# Advanced Logic Synthesis Midterm Project Report

Jie-Hong Liu*
jiehong0914@gmail.com
College of Semiconductor Research, National Tsing Hua University
Hsinchu, Taiwan

## Abstract

This is the mid-term report on Advanced Logic Synthesis. And the author is Jie-Hong Liu from NTHU CoSR. This report include the introduction and author's experiment on ABC and SIS tool.

***Keywords:*** logic synthesis, ABC, SIS

**ACM Reference Format:**
Jie-Hong Liu. 2023. Advanced Logic Synthesis Midterm Project Report. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 Introduction

SIS (A System for Sequential Circuit Synthesis) and ABC (A System for Sequential Synthesis and Verification) are academic tools developed by UC Berkeley. They are used for the synthesis and optimization of binary sequential logic circuits appearing in synchronous hardware designs. In this report, we will compare and analyze the result of the MCNC benchmark with SIS and ABC.

## 2 Methodology

These logic synthesis tools contain thousands of instructions to optimize the boolean network. Both SIS and ABC provide lots of scripts to help the user implement the logic synthesis. In our evaluation, we are going to use two scripts in SIS. Script 1 is the script named "script.rugged", and Script 2 is the script named "script", which we can find in "./sis1.3/sis/sis_lib/".

### 2.1 SIS

Script 1 is "script.rugged" in the SIS source code, while script 2 is "script" in the SIS source code. Both of them contain some operation to do logic synthesis on the boolean network.

The operation these two scripts did is introduced in the below description.

**Algorithm 1** Script "script.rugged" in SIS to do logic synthesis

```
sweep; eliminate -1
simplify -m nocomp
eliminate -1

sweep; eliminate 5
simplify -m nocomp
resub -a
fx

resub -a; sweep
eliminate -1; sweep
full_simplify -m nocomp
```

**Algorithm 2** Script "script" in SIS to do logic synthesis

```
sweep; eliminate -1
simplify
eliminate -1

sweep; eliminate 5
simplify

resub -a

gkx -abt 30
resub -a; sweep
gcx -bt 30
resub -a; sweep

gkx -abt 10
resub -a; sweep
gcx -bt 10
resub -a; sweep

gkx -ab
resub -a; sweep
gcx -b
resub -a; sweep

eliminate 0
decomp -g *

eliminate -1; sweep
```



*Conference'17, July 2017, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/XXXXXXX.XXXXXXX

- `sweep`: Eliminate all the single-input nodes and constant nodes
- `Eliminate [-l limit] threshold`: collapse, # cubes of node > limit, # literals in node <= threashold.
- `simplify -m nocomp`: espresso for each node.
- `fx`: finds all the double cube and single cube divisors of the nodes in the network. (Greedy)
- `full\_ simplify -m nocomp`: espresso for each node in the network using the local don't care.
- `gkx -abt threshold`: Generates all kernels, chooses the best kernel intersection, and extracts divisors only while their value exceeds the threshold.
- `gcx -bt threshold`: Extract common cube with a value greater than the threshold
- `Decomp -g`: use the good decomposition algorithm. to decompose the nodes

### 2.2 ABC

ABC uses AIG to represent a network. Script 3 is 'resyn' in the ABC source code, while script 4 is 'resyn2' in the ABC source code. Both of them contain some operations to do logic synthesis on the boolean network. Script 4 performs 10 passes over the network without any zero-cost replacements, resulting in a 12% improvement

**Algorithm 3** Script "resyn" in ABC

```
b;rw;rwz;
b;rwz;b
```

**Algorithm 4** Script "resyn2" in ABC

```
b;rw;rf;b;
rw;rwz;b;
rfz;rwz;b
```

The operation these two scripts did is introduced in the below description, noted that rwz/rfz has an opportunity to escape from a local optimal.

- `b`: balance
- `rw`: rewrite
- `rwz`: rewrite with zero cost
- `rf`: refactor
- `rfz`: refactor with zero cost

## 3 Experimental Results

In this paragraph, we are going to perform experiments on the same input, where the input format is BLIF (Berkeley Logic Interchange Format). Our input is composed of MCNC benchmarks. The experimental results are shown in this report. Our experimental flow chart is shown in the following subsection. In our experiment, we use a Python script to generate the TCL commands, and then execute them using the '-xf' argument. For instance, we can use 'os.system("./abc -xf 123.tcl")'. This script would open ABC and execute the commands in 123.tcl. The source code for our experiment can be found on my GitHub[1]. The sample TCL is provided in Algorithm 5. This TCL would read the testbench from MCNC, perform the AIG transformation, and finally write it in BLIF format, so that SIS can read the benchmark as AIG format.

**Algorithm 5** Sample TCL in ABC

```
read_blif ./mcnc/mlex/9symml.blif
strash
write_blif ./AIG_blif/9symml_aig.blif
```

### 3.1 First Approach - AIG

In the first approach, for the convenience of evaluation, we convert the input benchmarks into the same representation format and compare them on the same platform. We read the MCNC benchmark in ABC and perform the AIG transformation. Note that the 'Strash' command transforms the current network into an AIG using one-level structural hashing. The resulting AIG is a logic network composed of two-input AND gates and inverters represented as complemented attributes on the edges. Structural hashing is a purely combinational transformation that does not change the number or positions of latches. After this step, we have the benchmark represented as an AIG.

### 3.2 Second Approach - Optimization

In the second approach, to evaluate the benchmarks on the same platform and using the same representation, we first transformed the benchmarks into AIG representation in the previous section. Here, we use scripts from both ABC and SIS to optimize them. In the end, we compare their literals."

#### 3.2.1 SIS.

In SIS, we use the scripts 'script.rugged' and 'script', as shown in the Methodology section, to optimize the benchmarks. Table 1 shows the experimental results of optimizing the benchmarks using SIS. The first column is the benchmark name, and the second column is the original number of literals for each benchmark. The third and fourth columns show the results after optimization using the respective scripts. Table 2 shows the ratio of the number of literals before and after optimization, for each benchmark in Table 1. These tables indicate that 'script' is more likely to reduce the number of literals than 'script.rugged'.

#### 3.2.2 ABC.

In ABC, we use "resyn" and "resyn2", as shown in the Methodology section, to optimize the benchmark. Table 3 presents the experimental results of ABC, where the first column shows the benchmark name and the second column

**Table 1.** Experimental Results on SIS optimization

| name/script | origin | script.rugged | script |
|---|---|---|---|
| C17.blif | 12 | 14 | 10 |
| go.blif | 123 | 54 | 58 |
| count.blif | 254 | 159 | 169 |

**Table 2.** Ratio of Table 1

| name/script | origin | script.rugged | script |
|---|---|---|---|
| C17.blif | 1 | 1.167 | 0.833 |
| go.blif | 1 | 0.439 | 0.471 |
| count.blif | 1 | 0.625 | 0.665 |
| Average | 1.0 | 0.744 | 0.656 |

**Table 3.** Experimental Results on ABC optimization

| name/script | origin | resyn | resyn2 |
|---|---|---|---|
| C17.blif | 12 | 12 | 12 |
| go.blif | 123 | 67 | 69 |
| count.blif | 254 | 254 | 224 |

**Table 4.** Ratio of Table 3

| name/script | origin | resyn | resyn2 |
|---|---|---|---|
| C17.blif | 1 | 1 | 1 |
| go.blif | 1 | 0.545 | 0.561 |
| count.blif | 1 | 1 | 0.882 |
| Average | 1.0 | 0.848 | 0.814 |

shows the original number of literals of each benchmark. The 3-4 column shows the result after optimization (sourcing these scripts), and Table 4 presents the ratio of the number of literals in Table 3. For example, in the second row, "go.blif" optimized by "resyn" resulted in the number of literals becoming a fraction of its original number. The table shows that "resyn2" has a higher probability of reducing the number of literals than "resyn".

Furthermore, we also do another experiment on ABC. In order to inspire the maximum efficiency of resyn and resyn2, another approach is to execute them several times. In Table 5, 1st column is the number of iteration. We do multiple iterations to too-large.blif with resyn and resyn2. The col2 and col4 is the number of literals with sourcing resyn and resyn2. And col3 and col5 is the number of level with sourcing resyn and resyn2. This table is also plot as a chart (Figure 1.

### 3.3 Compare and Evaluation

In this paragraph, we are going to compare the result of SIS and ABC, and compare them in the same time. Noted that our literals is calculated by SIS "print-stats" command. Table 5 contains origin and optimized literals in each benchmark. Table 7 contains the ratio of Table 6. By the result, we find out

**Table 5.** optimize ABC with multiple iterations

| iterations | literals | level | literals(2) | level(2) |
|---|---|---|---|---|
| 0 | 1648 | 30 | 1648 | 30 |
| 1 | 954 | 24 | 920 | 24 |
| 2 | 910 | 24 | 860 | 24 |
| 3 | 898 | 24 | 782 | 24 |
| 4 | 892 | 24 | 758 | 24 |
| 5 | 884 | 24 | 754 | 24 |

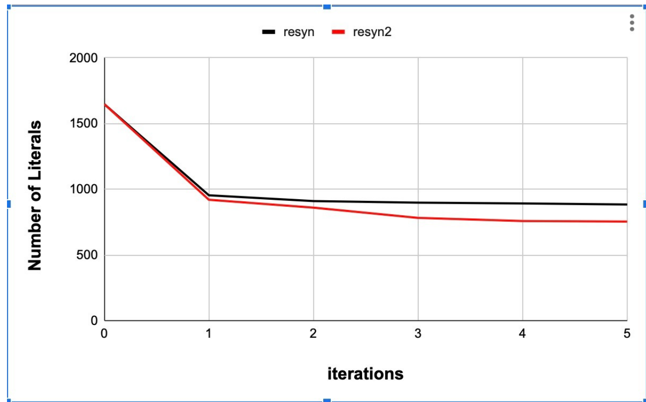


**Figure 1.** Line Chart of Table 5

that each script did well on literals optimization efficiency. By comparing the result, it shows that "script.rugged" outerperform on minimization the number of literals than other script. We guess that "script.rugged" optimize the benchmark by don't care set, which is a very useful technique in optimization the number of literals. Hence, it has the best result on literal-optimization. Figure 2 is the overall literal optimization, each y value means the sum of 20 benchmark literals, we compare them in the same plot, and it shows that "rugged" outer-perform than others on literal optimization. In Figure 1, we make Table 5 as a line chart, and this trend shows that with the number of iteration increase, the number of literal decrease. But note that, level is almost optimum with only 1 iteration, both in resyn and resyn2.

## 4 Conclusion

In conclusion, SIS and ABC are academic tools used for the synthesis and optimization of binary sequential logic circuits in synchronous hardware designs. This report compared and analyzed the results of the MCNC benchmark with SIS and ABC. The first approach was to transform the input benchmarks into the same AIG representation format and compare them in the same platform. The second approach was to optimize each benchmark using scripts from SIS and ABC. The experimental results showed that both SIS and ABC performed well on literal optimization efficiency, but "script.rugged" outperformed in minimizing the number

**Table 6.** Experimental results of 20 benchmarks and 4 optimization methods

| | origin | rugged | script | resyn | resyn2 |
|---|---|---|---|---|---|
| des_aig.txt | 8246 | 5906 | 7297 | 7132 | 7090 |
| i8_aig.txt | 6620 | 1509 | 1804 | 2238 | 2052 |
| i10_aig.txt | 5361 | 3179 | 3796 | 3725 | 3669 |
| k2_aig.txt | 3996 | 1552 | 2282 | 2534 | 2468 |
| t481_aig.txt | 3748 | 1819 | 4132 | 1752 | 1604 |
| pair_aig.txt | 3000 | 1942 | 2123 | 2578 | 2560 |
| dalu_aig.txt | 2742 | 1208 | 1547 | 2222 | 2212 |
| frg2_aig.txt | 2334 | 1031 | 1442 | 1470 | 1368 |
| vda_aig.txt | 1848 | 765 | 781 | 1278 | 1244 |
| i7_aig.txt | 1808 | 584 | 587 | 1288 | 1076 |
| i9_aig.txt | 1778 | 625 | 624 | 1204 | 1082 |
| x3_aig.txt | 1666 | 918 | 1162 | 1226 | 1212 |
| too_large_aig.txt | 1648 | 783 | 1087 | 954 | 920 |
| alu4_aig.txt | 1470 | 976 | 1335 | 1314 | 1304 |
| C2670_aig.txt | 1448 | 891 | 1212 | 1184 | 1172 |
| i6_aig.txt | 1384 | 457 | 457 | 910 | 910 |
| apex6_aig.txt | 1318 | 861 | 984 | 1234 | 1214 |
| rot_aig.txt | 1102 | 842 | 952 | 988 | 960 |
| C1355_aig.txt | 1008 | 558 | 568 | 784 | 780 |
| x4_aig.txt | 884 | 403 | 426 | 592 | 630 |

**Table 7.** Ratio of Table 6

| name/ratio | rugged | script | resyn | resyn2 |
|---|---|---|---|---|
| des_aig.txt | 0.716 | 0.885 | 0.865 | 0.86 |
| i8_aig.txt | 0.228 | 0.273 | 0.338 | 0.31 |
| i10_aig.txt | 0.593 | 0.708 | 0.695 | 0.684 |
| k2_aig.txt | 0.388 | 0.571 | 0.634 | 0.618 |
| t481_aig.txt | 0.485 | 1.102 | 0.467 | 0.428 |
| pair_aig.txt | 0.647 | 0.708 | 0.859 | 0.853 |
| dalu_aig.txt | 0.441 | 0.564 | 0.81 | 0.807 |
| frg2_aig.txt | 0.442 | 0.618 | 0.63 | 0.586 |
| vda_aig.txt | 0.414 | 0.423 | 0.692 | 0.673 |
| i7_aig.txt | 0.323 | 0.325 | 0.712 | 0.595 |
| i9_aig.txt | 0.352 | 0.351 | 0.677 | 0.609 |
| x3_aig.txt | 0.551 | 0.697 | 0.736 | 0.727 |
| too_large_aig.txt | 0.475 | 0.66 | 0.579 | 0.558 |
| alu4_aig.txt | 0.664 | 0.908 | 0.894 | 0.887 |
| C2670_aig.txt | 0.615 | 0.837 | 0.818 | 0.809 |
| i6_aig.txt | 0.33 | 0.33 | 0.658 | 0.658 |
| apex6_aig.txt | 0.653 | 0.747 | 0.936 | 0.921 |
| rot_aig.txt | 0.764 | 0.864 | 0.897 | 0.871 |
| C1355_aig.txt | 0.554 | 0.563 | 0.778 | 0.774 |
| x4_aig.txt | 0.456 | 0.482 | 0.67 | 0.713 |
| Average | 0.504 | 0.630 | 0.717 | 0.697 |

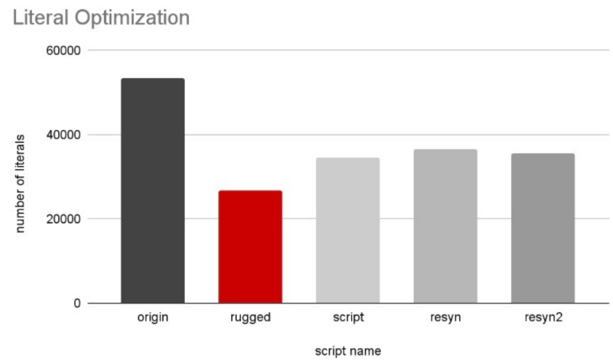


**Figure 2.** Literal Optimization

of literals. It is believed that "script.rugged" optimized the benchmark by the don't care set technique, which is a useful technique for optimizing the number of literals. Although SIS is an old tool to do the logic synthesis than ABC, it still has its strength on logic synthesis.

## References

[1] Author's Github repo
[2] Homework1 Spec
[3] A. Mishchenko, S. Chatterjee and R. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis," 2006 43rd ACM/IEEE Design Automation Conference, San Francisco, CA, USA, 2006, pp. 532-535, doi: 10.1145/1146909.1147048.

## A Appendix

### A.1 Report Thoughts and feeling

In this report, we have written several Python scripts to generate the TCL pattern and command them in two tools (ABC and SIS) to implement automation on the result. I believe this mid-term report has inspired me on how to use scripting languages to aid in experiment automation. Additionally, using CSV to save results has helped us record data, calculate averages, and plot graphs. Using these two tools has also helped me to understand the principles of logic synthesis more thoroughly. It's not just about typing commands from the command line, but also about understanding how knowledge from class is implemented in the real world.

Moreover, although the specifications did not require us to use LaTeX to finish the report, I took the initiative to use LaTeX on Overleaf. It was a brand new experience for me, and I had fun doing it.

In summary, this experience has brought me closer to the world of logic synthesis and has improved my writing techniques.