IWLS 2024 Programming Contest

Alan Mishchenko Yukio Miyasaka
UC Berkeley

In 2022 and 2023, IWLS Programming Contest focused on synthesizing the smallest possible correct circuits for a suite of Boolean functions representative of hardware designs, including random logic, arithmetic operators, and typical functionality of artificial neurons in machine learning.

The results of the IWLS 2023 Programming Contest clearly demonstrated that the methods used to synthesize the minimum circuits have not saturated. Indeed, the winner in 2023 produced circuits that were 15% smaller on average than those produced by the winner in 2022. Also, new methods were introduced to minimize the circuits, which were not compared against other methods or integrated into existing portfolio approaches.

This is the motivation to have this year's competition closely resemble the last year's competition when it comes to the rules, and only change the set of benchmarks used. As indicated in the IWLS 2023 Programming Contest announcement, we expect the participants to submit two sets of solutions, containing and-inverter graphs (AIGs) and xor-and-inverter graphs (XAIGs) for the given test cases. The results will be evaluated using the same criteria as in 2023.

We encourage the participant to carefully study the announcement of the past two competitions, appended at the end of this document, for other helpful information, including verification of the solutions using ABC.

The result should be submitted by the deadline using the following link
https://forms.gle/YwSAN3QoqDk6WXce8

IWLS 2023 Programming Contest

Alan Mishchenko, UC Berkeley

It has been recently observed that the cost of silicon wafers has started to grow steeply in the 3rd quarter of 2021. Please note the hockey stick shape of the second graph here https://morethanmoore.substack.com/p/tsmc-financial-year-2022 As a result, the design area contributes more to the design cost, and there is increasing interest in high-effort area optimization methods, which is the topic of this year's IWLS Programming Contest.

Please note that the contest this year is an extension of the contest held at IWLS 2022 (https://github.com/alanminko/iwls2022-ls-contest). Please read the description: https://www.iwls.org/iwls2022/contest/IWLS_2022_Programming_Contest.pdf

Similar to the last year, the participants are given 100 truth tables representing Boolean functions. The goal is to synthesize two circuits for each truth table: (1) the traditional and-inverter-graph (AIG), as in the last year's contest, and (2) the xor-and-inverter graph (XAIG) composed of any two-input gates, including exclusive-or gates. In the former case, the cost of a circuit, as before, is the number of internal two-input and-nodes. In the latter case, the cost is the number of any two-input gates, including xors.

The resulting score of each participant will be determined by the same formula as in the last year's competition. There will be three winners who submit functionally equivalent AIGs/XAIGs with: (i) the best score of the traditional AIGs, (ii) the best score of the XAIGs, and (iii) the best overall score. Functionally incorrect circuits as well as not submitted circuits, as before, will receive the zero score.

The question is, how to represent the resulting XAIGs. The known AIGER format (http://fmv.jku.at/aiger) for the traditional AIGs can only represent and-gates. The work-around below allows for representing xor-gates in this format.

The idea is to represent each two-input xor-gate as an equivalent combination of three and-gates with complemented attributes as needed. The resulting traditional AIG can be written in the AIGER format and verified against the original truth table, as described in the last year's competition announcement. To determine the cost of the XAIG, according to this year's competition rules, it is possible to use ABC to read the AIG, with xor-gate replaced by and-gates, using command "&read file.aig", convert it into an XAIG using command "&st –m –L 1", and list the node statistics using command "&ps –m".

For example, the benchmark "i10.aig" contains 2675 two-input and-nodes, which is the cost of this circuit as the traditional AIG, according to the competition rules:

```
abc 01> &r i10.aig; &ps; &st -m -L 1; &ps -m
i10      : i/o =    257/    224  and =    2675  lev =   50 (  15.54)  mem = 0.04 MB
```

When this circuit is converted into an XAIG, as discussed above, it has 151 two-input xor-gates and 2235 and-gates. The total two-input node count in this case is 2386, which is the cost of "i10.aig" as an XAIG, according to the competition rules:

```
abc 01> &r i10.aig; &st -m -L 1; &ps -m
Generated AND/XOR/MUX graph.
i10      : i/o =    257/    224  nod =    2386  lev =   50 (  15.54)  mem = 0.04 MB
XOR/MUX stats:  xor = 151  16.85 %   mux =  0   0.00 %   and = 2235  83.15 %  obj = 2386
```

IWLS 2022 Programming Contest

Alan Mishchenko, UC Berkeley Satrajit Chatterjee, Google AI

The goal of the contest this year is to synthesize small circuits for completely-specified multi-output Boolean functions represented using truth tables.

The benchmarks provided to the participants include the binary truth tables specified in text files containing strings composed of 0's and 1's. Each line of the file represents one output function. All functions are assumed to depend on the same variables.

A simple solution to the problem could be to read the truth tables into ABC, apply factoring, convert the result into an AIG, and synthesize the resulting AIG (*read_truth -xf* ex00.truth*; collapse; sop; strash; dc2; write* ex00.aig*; print_stats*). For many practical functions, the solution produced by this flow is not efficient. Thus, it is expected that the participants come up with novel break-through methods to solve the problem instead of tweaking and optimizing the baseline ABC flow.

The idea is not to look for a single best method, but find the best solution for different benchmarks using a variety of methods, including large scale search, enumeration, development of new algorithms, putting together new tool flows, etc, or perhaps an approach based on machine learning, which employs black box optimization/hyper-parameter search.

The deliverable is a zip file with AIG files ("ex00.aig", "ex01.aig", etc) generated for each input file ("ex00.truth", "ex01.truth", etc). The results are evaluated as follows:

(1) The AIGs are checked for functional equivalence against the input functions.

For example, AIG "ex00.aig", produced by synthesis, can be checked for equivalence with "ex00.truth" using ABC commands "*read_truth -xf* ex00.truth*; cec -n* ex00.aig".

(2) For all the participants who correctly solved a testcase, the number of points assigned for this testcase is equal to *floor*(*100**<*cost_of_best_solution*>/<*cost_of_this_solution*>) where a solution cost is determined as the number of two-input AND nodes in the AIG. If a testcase is solved incorrectly, the score for this testcase is 0.

For example, if a participant came up with a correct solution containing 53 AIG nodes, while the best solution for this testcase contains 41 AIG nodes, this participant will receive *floor*(100*41/53)=*floor*(77.35)=77 points.

The winner of the competition is the participant who received the largest floating-point average for all the testcases.