Scan Chain Reordering

Anshuman Tripathi
(07CS3024)

and

Manaal Faruqui
(07CS3011)

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur-721302

1 Problem Statement

Given a sequential circuit we try to minimize the power consumption for testing by reordering the f/f s in the scan chain. By reordering the scan chains only power consumed in the scan-in and scan-out phase of the circuit can be minimized. The power consumption of a D-f/f can be approximated as the number of transitions $0 \rightarrow 1$ and $1 \rightarrow 0$. Hence in this report we present a way to reorder the scan chains using the technique of simulated annealing to minimize the number of transitions in the scan chain during the scan-in and scan out phase.

2 Approach

In this section we throw some light on the step-by-step design and approach which giving insights for each of the steps.

2.1 Parsing

The project supports circuit description in ISCAS89 format. For this we borrowed the parser provided with the ISCAS89 specifications and remodeled the code to enable simulation of the circuit with given scan-in pattern and PI's. Simulation of the circuit is only necessary to find the corresponding scan-out pattern for the given input test vector. The code for parser is in the files :-

trans.l : Lexical analyzer for the ISCAS89 format (same as provided in the original ISCAS89 specifications)

trans.y : Bison specifications for the ISCAS89 format specification. Most of the code borrowed from the specifications has been changed to plug-in with the simulator code.

2.2 Simulator

As mentioned above the simulator is need to simulate circuit for one clock cycle to find the corresponding scan-out pattern for the scan-in pattern. The simulator is a simple-simulator implemented using dynamic programing approach to simulate the circuit. The Simulator tries to find the value of the output via recursively computing the values of corresponding input line. The the simulator computes the state each D-f/f if not already computed. The code for the simulator is in the following files:-

simulator.c : The implemention of the data structures and the simulator code. Export the function with signature `int simulate(int* input, int* dff, int* output)`, to simulate a circuit.

simulator.h : The header helper file for the simulator.

2.3 Simulated Annealing

As mentioned earlier the project uses search technique of simulated annealing to reorder the flipflops in the scanchain. We implemented a module to find the solution for a traveling salesman problem in a weighted clique using the technique of simulated annealing. The search technique of simulated annealing is adopted on the current project since it can converge to the globally optimal solution very quickly. Solving TSP using simulated annealing is very well studied in literature and provide good approximation results. The code for this component is in the following files:-

simanneal.c : The implementation of the simulated annealing algorithm. The file export a function `state simanneal()`, that can be used to find the solution to a TSP problem, given the weight of edges between each of the edges. The return type `state` contains the sequence of the nodes in the solution and the cost of the TS Path.

simanneal.h : Helper file to use the simnanneal code. The file defines an `extern` variable `ITS` that can be used to control the number of iterations simulated by the annealing.

2.4 Re-ordering

We reduce the problem of re-ordering the flip-flops to the problem of solving traveling salesman path as shown in [1]. For the sake of simlicity of implementing the simulated annealing part we use the following convention for scan-in and scan-out patterns:-

scan-in : The scan-in pattern is specified in the same order as it would appear in the scan chain after the scan-in phase. Thus for a scan pattern `0011` the actual order in which the bits would have been scanned in were `1100`.

scan-out : The representation for scan-out patterns is opposite of the scan-in i a way that the pattern is specified in the order as it would appear form the SO pin in the scan-out phase. That is for scan-out patter `0011` the actual order of bits from the SO pin is also `1100`.

The reason for using the above representations is that the number of transitions can be easily calulated by adding hamming distances for all the flipflop vectors and ust multiplying the results by $n_{f/f} - 1$. The code for the Re-ordering is in the following files:-

numtran.c : The main file for the code which reorders the scan chain (assuming the original scan-chain consists of the flip flops as they appear in the ISCAS89 specification file) and outputs the improvment and the final scan chain order.

define.c : Helper file that contains declaration of several tunable code parameters.

2.5 Generator

To test the efficacy of the approach one needs to generate test patterns to be applied in the testing phase. We created a random test pattern generator (this could be improved by integerating ATPG approch in the code) that generates a stream of test vectors where the probability of each bit being 1 (and thus for 0) is 0.5. The code for the generator is in file `generator.c`.

3 How To Use

The code is provided with a `Makefile`. The code needs `yacc` and `flex` installed prior on the system. To compile the code use:-

```
$ make
```

and to clean-up the output of compilation use:-

```
$ make clean
```

the compilation produces two executable files:-

generate : use as

```
./generate 12 100
```

here the first parameter is the size of a single test vector (depends on the PI's and f/f's in cct) and second parameter is the number of test vectors to be generated.

reorder : usage:

`./reorder data 100000 < s27.bench`

Here the first parameter is the file name for test patterns (generated using `generate`), second parameter is the number of iterations done by `simanneal()` and the file redirected in is the ISCAS89 specification of the circuit.

For easy use of the code a shell script `runall.sh` has been included in the code. Usage:-
`./runall.sh s298 17 100`
The first parameter is the name of the circuit (`.bench` file should be in a `DATA` folder in current directory). Second paramenter is the size of a single test vector and third argument is the number of testvectors.

the script simulated the procedure for 100 times and gives the averaged over ouput with original number of transitions, final transitions and the efficacy.

4 results

In this section we show some results got from running code on some wekk known ISCAS89 circuits. Table 4 shows the results for some well known ISCAS89 benchmark circuits. We notice

cct	**vector size**	**num_patterns**	**initial**	**final**	**efficacy**
s298	17	100	8915.46	5741.5	33.58%
s27	7	20	46.08	31.83	29.65%
s344	24	100	7946.02	5267.71	33.67%
s349	24	100	8090.19	5184.4	33.85%
s386	13	100	1475.28	1380.83	6.4%

Table 1: Efficiency (average over 100 runs)

that using the mentioned approach one can acheive improvements as high as 33%. Thus it can be said that the approach of simulated annealing does reduce the test power by appreaciable amount. Figure 1 shows the variation of the number of transitions in the s298 ISCAS89 circuit when the number of iteration of simulated annealing are increased. It can be see that the solution converges to the optimal (or close to optimal) at a very fast exponential rate. The number of test vectors were 100 for the above scenerio.

5 File Descriptions

Here we give the final list of files that constitute the code:-

numtran.c : The main file to reorder the scan chain in a circuit specification.

simulator.c : The implementation of the dynamic programming based simulator.

simulator.h : Helper file to expose the simulator interface.

simanneal.c : Implementation of the simulated annealing algorithm to solve TSP in weighted cliques.

simanneal.h : Helper file to expose the interface of simulated annealing.

define.h : Helper class that defines constants used during the parsing of a circuit specification.

trans.l : Flex specification as included in the ISCAS89 specifications

trans.y : Yacc specification as included in ISCAS89 specifications

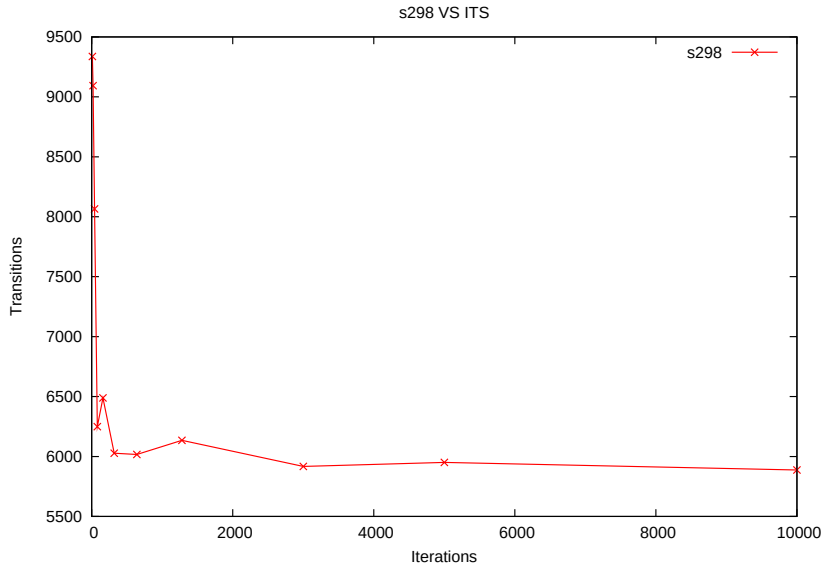

Figure 1: stabilization as number of iterations increase

tokens.h : Constants defined to be used in lexical analysis

run_sim.sh : helper shell script to run the code

take_avg.py : Helper python script to take average of results (used by `run_sim.sh`)

Makefile : The compiler specifications.

References

[1] U.S. Mehta, K.S. Dasgupta, N.M. Devashrayee, and K. Choksi. Hamming distance based distributed scan chain reordering for test power optimization. In *India Conference (INDICON), 2010 Annual IEEE*, pages 1 –5, dec. 2010.