Orthogonal Circuit Visualization Improved by Merging the Placement and Routing Phases

Thomas Eschbach
Wolfgang Günther
Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 202, Mai 2004

Orthogonal Circuit Visualization Improved by Merging the Placement and Routing Phases

Thomas Eschbach[1], Wolfgang Günther[2], and Bernd Becker[1]

[1] Institute for Computer Science
Albert-Ludwigs-University, Freiburg, Germany
{eschbach,becker}@informatik.uni-freiburg.de
[2] Infineon AG
CL DAT DF V, Munich, Germany
wolfgang.guenther@infineon.com

Abstract. Visualization of circuits is an important research area in electronic design automation. Locating errors in a large design may require a high-quality graphical representation of a circuit that allows humans to understand it. Usually, drawing a circuit is based on visualizing the corresponding graph or hypergraph structure where nodes are connected by straight lines, and nodes are located in a way that minimizes the crossings of these lines. Then the algorithms re-transforms this graph representation back to an orthogonal circuit structure, i.e. it replaces the straight lines by horizontal and vertical lines.
In contrast to many other approaches which route all the wiring after placing all nodes we focus on a new approach which dynamically reorders the nodes within the layers to further reduce the number of hyperedge crossings. An efficient algorithm is presented that minimizes the hyperedge crossings. Experimental results are provided that demonstrate the efficiency of the approach.

1 Introduction

The number of line crossings and the number of bends are commonly accepted measures for the readability of circuit drawings. High quality visualizations of circuits are required in the field of circuit design for several reasons. A fast visualization of the circuit can help the viewer to understand the functionality of the design. This is important in the debugging process. Other applications for circuit visualization include teaching of hardware design, presentation of new synthesis methods, and circuit related documentation.

Different goals and different restrictions make it impractical to use existing physical layout algorithms [12, 17] to compute readable circuit schematics. Two important points are:

- The result of circuit visualization is a two-dimensional drawing, while it is three-dimensional in the physical design process.
- On chip, the area constraints are more important than both the number of crossings and the number of bends. This leads to a compact but confusing routing of the wires.

A common framework for automated circuit visualization is based on visualization techniques for directed graphs first introduced by [18]. First approaches to extend the framework to orthogonal hypergraph visualization were given in [7]. An integration into an industrial toolset was discussed in [16]. Theoretical results can be found in [8]. These techniques split the process into several steps:

- All gates are placed onto a set number of layers. The inputs of the circuit are placed on the first layer and the outputs are placed on the last layer. Also, an even distribution of all elements over the layers is preferable. Promising approaches to establish this are given e.g. in [3].
- Next, the circuit is modeled as a graph (e.g. like proposed in [13]). Then, a crossing reduction algorithm is applied to obtain an ordering of all elements on each layer. After that, an algorithm is required to route all the hyperedges between the tracks. In contrast to all methods known from the area of physical design which mainly minimize the number of tracks used, the aim here is to improve the visual clarity of the drawing by only allowing every hyperedge to be part of one track (e.g. see Figures 1b, 6). Then a reordering of the tracks is done that reduces the number of hyperedge crossings.

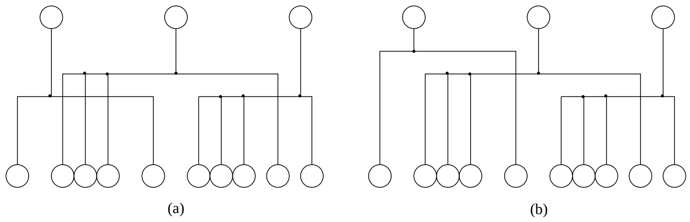

Fig. 1. Remark: A track's minimal embedding (a) does not imply a crossing minimal embedding (b).

In this work we extend the framework with a new step which dynamically reorders iteratively two nodes in one layer and repeats the orthogonal embedding of the hyperedges to further reduce the number of hyperedge crossings until a local optimum is reached. This new approach combines the placement and routing process with respect to the hypergraph structure in contrast to many known heuristic methods. Since finding exact solutions for both steps together means to traverse a huge search space, two fast heuristic methods are used by turns to compute the final solution. Combining two phases during optimization often leads to a significant increase in run time. As we are combining two very fast heuristics, this is not a disadvantage of our approach. Experimental results are given which strongly suggest the significant improvements iteratively of hyperedge crossings if the new phase is added.

2 Preliminaries

A circuit can be modeled as a hypergraph $H = (V, E)$ where gates are represented by nodes V and nets correspond to hyperedges (connecting a subset of V). Each hyperedge consists of all wires which are directly connected to each other. Nodes are created for every input, every output and every gate in the circuit. Then, the algorithm inserts the corresponding edges for every hyperedge E. An example of a circuit and its graph representation resulting from the above construction is given in Figure 2.

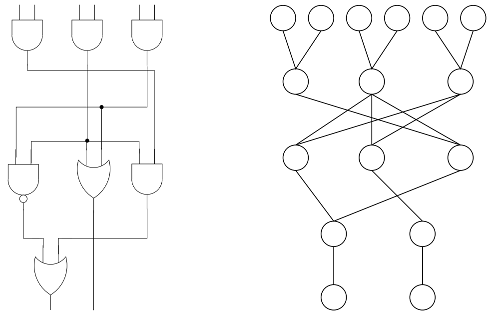

Fig. 2. Example of a circuit and its graph representation.

A directed graph $G = (V, E)$ can be converted into a *multi-layered* graph with d layers ($d \in I\!N$). By this, the node set V is partitioned into disjoint subsets V_1, V_2, …, V_d, i.e. $V_1 \cup V_2 \cup \ldots \cup V_d = V$ and $(\forall m \neq m')\ V_m \cap V_{m'} = \emptyset$, V_m is called the *m-th layer* of the graph. All edges in E connect nodes in different layers. A layering of a graph is called a *proper* layering if the edges are only connected to nodes of adjacent layers V_m and V_{m+1}. If a layering of a graph is not a proper layering, one can introduce *dummy nodes* along edges (u, v) if $\text{layer}(v) - \text{layer}(u) > 1$. We replace (u, v) by a path of length l ($u = v_1, v_2, \ldots, v_l = v$). In each layer between u and v, one dummy node has to be inserted. If multiple dummy nodes belonging to the same hyperedge exist on the same layer, they are merged. From now on we assume that the layering considered is proper.

It is important to note that the order of the nodes in a layer V_m, $m \in \{1, \ldots, d\}$, only affects the number of crossings with adjacent layers. To solve the exact multi-layer straight-line crossing minimization problem we have to determine an order ord_m for all layers m containing all the nodes in layer V_m so that the number of crossings is minimized. Unfortunately, minimizing edge crossings in graphs with two layers is NP-hard [10], and it remains NP-hard even if one ordering of a layer is fixed [6].

One of many known heuristic methods to solve the problem is the averaging heuristic method which was first introduced in [18]. It computes the position of a node n only with respect to all nodes in the above layer which are connected to it. More precisely, it computes the average of the x-coordinates of its neighbors for all nodes on one level. Then it sorts the nodes with respect to this value.

We can generalize the notation of the crossing number [5] introduced for layered graphs to layered hypergraphs if the orderings of all nodes are fixed. For each pair of nets we define the crossing number c_{ij} as the number of crossings between the nets i and j where net i is assigned to a higher track than net j. Furthermore, we define $c_{ii} = 0$ for all nets. On the right hand side of Figure 3, the crossing number $c_{1,5}$ between net 1 and net 5 is defined as one and the crossing number $c_{5,1}$ is defined as two. It is important to note that the definition of the crossing numbers and all algorithms in this paper can deal with hyperedges which are connected with r nodes in the upper layer and k nodes in the lower layer $(r, k \in I\!N)$.However, all experiments are carried out with real benchmark circuits [1, 14] where all hyperedges have a $1 : k$ relation.

To compute an orthogonal embedding of a circuit, the algorithm has to route the wires in the channel between the two layers. We divide this channel into tracks, and only one wire is allowed per track. This prevents wires within one track overlapping or being too close. Thus, every net is represented by at most one horizontal line and one or multiple vertical lines. This strategy is in contrast to many other routing algorithms which only try to minimize the number of tracks. For an example, see Figures 1, 5, 6.

An orthogonal hypergraph embedding can reduce the number of crossings compared to the straight line embedding of its corresponding graph as shown in Figure 3.

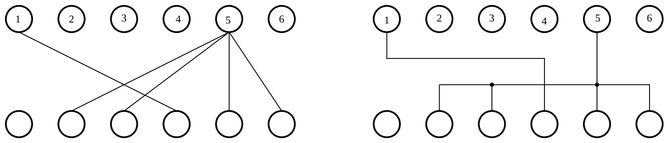

Fig. 3. An example graph and the corresponding orthogonal hypergraph embedding.

If π is the actual permutation of the nets in the channel and the permutation of all nodes is fixed, one can compute the number of crossing C as

$$C = \sum_{\pi(i)<\pi(j)} c_{ij}.$$

We now define the layered hypergraph crossing minimization problem:
Given a hypergraph $H = (L_0, \ldots, L_n, h)$ with n layers $L_0, \ldots, L_n$, a set of hyperedges h an ordering π_i for every layer L_i where $1 \leq i \leq n$. All hyperedges h are drawn in an orthogonal way and consist of at most one horizontal line which is located between the two layers. It is connected to all incident nodes with a vertical line. The layered hypergraph crossing minimization problem is now to find an ordering π_i for every layer L_i and an assignment of all hyperedges to different tracks which minimizes the number of hyperedge crossings.

3 Sifting and Reordering Heuristic Method

The computational complexity [8] of the layered hyperedge crossing reduction problem requires the use of fast heuristic methods, especially in the field of circuit visualization where a short response time is crucial. The next sections briefly describe a sifting and a reordering heuristic method.

3.1 Sifting

The sifting [15] algorithm was originally applied for minimizing the number of nodes in *Binary Decision Diagrams (BDDs)* [2] (a data structure frequently used in logic synthesis applications and formal verification of logic circuits). It chooses one net and moves it to the lowest track by repeatedly swapping it with its neighbor. Next, the chosen net is continuously swapped from the lowest track up to the topmost track. Finally, it is moved to its locally optimal position. The algorithm then proceeds with the next net until every net has been touched. It is easy to see that swapping a net with its neighbor is a local operation and that the number of crossings after the operation is

$$Crossings = Crossings_{before} - c_{ij} + c_{ji}.$$

Usually the algorithm is repeated until no further improvement of the solution can be achieved. Since the sifting heuristic method improves a given solution, all hyperedges are first assigned to one track starting at the top most track. First the algorithm computes the number of crossings for all hyperedges which are induced if the hyperedge is positioned above the remaining hyperedges. Then it chooses the hyperedge which causes the smallest number of crossings, fixing it to the top most track. The algorithm repeats this greedy selection step until all hyperedges are assigned to one track. Pseudo code for the first embedding is given below:

```
greedy_assign(crossing_matrix cm, hyperedges nets){
    for k = 1 to |nets| − 1 do {
        mincost = +∞
        for every net i ∈ nets do {
            cost = ∑_{j∈nets} c_ij
            if(cost < mincost) {
                best = i
                mincost = cost
            }
            nets = nets \ best
        }
        assign best to track k
    }
}
```

The algorithm can be improved in many ways. The sifting of one net takes $O(n)$ time, and sifting all nets summarizes to $O(n^2)$ time. For further details we refer to [7].

The algorithm can be improved by an intelligent strategy to select the next node to be sifted. In the field of circuit visualization it seems to make sense to sift nets which are connected to many nodes before the nets which are only connected to a few nodes. It is also possible to use a randomly selected net. To obtain high quality results the algorithm repeats the sifting procedure for all nodes until it has reached a local optimum.

There are two well known methods to speed up the implementation:

- Moving the net first to the top most or the deepest track, which ever one requires the fewest swappings. This idea was first introduced by [15].
- Using upper and lower bounds to prune the search space, as introduced in [4].

3.2 Reordering

Now we describe how the algorithms from the placement and routing phase are combined to further reduce the number of hyperedge crossings with respect to the hypergraph structure. Since finding exact solutions for both steps together implies to traverse a huge search space, two fast heuristic methods are used by turns to compute the final solution.

The new approach extends the well known greedy switch heuristic method for graphs to the hypergraph structure. It reorders iteratively two nodes in one layer and computes the orthogonal embedding of the hyperedges until a local optimum is reached. This new approach combines the placement and routing process with respect to the hypergraph structure in contrast to many known heuristic methods. The algorithm then repeats the reordering procedure until a local optimum is reached. The experimental results shows that the runtime for all instances which easily fit on one screen are short which is crucial for an interactive circuit visualization tool.

Pseudo code for the reordering procedure is given below:

```
reordering(hypergraph hg) {
    for each layer {
        for each node in actual layer {
            before = compute crossings using sifting
            swap actual node with its right neighbor
            after = compute crossings using sifting
            if (before ≤ after) {
                swap actual node with its right neighbor
            }
        }
    }
}
```

Figure 4 shows the result after the crossing minimization step and the orthogonal embedding of the example circuit in Figure 2.

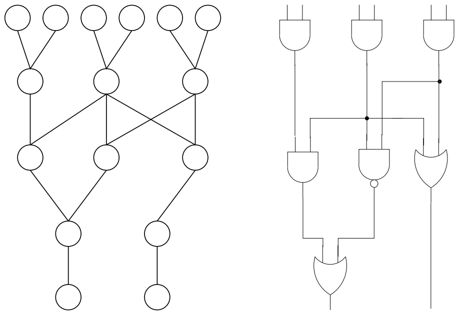

Fig. 4. The graph after the crossing minimization step and the orthogonal embedding of the circuit.

4 Experimental results

The algorithm has been implemented in C. All experimental results are based on examples which were taken from the sequential benchmark circuits in [1, 14]. The experiments were carried out on a 2 GHz personal computer with 1 GB main memory running linux OS. All run times are given in CPU seconds.
We utilized the barycenter heuristic method combined with the greedy switch method to obtain a fast embedding of the nodes. Like observed in [11, 9] the barycenter method combined with the greedy switch heuristic computes high quality results in a short period of time compared to many other well known methods. The algorithm then applies the sifting procedure to assign each hyperedge to one track in the corresponding channel. This is done to get the orthogonal layout of each channel with respect to the number of crossings. Then the reorder algorithm improves the result by switching nodes within the same level and recomputing a orthogonal embedding, if the number of hyperedge crossings does not increase.

In Table 1 the results are summarized. In the second column the number of gates of each benchmark circuit is shown. In the third column we present the number of straight line crossings for each graph using the barycenter implementation combined with the greedy switch methods. In the fourth column the number of hyperedge crossings using sifting are given. The final result in terms of hyperedge crossings after applying the sifting and reordering procedure can be found in the fifth column. Compared to an assignment of the hyperedges over the tracks without using the reordering heuristic method the new method reduced the number of crossings by 25 percent on average. The run times of the algorithm on each circuit are given in the last column. This demonstrates that in the VLSI CAD scenario, using the sifting heuristic together with the reordering method to compute the final orthogonal embedding is very effective.

Table 1. Benchmark results

circuit	gates	strl	sifting	siftplace	time
add6	96	170	136	105	0.44
alu1	31	116	89	55	0.02
alu2	71	415	352	249	0.39
alu3	88	572	465	345	0.62
adr4	64	147	102	75	0.06
co14	60	74	65	54	0.09
dk17	63	367	274	203	0.2
dk27	27	77	64	38	0.03
dk48	61	427	378	305	0.39
mish	73	60	60	49	2.7
radd	52	59	48	38	0.03
rd53	56	253	172	141	0.04
s208	96	291	284	175	1.05
s298	119	904	550	434	2.16
s382	158	745	468	367	2.21
s386	159	1793	1254	853	1.89
s400	162	838	562	427	2.45
vg2	73	184	154	126	0.55
x1dn	75	195	175	136	0.76
x9dn	79	228	210	156	0.76
z4	50	117	91	60	0.07
Z9sym	182	3901	2320	1788	16
$\sum$		11933	8273	6179	32.91

In Table 2 it is shown that fewer edge crossings in the graph embedding lead in almost all cases to less crossings in the orthogonal drawing if only the sifting algorithm is applied. To obtain less crossings in the graph model we post-processed the results computed with the averaging heuristic method with the windows optimization procedure [9]. The results are provided in the second and the third column, respectively. The windows optimization method uses a local optimization technique where subsets of nodes and edges are processed exactly. In contrast to most other heuristic methods more than two layers are considered simultaneously. It turned out that using the windows optimization technique reduces the number of hyperedge crossings in the final orthogonal representation on average by eight percent. However, using the sifting and reordering heuristic method together with windows optimization instead of only sifting and reordering does not lead to significant improvements in terms of hyperedge crossings. This is because the final result dose not depend as much on the given ordering of the nodes since the reordering heuristic improves the ordering of the nodes during

Table 2. Benchmark results

circuit	strl	winopt	sifting	wopt+splace	time/s splace	time/s winopt
add6	170	156	144	112	0.32	15
alu1	116	108	88	60	0.02	3
alu2	415	391	323	243	0.6	26
alu3	572	487	391	331	0.43	22
adr4	147	141	101	74	0.1	5
co14	74	69	65	52	0.13	6
dk17	367	308	249	188	0.22	17
dk27	77	71	57	45	0.02	2
dk48	427	397	341	290	0.29	32
mish	60	60	60	49	2.7	5
radd	59	52	43	37	0.04	2
rd53	253	218	152	126	0.09	14
s208	291	269	266	162	1.82	36
s298	904	854	515	428	2.38	27
s382	745	651	425	357	2.31	65
s386	1793	1636	1122	904	2.27	102
s400	838	725	512	400	3.63	59
vg2	184	168	151	131	0.58	9
x1dn	195	173	153	134	0.32	21
x9dn	228	206	185	158	0.27	12
z4	117	101	87	66	0.05	9
Z9sym	3901	3528	2160	1802	10.8	188
$\sum$	11933	10769	7590	6149	29.3	677

the crossing minimization. In the fourth and in the fifth column the number of hyperedge crossings after applying the sifting and the sifting combined with the reordering algorithm can be found. The runtime for the sifting combined with the reordering algorithm are published in the sixth column. The runtime for the windows optimization technique is given in the last column. In the last rows of Table 1 and Table 2 we give the total number of crossings for each column.

5 Conclusions

We have presented a new method to improve the "readability" of circuit schematics by reducing the number of hyperedge crossings. In contrast to other known algorithms our approach draws every net in each channel with a minimal number of bends which allows the viewer to easily extract the functionality of the circuit. The placement and the routing phase are merged by using to fast heuristic methods iteratively. The experiments have shown that the algorithm reduces the number of hyperedge crossings significantly compared to a layout method which does not reorder the nodes dynamically within a layer. Negligible runtime of our algorithm allows the visualization of large circuits in an interactive way.

6 Acknowledgment

We are grateful to Andre Markert, University of Freiburg, for providing us his implementation of the circuit visualization tool (see Figures 5, 6). He implemented it as part of his diploma thesis.

References

1. F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.
2. R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *TOC*, 35(8):677–691, 1986.
3. R. Drechsler, W. Günther, L. Linhard, and G. Angst. Level assignment for displaying combinational logic. *EUROMICRO*, pages 148–151, 2001.
4. R. Drechsler, W. Günther, and F. Somenzi. Using lower bounds during dynamic BDD minimization. *IEEE Trans. on CAD*, 20(1):51–57, 2001.
5. P. Eades and D. Kelly. Heuristics for reducing crossings in 2-layered networks. *Ars Combin.*, 21(A):89–98, 1986.
6. P. Eades and N. Wormald. Edge crossings in drawings of bipartite graphs. *Algorithmica*, 11:379–403, 1994.
7. T. Eschbach, W. Günther, and B. Becker. Crossing reduction for orthogonal circuit visualization. *Proceedings of the 2003 International Conference on VLSI, Las Vegas, USA, CSREA Press*, pages 107–113, 2003.
8. T. Eschbach, W. Günther, and B. Becker. Orthogonal hypergraph routing for improves visibility. *Great Lakes Symposium on VLSI (GLSVLSI 2004), Boston, USA*, pages 385–388, 2004.
9. T. Eschbach, W. Günther, R. Drechsler, and B. Becker. Crossing reduction by windows optimization. *Proceedings of the 10th International Symposium on Graph Drawing*, LNCS 2528:285–294, 2002.
10. M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic and Discrete Methods*, 4:312–316, 1983.
11. M. Jünger and P. Mutzel. 2-layer straightline crosing minimization: Performance of exact and heuristic algorithms. *J. Graph Agorithms Appl.*, 1(1):1–25, 1997.
12. T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Teubner, Wiley, 1990.
13. E. Mäkinen. How to draw a hypergraph. *International Journal of Computer and Mathematics*, 34:177–185, 1990.
14. K. McElvain. Benchmark set: Version 4.0. *International Workshop on Logic Synthesis*, 1993.
15. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *ICCAD*, pages 42–47, 1993.
16. G. Sander. Layout of directed hypergraphs with orthogonal hyperedges. *11th Int'l Symposium on Graph Drawing 2003*, LNCS 2912:381–386, 2004.
17. N. Sherwani. *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers, Norwell, Massachusetts, second edition, 1995.
18. K. Sugiyama, S.Tagawa, and M.Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transaction on Systems, Man and Cybernetics*, 11(2):109–125, 1981.

7 Appendix

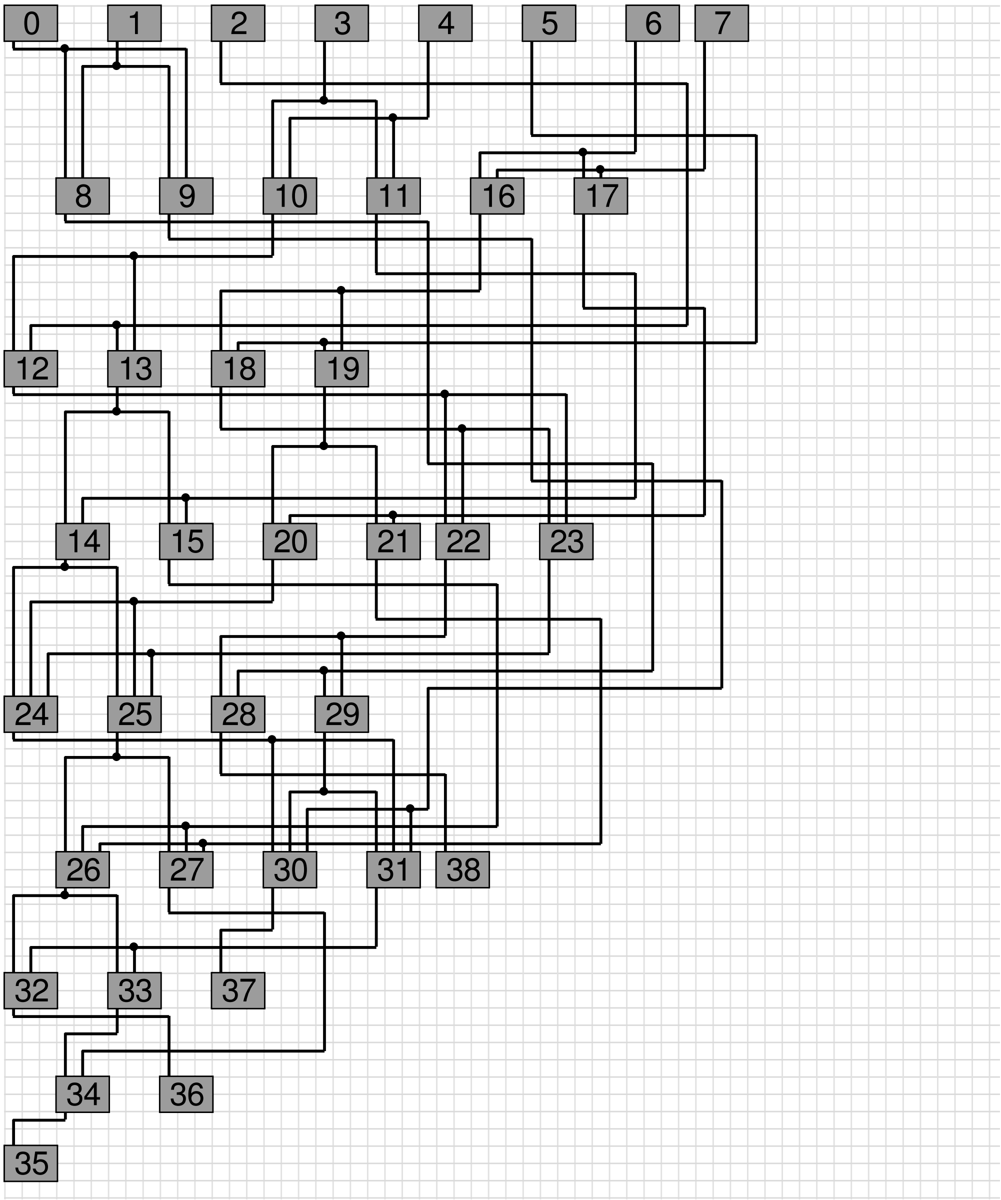

Fig. 5. The rd84 benchmark circuit without crossing minimization (118 wire crossings).

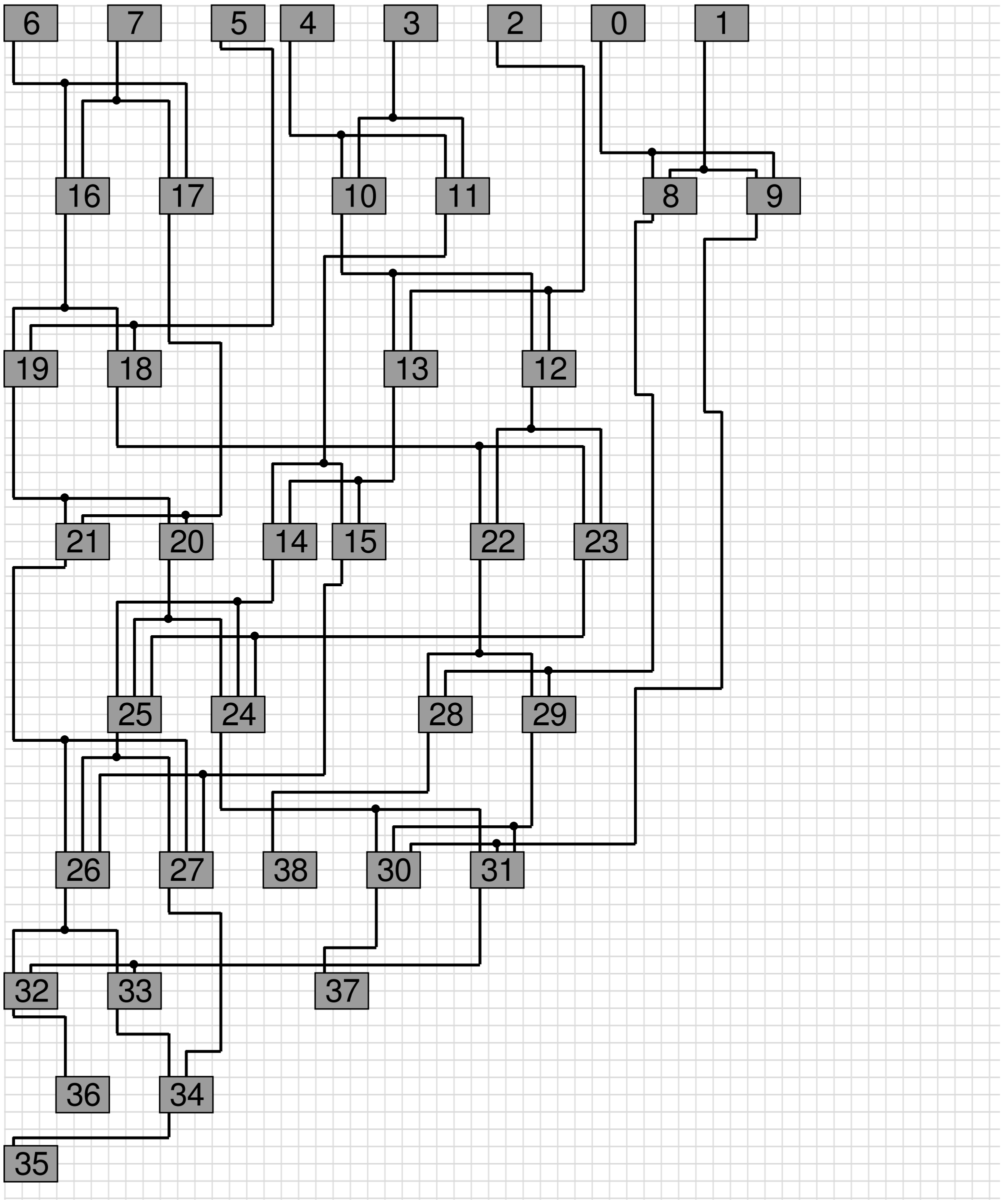

Fig. 6. The rd84 benchmark circuit after the crossing minimization (30 wire crossings).