Would Functional Tests Detect All Timing Issues On a Board?

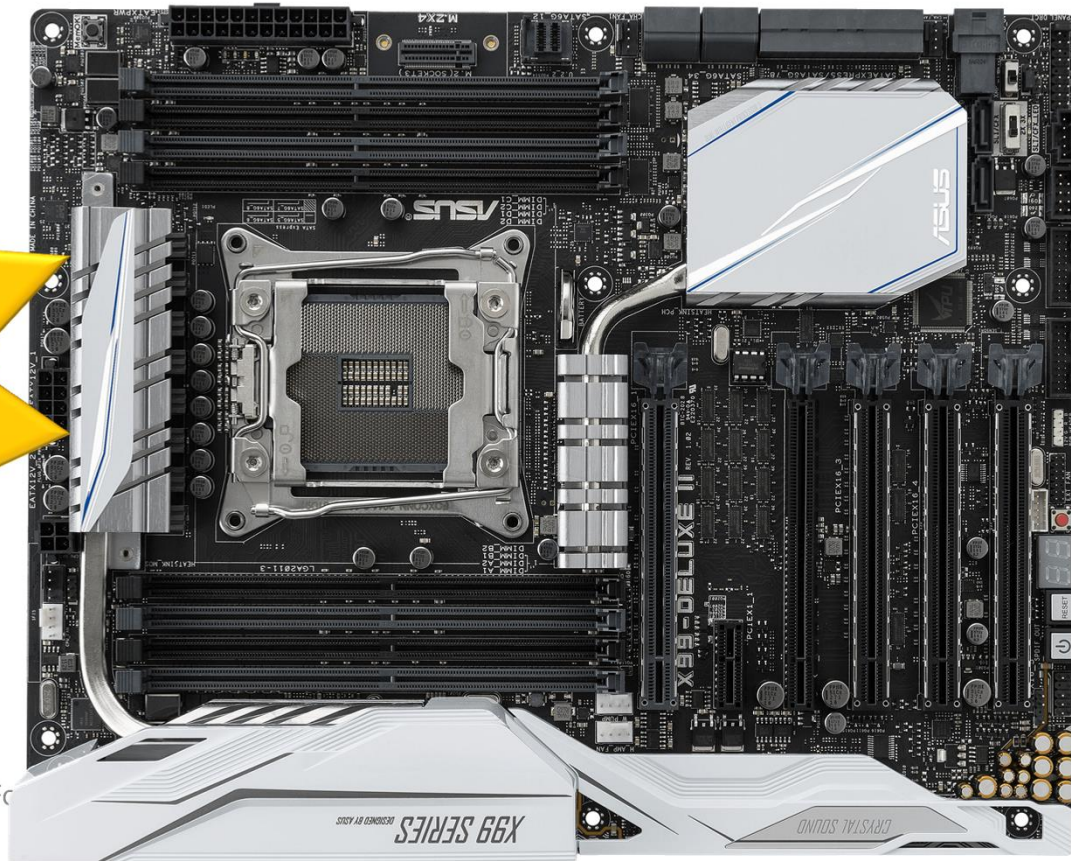
R. Cantoro, E. Sanchez, M. Sonza Reorda
DAUIN – CAD & Reliability group
Politecnico di Torino, Italy

A. Jutman, Testonica Lab

Testonica

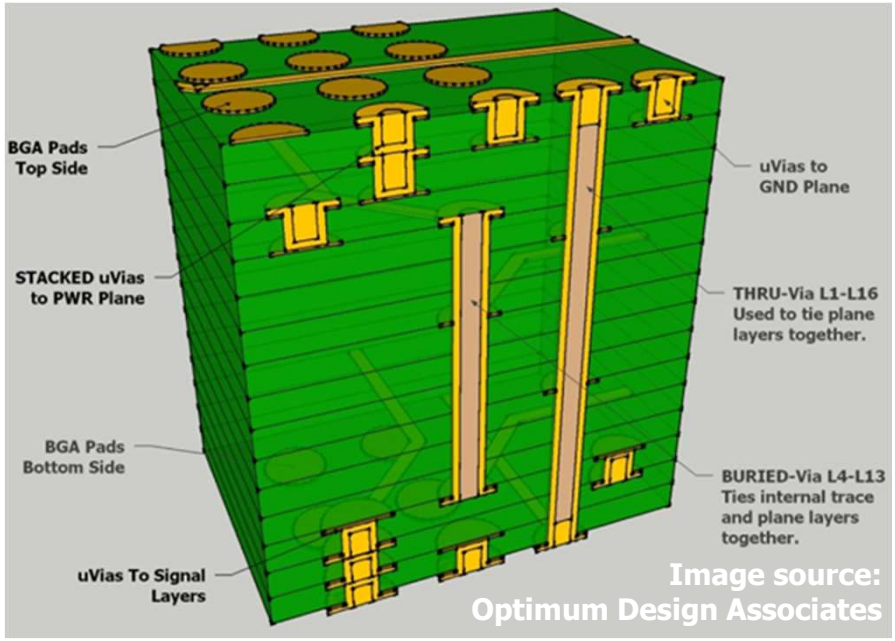
Typical PCB Today

- Up to quad channel DDR4-2133
- DDR4 goes up to 3.2 Gbit/s
- PCI-Express 4x add-in card up to 32Gbit/s
- 10 Gb LAN



High End PCB Features Today

- 3D object that may contain
  - dozens of hidden layers
  - stacked microvias
  - high density interconnect (HDI)
  - buried passive components



- High-speed signals
  - fine-tuned or even calibrated to deliver pitch perfect timing (e.g. for DDR3 / DDR4 memories)

- Data transmission rates on the board
  - reaching multi-gigabit ranges on a single channel

Motivations

- The *No Fault Found* (NFF) phenomenon is increasingly important as its economical cost is huge
- Timing and Performance Issues (e.g. *Delay Faults*) are supposed to significantly contribute to NFF
- How effective are current solutions to delay fault testing at the PCB level?
  - Functional test
    - Often based on running some application code and checking the produced results
    - Defect coverage hard to compute

Goal

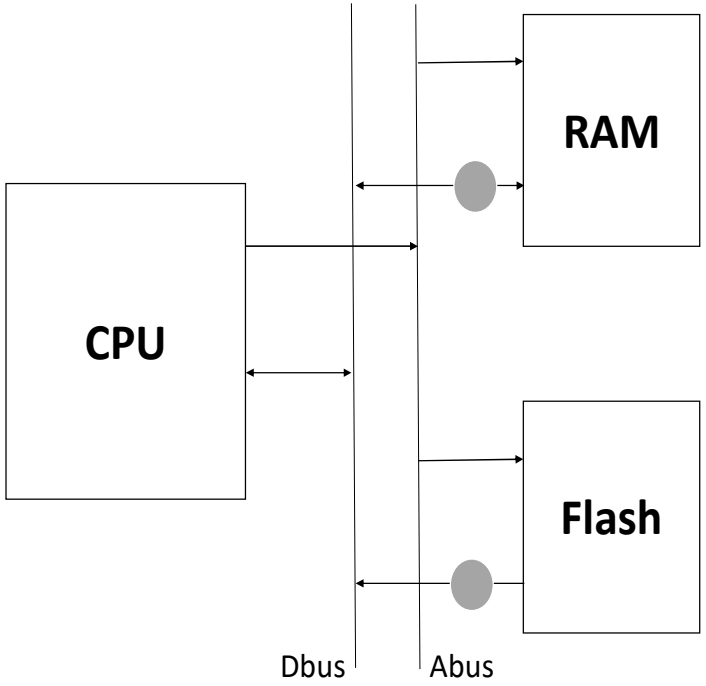
- To experimentally evaluate the delay fault coverage achievable at PCB level with different test programs
- To propose effective solutions for writing suitable test programs targeting delay faults
- Transition delay faults on the bus between CPU and memory are considered

Outline

- Experimental setup
- Delay fault coverage assessment
- New solutions
- Conclusions

Experimental setup

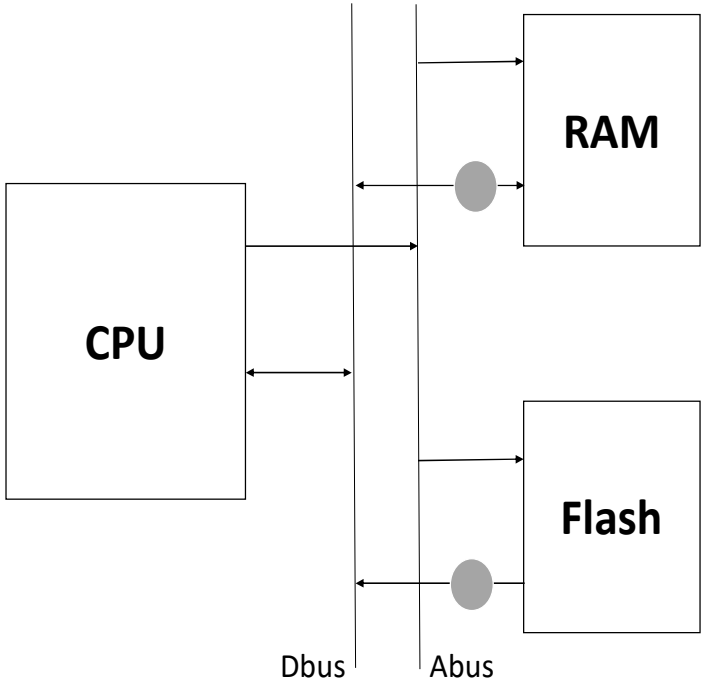
- We built a simple but representative system
  - OR1200 processor
  - 2MB RAM and 2MB Flash memory
- We considered the transition delay faults affecting the data lines connecting CPU and memory
- We investigated fault effects by resorting to *saboteurs*:
  - Verilog code is modified to support fault injection
  - Logic simulation is performed
  - Results are observed
  - Faults are possibly classified as detected



Fault types

We considered 4 types of faults:

- STR/STF on Read/Write operations
  - 128 faults on the RAM Dbus
  - 64 faults on the Flash Dbus



STR/STF – Slow To Rise/Fall

Approach

```
Perform fault free simulation;
Dump the final memory content;
For every fault
{     Inject the fault;
      Perform fault propagation;
      Dump the final memory content;
}
Compare the dump files and gather statistics;
```

Approach

```
Perform fault free simulation;
Dump the final memory content;
For every fault
{     Inject the fault;
      Perform fault propagation;
      Dump the final memory content;
}
Compare the dump files
```

As a reference, we also computed the Fault Covered achievable by constantly observing the Dbus.

Benchmark programs

Since test programs for PCB-level Delay Faults are generally not available, we have performed evaluation of some other typical types of test programs listed below:

1. Application programs
2. Test programs addressing stuck-at faults inside the CPU
3. Test programs for the memory
4. Ad hoc test programs

We performed an *STR/STF Fault Injection* campaign for each group of programs

Application programs - characteristics

	Duration (Clock Cycles)	Program size (KB)
bubble_sort	2,026	1.7
dijkstra	7,921	6.4
dijkVAR3	13,195	12.1
matrixMultiply	6,933	6.1
var3_10	134,418	8.0
bs_VAR3	3,502	3.6
var4p	6,681	6.3
var4p_10	126,018	9.3
var4pp	6,473	6.6
var4pp_10	121,134	9.6

Application programs – fault coverage

	Memory		Bus	
	#Detected faults	**Delay FC (%)**	**#Detected faults**	**Delay FC (%)**
bubble_sort	175	91.15%	182	94.79%
dijkstra	118	61.46%	151	78.65%
dijkVAR3	135	70.31%	167	86.98%
matrixMultiply	120	62.50%	122	63.50%
var3_10	114	59.38%	114	59.38%
bs_VAR3	134	69.79%	162	84.38%
var4p	121	63.02%	123	64.06%
var4p_10	115	59.90%	115	59.90%
var4pp	120	62.50%	122	63.50%
var4pp_10	113	58.85%	113	58.85%

Application programs – fault coverage

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
bubble_sort	175	91.15%	182	94.79%
dijkstra	118	61.46%	151	78.65%
dijkVAR3	13			
matrixMultiply				
var3_10				
bs_VAR3				
var4p				
var4p_10				
var4pp				
var4pp_10				

The achieved Fault Coverage

- **is rather low**
- **does not directly depend on program duration and size**

Stuck-at oriented test programs (I)

	Stuck-at Fault Coverage (%)	Duration (Clock Cycles)	Program Size (KB)
TP01	82.64	15,092	24.8
TP02	80.28	85,292	25.2
TP03	84.90	38,524	19.7
TP04	81.33	31,830	59.0
TP05	79.96	54,873	82.4
TP06	85.46	171,435	103.4
TP07	80.75	18,194	29.0
TP08	82.56	24,806	39.4
TP09	85.60	32,516	53.5
TP10	79.19	27,709	33.5
TP11	82.48	18,075	19.9
TP12	83.93	66,247	59.6
TP13	80.95	72,552	49.6
TP14	85.51	47,788	51.9
TP15	79.50	36,794	62.0

Stuck-at oriented test programs (II)

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
TP01	143	74.48%	192	100.00%
TP02	129	67.19%	167	86.98%
TP03	150	78.13%	192	100.00%
TP04	123	64.06%	192	100.00%
TP05	105	54.69%	168	87.50%
TP06	96	50.00%	96	50.00%
TP07	140	72.92%	173	90.10%
TP08	113	58.85%	192	100.00%
TP09	104	54.17%	192	100.00%
TP10	109	56.77%	192	100.00%
TP11	128	66.67%	192	100.00%
TP12	192	100.00%	192	100.00%
TP13	125	65.10%	151	78.65%
TP14	127	66.15%	152	79.17%
TP15	80	41.67%	192	100.00%

Stuck-at oriented test programs (II)

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
TP01	143	74.48%	192	100.00%
TP02	129	67.19%	167	86.98%
TP03	150	78.13%	192	100.00%
TP04	12			
TP05				
TP06				
TP07				
TP08				
TP09				
TP10				
TP11				
TP12				
TP13				
TP14				
TP15				

The relationship between stuck-at Fault Coverage and delay Fault Coverage is very loose

Test program for memory

- It implements the March-C algorithm

	Duration (Clock Cycles)	Program size (KB)
March-C	25,542	2.2

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
March-C	132	68.75%	168	85.42%

Test program for memory

- It implements the March-C algorithm

	Duration (Clock Cycles)	Program size (KB)
March-C	25.542	2.2

March-C	

Achieving a good Fault Coverage on the memory does NOT imply a good delay Fault Coverage

Ad hoc test program (I)

Approach

1. Write 00000000h to memory location X1
2. Write 11111111h to memory location X2
3. Write 00000000h to memory location X1
4. Read to a register R1 the content of memory location X1 (it should be 00000000h)
5. Read to a register R2 the content of memory location X2 (it should be 11111111h)
6. Read to a register R1 the content of memory location X1 (it should be 00000000h)
7. Write register R1 to memory location Z1
8. Write register R2 to memory location Z2

Ad hoc test program (I)

Approach

1. Write 00000000h to memory location X1
2. Write 11111111h to memory location X2
3. Write 00000000h to memory location X1
4. Read to a register R1 the content of memory location X1 (it should be 0000000
5. Read to a register R2 the [illegible] of memory location X2 (it should be 111
6. Read to a register R[illegible] X1 (it should be 000
7. Write register R1 to
8. Write register R2 to memory location 22

Triggering STR and STF transitions on each bus signals during a write operation

Ad hoc test pro

Triggering STR and STF transitions on each bus signals during a read operation

Approach

1. Write 00000000h t
2. Write 11111111h t
3. Write 00000000h to mem location X1
4. Read to a register R1 the content of memory location X1 (it should be 00000000h)
5. Read to a register R2 the content of memory location X2 (it should be 11111111h)
6. Read to a register R1 the content of memory location X1 (it should be 00000000h)
7. Write register R1 to memory location Z1
8. Write register R2 to memory location Z2

Ad hoc test pro

Approach

1. Write 00000000h t
2. Write 11111111h t
3. Write 00000000h to mem tion X1
4. Read to a register R1 the t of memory location X1 (it should be 0000000
5. Read to a register R2 the tent of memory location X2 (it should be 1111111)
6. Read to a register R1 the content of memory location X1 (it should be 00000000h)
7. Write register R1 to memory location Z1
8. Write register R2 to memory location Z2

Making the effects of possible faults observable

Ad hoc test program (II)

- A similar approach can be adopted for faults on the data bus line towards the Flash for read operations
- Since the Flash stores the program code, this requires
  - For triggering a transition
    - Fetching a couple of instructions, whose machine code is one the complement of the other
  - For observing the effects of possible faults
    - Storing in the memory the result of the instruction (thus checking whether the correct instruction has been executed)

Ad hoc test program I(II)

	Duration (Clock Cycles)	Program size (KB)
Ad hoc Test Program	1,120	4.1

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
Ad hoc Test Program	192	100.00%	192	100.00%

Ad hoc test program I(II)

	Duration (Clock Cycles)	Program size (KB)
Ad hoc Test Program	1,120	4.1

	Memory		Bus	
	#Detected faults	Delay FC (%)	#Detected faults	Delay FC (%)
Ad hoc Test Program	192			

Complete Fault Coverage can be achieved, with reasonable cost in terms of code size and duration

Conclusions

- A «normal» functional program does not guarantee the coverage of delay faults, despite its duration and size
- Any test program developed with different targets also fails in achieving full fault coverage
- A complete fault coverage can only be achieved by resorting to suitable devised test programs
- We provided guidelines for writing such programs

Thank you!