It Is Better to Run Iterative Resynthesis on Parts of the Circuit

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—**In this paper we investigate iterative logic synthesis processes. A well known academic logic synthesis tool ABC incorporates many synthesis algorithms and scripts which may be run iteratively to possibly improve the result. When iterating the synthesis process, the whole network is considered. We propose an alternative approach to iterative synthesis – only properly selected parts of the circuit are submitted to resynthesis, which is done iteratively. We show that a significant improvement in the result quality may be achieved. This observation is rather surprising and witnesses probably a lack of efficiency of the ABC resynthesis control. The observations are documented by numerous experiments on ISCAS and IWLS'93 benchmark circuits.**

Keywords-logic synthesis, resynthesis, iterative processes, ABC

I. INTRODUCTION

The classical combinational synthesis flow established in the 1980's is a cascade of specialized, distinguished steps: minimization, decomposition, technology mapping. Any description of the circuit is first brought to Sum of Products (SOP) form, and only then the circuit structure is built by decomposition. The advantage of such an approach is greater independence of the form of input description. To repeat such a process, however, is nearly meaningless, as structural information is lost during the first step.

The optimization processes and representations involved have serious and well-known scalability problems. Therefore, the next generation of synthesis algorithms relied on local transformations. As the forms of inputs and outputs of those processes were equal, the transformations could be iterated, which leads naturally to the notion of *resynthesis*.

Already in SIS, which uniformly works with Boolean networks, there were synthesis steps *recommended* for iteration. This was continued in ABC [1]. The unified representation is based on And-Inverter Graphs (AIGs) [2], with the ability to represent alternative representations (*chioces*) added [3], [4]. Most of the ABC synthesis procedures are of a local nature. For example, the *rewriting* [5], [6] process is conducted on 4-feasible cuts by default. The *refactoring* [7] algorithm uses larger cuts. The resubstitution [5], [6] process introduces the notion of a *window*, which is even a larger part of the circuit.

Form the point of view of combinatorial optimization, such complex iterative processes are impossible to analyze. Iteration in ABC and SIS is done at two levels: at the level of local transformations under an internal control and at the level of optimization steps. This level is controlled by *synthesis scritps* [1], which have been written by experience with series of benchmark circuits and which usually prescribe fixed sequences of optimizations. This seems to open up an opportunity to employ better iteration control. But first, we have to understand the iterative processes, even on an experimental basis. During the experiments we observed an interesting anomalous behavior: processes that processed the circuits by parts gave better results and, often, in shorter time.

This permitted us to construct an alternative iterative synthesis process, which we describe here and compare it with a standard iterative synthesis process in ABC on standard benchmarks. Only combinational circuits are assumed here.

II. PRELIMINARIES

A Boolean network N (circuit) is a structure of connected single-output *nodes* forming an acyclic graph. The network connections, which are naturally inputs and outputs of gates, will be denoted as *signals*.

The network *primary inputs* (PIs) are signals that are driven by the environment; there is no node driving these signals in the network. The *primary outputs* (POs) are signals that drive the environment. Primary outputs may be driving network nodes as well.

The *size* of the network, $|N|$ is the number of its nodes. Primary inputs and outputs are not considered as nodes. Let *cost*(N) of the network be $|N|$, for purposes of this paper.

The *fan-in* of a node is the number of its inputs. Since each input must be driven by exactly one node in the network, the fan-in term will be used interchangeably for gates driving the respective node. The *fan-out* of a node is the set of nodes it drives. The *transitive fan-in* of a node is a set of nodes that drive the node. The *transitive fan-out* is a set of nodes that are driven by the node.

The *distance* of two network nodes is the number of signals the one needs to pass to reach the other one. The *level* of a node is its maximum distance from any of the primary inputs. Primary inputs have the level equal to 0.

A *window* is a connected subcircuit N_w of a circuit (network) N. Formally, it is a Boolean network N_w, $N_w \subseteq N$, whereas for every node $n_i \in N_w$ there exists a path to every node $n_j \in N_w$, $i \neq j$. In the text, terms *window* and *part* will be used interchangeably, since they have the same meaning in the formal sense.

III. MOTIVATION

Let us suppose an iterative resynthesis process, i.e., process by which the solution could be improved when it is run several times consecutively. Let a network N^1 be obtained by running a resynthesis process P on N^0, i.e., $N^1 = P(N^0)$. Subsequent iterations of this process produce different networks, $N^i = P(N^{i-1})$. In an ideal case, $cost(N^i) \leq cost(N^{i-1})$ for each i. However, this may be not true in practice, depending on the process. For purposes of illustration of the problem, let us consider dividing the network N_0 into two disjoint parts: $N^0 = N^0_A \cup N^0_B$, $N^0_A \cap N^0_B = \varnothing$, nothing is said about $|N^0_A|$ and $|N^0_B|$. Now let's run the resynthesis process on N^0_A and N^0_B separately, yielding $N^1_A = P(N^0_A)$ and $N^1_B = P(N^0_B)$. By composing the obtained network back, we obtain $N^1_{AB} = N^1_A \cup N^1_B$. Obviously, networks N^0, N^1 and N^1_{AB} are functionally equivalent.

In our experiments we have found rather surprising cases, where $cost(N^1) > cost(N^1_{AB})$. Informally, a run of the resynthesis on *parts* of the circuit, rather on the whole circuit, could yield better results, in terms of the total circuit area (number of gates). In particular, we have divided the *e64* IWLS'93 circuit [10] into two parts, resynthesized them separately and merged again. The resulting circuit had 522 gates, whereas the resynthesis of the whole circuit yielded 530 gates. Moreover, the total runtime of this resynthesis was 2.33 seconds, while the total time of the resynthesis of the circuit halves (including the time needed for the circuit division) was 1.73 seconds.

This is apparently wrong; there *must* exist a case (sequence of cut/window selections), where the resynthesis of N^0 would be conducted in the same way, as for the separated N^0_A and N^0_B parts. Moreover, global information is lost in the latter case, thus it theoretically should produce worse results in general. Even the overall resynthesis runtime may be affected; usually the runtime of the resynthesis processes grows faster than linearly, thus resynthesis by parts takes less time.

For this reason, we have investigated possibilities of resynthesizing circuits by parts more thoroughly, with a hope of discovering reasons for the above-mentioned strange behavior of the synthesis and proposing a better synthesis process.

IV. ITERATIVE SYNTHESIS IN ABC

A. Selection of the ABC Synthesis Process

First of all, a "good" synthesis process that is to be iterated must be found, to be a basis of our experiments. The required synthesis process should be universal, in the sense of result quality and runtime. In other words, it should be able to produce good results in an acceptable time, independently of the circuit processed.

All of the experiments were conducted using a mix of 228 ISCAS'85 [8], ISCAS'89 [9] and IWLS'93 [10] benchmark circuits. The circuits were mapped onto arbitrary 2-input gates. For this purpose, the MCNC library restricted to 2-input gates was used, the circuit was mapped by the ABC *map* command and finally redundant buffers and inverters were removed by the *sweep* command.

We have compared several ABC synthesis processes, described below. The results, in terms of the sum of the number of gates of the 228 synthesized circuits, are shown in Table I.

The most naive synthesis process in ABC is a mere technology mapping, using the *map* command. In fact, no real synthesis is involved here, the network AIG is just mapped onto technology. However, results obtained by this process can serve as a baseline (Table I. "*map*" row).

The results may be improved by running the basic resynthesis script *resyn* prior to the mapping (row 2). A more advanced script *resyn2* interleaving rewriting [2] and refactoring [7] produces yet better results (row 3). The *choice* script incorporates these two scripts, together with using of "choices" [3], [4] (row 4). Even better results produced the *resyn2rs* script (row 5) and the *share* script (row 6). This brings us to the idea of combining these scripts in a way of the *choice* script, yielding a new script, *superchoice*, see Figure 1. Other resynthesis scripts were tested as well, however they did not produce better results, even when used with other scripts (combined by choices), or the runtime was "too high". Let us note here, that a script producing results of almost *any* quality could be generated this way, by combining numerous different synthesis scripts by using "choices" [3] together with the FRAIG package [4]. However, this will induce a longer runtime of a single script. Here naturally arises a question where is the "usability limit" of a particular process: when repeated application of a "fast" script will yield better results in the same time? This issue will be a part of further investigation.

Synthesis procedures like collapsing (*collapse*) and disjoint-support decomposition (*dsd*) [11] sometimes produce good results (sometimes they are essential [12]). They completely abandon the circuit structure by converting it into a SOP or by using global BDDs, but this also forms a scalability obstacle. Therefore, these processes cannot be used in general, even as parts of a synthesis script based on choices.

For the above mentioned reasons, the *superchoice* script followed by technology mapping was chosen as a good "universal" synthesis script.

For details on the scripts, readers are referred to the *abc.rc* configuration file [1].

TABLE I. ABC SYNTHESIS PROCESSES

#	*Process*	*Total gates*
1	*map; sweep*	168,279
2	*resyn; map; sweep*	143,308
3	*resyn2; map; sweep*	136,669
4	*choice; map; sweep*	135,245
5	*resyn2rs; map; sweep*	131,637
6	*share; map; sweep*	128,442
7	*superchoice; map; sweep*	126,131
8	*20x (superchoice; map; sweep)*	113,479
9	*1000x (superchoice; map; sweep)*	106,216

```
fraig_store; resyn
fraig_store; resyn2
fraig_store; resyn2rs
fraig_store; share
fraig_store; fraig_restore
```

Figure 1. The "*superchoice*" script

B. Iterating the Synthesis in ABC

Assuming that every ABC synthesis script is composed of several subsequent basic synthesis procedures and that the script will not likely deteriorate the network, the result may be effectively improved by iterating the script several times. This is even more emphasized when choices [3] are used, since many different network representations are stored simultaneously. Also authors of ABC claim that repeating the "*choice; map"* sequence several times improves the result [1]. We have studied this issue more into detail. The proposed *superchoice* script was used for testing purposes.

Results obtained from iterating the *superchoice* script followed by *map* and *sweep* 20-times are shown in Table I. row 8. The script was then iterated 1000-times to show "border limits" of ABC capabilities (Table I. row 9). Iterating the synthesis process 20-times improves the total area by 10%, iterating 1000-times yields 13.6% improvement. Iterating the synthesis more times mostly does not bring any more improvement.

To further justify the above-mentioned claims, we have tracked the progress of the iterative resynthesis for all 228 circuits with 1000 applications of the *superchoice* script. The results are shown in Table II. 85% circuits converged to a stable cost value in less than 20 iterations, only 3 circuits needed more than 500 iterations to converge. Therefore, we have set 1000 iterations as a basis for our experiments. Even though such a number is rather high, it ensures that ABC itself will (almost) never produce better results, when run longer.

We have also observed that the convergence *does not* depend on the circuit size. Even though the problematic circuits shown in Table II belong to the larger ones, much bigger circuits converged faster.

TABLE II. THE *SUPERCHOICE* SCRIPT CONVERGENCE

Iterations to converge	*# of cases*
< 20	194
20 – 100	27
100 – 500	4
500 – 1000	1 (*t481*)
> 1000	2 (*seq, too_large*)

V. CIRCUIT RESYNTHESIS BY PARTS

In contrast to iterative resynthesis of the whole circuit, we propose submitting only selected *parts* (*windows*) of the circuit to the ABC resynthesis. The motivation for this was presented in Section III.

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

A. The Synthesis Process

The basic and general principles of the proposed resynthesis process are described in Figure 2.

```
Resynthesize(Network N, opt) {
    do {
        (W, N_R)  = Extract_Window(N, opt);
        W' = resynthesize_by_ABC(W);
        N' = N_R ∪ W';
        if (cost(N') ≤ cost(N)) N = N';
    } while (!end());
}
```

Figure 2. The resynthesis by parts algorithm

At the beginning of each iteration, a part W of the network (window) is selected and extracted from the original network N. N_R is then the remainder of the original network, nodes included in W are not present in N_R. Primary inputs and outputs of N are retained, primary inputs and outputs of W are constructed as follows (see an example in Figure 3):

(1) Gate inputs that are not driven by any gate in W are assigned as W primary inputs (PI_1-PI_5 in the figure).

(2) Gate outputs that do not drive any gate in W are assigned as W primary outputs (PO_1, PO_2).

(3) Gate outputs that drive some gate in N_R are assigned as W primary outputs (PO_3).

(4) Gate outputs that are primary outputs of N are assigned primary outputs of W (PO_4).

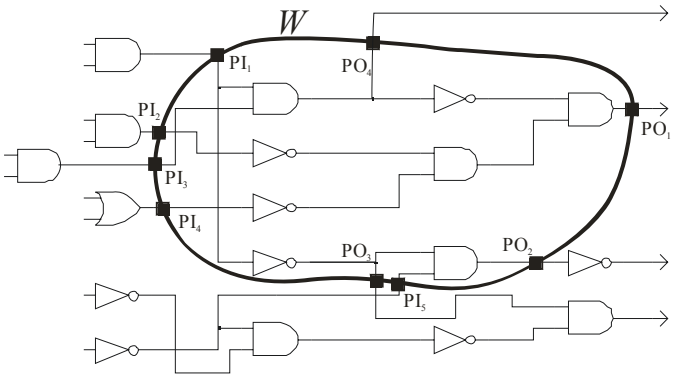

Figure 3. Window selection

The **Extract_Window** procedure is the pivotal step in the proposed resynthesis. Methods of part extraction will be described in detail later in this section.

The extracted window W is submitted to ABC synthesis. Any synthesis process may be used in general. In experiments presented in this paper we use one iteration of the *superchoice* script (Figure 1).

The resynthesized network W' is then "put back" into the network, by merging these two networks by their signals and primary input and output names. If the resynthesis brought any improvement, i.e., 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. Thus, the

resynthesis is greedy in the "first improvement" manner; non-improving iterations are discarded.

The whole procedure is iterated, until some 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. Part Selection Methods

We have implemented six algorithms of window selection, equipped with a mechanism to control window size.

Method 0 – Random selection

```
Random_Select(Network N, size) {
    n = random_node(N);
    W = {n};
    N^r = N - {n};
    while (|W| < size) {
        n = random_node(N^r);
        if (isConnected(n, W)) {
            W = W ∪ {n};
            N^r = N^r - {n};
        }
    }
    return (W, N^r);
}
```

Figure 4. Random window selection algorithm

This trivial algorithm (Figure 4) forms a basis of the three latter ones. The algorithm is parametrized by the number of gates of the extracted network. The window is constructed greedily and purely at random, only the condition of connected network must be satisfied.

Method 1 – MinimizePIs

```
MinimizePIs_Select(Network N, size) {
  n = random_node(N);
  W = {n};
  N^r = N - {n};
  while (|W| < size ) {
    candidate = N/A;
    for_each(n∈N^r) {
      if (isConnected(n, W)) &&
        FaninIncrease(n, N) <
          FaninIncrease(candidate, N)  )
          candidate = n;
    }
    W = W ∪ {candidate};
    N^r = N^r - {candidate};
  }
  return (W, N^r);
}
```

Figure 5. "MinimizeFanin" window selection algorithm

The second method locally minimizes the number of the window primary inputs. This could be beneficial for resynthesis procedures whose complexity depends on the number of inputs rather than the number of gates; this was not the case of presented experiments.

The asymptotic complexity increased $|N|$-times compared to Method 0. In each step of the main loop, every node connected to W is a candidate for selection and its cost function has to be evaluated.

FaninIncrease is the essential procedure here. It enumerates the number of primary inputs that have to be added to W, if a particular node was appended to P. Each primary input of the candidate node is checked, if it is driven by any W network node. If not, it induces an additional PI, by the rule (1) in Subsection V.A, and adds a penalty point for the candidate node. Let us note that a node driving formerly non-driven node input will be included into W in further steps, making this input an internal signal of W and decreasing the number of PIs.

Method 2 – MinimizePOs

Minimizing the total number of primary outputs of W becomes an apparent candidate for investigation. The cut selection algorithm is similar to the one shown in Figure 5, it only differs in the candidate node evaluation. Only the procedure **FaninIncrease** is modified, so that nodes inducing additional POs are penalized. The candidate node output is checked for conditions (2)-(4) stated in Subsection V.A. If any of them is satisfied, the candidate node induces an additional output, hence it obtains a penalty point. Like in the previous algorithm, some non-driven N' network nodes may be included into W in the node selection process, reducing W outputs.

Method 3 – MinimizePIs+POs

A combination of the selection criteria of Methods 1 and 2 yields a reduction of the overall reduction of the number of the W network external signals, thus it implicitly forces W to be as compact as possible, improving the chance for finding a better structure of the extracted circuit.

Method 4 – RadiusSelect

This algorithm significantly differs from the previously described four ones. The number of W inputs, outputs and nodes is not restricted. Instead, the most connected subcircuit of W is looked for intentionally. First, a *pivot* node is selected randomly. Then nodes reachable within a given radius from the pivot are moved to W. In particular, transitive fan-in and fan-out nodes of the pivot are selected. The pseudo-code of the algorithm is shown in Figure 6. A queue q of nodes is used to traverse the N network.

```
Radius_Select(Network N, radius) {
  n = random_node(N);
  q.enqueue(n);
  while (!q.empty()) {
    n = q.pop();
    W = W ∪ {n};
    N^r = N - {n};
    for_each(m∈N^r) {
      if (isConnected(m, W) &&
        distance(n, m) ≤ radius)
          q.push(j);
    }
  }
  N^r = N - W;
  return (W, N^r);
}
```

Figure 6. The RadiusSelect algorithm

Method 5 – Windowing-like Selection

Here we were inspired by the window selection algorithm used in the ABC resubstitution process [5]. As in the Method 4, the pivot node is selected first. Then, the minimum level of its transitive fan-in up to a given depth is found. After that, the transitive fan-out of the pivot is generated, up to a given depth and the transitive fan-in nodes of these are moved to *W*. Only transitive fan-in nodes having the level less than the level of the least pivot transitive fan-in level are considered.

VI. Experimental Results

Results of numerous experiments performed using a mix of 228 of the ISCAS'85 [8], ISCAS'89 [9] and IWLS'93 [10] benchmarks are presented in this section. In all the experiments, the *superchoice* script (Figure 1) followed by mapping into 2-input gates (*map*) and the *sweep* command is used for iterative resynthesis. This process was also run once on all the benchmarks, to obtain the initial circuits that are submitted to the experiments.

If not stated otherwise, the resynthesis was iterated 1000-times. All the experiments, where runtime is indicated, were run on the Ahtlon64 5600+ Dual Core CPU.

A. Area Measurement and Application to LUT Synthesis

The result quality measure is the number of 2-input gates (*not* AIG nodes). Look-up table (LUT) mapping can be also incorporated in the process. However, we do not expect better results than those obtained by post-synthesis LUT mapping. The 2-input gates offer more flexibility, due to a very low granularity of the design. Moreover, the complexities of gate-based and LUT-based syntheses correlate. To justify, we have run the *superchoice* script followed by either the *map* or *fpga* command 1000-times iteratively for all the 228 benchmark circuits. Numbers of obtained LUTs as a function of the obtained 2-input gates are shown in Figure 7. It can be seen that the dependency is linear.

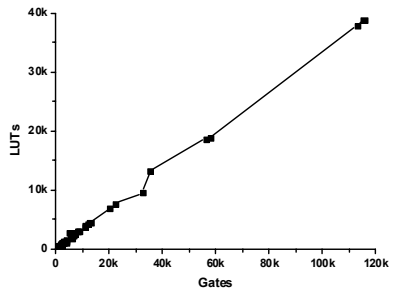

Figure 7. Gate vs. LUT synthesis

B. Characteristics of the Benchmarks

Since the characteristics of the resynthesis results heavily depend on the characteristic of the processed circuits, here we will present their statistics, namely concerning their sizes. A histogram of circuit sizes is shown in Figure 8. The numbers of circuit gates range from 5 to 11,210. The average circuit size is 600 gates, the median is 187.

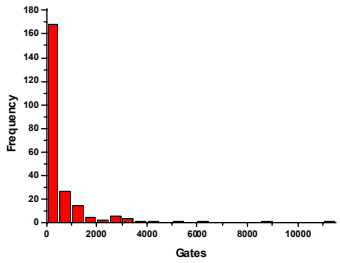

Figure 8. Benchmark circuits sizes distribution

Even though these circuits are relatively small, they represent a wide variety of parts of industrial designs. Conclusions derived in this section can be freely generalized to circuits of any size, since no dependency on the circuit size has been detected, in terms of the result quality.

C. Actual Window Sizes

To explore the nature of partial resynthesis, we need to control the size of the window, both in absolute terms and relatively to the circuit size. Methods 0-3 may lack connected nodes and return a window smaller than the requested size. In methods 4 (RadiusSelect) and 5 (Windowing-like), the window size is strictly given by the connectivity of the circuit and the selected pivot. The window size characteristics resemble the circuits' characteristics given in Subsection V.B for all methods. Therefore, we present characteristics of only one circuit, s38417 [9] with 8643 2-input gates in Table III.

TABLE III. Window size characteristics

	Max.	*Average*	*Median*
Method 0, 10%	864	776	805
Method 0, 20%	1728	1552	1600
Method 0, 30%	2592	2300	2389
Method 1, 10%	864	758	783
Method 1, 20%	1728	1510	1564
Method 1, 30%	2592	2263	2345
Method 2, 10%	864	758	788
Method 2, 20%	1728	1266	1576
Method 2, 30%	2592	2282	2361
Method 3, 10%	864	751	779
Method 3, 20%	1728	1517	1560
Method 3, 30%	2592	2229	2336
Method 4, radius 3	199	26	22
Method 4, radius 4	358	50	41
Method 4, radius 5	577	97	80
Method 4, radius 6	914	157	131
Method 4, radius 7	1606	261	193
Method 4, radius 8	1850	384	313
Method 4, radius 9	2342	552	475
Method 5, depth 3	429	68	33
Method 5, depth 4	720	87	53
Method 5, depth 5	685	96	63

	Max.	*Average*	*Median*
Method 5, depth 6	982	109	69
Method 5, depth 7	1025	111	68
Method 5, depth 8	1017	118	74
Method 5, depth 9	785	122	72

Maximum window sizes in methods 0-3 are equal to the respective percentage of the size of the original circuit. Average window sizes slightly differ between these methods, since different circuits are processed in latter iterations. The maximum, average and median values do not differ too much. Conversely, the maximum, average and median values significantly differ in methods 4 and 5, witnessing a rather steady distribution of window sizes. As an example, see the distributions for Method 4, radius 9 in Figure 9. In Method 5, the depth limit influences the window size only slightly. This can be observed especially from the median values.

Because of different characteristics of the window sizes, it is difficult to make a relevant comparison of Methods 0-3 with Methods 4 and 5. Therefore, we present results of these methods separately.

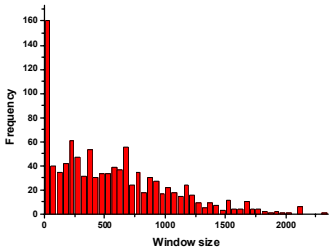

Figure 9. Window sizes distribution for Method 4, radius 9

D. The Window Selection Methods 0-3

In this subsection we experimentally evaluate effectiveness of the window selection methods. The comparison results, namely total gate counts of the synthesized 228 circuits, are shown in Table IV.

TABLE IV. COMPARISON OF WINDOW SELECTION METHODS

Size	*Method 0*	*Method 1*	*Method 2*	*Method 3*
4	126,261	126,423	125,851	125,707
5	126,005	125,816	125,462	125,031
6	125,655	125,160	125,135	124,378
10%	121,628	114,058	118,463	114,471
20%	120,190	110,308	114,905	111,014
30%	119,067	107,883	113,575	108,860

Two window size selection strategies were studied: in the upper part of the table, the window size was fixed to a given size (4, 5, 6 gates). Next, the window size was relative to the circuit size (10%, 20%, 30% of the circuit). In the latter case, the window size obviously significantly varies, depending on the circuit and the random pivot node selected. The study of actual window sizes will be presented in Subsection VI.C.

It can be seen that the Method 0 (random window selection) produces worst results in all cases (in terms of the total). Methods 1 (minimizing the window fan-in) and 2 (minimizing the window fan-out) produced very similar results for absolutely-sized windows, however Method 1 wins for percentage-sized windows. At any case, Method 3 (minimizing the sum of the window fan-in and fan-out) produces best results, in terms of quality.

The runtimes of the window construction differ for the four methods. For illustration, total window construction runtimes (sum of 1000 iterations) for the s38417 ISCAS'89 circuit for the window size of 20% of the circuit are shown in Table V. The runtimes spent by the ABC synthesis are shown as well. Surprisingly, with more advanced window selection methods these runtimes grow. Most probably this is due to more reconvergence present in the windows.

TABLE V. WINDOW SELECTION RUNTIMES, METHODS 0-3

	Constr. time [s]	*ABC time [s]*
Method 0	51,212	796
Method 1	93,811	970
Method 2	91,976	1,050
Method 3	111,536	1,084

Method 3 is the slowest one, since it combines methods 1 and 2. Still, it produces best results from these four.

Note that the window construction and ABC runtimes cannot be compared, since our resynthesis tool is rather slow, compared to ABC. The ABC code is targeted to speed efficiency, while our experimental tool is written in a more transparent code, up to ten times slower by our measurement.

E. The Window Selection Methods 4 and 5

Results obtained by methods 4 and 5 are shown in Table VI. Total numbers of gates for the 228 benchmarks, together with the total window construction and ABC runtimes for one circuit s38417 (in sense of Table V.) are given. An interesting observation can be made in Method 4: the number of gates decreases when increasing the radius, up to the threshold 8. Then, the quality lacks. The same behavior can be seen in Table VII (see Subsection VI.F). This gives us the first hint of better effectiveness of the resynthesis of smaller parts of circuits, rather than larger ones (or the entire circuit).

Even though a fully relevant direct comparison of the six window selection methods is not possible, comparison of results shown in Tables III - VI clearly show the winner. Methods 0-3 are extremely slow, in comparison to Methods 4 and 5. This is due to a large search space of methods 0-3 to be explored; the number of candidate node for evaluation in each selection algorithm step is rather high. The complexity of the algorithm is $O(n^2)$, where n is the number of network nodes. Conversely, Methods 4 and 5 proceed in a straightforward way; once the pivot node is selected, the window creation process is fully deterministic. Complexities of the methods are $O(n)$.

Regarding the result quality, Method 4, radius 8 produced the best result, out of all six methods. The lack of efficiency of Method 5 is apparent from TABLE III. The size of the

window increases only slightly with increasing the depth parameter.

TABLE VI. WINDOW SELECTION METHODS 4 AND 5

	Gates	*Constr. time [s]*	*ABC time [s]*
M. 4, radius 3	122,165	1,281	418
M. 4, radius 4	116,981	1,254	433
M. 4, radius 5	110,325	1,317	465
M. 4, radius 6	104,330	1,801	544
M. 4, radius 7	101,182	2,801	748
M. 4, radius 8	**100,929**	6,142	859
M. 4, radius 9	101,633	13,487	996
M. 5, depth 3	116,971	1,674	619
M. 5, depth 4	114,259	1,772	612
M. 5, depth 5	112,283	1,807	635
M. 5, depth 6	111,063	1,827	638
M. 5, depth 7	109,145	1,799	632
M. 5, depth 8	108,815	1,906	659
M. 5, depth 9	109,581	1,995	670

F. Comparison with ABC

A comparison of the performance of the proposed alternative resynthesis methods with ABC resynthesis run on the whole circuits is presented in TABLE VII. The reference quality value is the total number of gates of the original benchmark circuits. Average and maximum percentage improvements reached by the different resynthesis processes, with respect to this value are shown in the table.

The ABC *superchoice* script, when run 1000-times on the original circuit, reduces the total number of gates by 19.97% in average. The maximum improvement obtained by this process was 83.78% (particularly for *z4ml*, which has 111 gates originally and 18 gates after minimization).

TABLE VII. COMPARISON WITH ABC

Method	*Average impr.*	*Maximum impr.*
ABC	**19.97%**	**83.78%**
Method 0, const 5	11.79%	34.38%
Method 0, const 6	12.62%	37.63%
Method 0, 10%	14.62%	47.12%
Method 0, 20%	16.94%	54.80%
Method 0, 30%	18.21%	63.20%
Method 1, const 5	12.82%	41.02%
Method 1, const 6	13.77%	41.36%
Method 1, 10%	17.98%	51.19%
Method 1, 20%	21.84%	83.62%
Method 1, 30%	23.86%	91.06%
Method 2, const 5	12.66%	34.92%
Method 2, const 6	13.21%	36.61%
Method 2, 10%	16.24%	47.12%
Method 2, 20%	19.89%	72.40%
Method 2, 30%	21.29%	74.00%
Method 3, const 5	13.66%	43.73%
Method 3, const 6	14.55%	39.66%
Method 3, 10%	18.48%	62.40%
Method 3, 20%	22.10%	89.79%
Method 3, 30%	23.63%	85.96%
Method 4, radius 3	15.59%	41.44%
Method 4, radius 4	19.92%	88.72%
Method 4, radius 5	23.19%	91.49%
Method 4, radius 6	24.85%	90.21%
Method 4, radius 7	25.55%	91.06%
Method 4, radius 8	25.20%	89.79%
Method 4, radius 9	24.25%	89.57%
Method 5, depth 3	18.81%	84.68%
Method 5, depth 4	19.83%	82.88%
Method 5, depth 5	20.53%	90.64%
Method 5, depth 6	20.65%	82.88%
Method 5, depth 7	21.01%	90.85%
Method 5, depth 8	20.82%	82.88%
Method 5, depth 9	20.52%	82.88%

Shadowed cells in the table indicate cases, where an improvement w.r.t. ABC resynthesis run 1000-times on the whole circuit was reached. Methods 0-3 having constant window sizes up to 6 gates naturally do not overcome the ABC resynthesis of the whole circuit. However, when the circuits are iteratively resynthesized by, e.g., by halves, better results are reached in general. The Method 4 clearly justifies claims presented in Section III. There is an apparent minimum for the radius equal to 7. Increasing the window size involves a growth in the resulting circuit size. The influence of the size of the window on the result quality is discussed in the following subsection.

G. Influence of the Window Size

The negative impact of large window sizes led us to the following experiment: for a selected circuit and window selection method, we have run the iterative resynthesis with window sizes varied from 4 up to the full circuit size. Thus, the latter border case represents the case where the whole circuit is resynthesized in each iteration. The window selection method 0 (Random selection) was chosen for this experiment. The number of iterations of each fixed-sized window resynthesis process was 1000. First, we selected the *e64* IWLS'93 circuit [10], as a representative of circuits for which the resynthesis by parts brought the highest improvement. The resulting graph is shown in Figure 10. For small window sizes, the results naturally lack in quality. There is an apparent minimum around the window size 180. This is less than 28% of the original circuit size. With increasing the window size, the result area grows. This behavior corroborates our theory: resynthesis by parts is better than resynthesis of the whole circuit.

Next, we have done the same experiment with the *clip* circuit [10], a representative of circuits for which resynthesis by parts failed (particularly, 1000 iterations of ABC have reduced its 343 gates to 155, 1000 iterations of 50% Method 0 yielded 245 gates, which is a -37% difference). The graph is shown in Figure 10. It can be observed, that for windows larger than approx. 50%, the results substantially vary (ranging from

200 to 350), and have mostly random nature. Here it seems that the failure of 50% resynthesis was caused by "a bad chance", or better, the 100% resynthesis process just hit the right solution.

Graphs for other circuits and window selection methods are similar in character to those in Figure 10.

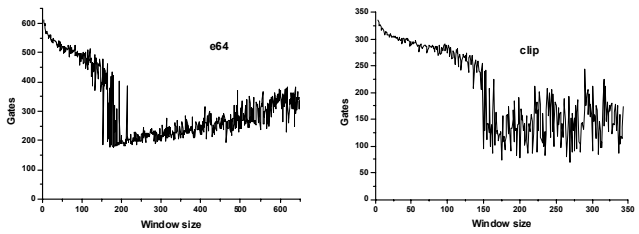

Figure 10. Varying window sizes

The convergence curves of the iterative resynthesis of the whole circuit and resynthesis by 50% parts, for the example *e64* and *clip* circuits, are shown in Figure 11. Here the reason for the *clip* failure is apparent. In both cases, the 100% circuit resynthesis process converges very quickly, while 50% resynthesis process convergence is much slower. In the *e64* case, 1000 iterations were enough for the latter process to reach a better solution. Moreover, the convergence curve indicates that the 50% resynthesis solution could be yet improved in further iterations. For the *clip* benchmark, the curves are much similar. ABC just converged to a "good" solution too quickly.

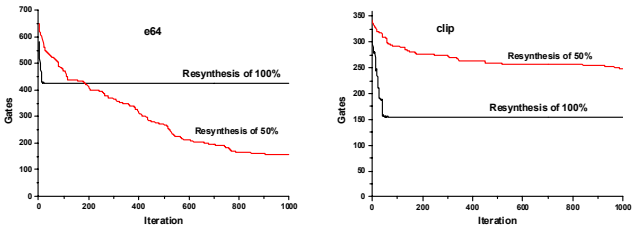

Figure 11. Convergence curves for *e64* and *clip*

VII. Discussion

The convergence curves shown in Figure 11 indicate that the resynthesis by parts is a process quite different from the resynthesis of the whole circuit. The convergence is much slower, which sometimes leads to local minima avoidance and better results. ABC aims at practical speed and must converge much faster.

Even though the slow convergence of the proposed resynthesis method could be a problem in practice, it offers a way of improving the synthesis limits. For example, we have found examples very difficult for scalable iterative synthesis, including ABC [12]. One class of these examples consists of ordinary circuits transformed into a really poor structure [13]. The only working remedy is to use a canonical structure, with all the scalability problems it brings. The presented process can be seen as an attempt at a scalable solution for the problem.

With this in mind, we can conjecture that partial resynthesis performs better – even in terms of time – *because splitting of the circuit shields the optimization from a misleading structure.*

There is an important point in interpreting Figure 10: do the curves indicate optimum *absolute* window size (which would lead to a scalable process) or a relative one? Table VI suggests that at least for Method 4 (Radius select), the size is absolute, but more data are needed to confirm this hypothesis.

VIII. Conclusions

We have tested the iterative behavior of ABC synthesis beyond the numbers employed by synthesis scripts. It appears that doing the resynthesis by parts of the circuit slows the convergence down, which, along with other phenomena, leads to better results. A process can be constructed along these lines that can be potentially more time-consuming but also much more resistant to difficult examples.

Acknowledgement

This research has been supported by MSMT under research program MSM6840770014 and by the grant of the Czech Grant Agency GA102/09/1668.

References

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

[2] A. Mishchenko, S. Chatterjee, R. K. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis" In 43th Annual ACM IEEE Design Automation Conference, San Francisco, CA, USA, 2006, pp. 532-535.

[3] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam, "Reducing structural bias in technology mapping", IEEE Trans. CAD, Vol. 25(12), December 2006, pp. 2894-2903.

[4] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification". ERL Technical Report, EECS Dept., UC Berkeley, March 2005.

[5] R. K. Brayton et al., "SAT-based logic optimization and resynthesis", In International Workshop on Logic Synthesis 2007 (IWLS), 2007, pp. 358-364.

[6] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure", Proc. IWLS '06.

[7] R. Brayton and C. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. ISCAS '82*, pp. 29-54.

[8] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan, Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985.

[9] F. Brglez, D. Bryan and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits, Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989.

[10] K. McElvain: IWLS'93 Benchmark Set: Version 4.O, distributed as part of the IWLS'93 benchmark distribution.

[11] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions", in Proc. ICCAD'97, 1997, pp. 78-82.

[12] P. Fišer, J. Schmidt, "The Observed Role of Structure in Logic Synthesis Examples", Proc. 18th of International Workshop on Logic and Synthesis 2009 (IWLS'09), Berkeley, California (USA), 31.7.-2.8.2009, pp. 210-213

[13] J. Cong and K. Minkovich, "Optimality study of logic synthesis for LUT-based FPGAs", IEEE Trans. CAD, vol. 26, pp. 230–239, Feb. 2007.