

CHAPTER 6

EXPERIMENTAL RESULTS

The gate-level timing verification method described in this paper has been implemented and tested on numerous examples. Table 6.1 compares our new gate-level timing verification method using standard benchmarks against results for the timed automata tool KRONOS [32], a conservative approximation method described in [43], and the ATACS explicit state timing verifier [41]. For the KRONOS runtimes, an entry with a question mark indicates the amount of time after which the verification ran out of memory. The runtimes for KRONOS and Pena's methods are taken from their papers while the runtimes for ATACS and our new method are from a 650 MHz Pentium III. For our new method, an entry of n/a indicates that this example has an internal cycle and cannot be analyzed using our new method. For the smaller examples, our method has comparable and usually better runtimes than the other methods. However, for larger examples such as *trimos-send*, our method is more than two orders of magnitude faster than KRONOS, twenty-five times faster than Pena's tool, and nearly three times faster than the explicit state method in ATACS.

Since our goal is to determine which gates have hazards on their outputs, we configured the explicit method in ATACS to continue after finding one hazard and identify all hazards. It should be noted that KRONOS did not check for hazards, but it instead was only checking conformance while Pena's tool halted after a hazard is found. The last column of the table indicates the number of gates that have hazards found by the explicit state method and our new method. Despite being a conservative approximation, our method found the exact number of hazards in most cases. However, in three examples, *rpdf*, *sbuf-ram-write*, and *sbuf-send-pkt2*,

Table 6.1. Comparison of standard benchmarks against other timing verification tools.

Example	Gates	KRONOS CPU Time	PENA CPU Time	ATACS CPU Time	New Method CPU Time	Hazards
alloc-outbound	11	0.09	3	0.37	0.14	0/0
chu133	9	0.63	1	0.14	0.11	1/1
converta	12	0.19	12	0.26	0.09	2/2
dff	6	0.19	3	0.18	n/a	3/?
ebergen	9	0.14	1	0.18	0.11	3/3
half	7	0.41	1	0.11	0.09	1/1
mp-forward-pkt	10	0.24	5	0.19	0.14	0/0
nowick	10	0.05	3	0.22	0.11	0/0
rcv-setup	6	0.22	1	0.22	0.10	0/0
rpdf	8	2.93	2	0.35	0.11	1/2
sbuf-ram-write	17	31.77	415	0.40	0.18	1/2
sbuf-read-ctl	10	0.13	2	0.17	0.12	0/0
sbuf-send-ctl	13	54	0.49	0.87	0.14	1/1
sbuf-send-pkt2	13	0.07	103	0.61	0.16	0/1
vme	12	0.39	30	0.49	n/a	1/?
mr1	16	607.43	317	0.31	n/a	0/?
tsend-bm	12	589.56	46	5.83	n/a	1/?
mmu	22	595.09?	480	0.66	n/a	0/?
mr0	20	593.24?	48	0.62	n/a	0/?
ram-read-sbuf	17	678.48?	550	0.41	0.18	0/0
trimos-send	24	580.33?	127	13.75	5.06	5/5

our new method found one additional false hazard.

The key advantage of our new method is its ability to be able to efficiently verify circuits with a large number of internal signals. In order to demonstrate this, we selected a few of our benchmark circuits derived from a variety of sources, and we derived gate-level circuits for them that use only 2-input NAND gates and inverters. Our results are shown in Table 6.2. In all the examples, our method is still able to check for hazards in less than a second while for the largest examples the explicit state method cannot complete.

Table 6.2. Comparison for decomposed netlists.

Example	Gates	ATACS CPU Time	New Method CPU Time	Hazards
slatch	13	3.28	0.16	0/0
lapbsv	14	0.89	0.12	0/0
scsiSV	14	0.96	0.12	0/0
elatch	17	88.0	0.23	0/0
cnt3	32	40.9	0.17	6/6
srgate	36	6.65	0.26	0/0
selopt	66	>1000	0.70	?/21
cnt11	86	>2000	0.99	?/26