Approximation-aware Testing for Approximate Circuits

Arun Chandrasekharan[1,+]
Stephan Eggersglüß[1,2,*]
Daniel Große[1,2]
Rolf Drechsler[1,2]

[1]University of Bremen, Germany
[2]DFKI Bremen, Germany
[+]now with OneSpin Solutions, Germany
[*]now with Mentor, a Siemens Business, Germany
grosse@informatik.uni-bremen.de

DFG
DR 287/29-1 & DR287/23-1

SyDe
graduate school, funded by
German Excellence Initiative

DAAD

Agenda

- Introduction
- Error Metrics
- Approximation-aware Testing for Approx. Circuits
  - Approximation Redundant Faults
  - Approximation-aware Fault Classification
- Experimental Results
- Conclusions

Why Approximate Computing?

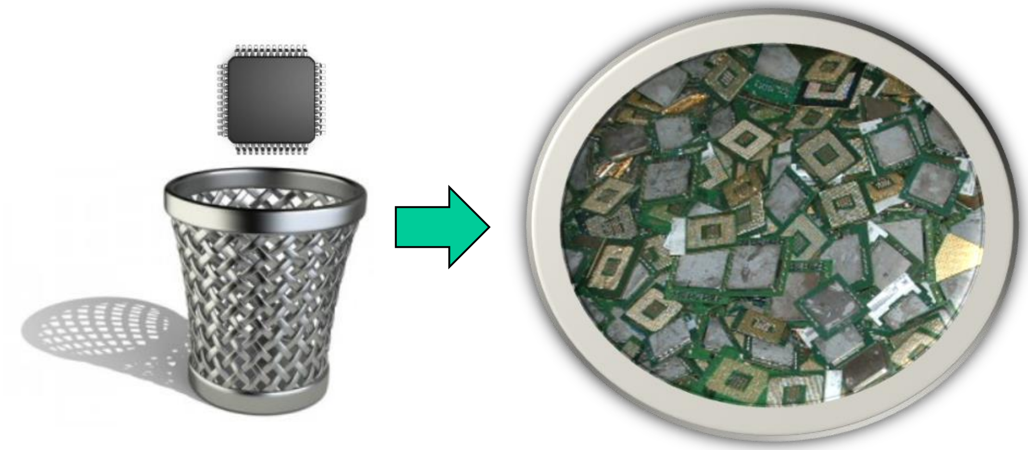
- Many real world applications tolerate inaccurate results
    - Audio, Video, Data mining, Websearch, AI

- Approximate Computing exploits this numerical inexactness for better performance
    - Accuracy vs [speed, power, area]
- How to introduce hardware approximations?
    - Timing induced errors – e.g. voltage scaling
    - Functional Approximations

Production Test and AC?

- After functional approximation standard design flow
- Test of AC chip:

  Test pattern fails ⇒ chip is sorted out



Maybe chip perfectly fine taking approx. during test into account?

Yield ☺

Measuring Inaccuracies in AC

- Error Metrics – how *different* results are?
- “Difference“ depends on application
  - Error-metrics quantify difference
  - Several categories of error-metrics
    - error-rate, error-magnitude, bit-flip error

1. Error Rate
   - Total number of errors at output due to approximation
     - Expressed as % in number of inputs combinations (5 out of 32 are errors etc)
     - eg: Binary to BCD, branch prediction logic

Error Metrics (2)

2. Error magnitude
   - Numerical difference between approximated-output and exact-output
     - Worst-case error: max among error magnitudes
     - Total error: sum of error-magnitudes
     - Average error etc
   - eg: Image processing
     - Pixel distortion due to error-magnitude

Error Metrics (3)

3. Bit-flip error
   - Number of bits in output with a different value (hamming distance)
     - eg: parity logic

- Error Metrics – statistical and exact flavors
  - Statistical
    - Uses error model, distribution (usually random or derived from application, inputs etc)
  - Exact: Guaranteed working
    - No input vectors, assumptions, error models
    - Formally verified

Test and ATPG

- Why test?
  - IC fabrication not ideal. Huge number of *defects*
  - Detect defects (faults) before shipping to customer
  - Ideally every fault must be tested

- Test as SAT-problem
  - Algorithm:
    - foreach fault in Netlist
      - Construct miter for netlist w/ and w/o fault
      - Run SAT-solver
      - SAT solution = ATPG pattern
  - Huge volume of test data
    - compaction, fault simulation, activation cone *etc*

Test for Approximate Computing

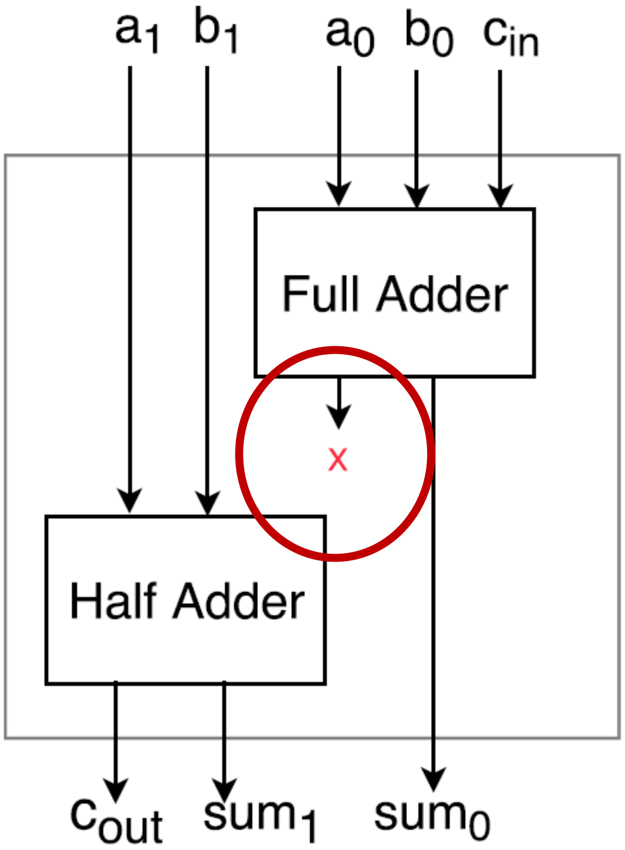
- Which fault to target in ATPG?
  - Skip defects within approximation error tolerance
  - Target only remaining faults

- *Approximation redundant faults*
  - *Faults that are guaranteed to have effects below the tolerable limit of AC*
  - No need to generate test

- Advantages
  - Improve **Manufacturing Yield**
  - Reduce test-time

Example: 2-bit Approximate Adder

$$c_{out}, sum \approx a + b + c_{in}$$

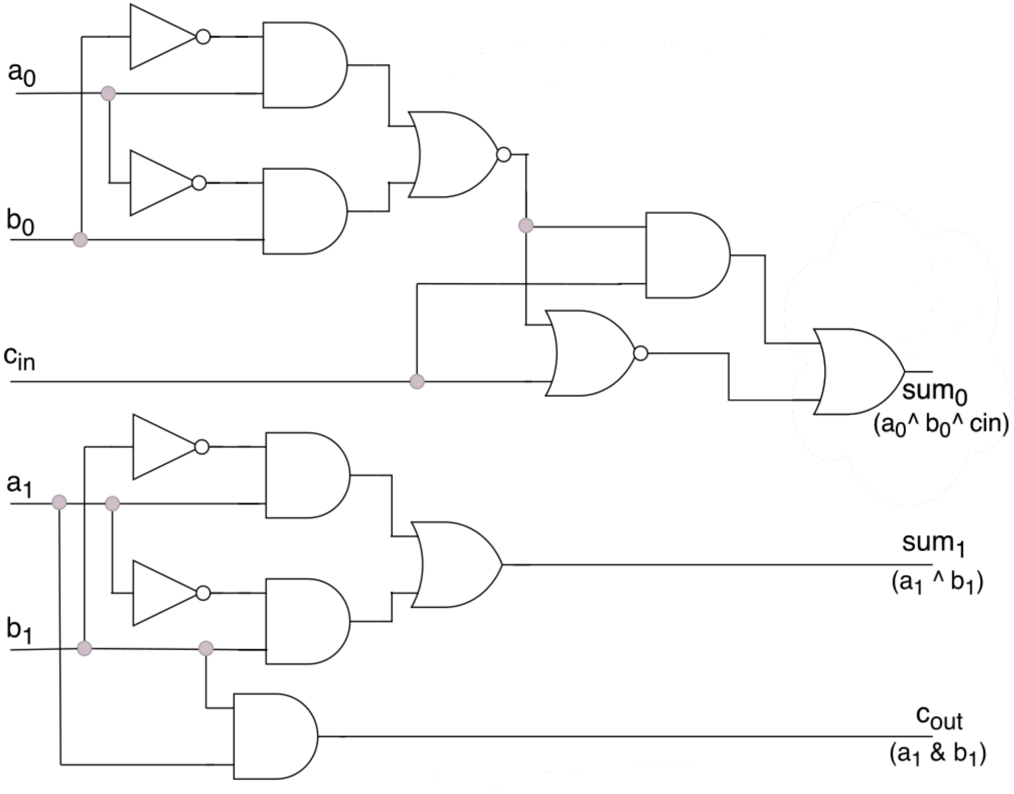


Functional approximation:
Cutting carry from FA to 2nd ~~FA~~ HA

Error metric:
Worst-case error

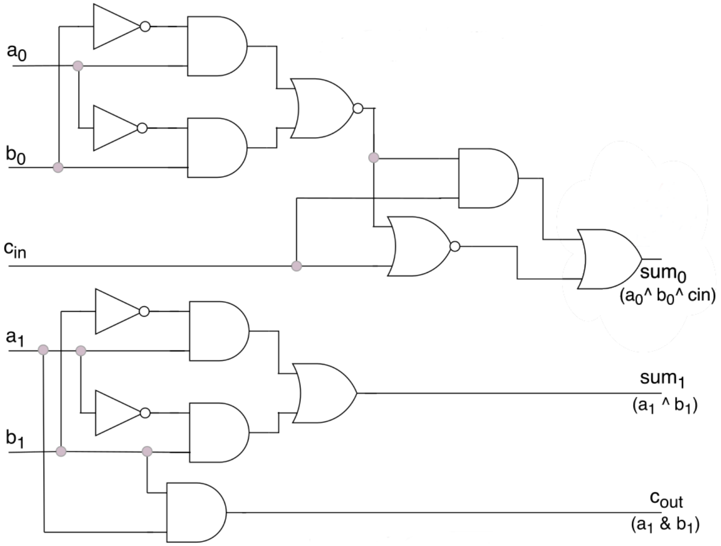
Error metric constraint:
Worst-case error: ≤ 2

Example: 2-bit Approximate Adder



c_{out}, $sum \approx a + b + c_{in}$

Example: 2-bit Approximate Adder



$C_{in}\, a_1\, a_0\, b_1\, b_0$

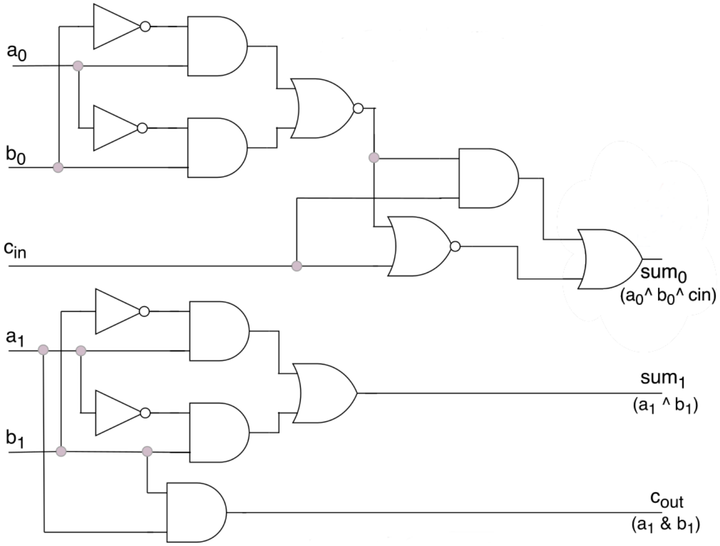
$C_{out}\, sum_1\, sum_0$

Correct†	
In	Out†
00000	000
00001	001
00010	010
00011	011
00100	001
00101	010
00110	011
00111	100

⋮

$c_{out}, sum \approx a + b + c_{in}$

Example: 2-bit Approximate Adder



$C_{in}\, a_1\, a_0\, b_1\, b_0$

$C_{out}\, sum_1\, sum_0$

Correct†		Appx‡	
In	Out†	Out‡	e‡
00000	000	000	0
00001	001	001	0
00010	010	010	0
00011	011	011	0
00100	001	001	0
00101	010	000	2
00110	011	011	0
00111	100	010	2

⋮

worst-case error as integer

$c_{out}, sum \approx a + b + c_{in}$

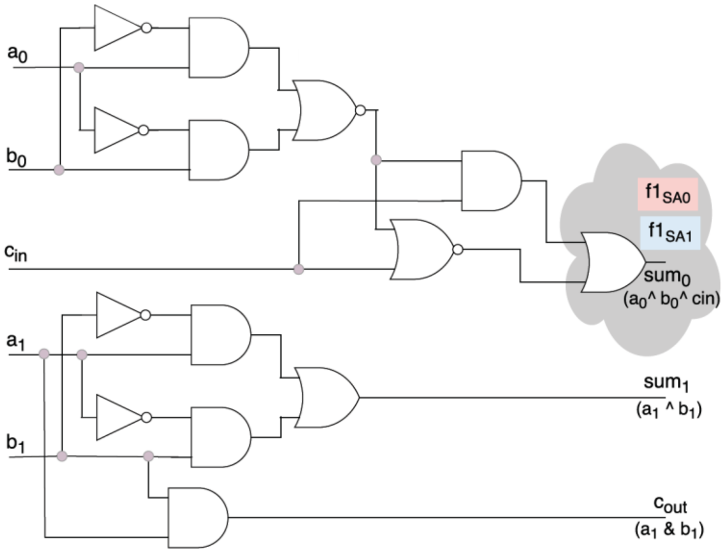
Example: 2-bit Approximate Adder

$C_{out}\, sum_1\, sum_0$

$C_{in}\, a_1\, a_0\, b_1\, b_0$

Correct†		Appx‡	
In	Out†	Out‡	e‡
00000	000	000	0
00001	001	001	0
00010	010	010	0
00011	011	011	0
00100	001	001	0
00101	010	000	2
00110	011	011	0
00111	100	010	2

$\vdots$



$c_{out}, sum \approx a + b + c_{in}$

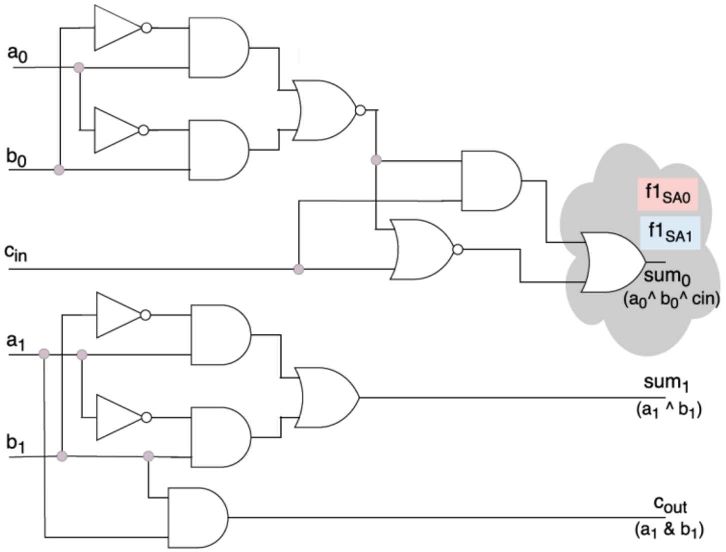
Example: 2-bit Approximate Adder

$C_{in}\, a_1\, a_0\, b_1\, b_0$

$C_{out}\, sum_1\, sum_0$

Correct†		Appx‡		Appx:SA0⋆		Appx:SA1±	
In	Out†	Out‡	$e^‡$	Out⋆	$e^⋆$	Out±	$e^±$
00000	000	000	0	000	0	001	1
00001	001	001	0	000	1	001	0
00010	010	010	0	010	0	011	1
00011	011	011	0	010	1	011	0
00100	001	001	0	000	1	001	0
00101	010	000	2	000	2	001	1
00110	011	011	0	010	1	011	0
00111	100	010	2	010	2	011	1

⋮



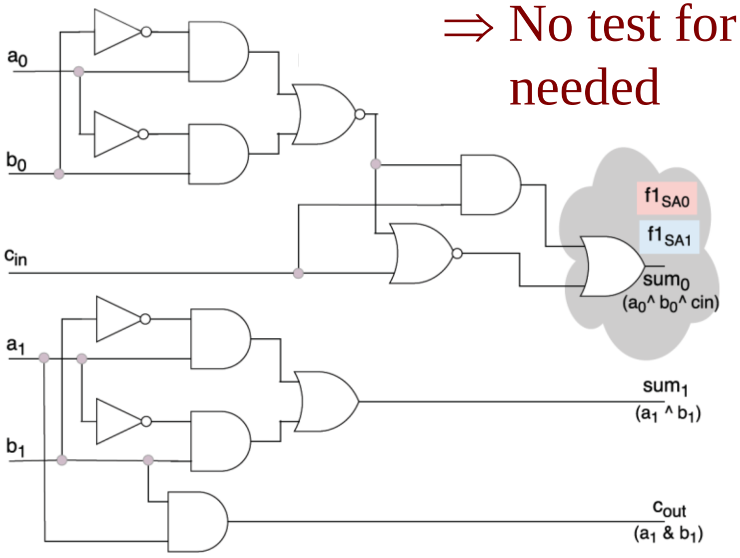
worst-case error
for appr. adder w/ SA0 and SA1
at output bit sum_0

$C_{out}, sum \approx a + b + c_{in}$

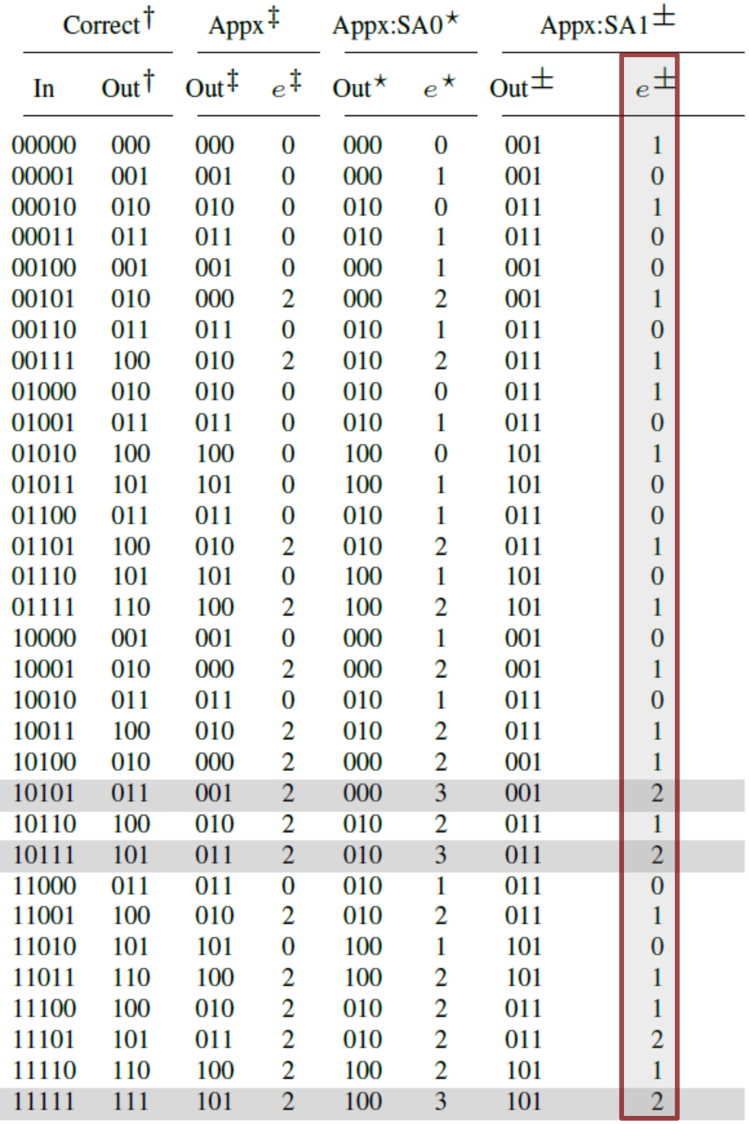
Example: 2-bit Approximate Adder

SA1 fault at sum_0 output is **approximation-redundant** because $e^{\pm}$ is always ≤ 2

⇒ No test for this fault needed



$c_{out}, sum \approx a + b + c_{in}$

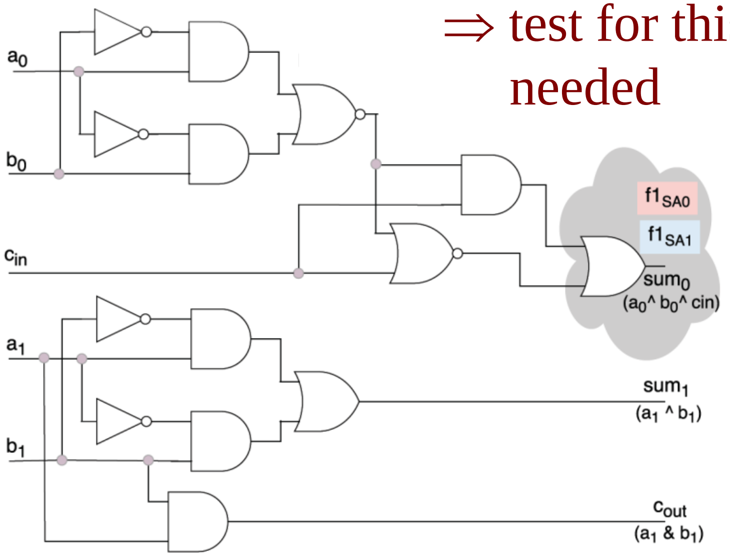


Correct†		Appx‡		Appx:SA0*		Appx:SA1$^{\pm}$	
In	Out†	Out‡	$e^{\ddagger}$	Out*	$e^{\star}$	Out$^{\pm}$	$e^{\pm}$
00000	000	000	0	000	0	001	1
00001	001	001	0	000	1	001	0
00010	010	010	0	010	0	011	1
00011	011	011	0	010	1	011	0
00100	001	001	0	000	1	001	0
00101	010	000	2	000	2	001	1
00110	011	011	0	010	1	011	0
00111	100	010	2	010	2	011	1
01000	010	010	0	010	0	011	1
01001	011	011	0	010	1	011	0
01010	100	100	0	100	0	101	1
01011	101	101	0	100	1	101	0
01100	011	011	0	010	1	011	0
01101	100	010	2	010	2	011	1
01110	101	101	0	100	1	101	0
01111	110	100	2	100	2	101	1
10000	001	001	0	000	1	001	0
10001	010	000	2	000	2	001	1
10010	011	011	0	010	1	011	0
10011	100	010	2	010	2	011	1
10100	010	000	2	000	2	001	1
10101	011	001	2	000	3	001	2
10110	100	010	2	010	2	011	1
10111	101	011	2	010	3	011	2
11000	011	011	0	010	1	011	0
11001	100	010	2	010	2	011	1
11010	101	101	0	100	1	101	0
11011	110	100	2	100	2	101	1
11100	100	010	2	010	2	011	1
11101	101	011	2	010	2	011	2
11110	110	100	2	100	2	101	1
11111	111	101	2	100	3	101	2

Example: 2-bit Approximate Adder

SA0 fault at sum_0 output is a **non-approximation fault** because worst-case error is 3

⇒ test for this fault needed



$c_{out}, \text{sum} \approx a + b + c_{in}$

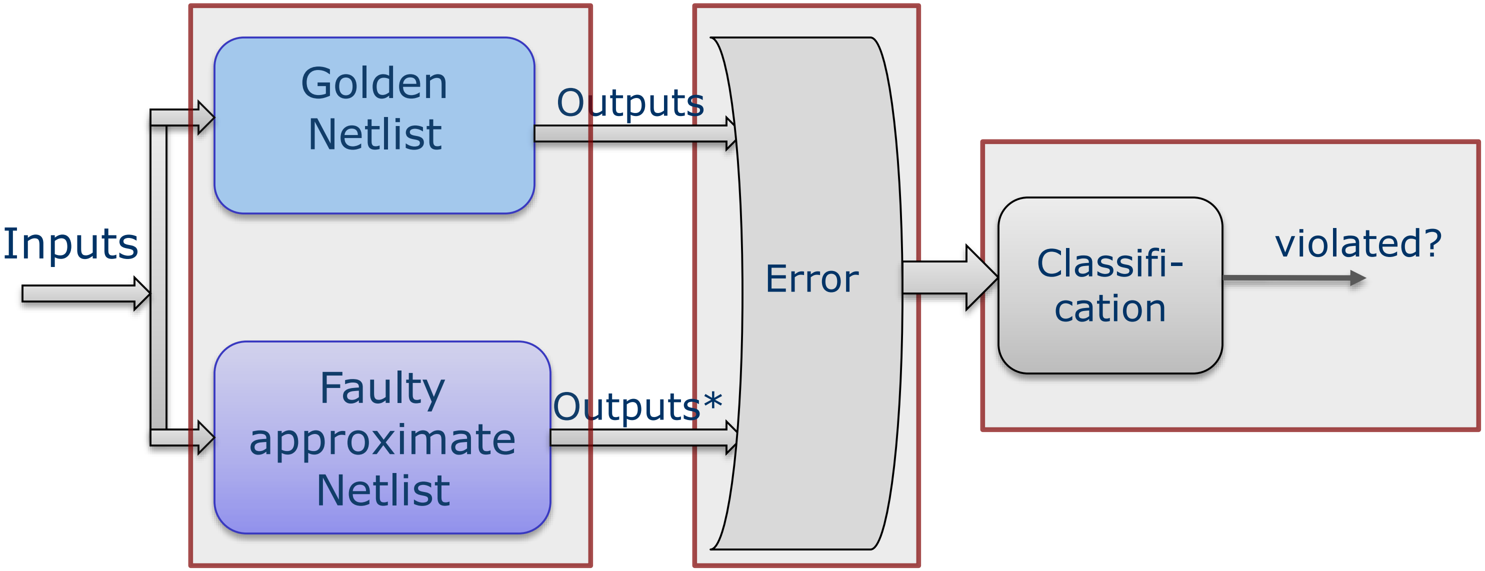
Correct†		Appx‡		Appx:SA0⋆		Appx:SA1±	
In	Out†	Out‡	e‡	Out⋆	e⋆	Out±	e±
00000	000	000	0	000	0	001	1
00001	001	001	0	000	1	001	0
00010	010	010	0	010	0	011	1
00011	011	011	0	010	1	011	0
00100	001	001	0	000	1	001	0
00101	010	000	2	000	2	001	1
00110	011	011	0	010	1	011	0
00111	100	010	2	010	2	011	1
01000	010	010	0	010	0	011	1
01001	011	011	0	010	1	011	0
01010	100	100	0	100	0	101	1
01011	101	101	0	100	1	101	0
01100	011	011	0	010	1	011	0
01101	100	010	2	010	2	011	1
01110	101	101	0	100	1	101	0
01111	110	100	2	100	2	101	1
10000	001	001	0	000	1	001	0
10001	010	000	2	000	2	001	1
10010	011	011	0	010	1	011	0
10011	100	010	2	010	2	011	1
10100	010	000	2	000	2	001	1
10101	011	001	2	000	3	01	2
10110	100	010	2	010	2	011	1
10111	101	011	2	010	3	11	2
11000	011	011	0	010	1	011	0
11001	100	010	2	010	2	011	1
11010	101	101	0	100	1	101	0
11011	110	100	2	100	2	101	1
11100	100	010	2	010	2	011	1
11101	101	011	2	010	2	011	2
11110	110	100	2	100	2	101	1
11111	111	101	2	100	3	01	2

Proposed Approximation-aware Testing

- Approach
  - Use SAT-based pre-processor to remove approximation-redundant faults

- Algorithm: Approximation Fault Pre-processor
  - foreach fault in Netlist
    - Construct Approximation Miter
    - Run SAT-solver
    - If UNSAT guaranteed to have effect below threshold
    - Skip UNSAT faults from ATPG
    - If fault cannot be classified, treated as non-approximation fault

Approximation Miter for Test



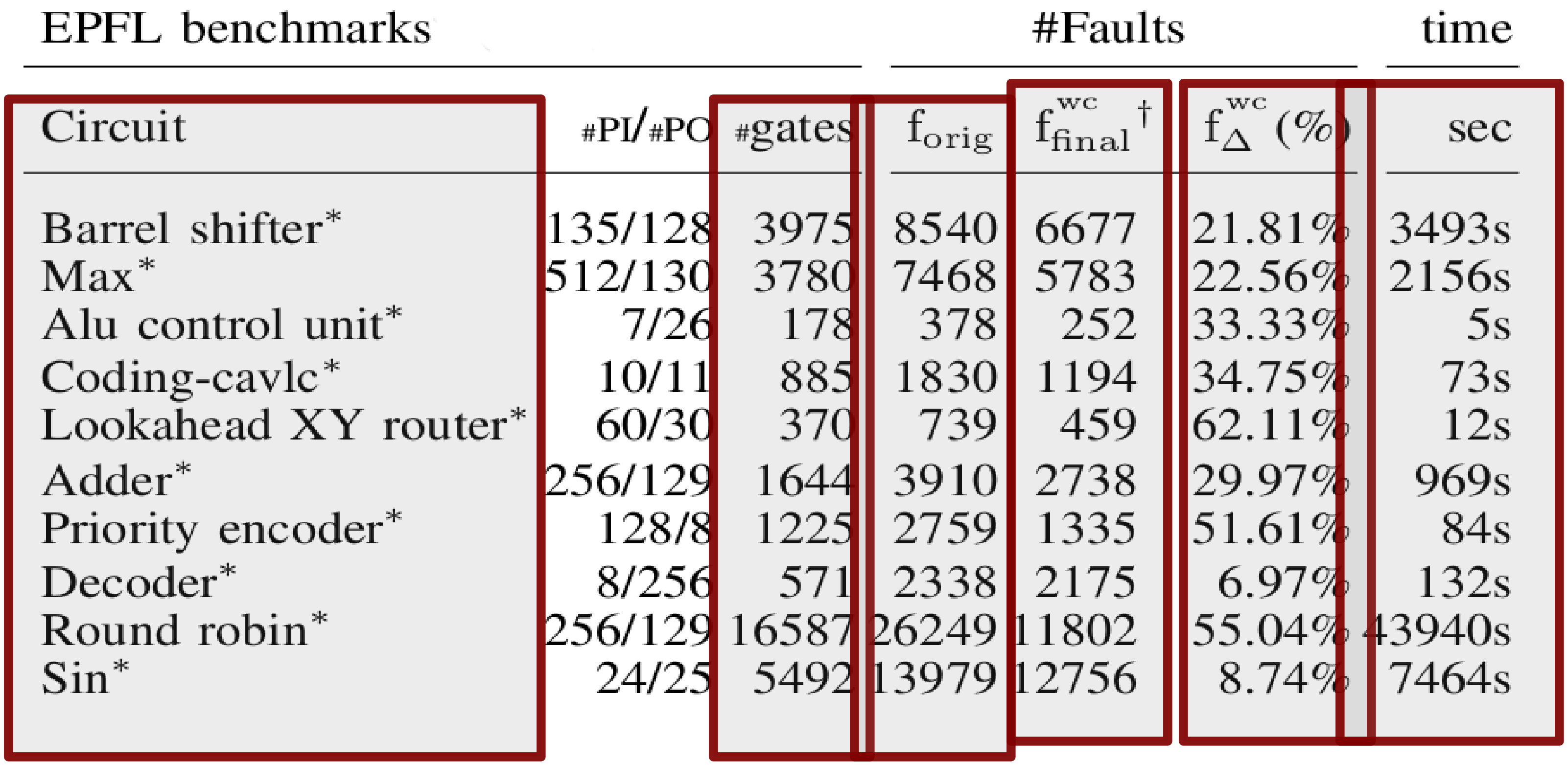
- Golden circuit
- Faulty approximated circuit
- Error = error computation network wrt. error metric
- Classification = fault classification network

  violated becomes 1, iff comparison violates error metric constraint

---

Approximation Miter proposed in ASP-DAC'16 and DAC'16

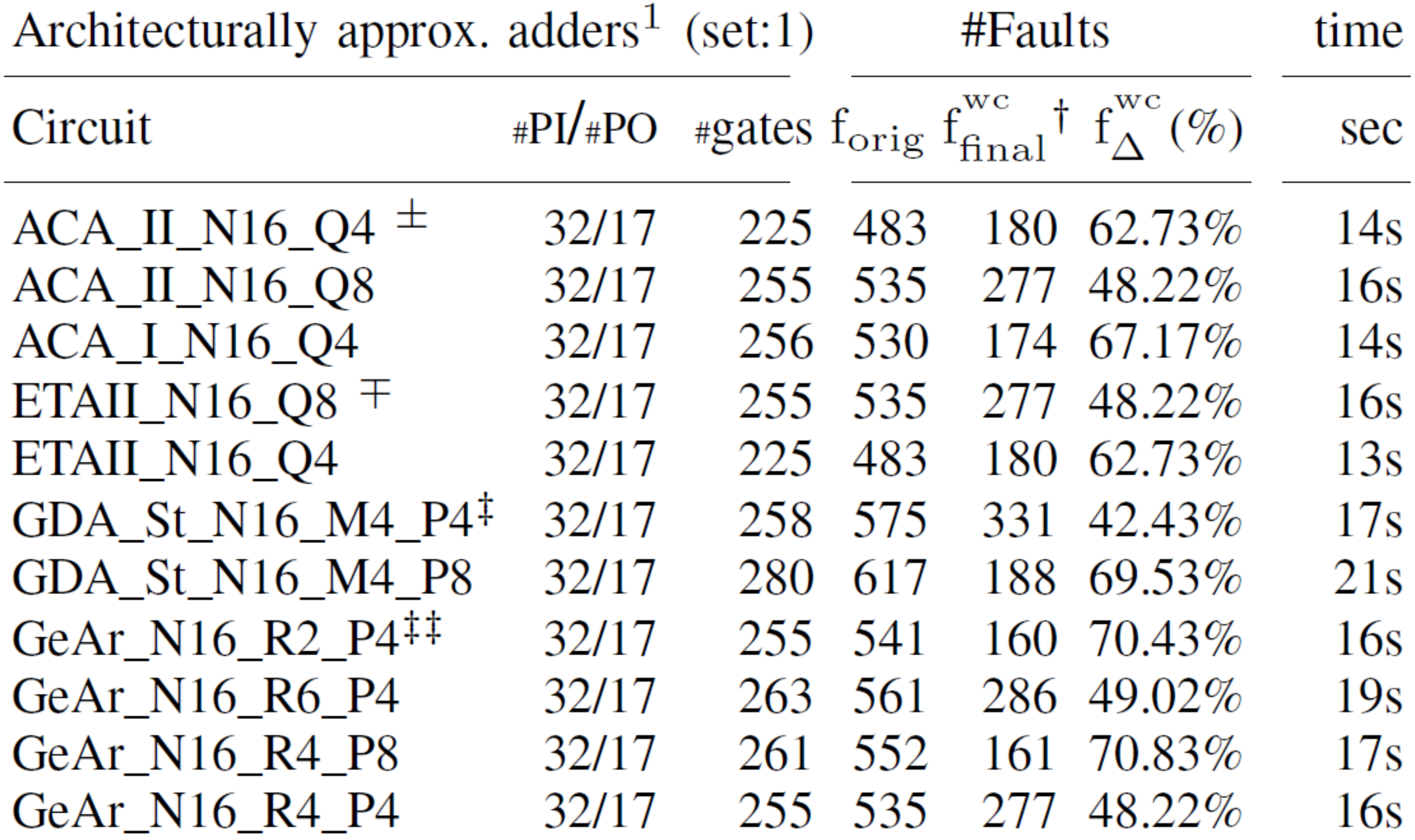
Results (1)



EPFL benchmarks			#Faults			time
Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc} †	$f_{\triangle}^{wc}$ (%)	sec
Barrel shifter*	135/128	3975	8540	6677	21.81%	3493s
Max*	512/130	3780	7468	5783	22.56%	2156s
Alu control unit*	7/26	178	378	252	33.33%	5s
Coding-cavlc*	10/11	885	1830	1194	34.75%	73s
Lookahead XY router*	60/30	370	739	459	62.11%	12s
Adder*	256/129	1644	3910	2738	29.97%	969s
Priority encoder*	128/8	1225	2759	1335	51.61%	84s
Decoder*	8/256	571	2338	2175	6.97%	132s
Round robin*	256/129	16587	26249	11802	55.04%	43940s
Sin*	24/25	5492	13979	12756	8.74%	7464s

[*] Worst-case error conditions, circuits taken from approximation synthesis (ICCAD’16)

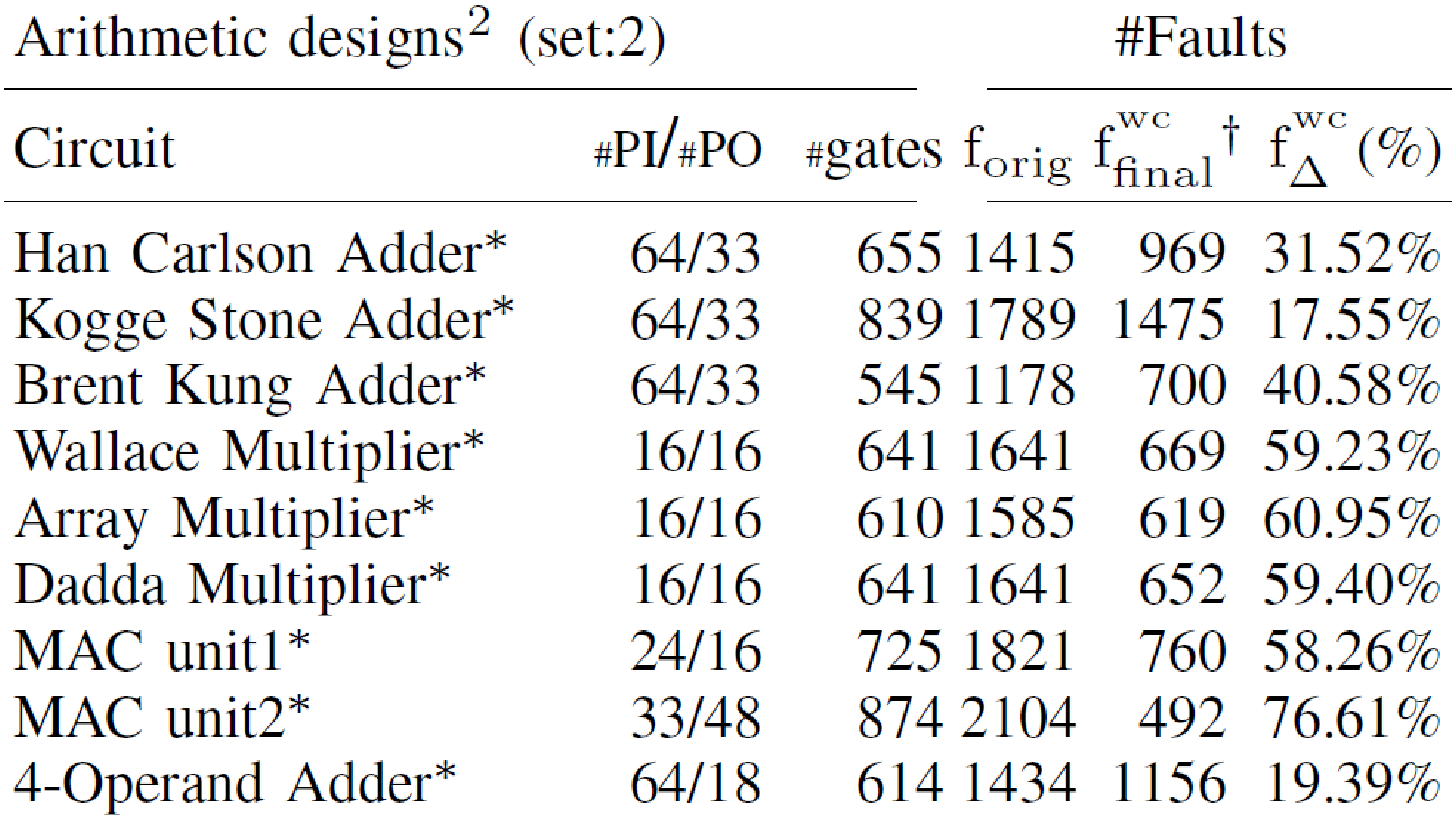
Results (2)



Architecturally approx. adders[1] (set:1)			#Faults			time
Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc} †	f_{Δ}^{wc} (%)	sec
ACA_II_N16_Q4 ±	32/17	225	483	180	62.73%	14s
ACA_II_N16_Q8	32/17	255	535	277	48.22%	16s
ACA_I_N16_Q4	32/17	256	530	174	67.17%	14s
ETAII_N16_Q8 ∓	32/17	255	535	277	48.22%	16s
ETAII_N16_Q4	32/17	225	483	180	62.73%	13s
GDA_St_N16_M4_P4‡	32/17	258	575	331	42.43%	17s
GDA_St_N16_M4_P8	32/17	280	617	188	69.53%	21s
GeAr_N16_R2_P4‡‡	32/17	255	541	160	70.43%	16s
GeAr_N16_R6_P4	32/17	263	561	286	49.02%	19s
GeAr_N16_R4_P8	32/17	261	552	161	70.83%	17s
GeAr_N16_R4_P4	32/17	255	535	277	48.22%	16s

manually architected approximation adders primary for image processing

Results (3)



Arithmetic designs^2 (set:2)			#Faults			
Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc} †	f_{Δ}^{wc} (%)
Han Carlson Adder*	64/33	655	1415	969	31.52%
Kogge Stone Adder*	64/33	839	1789	1475	17.55%
Brent Kung Adder*	64/33	545	1178	700	40.58%
Wallace Multiplier*	16/16	641	1641	669	59.23%
Array Multiplier*	16/16	610	1585	619	60.95%
Dadda Multiplier*	16/16	641	1641	652	59.40%
MAC unit1*	24/16	725	1821	760	58.26%
MAC unit2*	33/48	874	2104	492	76.61%
4-Operand Adder*	64/18	614	1434	1156	19.39%

[*] Worst-case error conditions, circuits taken from approximation synthesis (ICCAD'16)

Conclusions

- Approximation-aware Testing for Approx. Circuits
- Fault classification based on approximation error characteristics
  - Approximation-redundant fault vs
  - Non-approximation fault
- Approximation-redundant faults have effects below error threshold limits ⇒ no test needed
- Easy integration into standard test flows
- Significant yield improvement potential

Approximation-aware Testing for Approximate Circuits

Arun Chandrasekharan[1,+]

Stephan Eggersglüß[1,2,*]

<u>Daniel Große</u>[1,2]

Rolf Drechsler[1,2]

[1]University of Bremen, Germany
[2]DFKI Bremen, Germany
[+]now with OneSpin Solutions, Germany
[*]now with Mentor, a Siemens Business, Germany
grosse@informatik.uni-bremen.de

DFG

DR 287/29-1 & DR287/23-1

SyDe

graduate school, funded by
German Excellence Initiative

DAAD