Improving the Iterative Power of Resynthesis

Petr Fišer, Jan Schmidt
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

***Abstract*— We present a method of improving the iterative power of resynthesis of Boolean networks in this paper. In principle it is based on iterative resynthesis of parts of the network, instead of processing the network as a whole. The parts are randomly selected, thus more variability is introduced. The process is scalable, at least as much as the state-of-the-art. We show that our method performs better than the academic state-of-the-art, the ABC tool from Berkeley. This is documented by extensive experiments on LGSynth'93 benchmark circuits.**

Keywords- logic synthesis, resynthesis, iterative processes, ABC

I. INTRODUCTION

The synthesis process, where the forms of its input and output are equal (Boolean networks, AIGs), is called *resynthesis* [1]. Thus, by resynthesis we understand a process modifying the circuit in some way, while keeping the format of its description the same.

The state-of-the-art logic synthesis consists of two subsequent steps: the technology independent optimization and technology mapping, which may be *iterated*, i.e., repeatedly re-run several times, to improve the result quality.

The academic state-of-the-art logic synthesis tool is ABC [2] from Berkeley, a successor of SIS [3] and MVSIS [4]. Here iteration is suggested by the authors of ABC, too.

We have investigated the iterative power of iterative processes in ABC and have found it insufficient in many cases. Hence, we have looked for an alternative iterative process.

We propose a resynthesis method, where the circuit is iteratively resynthesized by parts only, instead of resynthesizing the circuit as a whole – *resynthesis by parts*. Let us point out that in ABC resynthesis, also parts of the circuits are being resynthesized at a time (e.g., using the *k*-feasible cuts [5] or *windowing* [1] techniques). However, these parts are relatively small, given by limits of the algorithms they are processed by. Conversely, we propose resynthesis where major parts of the circuits are extracted (up to 99%). "Resynthesis" here means applying the whole ABC synthesis process, not just, e.g., one rewriting step [5].

Such an approach may look weird and condemned to be less efficient than resynthesis of the whole (100%) circuit, since global information is lost. Surprisingly, this is not the case; we were able to obtain circuits more than 7-times smaller (or 87% improvement), compared to the standard resynthesis. This could be rationalized by an increase of the iterative power of the resynthesis – new structures can be discovered by intentionally obscuring the structure of the whole network.

The behavior of the *resynthesis by parts* process is studied thoroughly in the experimental part and its iterative power and effectiveness are compared with a standard iterative synthesis process in ABC on LGSynth'93 [8] and MCNC [9] benchmarks. Only combinational circuits are assumed here, however, the method could be extended to sequential circuits without any principal modifications.

II. THE PROPOSED ALGORITHM

Let us assume an iterative resynthesis process, i.e., a process which could gradually improve the solution when it is run several times consecutively. Let the network N^1 be obtained by running a resynthesis process P on N^0, i.e., $N^1 = \mathrm{P}(N^0)$. The subsequent iterations of this process produce different networks, $N^i = \mathrm{P}(N^{i-1})$. In an ideal case, $cost(N^i) \leq cost(N^{i-1})$ for each i, where $cost()$ is the selected optimization criterion (area, delay). Area, in terms of 2-input gates, will be considered throughout this paper.

The proposed *iterative resynthesis by parts* is based on dividing the processed network into two disjoint parts in each iteration, $N^i = N^i_A \cup N^i_B$, $N^i_A \cap N^i_B = \emptyset$, nothing is said about the sizes of N^i_A and N^i_B for now. Then one part (N^i_A) is resynthesized, to obtain a functionally equivalent network N^i_R. This network is then merged with the second part (N^i_B), to obtain a new network $N^{i+1} = N^i_R \cup N^i_B$. Obviously, networks N^i and N^{i+1} are functionally equivalent.

The overall synthesis process and part (window) selection algorithms are presented in this section.

A. The Synthesis Process

The basic and general principles of the proposed resynthesis process are shown in Figure 1.

```
Resynth_by_parts(Network N) {
    do {
        N_A = Extract_window(N);
        N_B = N - N_A
        N_R = resynthesize(N_A);
        N' = N_R ∪ N_B;
        if (cost(N') ≤ cost(N)) N = N';
    } while (!end());
}
```

Figure 1. The resynthesis by parts algorithm

At the beginning of each iteration, a part N_A of the network (window) is selected and extracted from the original network N. N_B then consists of the remainder of the original network; nodes included in N_A are not present in N_B.

The extracted window N_A is then submitted to ABC synthesis. Any synthesis process may be used in general.

The resynthesized network N_R is then merged with N_B. If the resynthesis has brought any improvement, i.e., if the network cost is reduced with respect to the original network, the old network is discarded and the new one is considered for the next iteration.

The whole procedure is repeated, until the stopping condition is satisfied. In experiments presented in this paper, we use a fixed number of iterations, for purposes of comparison. However, more sophisticated stopping criteria should be applied in practice.

B. Window Extraction Methods

The **Extract_window** procedure is the essential step in the proposed resynthesis procedure. Two window extraction algorithms will be described in detail in this subsection.

1) Random Extraction

The **Random Extraction** algorithm is the most naive one; nevertheless it gives surprisingly good results. The window (N_A) is gradually constructed by just *randomly* selecting nodes, while keeping the window network connected. The algorithm is parameterized by the number of gates of the extracted network (*size*).

We have also experienced with modifications of this basic algorithm minimizing the number of the window PIs, POs, or both. However, no significant improvement was observed [10].

2) RadiusExtract

This algorithm intentionally looks for the *most connected subcircuit*. First, a *pivot* node is selected *randomly* in the network. Then, nodes reachable in a given distance (radius) from the pivot are moved to N_A. The algorithm is parameterized by the radius or by the maximum window size, as in the previous case. Thus, the algorithm may operate in two modes, or their combination.

III. Experimental Results

As the resynthesis iterative step we have chosen the ABC *"choice*" script [2] followed by the *"map"* command, mapping the circuit into library gates. A library of all 2-input gates was chosen for simplicity of comparison.

Let us note here that any synthesis process may be used, without any loss of generality. Any structure-non-destroying resynthesis procedure may be applied, as well as any technology mapping process (standard cells, LUTs, etc.).

If not said otherwise, we have run all the resynthesis processes for 5,000 iterations in our experiments. This value is a little bit of an overkill, since only two of the examined 228 circuits needed more iteration to converge [10] using a standard synthesis process. However, it enables us to compare rather stable solutions and measure the convergence of the processes more precisely.

A. Influence of the Window Size

Here we investigate the influence of the window size on the result quality. First, we will examine the influence of the window size, relative to the resynthesized circuit size, for the Random Extraction algorithm. We have varied the window size from 10% to 100%, for all the 228 circuits. Then we have computed the average improvements obtained w.r.t. the repeated resynthesis of the whole circuit. The results are shown in Figure 2.

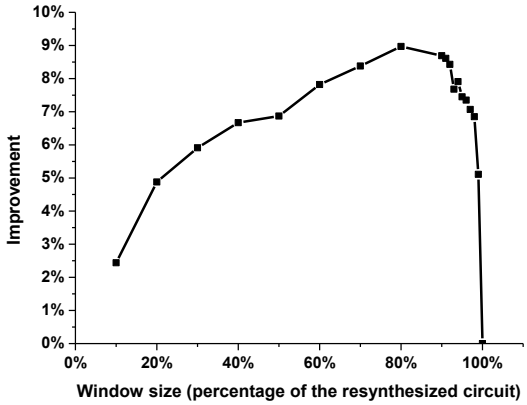

Figure 2. Influence of the window size – Random Extraction

We can see that the maximum improvement is achieved for window sizes ranging from 80 to 90% of the resynthesized circuit. If bigger windows are extracted, the quality of the result quickly drops.

Very similar behavior can be observed for the RadiusExtract. However, the maximum improvement is achieved for smaller windows (60-70%). This is because the RadiusExtract method naturally produces more compact windows.

The dependency of the improvement on the radius is shown in Figure 3. Here we see clearly, that best results are obtained for radii ranging from 5 to 7. Higher radius values produce inferior results, in most cases because window starts spanning the whole circuit, i.e., results of 100% resynthesis are obtained.

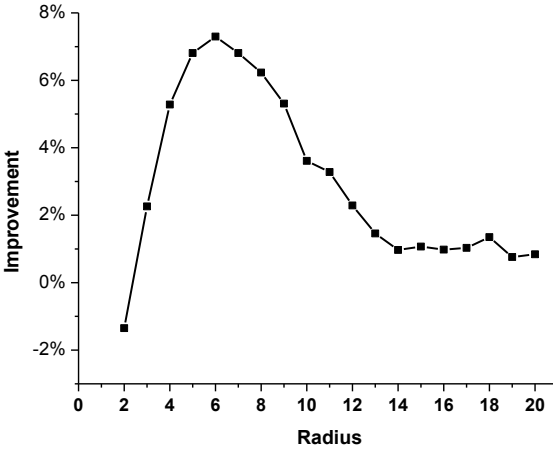

Figure 3. Influence of the window size – RadiusExtract

We have also observed that the optimum radii 6-7 generate windows of 60-70% of the resynthesized circuit, which coincides with the optimum relative window sizes, as mentioned above.

Then we have observed that the extracted window sizes *scale* with the resynthesized circuit size, even when a constant radius is set. This is a rather surprising observation, since the window extraction procedure is strictly local. This could be explained by the fact, that the extracted window often prematurely reaches "the border" of the circuit for smaller circuits. The scatter graph illustrating the dependency is shown in Figure 4. for all the 228 circuits resynthesized (5,000 iterations, radius 6).

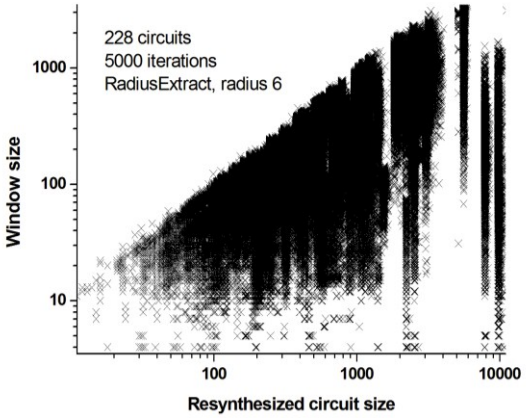

Figure 4. Extracted window sizes

B. Comparison with Standard Synthesis

Here we will present a comparison with the state-of-the-art, i.e., the iterative resynthesis of the whole circuit (100%). Results of 10 largest circuits from the 228 ones (plus *cordic*) are shown in TABLE I. All the iterative processes were run for 5,000 iterations.

After the benchmark name, its original size in terms of 2-input gates is given (*"orig."*). Then the number of gates obtained by 100% resynthesis is shown ("*100%*"). The "*conv. iters.*" column gives the number of iterations ABC needed to reach the final solution, thus possibly converge to a stable solution. However, very high values indicate that probably even better solutions could be reached, if iterated further (more than 5,000 iterations). Numbers of gates obtained by the RadiusExtract and Random Extraction methods and percentage improvements w.r.t. the 100% resynthesis follow. The "*eq. iters.*" columns indicate the numbers of iterations needed to reach the solution of at least the same quality as the one obtained by 100% resynthesis. Radius 6 and 80% circuit parts were extracted, for the RadiusExtract and Random Extraction, respectively. The summary (for numbers of gates) and average (for percentages and numbers of iterations, respectively) results are presented in the last table row.

We can see that resynthesis by parts, both RadiusExtract and Random Extraction, almost always produces better results than 100% resynthesis. Moreover, also a speedup can be seen – resynthesis by parts reaches the same solution as 100% resynthesis in significantly less iterations (8-times on average).

The results obtained from all the 228 examined benchmarks are also shown in Figure 5. The scatter-graph visualizes the improvement achieved by the resynthesis by parts (Random Extraction, 80%, 5000 iterations), as a function of the original circuit size (in terms of 2-input gates). Notice the logarithmic x-axis. Highest improvements are achieved for mid-size circuits here, however significant improvements can be seen even for larger circuits.

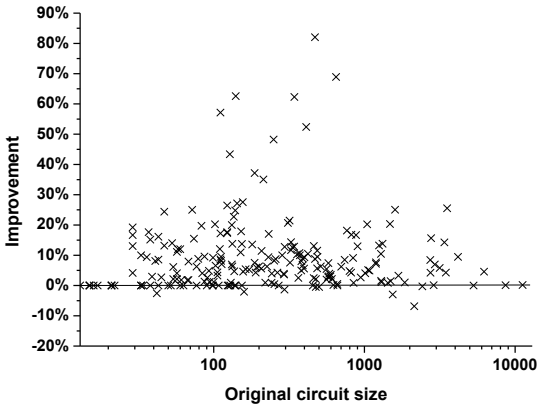

Figure 5. Summary results

C. Iterative Power

The main cause of the observed success of the method is an increase of the iterative power. Here we present representative results obtained from the *misex3* circuit [8]. This circuit belongs to the "hardest" ones, since even the 100% resynthesis converges rather slow, see TABLE I.

We have resynthesized this circuit using Random Extraction, window size 80%. Since the process is not deterministic, different runs may produce different solutions. Thus, we have run the resynthesis 20-times and observed the progress of the solution and the span in the result quality.

The convergence curves are shown in Figure 6. The topmost curve belongs to the 100% resynthesis case, the curves obtained from the 20 random runs are drawn below. We can see that the 100% resynthesis has never outperformed the resynthesis by parts in the 5,000 iterations.

Such a behavior can be observed for all circuits "difficult" for synthesis. For "easy" circuits the global optimum is found quickly by both methods.

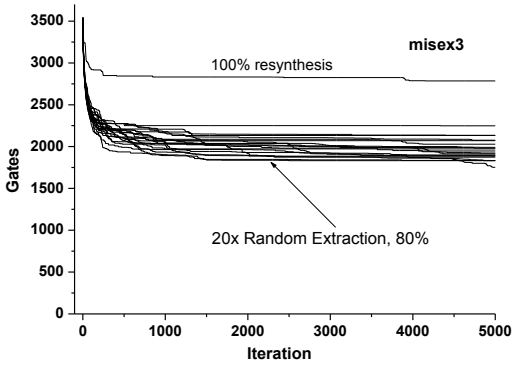

Figure 6. Convergence curves for *misex3*

IV. Discussion

A. Window Size Analysis

Experiments performed in Subsection III.A indicate that the two window extraction algorithms behave rather consistently. Even though the Random Selection naturally needs larger windows to be extracted, optimum window sizes are produced implicitly by RadiusExtract. This confirms the theory that the processes do not behave chaotically and purely at random.

Percentages 80% (60% respectively) represent feasible circuit portions to be resynthesized – only very local transformations would be enforced by smaller windows; for larger windows the synthesis quickly sinks to the 100% resynthesis behavior.

B. Convergence Analysis

The convergence curves in Figure 6. indicate that resynthesis by parts is a process quite different from the original one. The convergence is much slower, which sometimes leads to local minima avoidance and better results.

However, figures in TABLE I. indicate that resynthesis by parts is able to reach results of equal quality as 100% resynthesis in less iterations (compare the "*conv. iters.*" and "*eq. iters.*" columns).

Moreover, if the resynthesis by parts was implemented directly in ABC (e.g., upon AIGs), the process would naturally be faster than 100% resynthesis.

V. Conclusions

We have presented a new concept of iterative logic synthesis – the *resynthesis by parts*. Instead of repeatedly resynthesizing the whole circuit (which is the state-of-the-art), only partially randomly selected *large* parts (60-90%) are resynthesized at a time. This significantly increases the iterative power of the synthesis. We have shown experimentally that the process behaves consistently and the success does not consist in introducing randomness only.

The proposed method was evaluated on standard logic synthesis benchmarks. The average improvement w.r.t. the state-of-the-art, in terms of the area, was 9%. However, in some cases we have obtained up to 7-fold area reduction.

We have also shown that the method is able to produce equal results than the state-of-the-art in a significantly shorter time (8-times, on average).

The process is highly scalable, at least as much as the state-of-the-art resynthesis, for two reasons: 1) the window sizes scale with the design size, 2) large windows do not represent any problem; the method will lapse into the state of the-art 100% resynthesis in the limit case.

Acknowledgement

This research has been supported by the grant of the Czech Technical University in Prague, SGS12/094/OHK3/1T/18.

References

[1] R. K. Brayton et al., "SAT-based logic optimization and resynthesis", In Proc. of International Workshop on Logic Synthesis 2007, pp. 358-364.

[2] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", http://www.eecs.berkeley.edu/~alanmi/abc/

[3] E.M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis", Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992.

[4] M. Gao, Jie-Hong Jiang, Y. Jiang, Y. Li, S. Sinha, and R.K. Brayton, "MVSIS", In The Notes of the International Workshop on Logic Synthesis, Tahoe City, June 2001.

[5] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis". In Proc. of the 43th Design Automation Conference, San Francisco, 2006, pp. 532-535.

[6] H. Savoj and R.K. Brayton, "The Use of Observability and External Don't Cares for the Simplification of Multi-Level Networks", In Proc. of the Design Automation Conference, 1990, pp. 297–301.

[7] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure", In Proc. of IWLS 2006, pp. 15-22.

[8] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0", Mentor Graphics, May 1993.

[9] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991.

[10] P. Fišer and J. Schmidt, "It Is Better to Run Iterative Resynthesis on Parts of the Circuit", In Proc. of IWLS 2010, Irvine, CA, pp. 17-24.

TABLE I. Benchmarks results

name	*orig.*	*100%*		*Radius 6*			*Random 80%*		
		gates	*conv. iters*	*gates*	*impr.*	*eq. iters*	*gates*	*impr.*	*eq. iters*
s38584.1	11210	9752	1342	9692	0.6%	2187	9735	0.2%	1138
s38417	8643	7891	1934	7834	0.7%	808	7883	0.1%	261
prom1	6220	5829	3769	5548	4.8%	11	5562	4.6%	48
too_large	4182	3033	2467	3129	-3.1%	N/A	2746	9.5%	215
misex3	3539	2645	4147	2362	10.7%	2909	1970	25.5%	179
mainpla	3472	3091	4215	3027	2.1%	481	2958	4.3%	19
apex2	3394	2083	41	1998	4.1%	3165	1786	14.3%	275
des	3158	2915	1233	2815	3.4%	74	2746	5.8%	39
xparc	2930	2540	396	2406	5.3%	108	2363	7.0%	14
seq	2771	2024	2161	1803	10.9%	1157	1707	15.7%	129
cordic	470	334	8	44	86.8%	17	60	82.0%	19
Sum/avg.	**136,755**	**117,215**	**398.2**	**110,923**	**7.3%**	**102.0**	**109,335**	**9.0%**	**49.5**