IWLS 2027 Programming Contest

Alan Mishchenko, UC Berkeley Attila Jurecska, Siemens EDA Walter Lau Neto, Synopsys

The IWLS Programming Contest this year continues the logic-synthesis competitions held at IWLS in recent years. As in 2026, the objective is not to find a single "best" solution per benchmark. Instead, the goal is to explore the tradeoff between two discrete optimization objectives, delay and area, and to produce a high-quality Pareto curve (Pareto set) of (delay, area) tradeoff points for each benchmark. What changes this year is the circuit representation and the way area and delay are measured: instead of counting AIG nodes and AIG logic levels, solutions are gate-level networks composed of AOI/OAI gates and evaluated using a transistor-level cost model described below.

Motivation: In previous contests, area was the number of two-input AND nodes in the AIG and delay was the number of AIG levels. These metrics are convenient but suffer from structural bias, and they correlate poorly with the area and delay after technology mapping into a standard-cell library, such as the freely available ASAP7. There are three main reasons: (1) complemented edges in an AIG are free, whereas in a real circuit every inversion costs transistors and delay; (2) every AND node has the same cost, regardless of whether it can be absorbed into a complex gate (such as AOI22) during technology mapping; (3) fanout is ignored, whereas in a real circuit a gate driving many loads is slower than a gate driving one load. Rather than adopting a specific standard-cell library, which would lock the contest into one technology and one set of vendor-specific cells, this year's contest uses a technology-independent "virtual" library of AOI/OAI gates, whose area is measured in transistors and whose delay is measured in transistors on the longest chain in the transistor network of the gates, taking fanout into account. This cost model is much closer to any CMOS standard-cell library than AIG node and level counts, while remaining library-independent.

The virtual gate library: An AOI (AND-OR-INVERT) gate computes the complement of an OR of AND terms over disjoint sets of inputs; for example, AOI221(a,b,c,d,e) = NOT(ab + cd + e). An OAI (OR-AND-INVERT) gate is the dual: it computes the complement of an AND of OR terms; for example, OAI22(a,b,c,d) = NOT((a+b)(c+d)). A gate is characterized by its number of terms k and term sizes (x1, …, xk). Inverters, NANDs, and NORs are special cases: INV is AOI with one single-input term, NANDn is AOI with a single term of size n, and NORn is AOI with n single-input terms (equivalently, OAI with a single term of size n; the dual representations have identical cost). In this contest, a gate is legal if it satisfies all of the following conditions:

- it has at most three terms ($k \leq 3$),
- each term has at most three inputs ($xi \leq 3$),
- the total number of inputs is at most six ($x1 + \ldots + xk \leq 6$).

Exception: NAND4 and NOR4 are additionally allowed. These conditions are motivated by realistic libraries: in static CMOS, the term sizes and the term count determine the lengths of the transistor stacks in the pull-down and pull-up networks, and practical libraries cap stack lengths at three, with four-input NAND/NOR gates as a common exception. Thus, AOI33, AOI222, and AOI321 are legal, while AOI42 (a stack of four inside a complex gate) and AOI331 (seven inputs) are not. Note that the delay model below naturally penalizes long stacks, so NAND4/NOR4 are useful mainly as an area optimization off the critical path, as in real designs.

XOR and MUX functions: The library deliberately contains no dedicated XOR, XNOR, or MUX cells. These functions are implemented by combinations of primitive gates and inverters, with the

following canonical decompositions and costs, matching the typical costs of these cells in standard-cell libraries: XOR2(a,b) = AOI22(a,b,a',b') and XNOR2(a,b) = OAI22(a,b,a',b'), each requiring two input inverters (12 transistors, worst pin delay 3); the inverting multiplexer MUXI(s,a,b) = !(s ? a : b) = AOI22(s,a,s',b), requiring one inverter on the select input (10 transistors; delay 3 from the select and 2 from the data inputs); and MUX2, which is MUXI followed by an output inverter (12 transistors; delay 4 from the select and 3 from the data inputs). Dedicated composite cells are omitted deliberately: internally, they are two-stage circuits rather than single transistor networks, and their inputs (an XOR input, or a MUX select) are attached to two transistor pairs, presenting twice the input capacitance to the driving gate. Modeling this faithfully would require per-pin input loads, an exception that is easy to overlook and that simple genlib-based flows, such as the one of ABC, do not apply by default, whereas pricing such cells without it would make them strictly cheaper than their own decompositions. In the decomposed form, the correct cost is charged automatically: the input net of an XOR drives two ordinary pins, so its driver pays two units of load delay, with no special rule. The decomposed form also enables optimizations that a monolithic cell cannot express, such as sharing one inverted signal between several XOR gates. As a result, every input pin in this library is attached to exactly one transistor pair, all input loads are equal, and the fanout term of the delay model reduces to simply counting driven pins.

Area: In a static CMOS gate, each input drives one NMOS and one PMOS transistor. Therefore, the area of an AOI/OAI gate is defined as twice its number of inputs: INV = 2, NAND2 = 4, AOI22 = 8, AOI222 = 12 transistors, etc. The area of a circuit is the sum of the areas of its gates. Note that, unlike in an AIG, inverters are explicit and contribute to both area and delay, while buffering a high-fanout signal can be performed using a BUF cell or, equivalently, a chain of two inverters.

Delay: The delay of a gate consists of an intrinsic component, determined by its transistor stacks, and a load-dependent component, determined by its fanout count. We only care about the maximum delay of any pin of a gate, and make all pin delays uniform. Accordingly, the intrinsic delay of a gate with k terms whose largest term has x inputs is defined as D = max(x, k): the number of series transistors on the longest chain in the transistor network of the gate (the pull-up and pull-down networks are duals, and the longest chain equals the largest term size in one of them and the number of terms in the other). For example, AOI22 has intrinsic delay max(2, 2) = 2, NAND4 has max(4, 1) = 4, and AOI32 has max(3, 2) = 3. For a few pins of AOI31 and AOI32 (and their OAI duals), the uniform delay conservatively rounds the pin delay up by one unit; all pins of every other gate are equal anyway under the worst-edge (slower of rising and falling) timing used here, and this convention keeps the library maximally simple: one delay per gate. In addition, if the output of a gate drives F input pins (each fanout gate pin and each primary output counts as one), the delay of every pin-to-pin path through this gate is increased by (F – 1). In summary, the delay from an input pin to the output of a gate is:

$$delay(pin \rightarrow output) = D + (F - 1)$$

For example, an AOI22 gate whose output drives three pins has delay 2 + 2 = 4 from each of its inputs to its output. The delay of a circuit is computed by standard static timing analysis: primary inputs arrive at time 0 and are free and ideally driven, with unlimited load bearing — no load penalty is applied to a primary input, regardless of how many pins it drives, matching the default assumption of static timing analysis in the absence of specified input drivers. The circuit delay is the largest arrival time over all primary outputs, with each primary output contributing one unit of load to its driving gate. With these definitions, all delays are integer.

Why the delay model is linear in fanout: The linear load term (one extra unit of delay per additional fanout pin) is intentional and reflects the underlying physics: driving a net is an RC charging process, and each fanout pin adds a fixed increment of gate and wire capacitance, so the delay of a single gate is linear in its load. In terms of the theory of logical effort, with unit-sized gates the electrical effort of a stage equals the number of fanout pins it drives, and one additional pin adds about one unit of effort delay, comparable to the intrinsic delay of an inverter; this motivates charging exactly one unit of delay per additional fanout pin, which also keeps all delay values integer, so that both optimization objectives remain simple integer counts. The load term additionally serves as a first-order wire-load proxy, since each fanout connection adds wire capacitance as well as gate capacitance; wire delay proper is intentionally out of scope, since it cannot be evaluated without placement information. Practically, the load term closes an important gap of the AIG metrics: a signal driving hundreds of loads is no longer free, and buffering and logic replication become meaningful delay-optimization techniques, as in real design flows.

Buffering of high-fanout nets: The logarithmic delay commonly associated with high-fanout nets in real designs is not a property of a single gate: it is achieved by inserting a buffer tree, which costs area and intrinsic delay. The proposed model reproduces this behavior rather than granting it for free: a net with F fanouts can be driven through a tree of inverters with branching factor b, whose delay is about $b \times \log_b(F)$ (minimized at $b = 3$, giving roughly $1.9 \times \log_2(F)$), so that logarithmic delay is available at an explicit area cost, exactly as in real design flows. Charging a logarithmic fanout penalty directly would instead model an ideal zero-cost buffer tree that is not present in the netlist, and would make actual buffering counterproductive. Consequently, high-fanout internal nets are expected to be buffered by the participants in the submitted solutions. Note that the polarity bookkeeping of inverter trees is nearly free in this library: every gate has an AOI/OAI dual, so an extra inversion can often be absorbed by switching the form of a fanout gate. Note also that no load penalty applies to primary inputs, which are free and ideally driven regardless of their fanout; fanout buffering is therefore a consideration for internal nets only.

Representative legal gates and their costs: The table below lists selected legal gates in the AOI form together with their costs (the OAI duals, such as OAI21, are legal as well and have the same costs). The intrinsic gate delays follow directly from the formula $D = \max(x, k)$, uniform over the pins of each gate. The complete library is distributed in the traditional genlib format as iwls27.genlib; the header of this file also summarizes the genlib format and the delay computation, so that participants do not need to consult the genlib documentation. The genlib delay fields encode the contest delay model exactly: the block delay of every pin of a gate is $D - 1$, the fanout delay coefficient is 1, and all pin input loads are equal to 1, so that the standard genlib delay computation, block + (sum of driven input loads), coincides with the contest delay $D + (F - 1)$. In particular, ABC can be used to obtain a legal baseline directly: “read_truth -xf ex200.truth; strash; read_genlib iwls27.genlib; map; print_stats”. Note that the delay reported by ABC’s “print_stats” for mapped networks has recently been updated to include the fanout-delay term if the genlib file includes non-zero fanout-delay.

Gate	Function (before final inversion for AOI/OAI)	Terms (k)	Area (tr.)	Gate delay D
INV	a	1	2	1
NAND2	ab	1	4	2
NOR2	a + b	2	4	2
NAND3	abc	1	6	3
NAND4	abcd (exception)	1	8	4
NOR4	a + b + c + d (exception)	4	8	4
AOI21	ab + c	2	6	2
AOI22	ab + cd	2	8	2
AOI32	abc + de	2	10	3
AOI33	abc + def	2	12	3
AOI211	ab + c + d	3	8	3
AOI221	ab + cd + e	3	10	3
AOI222	ab + cd + ef	3	12	3
AOI321	abc + de + f	3	12	3

Validation of the cost model: The purpose of the proposed cost model is not to predict the absolute post-mapping area and delay in a particular technology, but to rank alternative implementations of the same function approximately as technology mapping into a modern standard-cell library would. To validate this, the model and several of its variations were benchmarked against a number of actual technologies, on larger designs as well as smaller focused circuits, checking both whether the model exposes the correct groups of critical paths and how well the achievable areas and delays correlate. The results confirm the choices made above. The AOI/OAI library alone is an excellent area model: the geomean deviation of the area scaling factors between designs is at most 3.5% across the tested technologies, with a worst design-pair difference of 11%, and adding further cells is not necessary: dedicated MUX cells ease routing in physical synthesis but do not improve area accuracy, while a full-adder cell improves the geomean error by only about 2% even on arithmetically dense designs, which is below technology-specific effects. The uniform per-gate delay is similarly validated, deviating by only 5% in the geomean (13% for the worst design pair), and the linear form of the fanout penalty is confirmed as well.

Deliberate simplifications and possible refinements: The same experiments quantify the cost of the intentional simplifications, for readers interested in higher-fidelity variants of the model. A per-pin timing model without separate edges brings no improvement at all (and slightly worsens the area correlation), while a per-pin model with separate, still integer, rising and falling delays improves the geomean deviation from 5% to 2.5% and the worst design pair from 13% to 8.5% – a real improvement, but not large enough to justify complicating the contest model. Similarly, the unit fanout penalty is slightly conservative: the linearity assumption is accurate, but a smaller per-fanout coefficient improves the area and delay correlation by a few percent, at the price of giving up integer delays. These refinements are noted as known next steps rather than adopted rules.

Track and circuit representation: There is one track: synthesizing gate-level networks composed of the cells of iwls27.genlib. Submissions are mapped netlists that reference the library cells by

name in mapped BLIF, in which every cell is one .gate line (for example, ".gate AOI22 a=n1 b=n2 c=n3 d=n4 O=n5"), as produced by ABC's write_blif after mapping. This makes the legality rule simple: a submission is legal if and only if it is a combinational netlist (no latches) that reads successfully against the official iwls27.genlib. Unknown cell names, wrong pin counts, and wrong formal pin names are rejected by the netlist reader itself, so no separate legality checker is required.

There are no special-case cost conventions: every cell instance, including every inverter and buffer, costs its library area and delay. In particular, the constant cells ZERO and ONE have area 0 as specified in the library; a primary output that repeats a primary input is implemented with a BUF cell, which costs its regular 4 transistors and delay 2 like any other buffer; and dangling cells count toward area. Primary inputs, in contrast, are free and ideally driven, with unlimited load bearing: no load penalty is applied to a primary input regardless of how many pins it drives, matching the default assumption of static timing analysis in the absence of specified input drivers. As in previous contests, all submitted circuits must be functionally equivalent to the given truth table.

Benchmarks: The contest reuses the 100 practical benchmarks of the 2026 contest, with file names ex200.truth through ex299.truth. The high-level descriptions are not disclosed because this is a logic synthesis competition and the participants are encouraged to develop optimization methods that are universally applicable, rather than tailored to specific known circuits. Reusing the 2026 benchmark set makes it possible to directly compare the solutions produced under the new cost model with the AIG-based solutions of the previous contest.

Submissions: Each team submits one zip file whose name includes the team's affiliation. Please do not submit many zip files and do not create subdirectories inside the archive, only optimized netlist files with the proper file naming in one zip file. For each input truth table "ex2NN.truth", a team may submit any number of netlist files corresponding to different (delay, area) tradeoff points. The required file naming convention is ex2NN_DDD.blif, where DDD is the circuit delay of a testcase, computed as defined above, with leading zeros up to three decimal digits. For example, some of the submitted files may look as follows: ex200_045.blif, ex200_047.blif, ex200_052.blif, etc, meaning that this is the first testcase (ex200.truth) optimized as a series of networks having delay 45, 47, 52, etc.

Important: Teams are strongly encouraged to submit only Pareto points (see below). If a team submits redundant points, compared to other points submitted by the same team, they will be automatically removed and not considered for evaluation.

Functional correctness and legality: Each submitted network is checked for functional equivalence against the corresponding truth table using ABC (as in previous years). For example, a network for "ex200.truth" can be checked with "read_genlib iwls27.genlib; read_blif ex200_045.blif; strash; &get; &cec -t ex200.truth". Legality is established by the same flow: a file that does not read successfully against the official iwls27.genlib is not a legal submission.

Pareto points: Each submitted correct network corresponds to one point (delay, area) in the solution space. A point is considered "non-Pareto" (dominated) and will be removed if the same team has another submitted point that is at least as good in both objectives and strictly better in one of them. In other words, a point is removed if the team's submission includes a point that has better area and better delay, the same area and better delay, or the same delay and better area.

The "virtual best solver": For each benchmark, after processing all teams' submissions, as described above, the organizers will form the union of all teams' points and again remove

dominated (non-Pareto) points. The remaining global Pareto set is called the "virtual best solver" for this benchmark. Intuitively, it is the best tradeoff curve that could be obtained by taking, for each tradeoff region, the best point of any participant.

How participants are compared to the virtual best solver: The comparison is performed "delay by delay" using the discrete delay values appearing in the virtual best solver's Pareto set:

- Consider one Pareto point of the virtual best solver with a specific delay D. This point represents the best known area achieved by any participant at that delay.
- For the participant being scored, we find the participant's best available area among their own Pareto points whose delay is not larger than D (i.e., any of their points that are at least as fast as D). If the participant has no such point, the participant receives 0 points for this delay value.
- Otherwise, the participant receives an integer number of points based on how close their area is to the virtual best area at this delay. As in prior IWLS contests, this is computed as the integer part of: 100 × (virtual-best area) / (participant's area). Thus, matching the virtual best area gives 100 points at that delay value; larger areas give proportionally fewer points.

Benchmark score and final score: For each benchmark, the participant's score is the average of the points obtained over all delay values (Pareto points) in the virtual best solver's Pareto set for that benchmark. The participant's final contest score is the floating-point average of the benchmark scores over all 100 benchmarks. The winner is the team with the highest final score.

Acknowledgments: The organizers thank Philippe Sauter for independently benchmarking the proposed cost model, and several of its variations, against actual technologies. This analysis confirmed the AOI/OAI library as an area model, the uniform per-gate delay, and the linear fanout penalty, quantified the accuracy of the intentional simplifications summarized above, and sharpened the wording of this document.