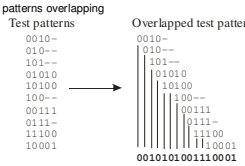
Test Patterns Compression Technique Based on a Dedicated SAT-Based ATPG

Jiří Balcárek, Petr Fišer, Jan Schmidt

Czech Technical University in Prague,
Faculty of Information Technology, Dept. of Digital Design
balcaji2@fit.cvut.cz, fiserp@fit.cvut.cz, jan.schmidt@fit.cvut.cz

Test Patterns Compression

- Based on test patterns overlapping



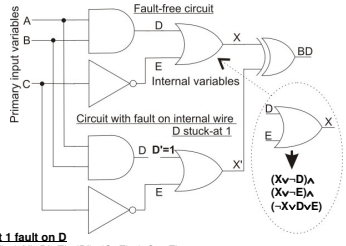
Test bitstream:
0010-010--**101**--0101010**1010**0100--**00111**0111-**11100**10001

Compressed test bitstream:
0010101001110001

- **How can we generate a set of test patterns with maximal overlap?**
- Test patterns are **not pre-generated by an ATPG** (Automatic Test Patterns Generator) but they are **generated on the fly**, during the test compression process
- Test patterns set for each fault is **represented implicitly in CNF**
- Most suitable test patterns to be overlapped are obtained by **applying of constraints** to the CNF
- SAT-Compress algorithm has been created to show possibilities of implicit representation in CNF

Fault-to-CNF Conversion

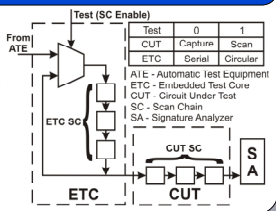
- Two copies of the circuit are created: the fault-free circuit and circuit with a particular fault (faulty circuit)
- Outputs of these two circuits must differ to detect the fault
- Each gate in circuit is described by its characteristic function and CNF is obtained as their conjunction



CNF of stuck-at 1 fault on D
(X'v¬D')∧(X'v¬E)∧(¬X'vD'vE)∧(D')∧(CvE)∧(¬Cv¬E)∧
(Xv¬D)∧(Xv¬E)∧(¬XvDvE)∧(¬DvA)∧(¬DvB)∧(Dv¬Av¬B)∧
(¬XvX'vBD)∧(Xv¬X'vBD)∧(XvX'v¬BD)∧(¬Xv¬X'v¬BD)
Satisfied for assignment: A=1, B=0, C=1, D=0, D'=1, E=0, X=0, X'=1, BD=1.
Therefore, the stuck-at-1 at D fault is detected by the pattern (A, B, C)=(101).

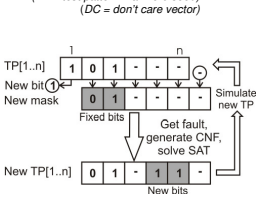
RESPIN Decompression Architecture

- RESPIN (REusing Scan chains for test Pattern decompression) is intended for testing systems on chip (SoCs)
- Based on IEEE 1500 proposed standard for testing of embedded cores
- Scan chains of different cores are reused for updating of the content of the tested core scan chain (SC)



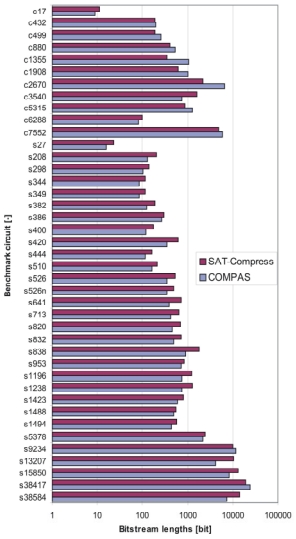
SAT-Compress Algorithm

1) Generate FL *(FL = fault list)*
2) TP[1..n] = "0..0" *(TP = test pattern = all-zero seed)*
   mask[1..n] = DC *(DC = don't care vector)*
3) FL = FL - detected_by_simulation(TP[1..n])
   compressed_bitstream += TP[1]
4) mask[1..n] = TP[2..n] + DC[1]
5) ***do*** {
   for each fault in FL {
   Create CNF
   Apply mask to PIs in CNF
   if CNF is SAT {
   break the *for* loop
   }
   }
   TP[1..n] = CNF_Solution[1..n]
   FL = FL - detected_by_simulation(TP[1..n])
   compressed_bitstream += TP[1]
   mask[1..n] = TP[2..n] + DC[1]
   }
6) ***while*** (FL!=0 or (mask[1..n]==DC and All_CNF==UNSAT))



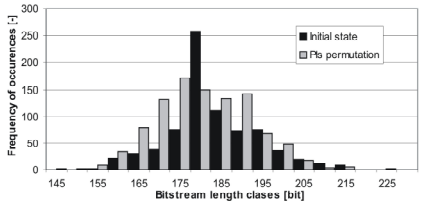
Comparison with COMPAS algorithm

Bench.	COMPAS			SAT-Compress			
	bits	*avg.*	*var.*	*bits*	*avg.*	*var.*	*time [s]*
c17	0	-	9	1.5	11	12.3	1	0
c432	195	218	30.6	189	198.2	21.8	3
c499	260	303.7	47.6	187	198.2	8.7	6
c880	540	412.7	44.3	410	561.5	60.99	10
c1355	1040	1126	68.6	349	-	-	74
c1908	1009	989.8	49.6	624	-	-	229
c2670	6553	5940	269.7	2223	-	-	2305
c3540	747	743.7	32.9	1622	-	-	3569
c5315	1255	1159.5	61.8	881	-	-	156
c6288	83	95.2	36.6	97	-	-	165
c7552	6005	6430.7	345.2	4840	-	-	7406
s27	16	13.2	2.3	23	21.5	2.1	0
s208	130	124.1	9.1	202	209	15.9	0
s298	101	79.6	5.5	137	139.7	8.2	0
s344	85	80.9	6.7	116	114.7	10.2	0
s349	85	80.1	6.8	116	114.4	10	0
s382	123	111.3	6.6	191	179.2	12.8	0
s386	261	255	10.8	304	310	12.5	4
s400	121	109	7.2	173	170.9	9.7	0
s420	352	315.9	29.1	624	574.8	46.4	11
s444	116	107	7.5	160	150.6	10.5	0
s510	160	156	8.9	210	211.8	10.3	2
s526	344	349.8	18.4	521	482.8	22.7	4
s526n	344	350.2	18.1	493	-	-	3
s641	397	393.3	24.2	710	670.7	28.4	15
s713	428	403.8	26.1	642	672	38.7	32
s820	460	504.6	21.9	697	697.7	26.3	28
s832	494	498.8	19.5	729	690.7	24.1	36
s838	920	762.3	79.6	1800	1638.3	103.5	136
s953	723	700.4	24.7	825	-	-	51
s1196	740	738.5	23.5	1211	1202.2	41.9	173
s1238	741	769.2	24.3	1300	1268.7	41.7	365
s1423	596	621.5	38.6	794	827.6	43.9	72
s1488	488	461.5	15.1	546	-	-	20
s1494	431	451.3	16.6	573	-	-	25
s5378	2148	1995.6	73.8	2407	-	-	631
s9234	11594	11309.6	310.7	9928	-	-	68119
s13207	4163	-	-	10457	-	-	67751
s15850	8234	-	-	12987	-	-	114932
s38417	24198	24926.3	1717.7	19291	-	-	91713
s38584	7291	-	-	14271	-	-	122143



- COMPAS algorithm compresses pre-generated test patterns by their overlapping
- SAT-Compress algorithm generates compressed test patterns on the fly
- SAT-Compress algorithm can reach better compression ratio than COMPAS
- Compression efficiency seems to be much better for ISCAS'85 than ISCAS'89 benchmark circuits

Dependences of Compression Efficiency



- Our extensive evaluation shows that the result quality depends on the initial state (initial seed) as much as on the order of primary inputs (see example for c432)

Comparison with Other Compression Algorithms

Bench.	MinTest	Stat. Coding	LFSR Reseeding	Illinois Scan	FDR Codes	EDT	RESPIN++	COMPAS	SAT-Compress
s5378	20,758	15,417	6,180	14,572	12,346	-	17,332	2,148	2,407
s9234	25,935	19,912	12,112	27,111	22,152	-	17,198	11,594	9,928
s13207	163,100	52,741	11,285	109,772	30,880	10,585	26,004	4,163	10,457
s15850	58,656	49,163	12,438	32,758	26,000	9,805	32,226	8,234	12,987
s35932	21,156	-	-	-	22,744	-	-	1,860	5,096
s38417	113,152	172,216	34,767	96,269	93,466	31,458	89,132	24,198	19,291
s38584	161,040	128,046	29,397	96,056	77,812	18,568	63,232	7,291	14,271

- SAT-Compress can obtain similar results such as state-of-the-art compression methods

Conclusion and Future Work

- SAT-Compress algorithm utilizes a CNF (Conjunctive Normal Form) implicit representation of test patterns and tries to compress the test patterns by overlapping
- SAT-Compress algorithm does not rely on a pre-generated test set; most suitable test patterns are being generated on the fly
- The proposed method was compared with seven state-of-the-art test compression algorithms and a detailed comparison with COMPAS has been made
- Dependences on the initial state and the order of primary inputs show that the compression efficiency (ratio) can be further increased

This research has been supported by MSMT under research program MSM6840770014, by the grant of the Czech Grant Agency GA102/09/1668 and the grant of the Czech Technical University in Prague, SGS10/118/OHK3/1T/18.