

Exploiting post-silicon debug hardware to improve the fault coverage of Software Test Libraries

Original

Exploiting post-silicon debug hardware to improve the fault coverage of Software Test Libraries / Cantoro, Riccardo; Garau, Francesco; Masante, Riccardo; Sartoni, Sandro; Singh, Virendra; Reorda, Matteo Sonza. - (2022), pp. 1-7. (Intervento presentato al convegno 2022 IEEE 40th VLSI Test Symposium (VTS) tenutosi a San Diego (USA) nel 25-27 Aprile 2022) [10.1109/VTS52500.2021.9794219].

Availability:

This version is available at: 11583/2968143 since: 2022-06-17T17:08:55Z

Publisher:

IEEE

Published

DOI:10.1109/VTS52500.2021.9794219

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

IEEE postprint/Author's Accepted Manuscript

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

Exploiting post-silicon debug hardware to improve the fault coverage of Software Test Libraries

Riccardo Cantoro*, Francesco Garau*, Riccardo Masante*,
Sandro Sartoni*, Virendra Singh[†] and Matteo Sonza Reorda*

*Department of Computer and Control Engineering
Politecnico di Torino
Turin, Italy

[†]Department of Electrical Engineering
Indian Institute of Technology Bombay
Mumbai, India

Abstract—Functional test using a Software Test Library (STL) is becoming a standard solution for the in-field test of safety-critical systems, in compliance with functional safety standards, such as the ISO26262 for the automotive domain. However, developing high-quality test programs is considerably more challenging than generating scan test patterns through commercial tools, mainly due to the lack of mature EDA tools. As a result, in many cases, the effort needed to reach the target fault coverage is not affordable. In this paper, we propose a methodology to improve the fault coverage of an STL using already available hardware resources. The proposed approach identifies the set of sequential cells that capture fault effects before being masked during their propagation towards observable points. Using a heuristic set covering approach, we select the subset of flip-flops needed to reach the target fault coverage, and exploit post-silicon debug hardware to make fault effects observable. Experimental results gathered on an open-source RISC-V core show significant improvements in the stuck-at and delay fault coverage values.

Index Terms—software-based self-test, software test libraries, in-field test, post-silicon debug, safety, functional test

I. INTRODUCTION

Functional test of integrated circuits is becoming a standard solution to implement in-field test for safety-critical systems. Such solution is typically developed in the form of Software-Based Self-Test (SBST) [1], [2], in which the device under test (DUT) executes Software Test Libraries (STLs), i.e., a collection of test programs. The results produced while executing STLs are compacted through software means and compared against pre-computed signatures to detect structural faults (e.g., stuck-at faults). SBST is a well-studied topic in academia and has been proven effective when targeting processor cores [3]–[11], peripherals [12], [13] and memories [14]. Moreover, several companies provide STLs as a Functional Safety (FuSa) solution to test latent faults in the field [15]–[17]. The main advantages of using SBST solutions are its proven reliability and flexibility. For example, test engineers can schedule STLs to run only during idle slots not to interfere with the DUT's main mission, hence avoiding service interruptions. Being an inexpensive technology, as no additional hardware is required, these properties make functional testing particularly suitable

whenever compliance to the FuSa standards like ISO26262 is needed. However, the development of STLs is very expensive and requires a significant manual effort. The main difficulty of this process resides in the necessity of exciting and propagating fault effects throughout the whole circuit to observable points, i.e., primary outputs or memory locations, by only using instructions (test program) from the instruction set architecture (ISA) [18].

The work in [19] moves the first steps in providing systematic strategies to automatically improve test programs by identifying two groups of excited but not observed (NO) faults. The first group includes faults reaching user-visible registers, while the other one is composed by faults reaching to hidden registers. [19] gives preliminary solutions to cover faults belonging to the first category. Contrarily, covering faults that are captured by hidden registers through functional software means is still an open problem. For this reason, we intend to employ debug hardware that is already available in SoCs to observe their effects.

This paper presents a mixed hardware/software methodology that allows observing fault effects from the latter, hard-to-test, fault category, targeting transition delay faults (TDFs). We first perform a fault simulation analysis that extracts spatial (i.e., at which sequential nodes fault effects stopped propagating) and temporal (i.e., at what time instant such propagation occurs) information. Next, based on the availability of features to support the test, we devise systematic algorithms to evaluate the best configuration in terms of signals and clock cycles to look for the aforementioned faults. The main contributions of this work are:

- a method to identify a set of hard-to-test TDFs through SBST means, together with a set of internal signals where such fault effects propagate, paired with timing instants at which such faults are observed;
- three different approaches to improve the available STL to detect as many additional faults as possible with as little resources as possible; and
- integration of available mechanisms (e.g., trace buffers originally included for post-silicon debug) and SBST methodologies to support the test engineer developing STLs while keeping the number of signals observed

Exploiting post-silicon debug hardware to improve the fault coverage of Software Test Libraries

Original

Exploiting post-silicon debug hardware to improve the fault coverage of Software Test Libraries / Cantoro, Riccardo; Garau, Francesco; Masante, Riccardo; Sartoni, Sandro; Singh, Virendra; Reorda, Matteo Sonza. - (2022), pp. 1-7. (Intervento presentato al convegno 2022 IEEE 40th VLSI Test Symposium (VTS) tenutosi a San Diego (USA) nel 25-27 Aprile 2022) [10.1109/VTS52500.2021.9794219].

Availability:

This version is available at: 11583/2968143 since: 2022-06-17T17:08:55Z

Publisher:

IEEE

Published

DOI:10.1109/VTS52500.2021.9794219

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

IEEE postprint/Author's Accepted Manuscript

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

Algorithm 2: Variable flip-flop selection

input : A quadruplet $(D, L_{\max}, T_{\max}, C_{\min})$ where
 D is a list of triplets (F_i, P_i, T_i) where
 F_i is a fault
 P_i is a flip-flop where a F_i is captured
 T_i is the time when F_i is captured in P_i
 L_{\max} is the max number of flip-flops to select
 T_{\max} is the max observation time
 C_{\min} is a target coverage of recoverable faults

output: A list of pairs (S_j, T_j) where S_j is a set of flip-flops to select at time T_j

$S :=$ empty set of flip-flops;
 $R :=$ empty list of (set of flip-flops, time) ;
 $U \leftarrow$ all untested faults in D ;
order D by time;
while $coverage < C_{\min}$ or D is not empty **do**
 $(P_{next}, F_{next}, T_{next}) \leftarrow$ extract first el. from D ;
 if F_{next} in U **then**
 if $(Length(S) < L_{\max}$ or P_{next} in $S)$ and
 $T_{next} - T_{flush} < T_{\max}$ **then**
 remove F_{next} from U ;
 if P_{next} not in S **then**
 add P_{next} to S ;
 end
 $T_{add} \leftarrow T_{next}$;
 else if $T_{next} > T_{add}$ **then**
 add (S, T_{flush}) to R ;
 remove F_{next} from U ;
 clean S and add P_{next} ;
 $T_{add} \leftarrow T_{next}$;
 $T_{flush} \leftarrow T_{next}$;
 end
 end
 end
end
return R

percentage of the total number of faults (*Recoverable FC*). Since all STLs were developed to tackle SAFs, we have similar stuck-at fault coverages, while for TDFs STL2 is significantly less effective than the others, although all fault coverages are not very high. However, we purposely report TDF figures to show the scalability of the proposed approach.

B. Experimental setup

Fault simulations have been carried out using Synopsys Z01X, a commercial tool devised specifically for FuSa. As a result, the full flow of top-level and combinational level stuck-at and transition delay fault simulations took no longer than 5 hours on an Intel Xeon CPU E5-2680 v3 server with a clock frequency up to 3.3GHz. The second fault simulation performed on the combinational logic has been conducted by dropping each fault after 50 detections, which led to fault dictionaries not larger than 15MB. The two post-processing algorithms were developed in Python and require few seconds to analyze each fault dictionary.

C. Algorithm 1

Let us start first by analyzing data obtained when Algorithm 1 is applied, as showed in Fig. 1. First, we present and discuss results achieved on TDFs as this fault model is the main goal of the article, followed by those achieved on SAFs.

Testing a slow-to-rise TDF on a net implies testing the relative stuck-at-0, and the same applies to slow-to-fall and stuck-at-1 faults: for this reason, improving the transition delay fault coverage enhances the stuck-at one as a byproduct.

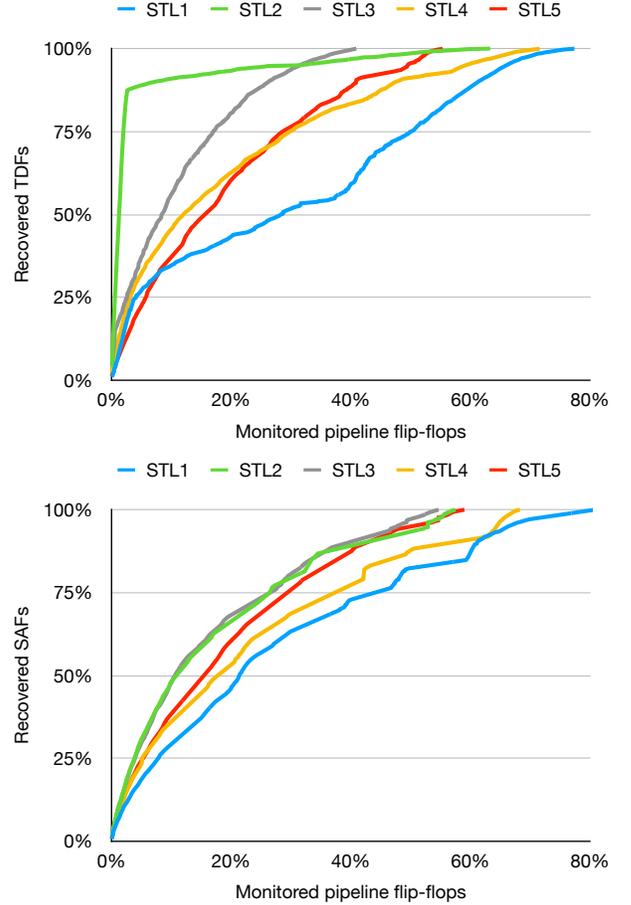


Fig. 1. Undetected TDFs and SAFs faults recovered using a fixed selection of pipeline flip-flops to monitor.

Each graph reports the percentage of *recovered* faults, i.e., faults that become detected (y -axis) when a given percentage of flip-flops are monitored (x -axis). The TDF graph (above) shows the results for all STLs. Starting with the curve for STL2, we see that it has a quite steep slope at the very beginning, allowing to recover more than 80% of TDFs by observing just 2.24% of all flip-flops. It is noted, however, that the transition delay fault coverage of this test program was quite low to begin with as shown in Table I, and the curve, once reached this value, markedly changes its slope, requiring 63.30% of all flip-flops to reach 100%.

Looking at other test programs, detecting 50% of faults requires observing 28% for STL1, 8.5% for STL3, 12% for STL4 and 15% for STL5 of all flip-flops. Moving to the 75% mark, we must monitor a percentage of flip-flops equal to 50% for STL1, 17% for STL3, and about 30% for STL4 and STL5. As for the previous case, trying to detect all the excited but not detected TDFs requires a significant number of flip-flops to be observed, the worst case scenario being 78% for STL1.

TABLE I
STLS GENERAL INFORMATION

Program	Duration [clock c.]	Size [kB]	Detected faults		Recoverable faults		Fault coverage %		Recov. FC (% total)	
			SAF	TDF	SAF	TDF	SAF	TDF	SAF	TDF
STL1	17,308	27.32	151,558	117,758	4,581	7,669	81.66	63.09	2.44	4.08
STL2	31,158	27.86	152,269	75,826	2,842	31,338	82.02	40.74	1.51	16.68
STL3	80,455	16.68	152,801	118,374	2,327	3,161	82.32	63.41	1.24	1.68
STL4	64,541	36.04	160,149	119,367	4,213	5,007	86.22	63.94	2.24	2.67
STL5	118,137	35.61	156,038	123,060	3,412	3,562	84.03	65.91	1.82	1.90

TABLE II
EXPERIMENTAL RESULTS ON TRANSITION DELAY FAULTS USING VARIABLE FLIP-FLOP SELECTION

Program	Slots Bits	#Configurations					Recovered transition delay faults					Recovered transition delay fault coverage %				
		16	32	64	128	inf.	16	32	64	128	inf.	16	32	64	128	inf.
STL1	4	551	531	515	510	503	4,547	4,514	4,517	4,515	4,531	59.29	58.86	58.90	58.87	59.08
	8	359	328	311	298	288	5,473	5,434	5,427	5,383	5,384	71.37	70.86	70.77	70.19	70.20
	16	249	221	197	179	164	6,377	6,338	6,336	6,285	6,346	83.15	82.64	82.62	81.95	82.75
	32	198	159	133	112	90	7,259	7,175	7,195	7,176	7,099	94.65	93.56	93.82	93.57	92.57
	64	173	123	90	65	40	7,591	7,538	7,534	7,495	7,453	98.98	98.29	98.24	97.73	97.18
	128	166	112	75	48	16	7,669	7,669	7,609	7,589	7,503	100.00	100.00	99.22	98.96	97.84
STL2	4	936	890	865	847	835	21,877	21,652	21,492	21,163	21,137	69.81	69.09	68.58	67.53	67.45
	8	619	553	498	471	451	26,567	26,341	26,117	26,090	25,938	84.78	84.05	83.34	83.25	82.77
	16	477	395	323	278	234	29,976	29,912	29,837	29,595	29,362	95.65	95.45	95.21	94.44	93.69
	32	400	304	217	165	88	31,048	31,058	30,994	30,971	30,993	99.07	99.11	98.90	98.83	98.90
	64	378	277	191	132	30	31,272	31,215	31,182	31,177	31,197	99.79	99.61	99.50	99.49	99.55
	128	377	273	186	123	13	31,338	31,338	31,336	31,248	31,266	100.00	100.00	99.99	99.71	99.77
STL3	4	161	160	160	159	158	1,652	1,652	1,652	1,648	1,648	52.26	52.26	52.26	52.14	52.14
	8	104	102	102	101	100	2,050	2,049	2,049	2,043	2,043	64.85	64.82	64.82	64.63	64.63
	16	64	61	60	60	58	2,477	2,456	2,456	2,456	2,440	78.36	77.70	77.70	77.70	77.19
	32	44	38	38	37	35	2,797	2,801	2,805	2,805	2,793	88.48	88.61	88.74	88.74	88.36
	64	30	22	20	19	18	3,069	2,986	2,988	2,988	3,021	97.09	94.46	94.53	94.53	95.57
	128	28	16	12	9	8	3,134	3,148	3,139	3,095	3,099	99.15	99.59	99.30	97.91	98.04
STL4	4	587	560	536	507	466	3,614	3,607	3,598	3,599	3,573	72.18	72.04	71.86	71.88	71.36
	8	418	386	358	320	258	4,188	4,180	4,164	4,151	4,088	83.64	83.48	83.16	82.90	81.65
	16	331	289	249	214	136	4,616	4,603	4,581	4,561	4,441	92.19	91.93	91.49	91.09	88.70
	32	286	238	201	155	67	4,816	4,813	4,810	4,864	4,722	96.19	96.13	96.07	97.14	94.31
	64	266	220	176	133	29	5,004	5,005	4,955	4,954	4,927	99.94	99.96	98.96	98.94	98.40
	128	262	216	171	126	13	5,007	5,007	5,004	5,004	4,979	100.00	100.00	99.94	99.94	99.44
STL5	4	669	643	598	570	504	3,380	3,375	3,378	3,378	3,362	94.89	94.75	94.83	94.83	94.39
	8	457	421	365	330	246	3,514	3,497	3,506	3,507	3,489	98.65	98.18	98.43	98.46	97.95
	16	366	319	262	224	114	3,556	3,554	3,554	3,550	3,548	99.83	99.78	99.78	99.66	99.61
	32	328	277	219	180	53	3,562	3,530	3,530	3,530	3,528	100.00	99.10	99.10	99.10	99.05
	64	322	269	204	162	24	3,562	3,562	3,562	3,562	3,558	100.00	100.00	100.00	100.00	99.89
	128	319	265	202	158	11	3,562	3,562	3,562	3,560	3,562	100.00	100.00	100.00	99.94	100.00

Looking at SAF curves, some of them present significant differences with respect to the TDF case. Such differences can be explained by noting that the available STLs have been developed keeping the stuck-at fault model in mind. Covering 50% of excited but not detected SAFs requires to observe 10% of flip-flops for STL2 and 3, 18% of flip-flops for STL4 and STL5, and more than 20% for STL1. However, if we increase the amount of recovered SAF faults to 75%, we must observe 29% of all flip-flops for STL2, STL3, and STL5, 38% of flip-flops for STL4 and 44% for STL1. If we aim at recovering *all* SAF faults, we need to monitor 58% of flip-flops for STL2, STL3, and STL5, 68% for STL4 and 80% for STL1. In the worst case scenario we would have to observe 960 flip-flops for the whole duration of the test procedure.

D. Algorithm 2

Results of Algorithm 2 are reported in Tables II and III for TDFs and SAFs, respectively. These tables report data on

recoverable faults with respect to the trace buffer width (*Bits*), and the number of clock cycles during which a configuration of observed flip-flops is kept (*Slots*). The value *inf.* defined for *Slots* means that there is no fixed number of clock cycles for the trace buffer to observe, hence the configuration can be kept for as many clock cycles as necessary; this situation implies for example the presence of a MISR to compact the values of the monitored flip-flops.

Although different in absolute values, Tables II and III report very similar trends for all test programs. To get more into details, it is possible to see that the larger the trace buffer width, the higher the amount of recovered faults: providing 128 bits, a size usually adopted with trace buffers, allows to recover all TDFs and SAFs in almost every case. Looking at TDFs, 32-bits trace buffers are required to observe more than 90% of faults, with the sole exception of STL3, where we can only recover about 84% of faults with minimal fluctuations

TABLE III
EXPERIMENTAL RESULTS ON STUCK-AT FAULTS USING VARIABLE FLIP-FLOP SELECTION

Program	Slots Bits		#Configurations					Recovered stuck-at faults					Recovered stuck-at fault coverage %				
			16	32	64	128	inf.	16	32	64	128	inf.	16	32	64	128	inf.
STL1	4	464	452	444	438	429	3,793	3,783	3,781	3,785	3,756	82.80	82.58	82.54	82.62	81.99	
	8	302	281	270	258	250	4,291	4,278	4,300	4,279	4,260	93.67	93.39	93.87	93.41	92.99	
	16	198	171	155	145	131	4,476	4,484	4,481	4,464	4,399	97.71	97.88	97.82	97.45	96.03	
	32	148	118	98	85	66	4,523	4,516	4,524	4,496	4,506	98.73	98.58	98.76	98.14	98.36	
	64	126	92	71	56	32	4,581	4,568	4,531	4,536	4,541	100.00	99.72	98.91	99.02	99.13	
	128	117	84	58	43	15	4,581	4,581	4,572	4,572	4,557	100.00	100.00	99.80	99.80	99.48	
STL2	4	370	362	352	350	334	2,600	2,600	2,600	2,601	2,592	91.48	91.48	91.48	91.52	91.20	
	8	236	223	211	203	185	2,794	2,793	2,791	2,790	2,793	98.31	98.28	98.21	98.17	98.28	
	16	150	138	127	116	93	2,795	2,793	2,793	2,793	2,790	98.35	98.28	98.28	98.28	98.17	
	32	106	97	85	75	46	2,841	2,841	2,841	2,841	2,841	99.96	99.96	99.96	99.96	99.96	
	64	90	75	62	53	23	2,842	2,802	2,842	2,842	2,842	100.00	98.59	100.00	100.00	100.00	
	128	83	67	54	44	11	2,842	2,842	2,842	2,842	2,842	100.00	100.00	100.00	100.00	100.00	
STL3	4	244	243	243	242	242	1,753	1,753	1,753	1,749	1,749	75.33	75.33	75.33	75.16	75.16	
	8	155	154	154	153	153	2,111	2,115	2,115	2,111	2,111	90.72	90.89	90.89	90.72	90.72	
	16	86	82	81	81	81	2,236	2,212	2,213	2,213	2,223	96.09	95.06	95.10	95.10	95.53	
	32	50	44	42	42	42	2,296	2,302	2,302	2,302	2,295	98.67	98.93	98.93	98.93	98.62	
	64	31	24	22	21	20	2,315	2,314	2,309	2,309	2,315	99.48	99.44	99.23	99.23	99.48	
	128	24	16	12	12	9	2,298	2,297	2,297	2,297	2,295	98.75	98.71	98.71	98.71	98.62	
STL4	4	476	461	451	440	417	3,598	3,598	3,603	3,603	3,601	85.40	85.40	85.52	85.52	85.47	
	8	319	300	284	267	238	3,983	3,983	3,983	3,978	3,969	94.54	94.54	94.54	94.42	94.21	
	16	213	189	169	154	123	4,103	4,106	4,106	4,103	4,114	97.39	97.46	97.46	97.39	97.65	
	32	164	140	117	102	59	4,212	4,061	4,061	4,061	4,165	99.98	96.39	96.39	96.39	98.86	
	64	140	115	92	78	29	4,213	4,213	4,213	4,213	4,187	100.00	100.00	100.00	100.00	99.38	
	128	131	105	82	67	14	4,213	4,213	4,213	4,213	4,213	100.00	100.00	100.00	100.00	100.00	
STL5	4	461	445	427	415	385	2,995	2,995	2,995	2,994	2,993	87.78	87.78	87.78	87.75	87.72	
	8	322	299	277	257	219	3,278	3,278	3,278	3,278	3,276	96.07	96.07	96.07	96.07	96.01	
	16	240	212	186	167	116	3,376	3,376	3,376	3,376	3,379	98.94	98.94	98.94	98.94	99.03	
	32	190	158	136	113	58	3,411	3,411	3,411	3,411	3,410	99.97	99.97	99.97	99.97	99.94	
	64	168	138	116	92	29	3,412	3,412	3,412	3,412	3,411	100.00	100.00	100.00	100.00	99.97	
	128	162	130	107	82	14	3,412	3,412	3,412	3,412	3,412	100.00	100.00	100.00	100.00	100.00	

due to the different timing slots; when dealing with SAFs, on the other hand, even 16 bits trace buffers allow to observe more than 90% of faults. Let us focus now on STL3, only: even in the best case scenario, the greedy algorithm does not achieve 100% coverage. The reason for this lies in how this STL is implemented: this test program provides a large amount of faults to be observed at the same time, which translates in a large amount of flip-flops to be monitored by the trace buffer. If there are more flip-flops to be monitored than the maximum trace buffer size, this will inevitably lead to the discard of some of them, having some untested faults as a consequence. More sophisticated search algorithms could be implemented to try to recover this situation, or to prove those faults are untestable using a single STL run. Please note that further STL runs would allow the full detection of recoverable faults. Finally, if we take a look at the table values with respect to the time slots, we see that the smaller the slot size the higher the final coverage. This is expected, as shorter slots allow for more configurations, hence observing more signals throughout the whole test procedure. Results from the *inf.* column fluctuate and are slightly better or slightly worse than those achieved by having time slots of 128 and, in some cases, 64 clock cycles. This can be traced back to the peculiarities of the single test program. Some STLs might require more frequent configuration changes to achieve higher coverages, a

feature that cannot be achieved when large observation slots are scheduled with no flexibility.

V. CONCLUSIONS

This work introduced a systematic methodology to identify and detect hard-to-test, not directly observable in functional mode transition delay faults in complex pipelined processor cores through SBST. Our methodology assumes that the test is based on STLs. First, it identifies a set of excited but still unobserved faults through fault simulation, then produces a configuration schedule for available monitoring features like programmable trace buffers by means of carefully devised algorithms. We have shown with experimental results that a fixed selection of the signals to monitor does not allow to efficiently detect hard-to-test faults, while the proposed variable selection is capable of detecting 100% of the targeted transition delay and stuck-at faults for 4 out of 5 STLs, given enough configurations and 128 bits of trace buffer size. Even in the worst case scenario, we recover 99.15% of TDFs and 98.75% of SAFs. If such size cannot be afforded, 32 bits trace buffers still achieve significant results and can be considered the best trade-off in terms of size vs. coverage. The proposed solution aims at showing how to adopt trace buffers to easily increase the final fault coverage for in-field testing. Future works will include developing strategies for concurrent testing.

REFERENCES

- [1] M. Psarakis *et al.*, “Systematic software-based self-test for pipelined processors,” in *2006 43rd ACM/IEEE Design Automation Conference*, 2006, pp. 393–398.
- [2] M. Psarakis *et al.*, “Microprocessor Software-Based Self-Testing,” *IEEE Design & Test of Computers*, vol. 27, no. 3, pp. 4–19, 2010.
- [3] Wei-Cheng Lai *et al.*, “Test program synthesis for path delay faults in microprocessor cores,” in *IEEE International Test Conference (ITC)*, 2000, pp. 1080–1089.
- [4] C. H. Wen *et al.*, “On a software-based self-test methodology and its application,” in *IEEE VLSI Test Symposium (VTS)*, 2005, pp. 107–113.
- [5] V. Singh *et al.*, “Instruction-Based Self-Testing of Delay Faults in Pipelined Processors,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 11, pp. 1203–1215, Nov 2006.
- [6] K. Christou *et al.*, “A Novel SBST Generation Technique for Path-Delay Faults in Microprocessors Exploiting Gate- and RT-Level Descriptions,” in *IEEE VLSI Test Symposium (VTS)*, April 2008, pp. 389–394.
- [7] P. Bernardi *et al.*, “A Deterministic Methodology for Identifying Functionally Unstable Path-Delay Faults in Microprocessor Cores,” in *International Workshop on Microprocessor Test and Verification (MTV)*, Dec 2008, pp. 103–108.
- [8] —, “Development flow for on-line core self-test of automotive microcontrollers,” *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, 2016.
- [9] N. Hage *et al.*, “On Testing of Superscalar Processors in Functional Mode for Delay Faults,” in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, 2017, pp. 397–402.
- [10] R. Cantoro *et al.*, “New perspectives on core in-field path delay test,” in *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1–5.
- [11] A. S. Oyeniran *et al.*, “Implementation-Independent Functional Test for Transition Delay Faults in Microprocessors,” in *Euromicro Conference on Digital System Design (DSD)*, 2020, pp. 646–650.
- [12] A. Apostolakis *et al.*, “Test Program Generation for Communication Peripherals in Processor-Based SoC Devices,” *IEEE Design & Test of Computers*, vol. 26, no. 2, pp. 52–63, 2009.
- [13] R. Cantoro *et al.*, “In-field functional test of can bus controllers,” in *IEEE VLSI Test Symposium (VTS)*, 2020, pp. 1–6.
- [14] A. van de Goor *et al.*, “Memory testing with a RISC microcontroller,” in *Design, Automation & Test in Europe Conference Exhibition (DATE)*, 2010, pp. 214–219.
- [15] Hitex, “Microcontroller self-test libraries.” [Online]. Available: <https://www.hitex.com/tools-components/software-components/selftest-libraries-safety-libs/pro-sil-safetlib/>
- [16] ARM, “Enabling Our Partnership to Bring Safer Solutions to the Market Faster.” [Online]. Available: <https://developer.arm.com/technologies/functional-safety>
- [17] Microchip Technology Inc., “16-bit CPU Self-Test Library User’s Guide,” 2012. [Online]. Available: <http://ww1.microchip.com/downloads/en/DeviceDoc/52076a.pdf>
- [18] J. Perez Acle *et al.*, “Observability Solutions for In-Field Functional Test of Processor-Based Systems,” *Microprocessors and Micros.*, 2016.
- [19] R. Cantoro *et al.*, “Self-test libraries analysis for pipelined processors transition fault coverage improvement,” in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, 2021, pp. 1–4.
- [20] A. Ruospo *et al.*, “On-line Testing for Autonomous Systems driven by RISC-V Processor Design Verification,” in *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2019, pp. 1–6.
- [21] A. Jasnetski *et al.*, “On automatic software-based self-test program generation based on high-level decision diagrams,” in *IEEE Latin-American Test Symposium (LATS)*, 2016, pp. 177–177.
- [22] —, “Automated software-based self-test generation for microprocessors,” in *International Conference Mixed Design of Integrated Circuits and Systems (MIXDES)*, 2017, pp. 453–458.
- [23] E. Sanchez *et al.*, “Automatic generation of test sets for sbst of microprocessor ip cores,” in *2005 18th Symposium on Integrated Circuits and Systems Design*, 2005, pp. 74–79.
- [24] B. Kumar *et al.*, “A methodology to capture fine-grained internal visibility during multisession silicon debug,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 1002–1015, 2020.
- [25] J.-S. Yang *et al.*, “Improved trace buffer observation via selective data capture using 2-d compaction for post-silicon debug,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 320–328, 2013.
- [26] H. Oh *et al.*, “An on-chip error detection method to reduce the post-silicon debug time,” *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 38–44, 2017.
- [27] S. Chandran *et al.*, “Managing trace summaries to minimize stalls during postsilicon validation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 6, pp. 1881–1894, 2017.
- [28] ETH Zurich and Università di Bologna, “PULPino microcontroller system.” [Online]. Available: <https://github.com/pulp-platform/pulpino>
- [29] Silvaco, “Silvaco 45nm open cell library.” [Online]. Available: https://www.silvaco.com/products/nangate/FreePDK45_Open_Cell_Library/