Cut-based technology mapper with optimizations

Daniil Garyaev
ISP RAS
Plekhanov REU
Moscow, Russia
dgaryaev@ispras.ru

Egor Belin
ISP RAS
MIEM HSE
Moscow, Russia
esbelin@miem.hse.ru

Grigory Mazov
ISP RAS
MIEM HSE
Moscow, Russia
gamazov@edu.hse.ru

Vladislav Shtrenev
ISP RAS
MIEM HSE
Moscow, Russia
vsshtrenev@edu.hse.ru

Mikhail Chupilko
ISP RAS
Plekhanov RUE
Moscow, Russia
chupilko@ispras.ru

Abstract—This paper aims at the problem of construction of optimization-oriented technology mapper for logic synthesis. We describe our implementation of cut-based technology mapper used in our prototype of logic synthesis tool. Our implementation supports several optimization strategies: by target design area, power, and timing. A comparison with a technology mapper used in OpenLane is provided. The experiments have been conducted on BENCH descriptions from OpenABC-D dataset, showing the selected optimization strategy to allow us in majority cases to get the desired characteristics of the selected target parameter.

Keywords—technology mapping, integrated circuits, standard cells, optimization

I. INTRODUCTION

The process of integrated circuits (ICs) production starts from specifications development, which states characteristics and functionality the future IC should satisfy. Then, a design written in the Verilog or VHDL language is developed. In this paper we assume Verilog to be selected. This design should be checked against its specification. To conduct such a check, one should use functional verification step. When this step is done, the process of logic synthesis is started.

Logic synthesis allows to obtain a technology-mapped representation with the specified functionality. The first step of logic synthesis is a parsing of Verilog description and construction of internal representation in form of graph. One of the typically used graphs is so-called AIG. An AIG (And-Inverter Graph) is a compact graphical representation of Boolean functions, consisting of two elements: AND gates and inverters. The nodes in the graph represent AND gates, which connect through direct edges or through inverted edges that signify logical negation. This structure is used for efficient circuit optimization and verification. Then this representation is undergone logic optimization that can improve the effectiveness of a logic circuit based on several criteria.

After logic optimization, the step of technology mapping comes, that is the point of the current paper. It is assumed here that the target of the mapping is IC based on standard cells (not FPGA). Also, in this research we aim at functional mapping, when the input graph representing the desired functionality is transformed into new one by means of matching truth tables. Our research show that there is a lack of optimization-oriented technology mappers and we propose our own method to construct them and developed of its implementation.

The paper is organized as follows. The second chapter touches upon the related works. The third chapter presents some details of our implementation. The next chapter shows the results of our experiments and comparison against OpenLane's standard technology mapper. The fifth chapter concludes the paper.

II. RELATED WORK

Technology mapping in digital circuit design has been profoundly influenced by a multitude of scholarly contributions, among which the works of Alan Mishchenko [1] are of importance. The essence of his algorithm consists in the preliminary selection of the best cut (A cut is defined as a subset of nodes that isolates a specific portion of the circuit, effectively capturing a sub-function within the circuit for targeted optimization) for replacement without using the cells physical parameters from the technology library.

Initially, the algorithm computes K-feasible cuts for each vertex, with K-feasible referring to a partition of the graph that contains no more than K leaves. After establishing these cuts, the algorithm processes the vertices in a topological order, starting from the primary inputs (PIs) and moving toward the primary outputs (POs). In this step, it assesses the depth of all cuts for each vertex and selects the cut that minimizes delay, which is crucial for the speed optimization of the circuit.

Subsequent to the timing optimization, the algorithm undertakes area recovery to minimize the overall logic gate count, thereby optimizing the circuit's area. This is achieved through two strategies: using area flow and using exact local area. Area flow optimizes the logic sharing between functions, while exact local area focuses on minimizing the number of gates for individual cuts.

Once the area recovery is complete, the algorithm selects the best cover for the AIG in reverse topological order, from the primary outputs back to the primary inputs. This ensures that the optimization process does not negatively impact the parts of the graph that have already been optimized.

The final output is a mapped netlist, which provides a detailed blueprint of the optimized circuit. This includes specific gates and their connections, ready for further simulation or fabrication.

This method is implemented in open-source ABC logic optimization tool [2] that works in pair with also open-source Yosys logic synthesis tool [3], that is, in order, is the only logic synthesizer that is used by open IC design flow OpenLane [4].

There are of course some other open-source tools. E.g., there is an open-source Mockturtle tool [5]. It is a C++ library for logic synthesis and optimization of digital circuits. It supports various network representations like AIGs and

MIGs, and offers a range of optimization techniques. The task of technology mapping in the library is solved similarly to Mishchenko's algorithm but with some modifications. For instance, in Mockturtle, multi-output cuts are sought [6], and algorithms for Boolean and structural matching are combined [7].

There are some limitations of all the existing algorithms that we would like to cover in this paper. E.g., it is impossible to run a delay- or area- or power-optimization, as these tools allow to run only some optimization from the restricted list (say, "area flow") and the influence on the target characteristics is now guaranteed. The aim of our on-going research is develop a technology mapping facilities that fix this gap by implementing an optimization-with-guarantee mapper.

III. Description of the Approach

Our implementation is developed on the base of Utopia EDA [8] project, using its internal representation of logic graph. Utopia EDA has facilities to construct the internal representation out of Verilog and GraphML files (that be constructed out of BENCH file). In the description of the approach we suppose the internal representation for the selected input RTL model to be already constructed and premapped to AIG. So, the proposed cut-based technology mapper consists of C++ main classes as follows.

a) *Class TechMapper* is the main class that manages the technology mapping process. This class coordinates actions between different subsystem components. Some details of the class are as follows.

- *Enum Strategy* includes all the names of different technology mapping strategies.

- *The TechMapper class constructor* receives the path to the Liberty library [9] (a standard format in the semiconductor industry to describe the timing and power characteristics of electronic components, such as standard cells and complex cells in integrated circuits) to call Utopia EDA's Liberty parser, the technology mapping strategy (Strategy), the SDC class consisting of physical constraints for the circuit mapping.

- *Method techmap* executes the mapping process using provided input vertexes and links between them in the input representation format.

b) *Class CombMapper* defines the basic interface for all technology mappers, providing template methods for their implementation. Some details of the class are as follows.

- *Method mapping* is responsible for itself mapping of internal representation vertex to Liberty cells.

- *Struct BestReplacement* is used by *mapping()* to store the selected "the most effective cells" for each input representation vertex within the initial circuit.

c) *Class Assembly* is responsible for assembling the final result from individually mapped components.

- *Method assemble()* combines mapped components into a single output format or structure.

The interaction between the components above is done in the following way.

1) *Initialization*: *TechMapper* initializes the process by loading configurations and input data.

2) *Mapping*: Using an implementation of *TechMapper* following the interface provided by *BaseMapper*, input data components are iterated over to be found the most suitable mapping (using an instance of *BestReplacement*).

3) *Assembly*: After each input component have been mapped, Assembly compiles the mapped model in the Utopia EDA's internal representation. This mapped model can be printed by Utopia EDA's Verilog printer.

4) *Finalization*: TechMapper concludes the process, saving results and performing cleanup.

The following three subchapter reveals our approaches to development of BaseMapper's inheritances aimed to power, area, and delay optimizations.

1. Power-targeted optimization

It is commonly known that the most power dissipation in ICs is due to dynamic power and can be characterized by the equation [10]:

$$P_{dynamic} = \frac{1}{2} f \cdot V^2 \sum_{i \in signals} C_i \cdot s_i$$

where f is the clock frequency, V the supply voltage, C_i the capacitance switched by signal i, and s_i is the probability of signal i making 0 to 1 or 1 to 0 transitions. The general idea of our mapping strategy is to reduce total switching activity (i.e. the number of switches) in the mapped network.

The main metric that is used in our approach to optimize power consumption is *Switching flow* (that correlates with Mischenko's paper [10]) and it is defined as follows:

$$SwitchFlow(n) = Switch(n) + \sum_i \frac{SwitchFlow\left(Leaf_i(n)\right)}{NumFanout\left(Leaf_i(n)\right)}$$

where $Leaf_i(n)$ is the i-th leaf of the representative cut of a vertex n and $NumFanouts(Leaf_i(n))$ is the number of fanouts of a vertex $Leaf_i(n)$ in the currently selected mapping. $Switch(n)$ is the total switching activity at the output of vertex n, computed with simulation.

The power optimization strategy is described in class *PowerMap* which is inherited from *CutBaseMapper* (that is inherited in its order from *BaseMapper*). *PowerMap* implements the *findBest* method. The findBest method makes one pass in topological order over internal representation graph and incrementally computes two metrics *SwitchFlow* and *AreaFlow* (described in section 2) for each cut of the considered vertex. When comparing two cuts, their *switch flows* are compared first, and *area flow* is used as a tie-breaker. For the best cut the technology cell is chosen from the *sky130* [4] library. The mapper chooses the technology cell among those having the required by the cut truth table, and having the minimal *CellPower* which is computed as follows:

$$CellPower(cell) = \sum_i rPower\left(pin_i(cell)\right) \cdot rSwitch\left(pin_i(cell)\right) + fPower\left(pin_i(cell)\right) \cdot fSwitch\left(pin_i(cell)\right)$$

where $rPower\left(pin_i(cell)\right)$ is power that dissipates when happens 0 to 1 transition on i-th pin of the *cell*, $fPower\left(pin_i(cell)\right)$ is the opposite. These values are derived from technology library. $rSwitch\left(pin_i(cell)\right)$ and

$fSwitch(pin_i(cell))$ denotes the number of 0 to 1 and 1 to 0 transitions respectively. This way the *findBest* method finds *techCells* for each internal representation graph vertex and stores them in an instance of *BestReplacement*.

2. Area-targeted optimization

Considering each inner vertex v, we choose the best one among all cones with a root in v that we can match to an element of the technology library. The best cone is the cone with the minimum $AreaFlow$ among the other cones of the selected vertex (that correlates with papers [10] and [11). The cone corresponds to a vertex in the mapped net and the $AreaFlow$ is calculated using the formula:

$$AreaFlow(v) = A_v + \sum_{i \in \text{iedges(v)}} \frac{AreaFlow(head(i))}{|oedge(head(i))|},$$

where A_v is equal to the area of the technical library cell that we have mapped to the cone in question, A_v = 0 for PIs/POs. $\frac{iedges(v)}{oedges(v)}$ are the sets of incoming/outgoing edges for vertex v, $head(e)$ is the vertex from which edge e exits.

This strategy is implemented similar to *PowerMap* class. For each vertex it selects the matching technology cell which has minimal area in sky130 library and leads to reducing the total area of mapped net.

3. Time-targeted optimization

The main idea to make time-targeted strategy is to rely on so-called "Wire-Load Models" (WLM) [12], which are based on some heuristic models developed on the basis of previously carried out physical synthesis runs for various other circuits. Note that the most accurate information about the circuit delay appears after the physical synthesis stage.

Currently, one can find several WLMs in the public domain, which are presented in the Liberty format. Note that after 2001-2002, the main commercial tools began to use WLMs, described in their own internal formats, taking into account the results of physical synthesis [13]. In this paper we use WLM provided with sky130 technology library. This model is shown in listing 1.

Listing 1. Sky130 WLM

```
wire_load("sky130") {
      capacitance : 1.42e-05;
      resistance : 0.0745;
      slope : 8.3631;
      fanout_length( 1, 23.2746);
      fanout_length( 2, 32.1136);
      fanout_length( 3, 48.4862);
      fanout_length( 4, 64.0974);
      fanout_length( 5, 86.2649);
      fanout_length( 6, 84.2649);
    }
```

The parameters here are capacitance and resistance, taken from DEF files describing the characteristics of metal used in the given technology.

To optimize the time values of the synthesized circuit, the Wire-Load Model and Non-Linear Delay Model are used. The first one provides the physical characteristics of the circuit lines, i.e. resistance and capacitance, depending on the number of fanouts for calculating the wire delay and in the cell itself. Non-Linear Delay Model is used to calculate the delay values exactly inside the cell, based on data from LookUp Tables stored in Liberty format, and using interpolation from the values obtained from these tables.

These two models have been implemented as a time estimation module. In turn, this module is used in the *delay* strategy in terms of *CutBasedMapper*. It selects a cut and a matching technology cell based on the delay derived from the time-estimation module.

IV. Experiments results

The idea of our experiments is to check what is the difference between results of our techmappers (optimization-oriented implementations of a class CutBaseMapper) and the basic techmapper for OpenLane (Yosys working under OpenLane's AREA0 strategy; the name of strategy here denotes a long set of logic optimizations called in ABC and Yosys to be performed under the target design).

We made several experiments, using nineteen BENCH (in GraphML format) descriptions from OpenABC-D [14] dataset as input circuits for technology mapping. We derived the estimations for the worst arrival time (referred to as 'delay'), area, and power based solely on the results of placement and routing, without conducting the other steps of physical synthesis.

Given that this research is ongoing, the current findings are primarily focused on power optimization. Table 1 shows that in most cases our "power" strategy allows to get the resulted design better that those received from Yosys in terms of its power consumption. The area optimization should have the similar story because it is based on the similar idea (AreaFlow algorithm with selection of the best cell from the point of view of its physical characteristics). The timing strategy results is a priory dependent on the adequacy of the WLM model used and here more experiments are needed.

Conclusion

We have developed an implementation of the cut-based approach to technology mapping based on Utopia-EDA project. In the experiments conducted using some examples from OpenABC-D, we generated netlists composed of technology cells by means of our implementation, by Yosys+ABC, estimate their characteristics by means of OpenLane and compare them. The characteristics of the netlists estimated by OpenLane are similar to those in the netlists generated by Yosys+ABC, regardless of whether logic optimization is enabled or disabled in Yosys+ABC. The future work in our research is to use logic-level optimizations in Utopia EDA, improve the implementation's performance and improve our results in data arrival time.

References

[1] A. Mishchenko, Sungmin Cho, Satrajit Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in Proc. ICCAD, 2007.

[2] ABC tool – URL: https://people.eecs.berkeley.edu/~alanmi/abc/

[3] YOSYS tool – URL: https://github.com/YosysHQ/yosys

[4] OpenLane tool – URL: https://github.com/The-OpenROAD-Project/OpenLane

[5] Mockturtle tool – URL: https://github.com/lsils/mockturtle

[6] A. T. Calvino and G. De Micheli, "Technology mapping using multioutput library cells," Proc. ICCAD, 2023.

[7] G. Radi, A. Tempia Calvino, and G. De Micheli, "In Medio Stat Virtus: Combining Boolean and Pattern Matching," ASP-DAC, 2024.

[8] *Utopia EDA. - URL: https://gitlab.ispras.ru/mvg/utopia-eda*

[9] Liberty standard "User Guides and Reference Manual Suite Version 2017.06"

[10] A. Mishchenko, R. Brayton, S. Jang, K. Chung. A power optimization toolbox for logic synthesis and mapping. Proc. IWLS'09, pp. 1-8.

[11] V. Manohara-rajah, S. D. Brown, and Z. G. Vranesic, "Heuristics for area minimization in LUT-based FPGA technology mapping," Proc. IWLS'04, pp. 14-21.

[12] Golson S. Resistance is Futile! Building Better Wireload Models // Proceedings of SNUG'99, pp. 1-18, 1999.

[13] Digital VLSI Chip Design with Cadence and Synopsys CAD Tools. Example Liberty File. – URL: https://users.cs.utah.edu/~elb/cadbook/Chapters/AppendixC/example.lib.txt.

[14] OpenABC-D test set – URL: https://github.com/NYU-MLDA/OpenABC/tree/master/bench_openabcd_data_set: designs in BENCH format

TABLE I. COMPARISON OF POWER-AREA-DELAY IN EXPERIMENTS

Design name	Vertex / edges number	Input/Output number	Estimation of power consumption, target area and max data arrival time for different strategies			
			Power-opts	***Area-opts***	***Delay-opts***	***Yosys+ABC***
ac97_ctrl	*22 060 / 33 524*	*4 476*	*19 300 uW*	***19 000 uW***	*32 900 uW*	*28 100 uW*
			328 032 um²	***326 961 um²***	*419 108 um²*	*331 315 um²*
			26.27 ns	*9.72 ns*	*8.59 ns*	***7.09 ns***
aes	*39 215 / 68 140*	*1 212*	***12 400 uW***	*16 900 uW*	*19 900 uW*	*14 900 uW*
			120 810 um²	***117 311 um²***	*251 973 um²*	*136 239 um²*
			198.20 ns	*70.76 ns*	*161.53 ns*	***6.94 ns***
des3_area_orig	*7 766 / 12 737*	*367*	***636 uW***	*1 460 uW*	*1 870 uW*	*3 980 uW*
			19 774 um²	***18 742 um²***	*41 910 um²*	*23 759 um²*
			3.33 ns	*9.84 ns*	*46.07 ns*	*6.68 ns*
dynamic_node	*33 761 / 51 855*	*5 283*	*55 500 uW*	*58 800 uW*	*143 000 uW*	***40 600 uW***
			408 776 um²	***406 698 um²***	*550 712 um²*	*417 902 um²*
			98.01 ns	*51.86 ns*	*433.43 ns*	***10.88 ns***
fir	*9 002 / 13 560*	*761*	*1 640 uW*	***810 uW***	*3 630 uW*	*8 710 uW*
			28 263 um²	***26 853 um²***	*56 982 um²*	*33 143 um²*
			55.56 ns	*55.15 ns*	***2.77 ns***	*7.51 ns*
fpu	*55 935 / 85 558*	*1 041*	*19 900 uW*	***12 700 uW***	*n/a*	*n/a*
			138 358 um²	***135 185 um²***	*n/a*	*n/a*
			286.47 ns	***256.75 ns***	*n/a*	*n/a*
i2c	*2 018 / 3 187*	*305*	*384 uW*	***374 uW***	*622 uW*	*434 uW*
			5478 um²	***5 251 um²***	*10 774 um²*	*5 577 um²*
			12.81 ns	*12.16 ns*	*16.33 ns*	***4.58 ns***
iir	*13 645 / 20 623*	*935*	*2260 uW*	***1 300 uW***	*6 040 uW*	*29 700 uW*
			44 013 um²	***42 555 um²***	*87 393 um²*	*48 583 um²*
			23.95 ns	*31.99 ns*	*58.50 ns*	***9.81 ns***
mem_ctrl	*29 814 / 47 906*	*2 149*	*18 200 uW*	*14 700 uW*	*25 000 uW*	***7 770 uW***
			135 944 um²	*133 123 um²*	*229 775 um²*	***108 764 um²***
			273.24 ns	*225.64 ns*	*246.06 ns*	***9.52 ns***
pci	*38 279 / 57 826*	*6 586*	*38 100 uW*	***37 100 uW***	*69 600 uW*	*59 200 uW*
			619 649 um²	***616 840 um²***	*773 795 um²*	*620 715 um²*
			165.68 ns	*216.26 ns*	*230.85 ns*	***12.24 ns***
sasc	*1 214 / 1 827*	*260*	*197 uW*	***191 uW***	*419 uW*	*406 uW*
			3 269 um²	***3 221 um²***	*8 523 um²*	*3 700 um²*
			3.80 ns	*4.36 ns*	*10.02 ns*	***3.32 ns***
sha256	*28 691 / 44 507*	*2 985*	*64 800 uW*	*n/a*	*92 700 uW*	***36 300 uW***
			176 200 um²	*n/a*	*235 301 um²*	***174 664 um²***
			376.55 ns	*n/a*	*208.07 ns*	***11.29 ns***
simple_spi	*1 764 / 2 694*	*296*	***306 uW***	*330 uW*	*547 uW*	*484 uW*
			4 673 um²	***4 622 um²***	*10 729 um²*	*4 805 um²*
			10.59 ns	*11.64 ns*	*11.93 ns*	***3.29 ns***
spi	*8 311 / 12 530*	*492*	*4 590 uW*	*8 109 uW*	*17 000 uW*	***2 080 uW***
			22 024 um²	*19 979 um²*	*43 450 um²*	*18 398 um²*
			36.88 ns	*28.34 ns*	*67.72 ns*	***6.13 ns***
ss_pcm	*762 / 1 165*	*194*	***124 uW***	*150 uW*	*208 uW*	*281 uW*
			2 186 um²	***2 136 um²***	*4 516 um²*	*2 367 um²*
			2.28 ns	*5.28 ns*	*8.26 ns*	***2.19 ns***
tv80	*19 241 / 30 569*	*997*	*8 740 uW*	*9 250 uW*	*13 700 uW*	***5 160 uW***
			62 451 um²	*59 359 um²*	*114 158 um²*	***55 543 um²***
			157.37 ns	*197.53 ns*	*254.95 ns*	***8.60 ns***
usb_phy	*893 / 1 380*	*222*	***172 uW***	*180 uW*	*260 uW*	*239 uW*
			2 576 um²	***2 530 um²***	*5 235 um²*	*2 729 um²*
			8.46 ns	*8.62 ns*	*9.81 ns*	***2.43 ns***
wb_conmax	*81 107 / 128 947*	*4 197*	*121 000 uW*	*98 500 uW*	*181 000 uW*	***94 600 uW***
			440 237 um²	***434 118 um²***	*705 276 um²*	*461 569 um²*
			214.48 ns	*336.64 ns*	*169.63 ns*	***10.35 ns***
wb_dma	*8 231 / 12 818*	*1 530*	*4 860 uW*	*4980 uW*	*7 410 uW*	***4 290 uW***
			56 335 um²	***55 792 um²***	*88 552 um²*	*55 944 um²*
			35.23 ns	*63.23 ns*	*24.71 ns*	***7.09 ns***