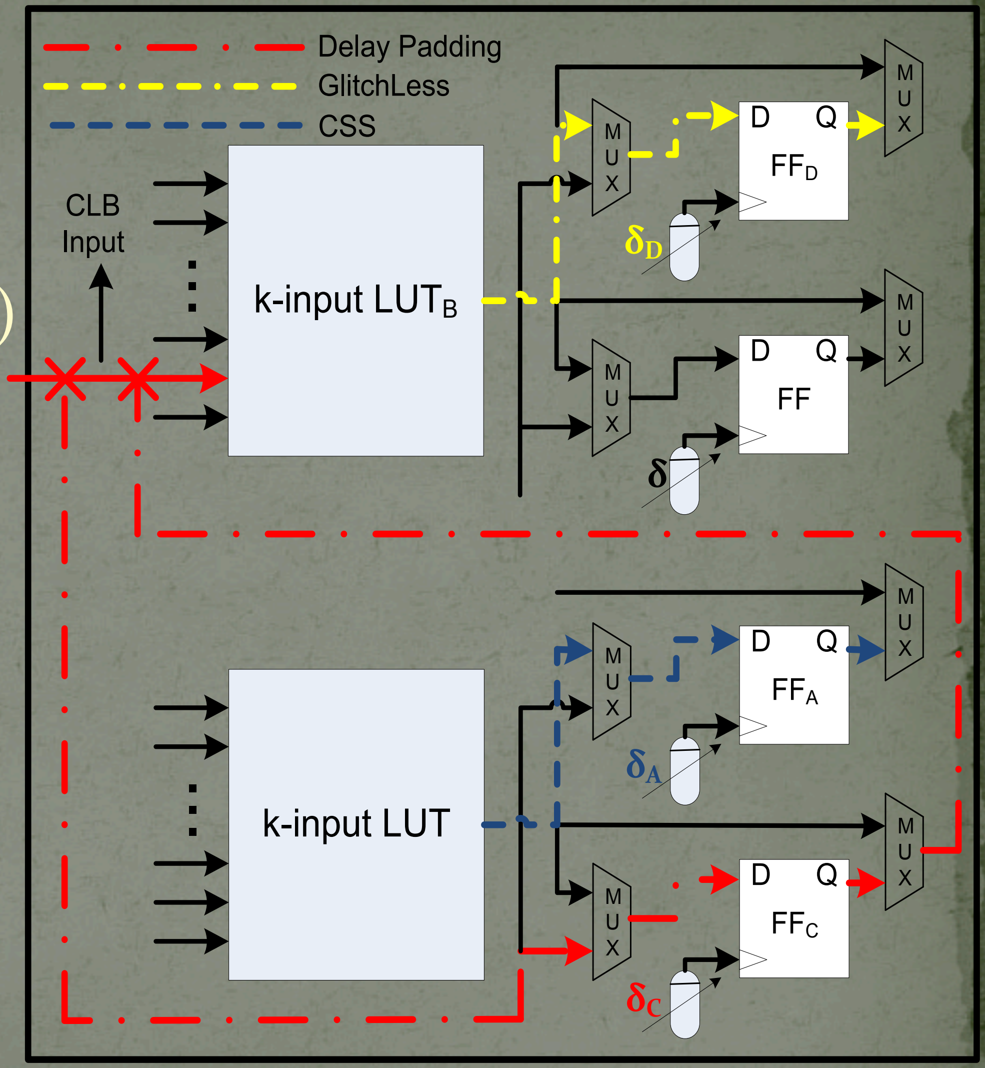
Outline

- Introduction/Motivation
- Concept
- Implementation
- Results
- Conclusion
- Future work

Architecture 1

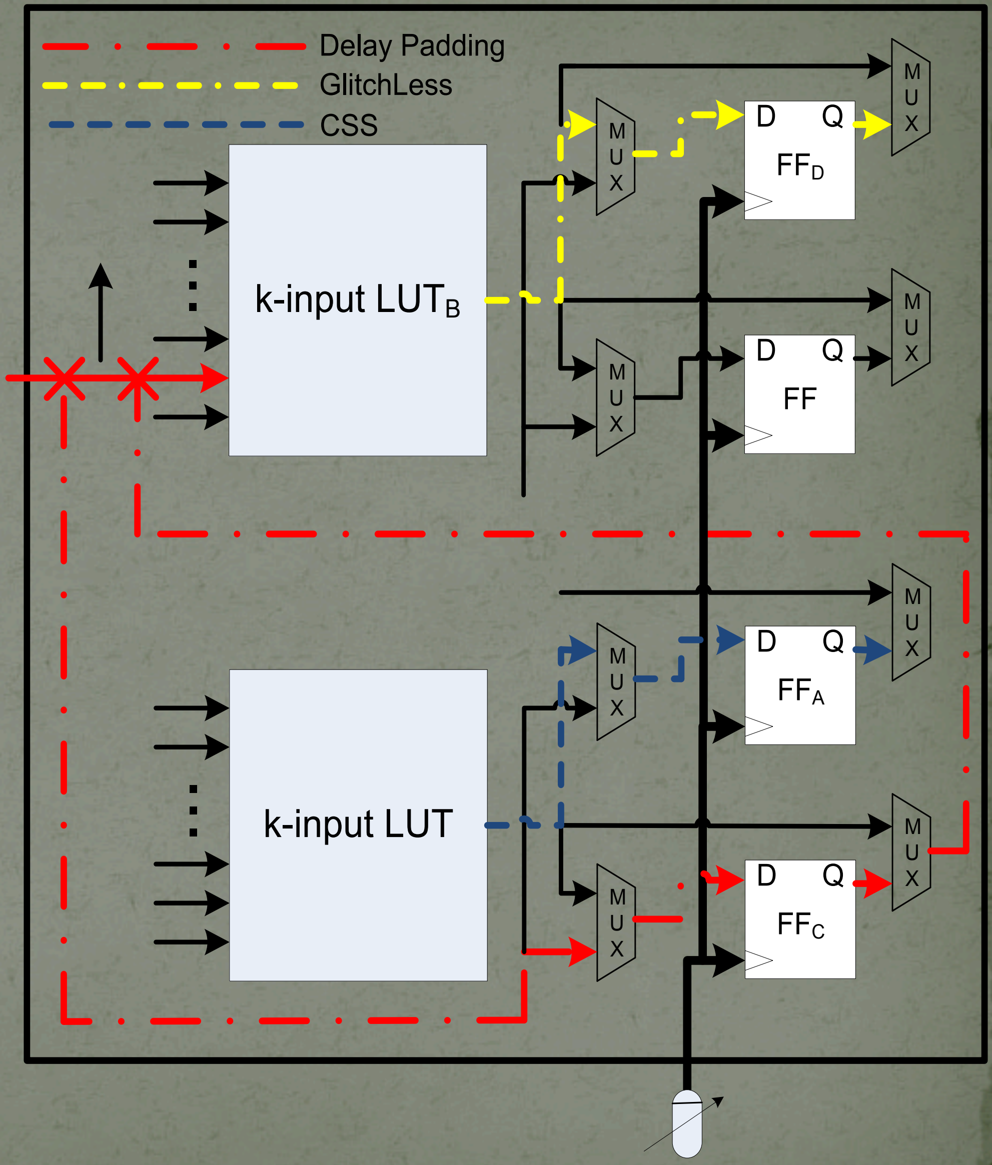
- 1 PDE per FF
  - 20 PDEs (2 FFs per LUT)
  - ~10% area cost

- CSS – add δ to FF clock
- DP – rerouting
- GL – insert FF on path



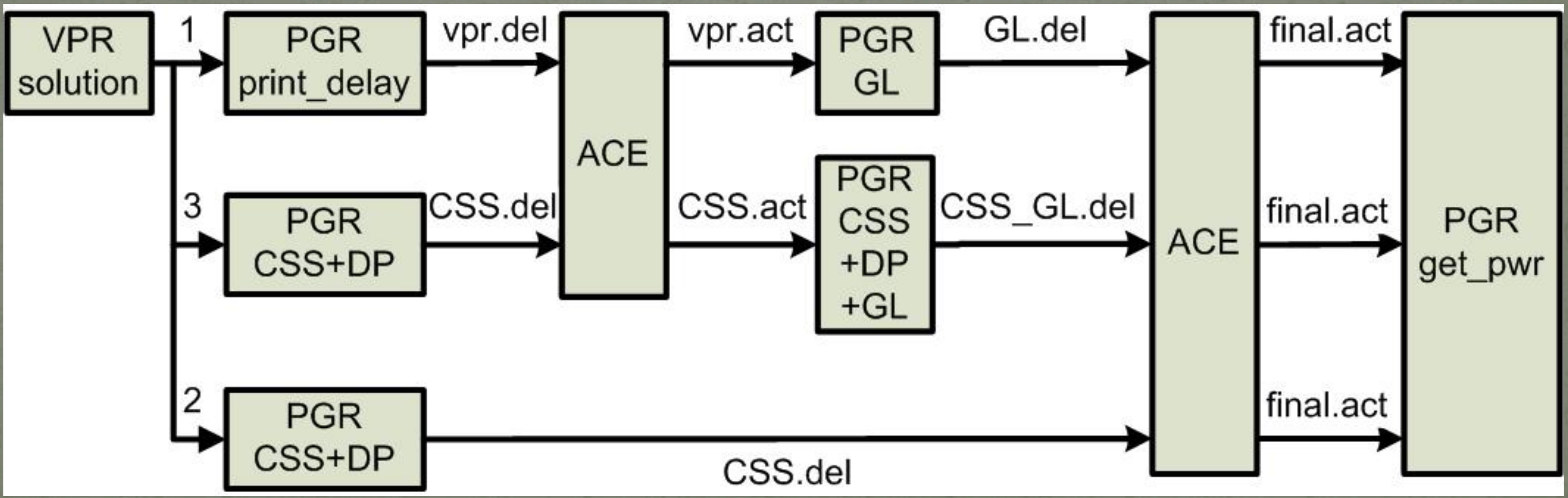
Architecture 2

- Objective: save area
  - 1 PDE per CLB
  - Share PDE with all FFs
  - ~0.5% area cost
- CSS – add δ to FF clock
- DP – rerouting
- GL – insert FF on path



Algorithm – Overall

- Two choices:
  - Choice “1”: GlitchLess Only
  - Choice “2”: CSS+DP
  - Choice “3”: CSS+DP+GlitchLess



Outline

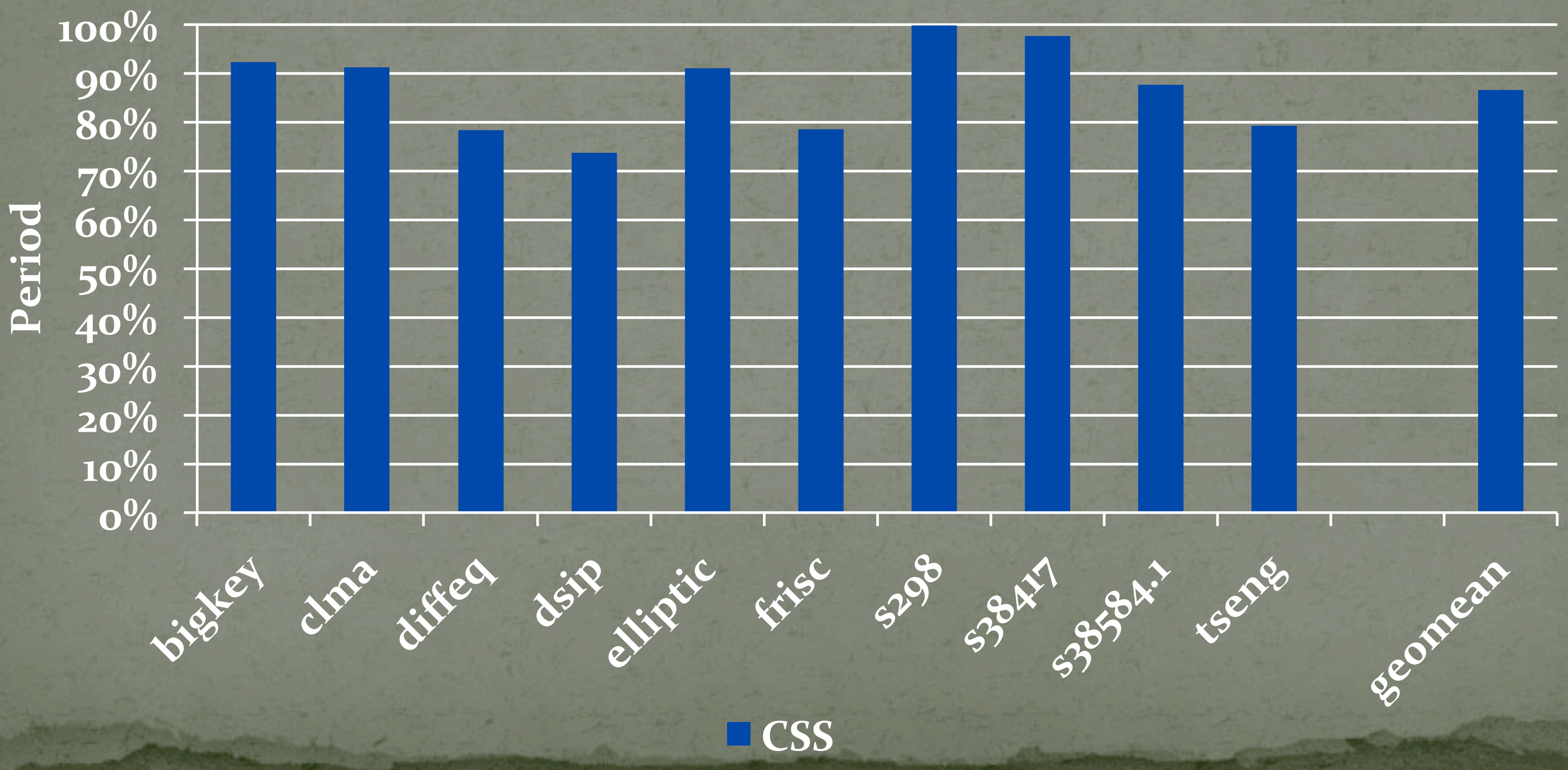
- Introduction/Motivation
- Concept
- Implementation
- Results
- Conclusion
- Future work

Results – Benchmarking

- 10 largest MCNC sequential circuits
- VPR 5.0: timing driven place and route
- route_chan_width = 104
- Architecture
  - 65nm technology
  - k=4, N=10, I=22
  - (k=6, N=10, I=33 not shown)
- Glitch estimation:
  - Modified ACE 2.0
  - 5000 pseudo-random input vectors

Results – CSS+DP Only

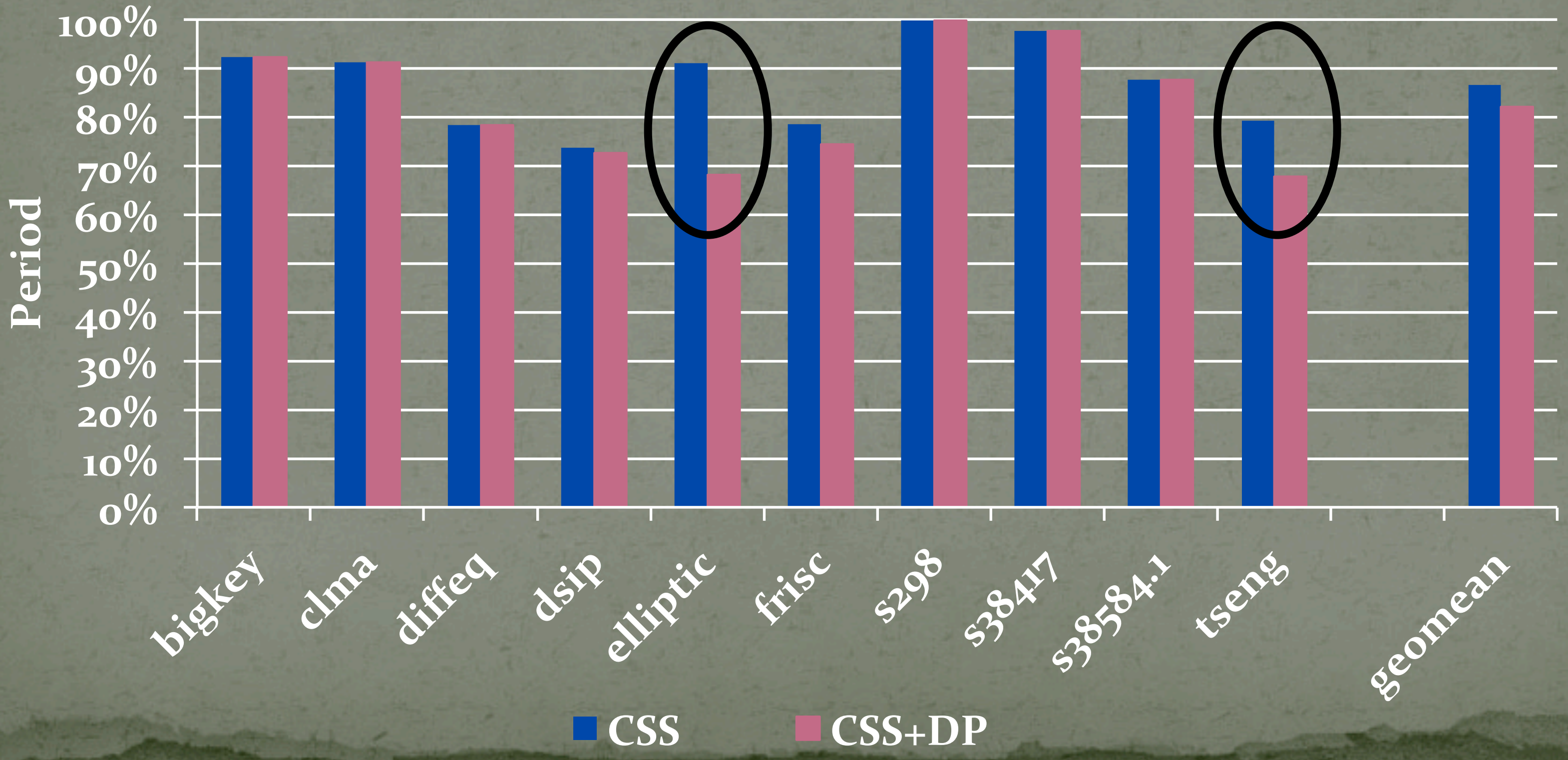
- All saving percentages are % of original period
- CSS geomean 13%



Results – CSS+DP Only

- All saving percentages are % of original period

- CSS geomean 13%
- CSS+DP geomean 18% (up to 32%)
  - Delay padding benefits 4 circuits (up to 23%)

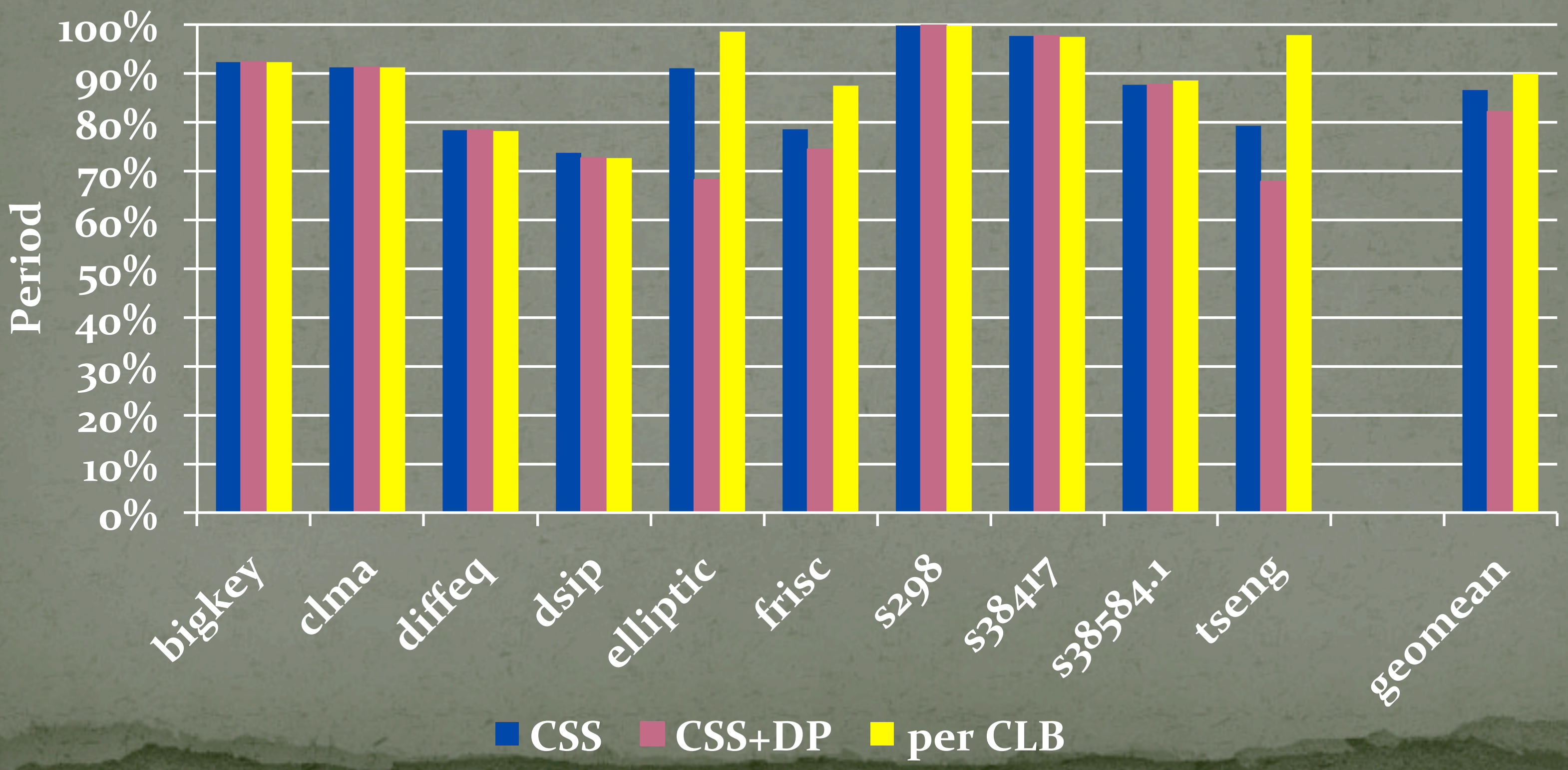


Results – CSS+DP Only

- All saving percentages are % of original period

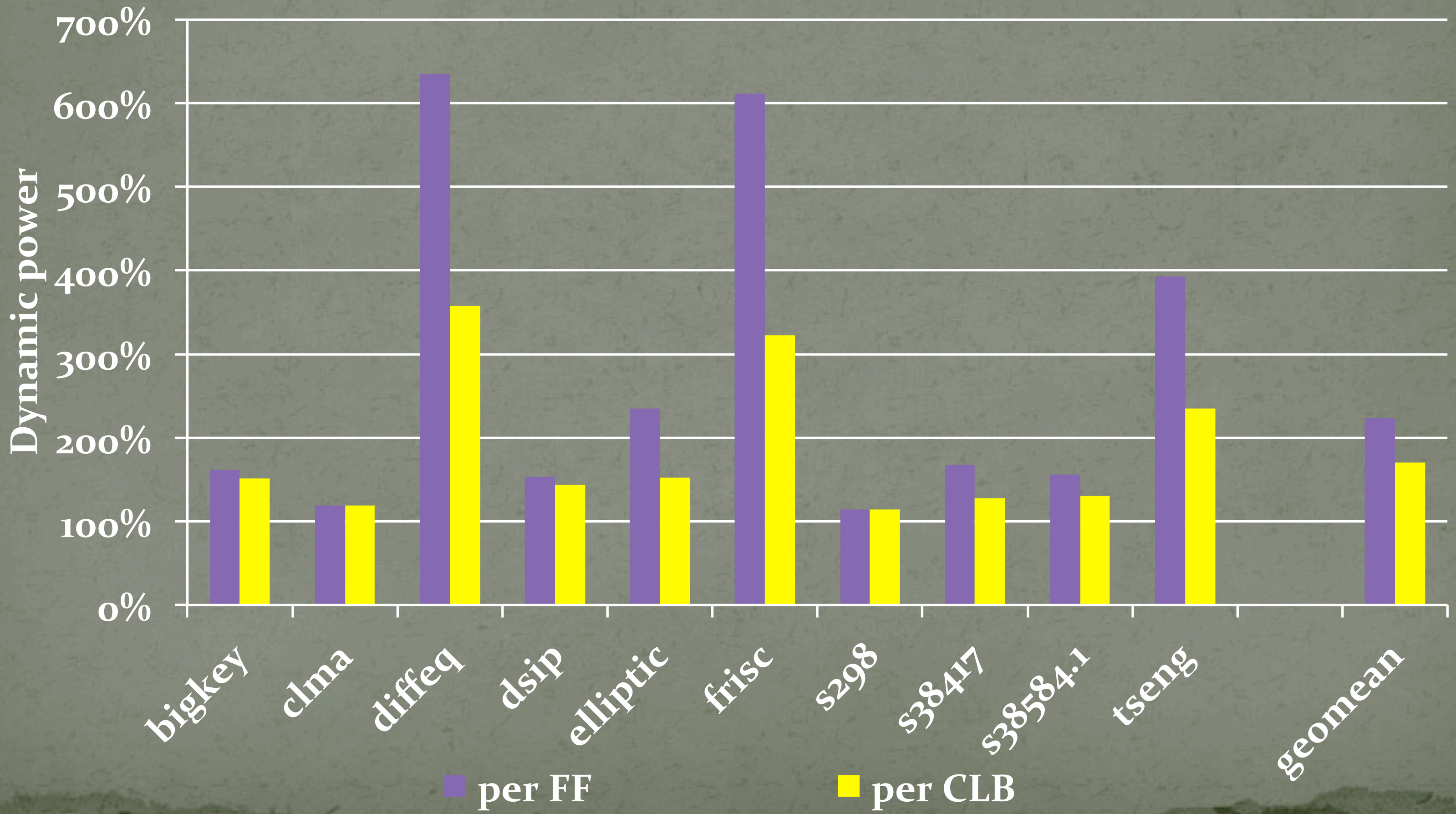
- CSS geomean 13%
- CSS+DP geomean 18% (up to 32%)
  - Delay padding benefits 4 circuits (up to 23%)

- 1 PDE per CLB restriction:
  - DP not achievable
  - Geomean 10%



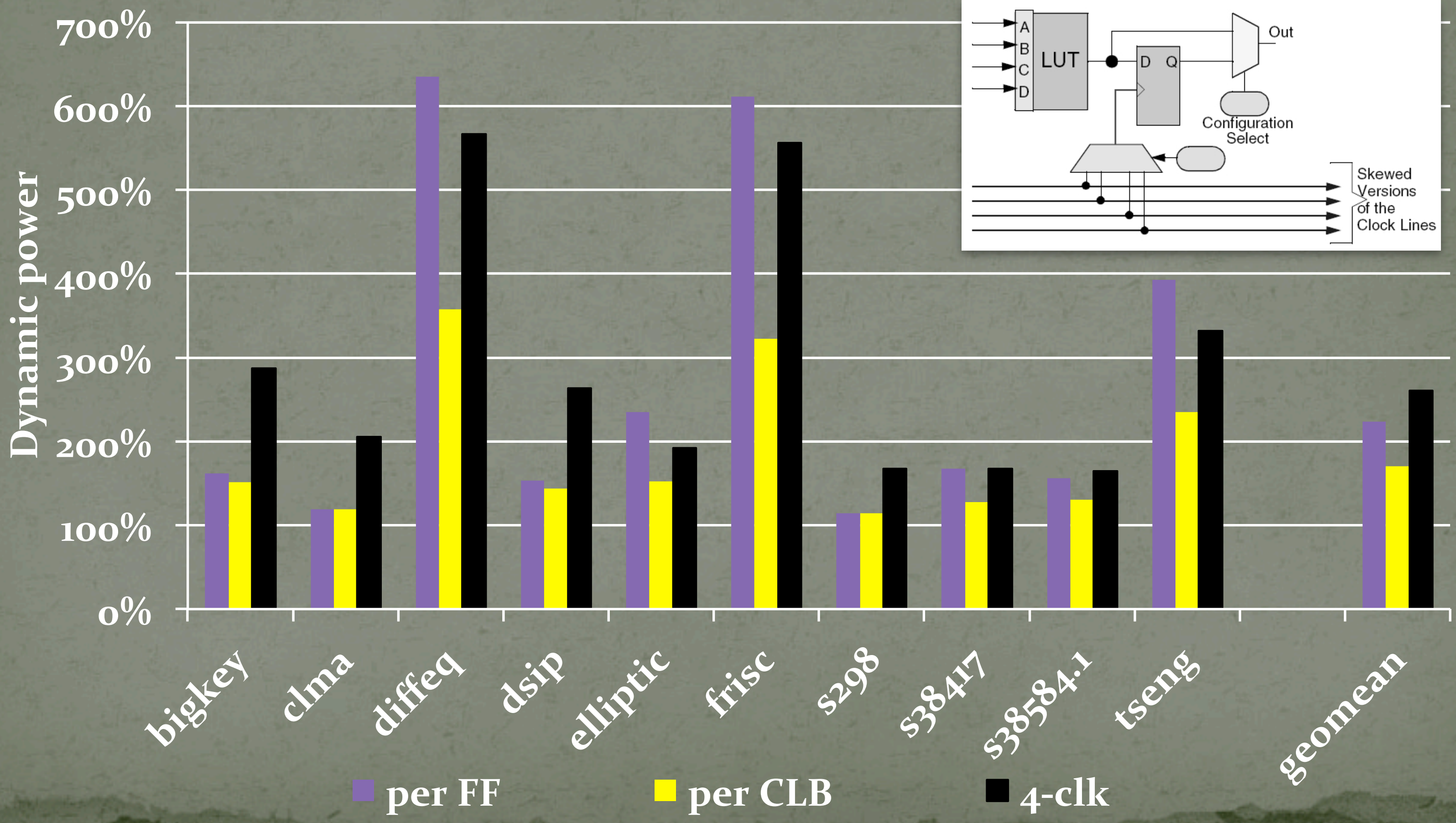
Results – CSS+DP Power Implications

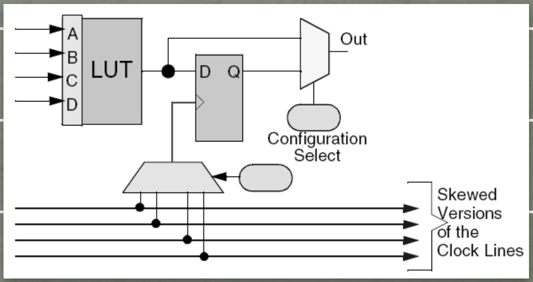
- PDEs need power – clock has activity = 1 !
- 1 PDE per CLB – significantly lower power



Results – CSS+DP Power Implications

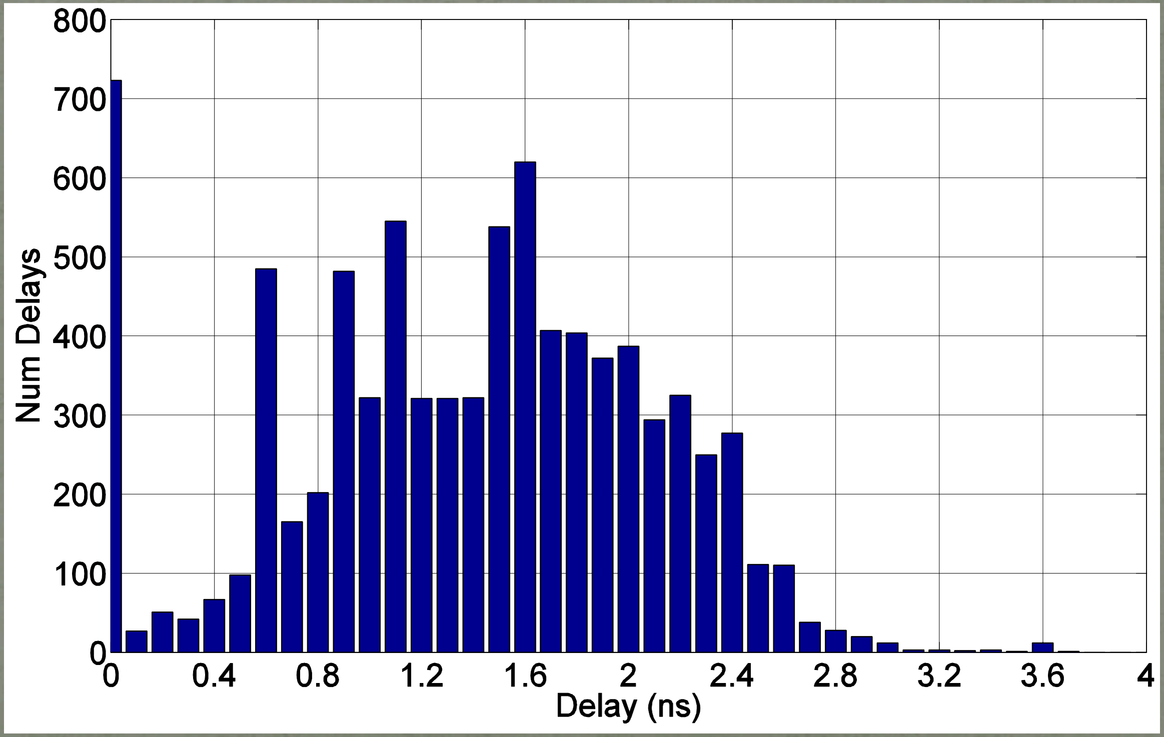
- PDEs need power – clock has activity = 1 !
- 4-clk: power overhead with 3 extra global clocks





Results – Skew Distribution

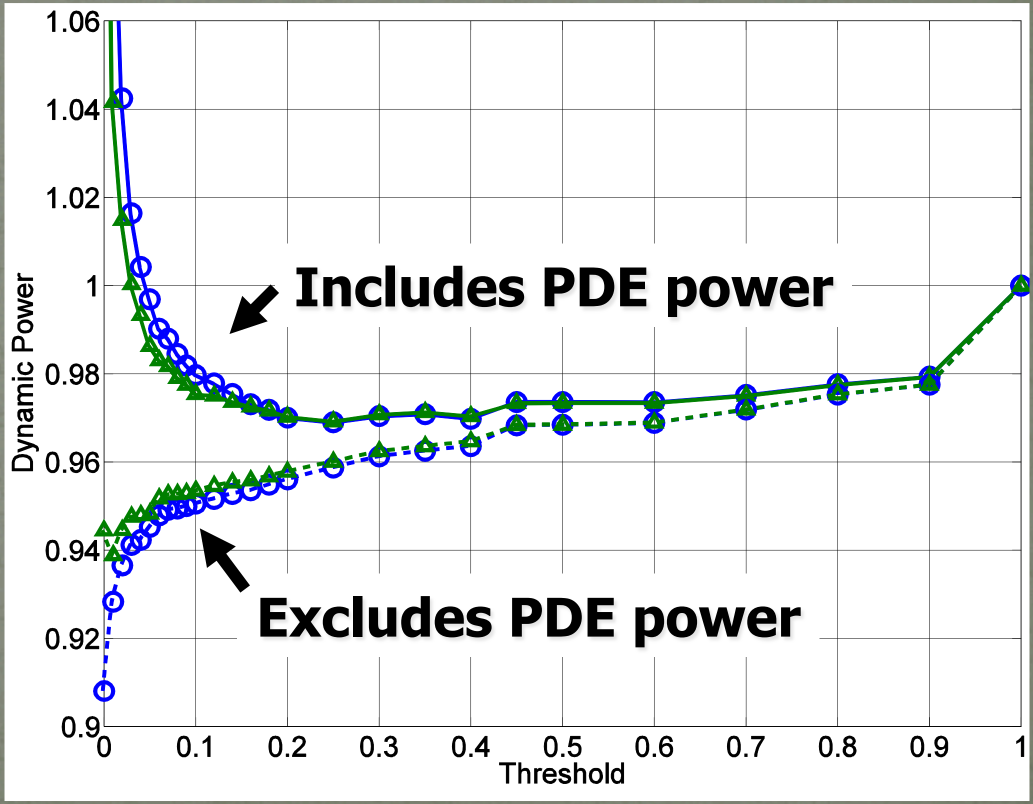
- PDE settings aggregated over all circuits
- Skew is relatively "spread out"



Results – GlitchLess Only

- Select nodes above "threshold"
    - Power of node with most glitching = 1.0
    - Threshold filter selects nodes with most glitching
- Few (~ 10) high glitch power nodes
    - Most nodes w/ small glitch power
- Threshold < 0.2
    - PDE power overhead swamps glitch savings

Blue lines → 20 PDE per CLB
Green lines → 1 PDE per CLB



Outline

- Introduction/Motivation
- Concept
- Implementation
- Results
- Conclusion
- Future Work

Conclusion

- 20 PDEs per CLB
- CSS+DP speedup
  - k=4: geomean 18% (up to 32%)
  - k=6: geomean 20% (up to 38%)
- Dynamic power reduction
  - Best case savings
    - k=4: average 3% (up to 14%)
    - k=6: average 1% (up to 8%)
  - Swamped by PDE power → need low-power PDE
- Area penalty
  - k=4: 11.7%
  - k=6: 7.6%

- 1 PDE per CLB
- CSS speedup
  - k=4: geomean 10% (up to 27%)
  - k=6: geomean 10% (up to 38%)
  - Can't do delay padding
- Dynamic power reduction
  - Similar
- Area Penalty
  - k=4: 0.6%
  - k=6: 0.4%

Future Work

- Improve glitch power estimation
  - Done: fast glitches, analog behavior on single net
  - To do: propagate analog glitches through LUTs
- Reduce PDE power overhead
  - Low-power PDE (circuit design)
- Newer benchmarks
  - Bigger, more recent circuits

Conclusion

THANK YOU!

- <u>20 PDEs per CLB</u>
- CSS+DP speedup
  - k=4: geomean 18% (up to 32%)
  - k=6: geomean 20% (up to 38%)
- Dynamic power reduction
  - Best case savings
    - k=4: average 3% (up to 14%)
    - k=6: average 1% (up to 8%)
  - Swamped by PDE power → need low-power PDE
- Area penalty
  - k=4: 11.7%
  - k=6: 7.6%

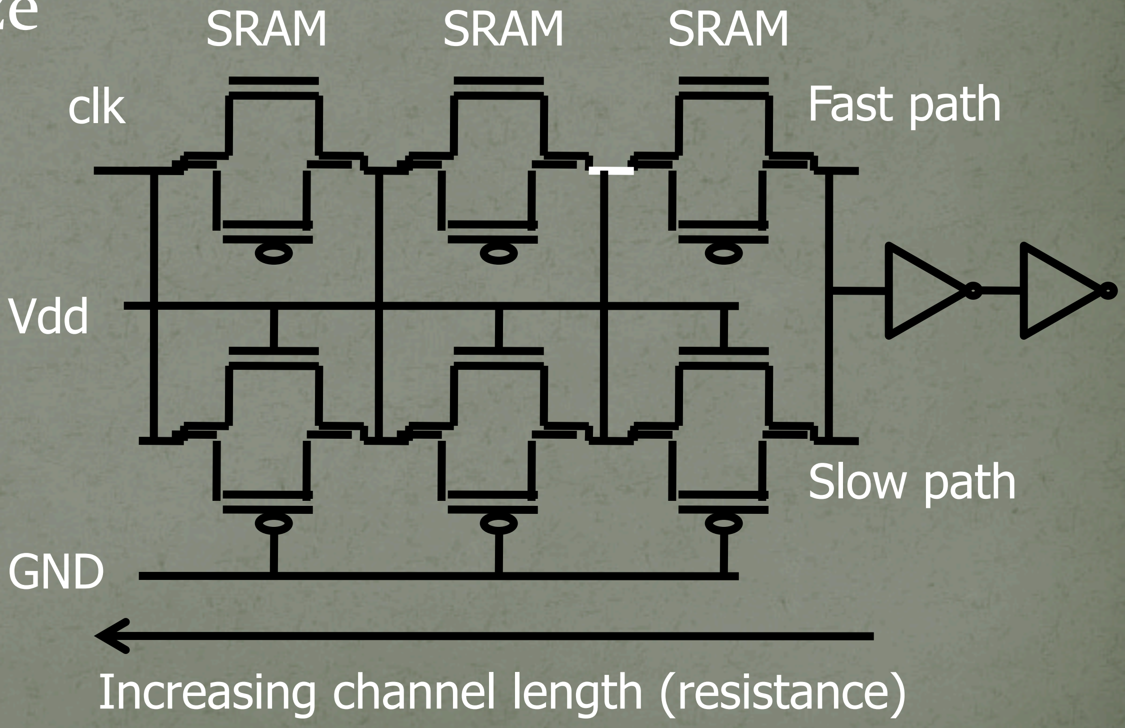
- <u>1 PDE per CLB</u>
- CSS speedup
  - k=4: geomean 10% (up to 27%)
  - k=6: geomean 10% (up to 38%)
  - Can't do delay padding
- Dynamic power reduction
  - Similar
- Area Penalty
  - k=4: 0.6%
  - k=6: 0.4%

Questions?

Architecture – PDE

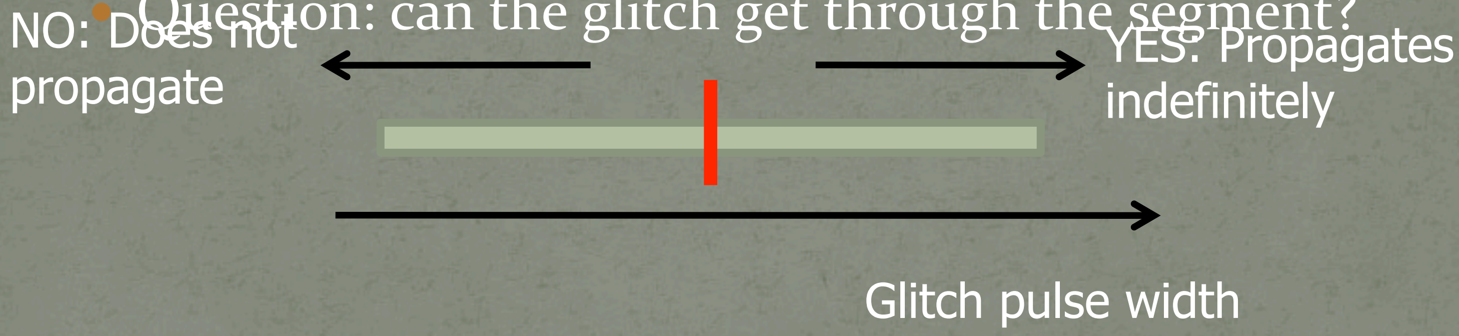
- PDE adapted from GlitchLess (TVLSI 2008)
- 2^n delay values
- Fast path – min size

Delay	Fast path state
000	on on on
001	on on off
010	on off on
011	on off off



Glitch Estimation

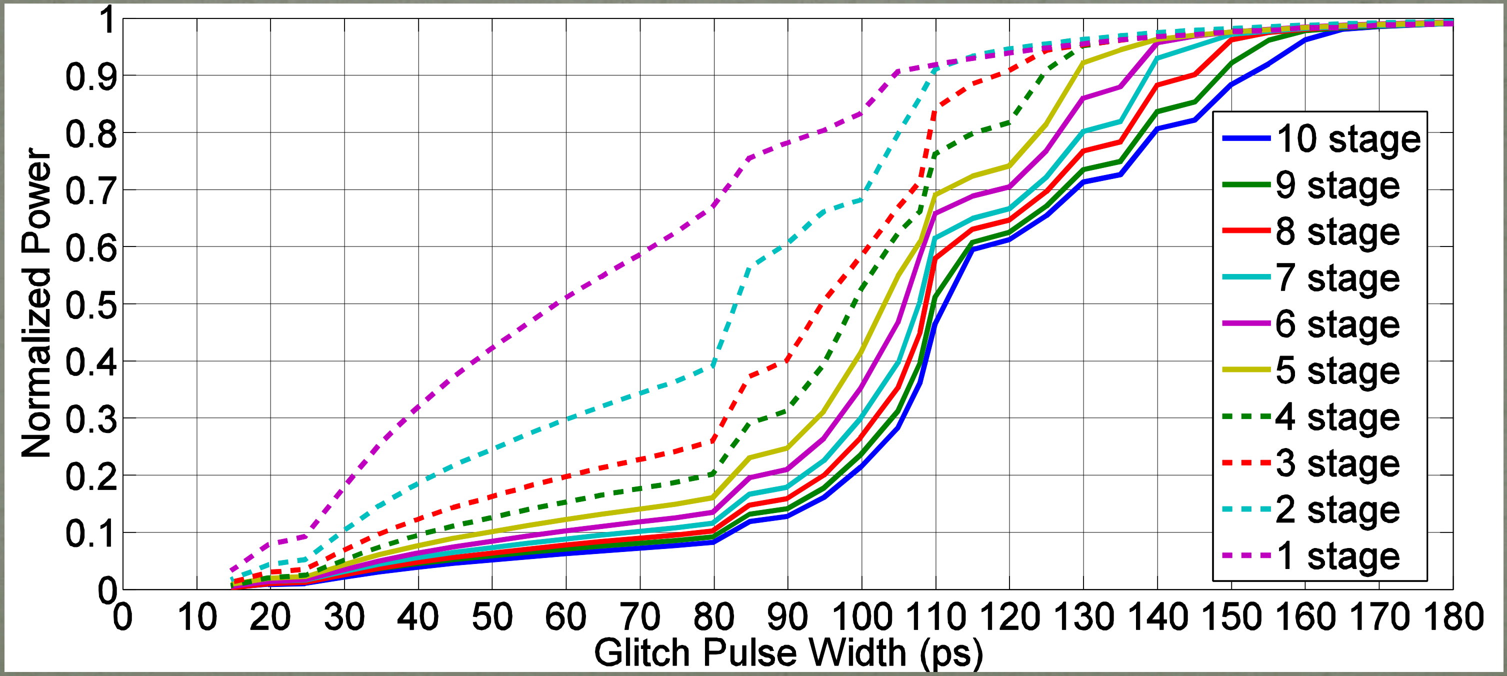
- Need good activity estimator for good power estimates
  - Previous work: ACE 2.0
  - Uses threshold to determine glitch propagation
    - Threshold = one length-4 segment
    - Question: can the glitch get through the segment?



- Real glitches have analog behavior
- Short pulses GRADUALLY damps out

Glitch Estimation

- Real glitches have analog behavior
- Short pulses GRADUALLY damps out
- Group pulse widths into bins – X axis



Glitch Estimation

- Positive = original ACE underestimates
- More underestimates for k=4 → arrival time differences for smaller LUTs are smaller

circuit	k = 4			K = 6		
	Bins	Original	% diff	Bins	Original	% diff
bigkey	913	471	48.4	560	629	-12.4
clma	3794	3407	10.2	2955	3303	-11.8
diffeq	136	129	4.9	63	58	6.7
dsip	698	512	26.6	574	557	3.2
elliptic	11607	10462	9.9	6408	6944	-8.3
frisc	1185	1096	7.5	1045	1088	-4.1
s298	5350	3906	27	4956	5585	-12.7
s38417	29292	19195	34.5	7036	8111	-15.3
s38584.1	10455	9246	11.6	4052	4395	-8.5
tseng	1334	1326	0.6	590	608	-3.2

Algorithm – CSS+DP Top Level

```
iteration = 0;
solution[iteration] = CSS ( Pmax, Pmin );
num_edges = find_critical_hold_edges ( edges[iteration] ); //delete edges
while (num_edges > 0) {
    find_deleted_edge_nodes ( edges[iteration] ); //for delay padding later
    recalculate_binary_bound ( &Pmax, &Pmin );
    iteration ++;
    solution[iteration] = CSS ( Pmax, Pmin );
    num_edges = find_critical_hold_edges ( edges[iteration]) ;
}
while (iteration >= 0) {  //in case delay padding fails for current iteration
    success = delay_padding ( edges[iteration], solution[iteration] );
    if (success) break;
    iteration –= 1;
}
```

Algorithm – DP

- During delay padding for each edge:

```
for each node “n” on deleted edge “iedge” {
    max_padding = get_max_padding( n );
    skew = roundup ( fanin→arrival +
            Ts + MARGIN + fanin_delay(n, fanin), PRECISION ); //for early clock
    delay = skew – fanin→arrival – fanin_delay( n, fanin );
    needed_slack = delay + MARGIN; //for late clock
    while (delay < needed_delay && needed_slack <= max_padding)
        increment skew, delay and needed_slack by PRECISION;
    needed_delay –= delay;
    if ( needed_delay <= 0.0 ) {done = 1; break;}
}
if ( done ) check_other_paths(); //check other paths with same source/sink
else success = 0;
```

Algorithm – GlitchLess

- Similar to delay padding

```
for each level in breadth-first timing graph {
    rank_nodes ( &list, threshold ); //only nodes with glitch power > threshold
    for each node “n” in list {
        skew = roundup ( n→arrival
            + Ts + MARGIN, PRECISION ); //for early clock
        needed_slack = skew – n→arrival + MARGIN; //for late clock
        if ( needed_slack < n→slack ) {
            for each fanin “f” of node “n” {
                needed_delay = n→arrival – f→arrival – fanin_delay( n, f );
                fanin_delay( n, f ) += needed_delay + needed_slack ;
            }
        }
    }
}
```