Layout Driven Decomposition with Congestion Consideration

Thomas Kutzschebauch
Leon Stok
IBM TJ Watson Research Center
Yorktown Heights, NY
{kutzsche, stokl}@watson.ibm.com

Abstract

We present a novel algorithm that applies physical layout information during common subexpression extraction to improve wiring congestion and delay, resulting in improved design closure.

As feature sizes decrease and chip sizes increase, the traditional separation of physical design and logic synthesis proves to be increasingly detrimental. Interconnect delay and wiring congestion, among the most critical objective functions to meet design closure, are not considered during logic synthesis. On the other hand, physical design is too deep in the design process to be able to significantly restructure the already technology mapped netlist. While this problem has been addressed previously, the existing solutions only apply simple synthesis transforms during physical design. Hence they are generally unable to reverse decisions made during logic restructuring which have a major negative impact on the circuit structure.

In our novel approach, we propose a layout driven algorithm for the concurrent extraction of common subexpressions, one of the most important steps that affect the overall circuit structure, and consequently congestion and wire length during logic synthesis. In addition, we consider dependency relations between cube divisors to improve the extraction process. As a result, our layout driven decomposition algorithm combines logic synthesis and physical layout information to effectively decrease wire length and improve congestion for improved design closure.

1 Introduction

Due to the increasing complexity of microelectronic designs and the continuous development of finer featured fabrication processes, interconnect delays and overall routability play an increasingly important role in chip design. One of the most important problems in the design of VLSI circuits is the interaction of logical and physical domain. In the continuous quest to achieve design closure, the traditional separation of logic and physical design proves to be disadvantageous as the cost functions employed during logic synthesis become increasingly inaccurate during later design stages.

While this problem has been addressed previously, the existing solutions employ only trivial logic synthesis transforms late in the design process, during physical design [1, 2]**.** Recent improvements use a seamlessly integrated synthesis-placement flow instead of an iterative process. However, transformations are mainly local and have little impact on the global structure of the net list. These transformations are generally unable to reverse adverse decisions made during earlier technology-independent logic synthesis based on inaccurate cost functions, namely literal count and gate-based path delay, not reflecting the physical properties of the design. Decomposition of a Boolean expression refers to extracting subexpressions common to one or more functions such that they can be shared across the network, effectively reducing the size of the synthesized design. As a result, it has a major impact on the overall circuit structure. Traditional common subexpression extraction by Brayton et. al., widely known as kernel extraction [3, 4], and Rajski et. al. [5, 6] only optimizes cell area by minimizing the literal count. Due to the sharing of expressions, this process tends to create gates with high fanout and fanin, which in turn results in increased wiring congestion. Existing layout driven logic synthesis algorithms [7, 8] only exploit the layout information for improved timing information utilizing estimated wire delay and to minimize area employing cell and estimated wiring area. However, the reduction of overall wiring congestion for better routability, a very important objective function besides timing closure, in conjunction with timing and area optimization, has not been actively addressed.

To address this problem, we propose a layout driven decomposition algorithm employing physical layout information early during logic independent restructuring. We first create an initial placement of the technology-independent net list and use the layout information during logic restructuring. By utilizing the geographical location of the objects in the net list, for example the location of a signals source and sinks, and the estimated wire length, we are able to perform congestion aware extraction of common subexpressions. By extracting expressions depending on the location of their sources, we are able to group signals that are located close to each other, which yields an overall reduction of wire length. In addition, we decrease routing congestion by distributing the wiring instead of concentrating it in small local areas, as happens during regular decomposition. Furthermore, we consider dependency relations between candidate cube divisors to efficiently select the kernel with the best layout cost.

In conclusion, the main contribution of this paper is a layout driven decomposition algorithm that uses physical layout information and cube divisor dependencies, to effectively decrease wire length and congestion.

The remainder of this paper is organized as follows. In section 2 we give a brief overview of common subexpression extraction based on the *fast extract* algorithm by Rajski et. al. [5, 6]. Section 3 describes the motivation, problem formulation and implementation of our layout driven decomposition algorithm. In section 4 we present our overall synthesis flow, while section 5 shows experiments and results, followed by conclusions and future work in section 6.

```
begin
   Generate double-cube divisors with
     associated weights;
while (weight >= 0)
   Choose double-cube divisor dcd with
      maximum weight w_dc;
   Choose single-cube divisor scd with
     maximum weight w_sc;
   Select the divisor dcd or scd with
     maximum weight;
   Substitute extracted expression;
   Update weights of conflicting
     double-cube divisors;
end.
```

Figure 1: Fast Extract Algorithm

2 Overview

Decomposition plays an important role in the design process because it has a major impact on the overall structure of the circuit. During the decomposition process, boolean subexpressions common to one or more functions are identified and extracted such that they can be shared across the network. The most common decomposition algorithms of practical importance are the method by Brayton [3, 4], which limits potential divisors to cube-free multiple-cube divisors (kernels), generally known as *kernel extraction*, and the concurrent decomposition method known as the *fast extract* algorithm by Rajski [5,6]. Our layout driven decomposition algorithm is based on the later method, hence we shall give a brief overview of the underlying procedure.

The *fast extract* approach limits itself to identifying and extracting *double-cube divisors*, that is, cube-free multiple-cube divisors having exactly two cubes, and *single-cube divisors* having exactly two literals, considered concurrently with their complements. By limiting objects to size 2, the complexity of candidate subexpression generation (the cube divisors) is polynomial, yet algebraic divisors of arbitrary size can be extracted by way of multiple subsequent extractions.

A simplified version of the *fast extract* algorithm is shown in Figure 1. Double-cube divisors are generated only once, and updated during the extraction process. Their extraction is done separately for each node in the network. However, double-cube divisors can be shared between different nodes, that is, when the divisor has been identified for each node individually. In contrast, single-cube divisors are extracted from cubes belonging to different nodes in the network. At each iteration of the algorithm, the double- or single-cube divisor with the highest weight, that is, the largest literal saving is chosen. Please note that the algorithm is entirely greedy, the cube divisor with the maximum literal saving is chosen and fully extracted, even though it might affect the weight of other double-cube divisors. In fact, some kernels may entirely cease to exist as divisors of the remaining network.

3 Layout Driven Decomposition

To combine logic restructuring and physical design, we create an initial placement of the technology-independent net list and use the placement coordinates of the objects to improve synthesis transformations. We use a quadratic placement solver with quadrisection based on the conjugate gradient method to minimize quadratic net length [11]. Contrary to min-cut based approaches this type of placement algorithm is relatively stable with respect to local changes in the net list. To accurately represent technology-independent cells, we create a physical description based on the actual technology library the net list is going to be implemented in. The placement algorithm uses a density function that allows the circuit to be spread across the layout image accordingly, which can be tuned to achieve an accurate placement representation of the technology-independent circuit that provides a good estimation of the actual location of the objects in the final technology-dependent implementation.

Decomposition, the extraction of common subexpressions, is one of the most important steps affecting overall congestion and wire length, because it has a significant impact on the structure of the netlist by decomposing large two-level structures in the sum-of-products form (SOP) into smaller multi-level representations.

Our decomposition algorithm first transforms the net list into a structure consisting of NANDs with a large fanin. Hence, each node in the network can be represented as a sum-of-products form, which provides the input for the subsequent cube-divisor generation.

Example:

Traditionally, the decomposition algorithm minimizes cell area by selecting the cube divisor with the maximum literal saving. In our layout aware approach, we present a decomposition procedure with the objective of optimizing wire length and wiring congestion. Let us first demonstrate the importance of geographical location during the extraction process on a simple example. Let $f = abc + abd + bcd$ as shown in Figure 2a. The following double-cube divisors can be extracted: $a+d$, $a+c$, and $c+d$. Each of the divisors provides the same literal savings when extracted, however, the selection of the right kernel has a major impact on the circuit structure. As shown in Figure 2a, assume that the sources of signals (a,b) and (c,d) are located

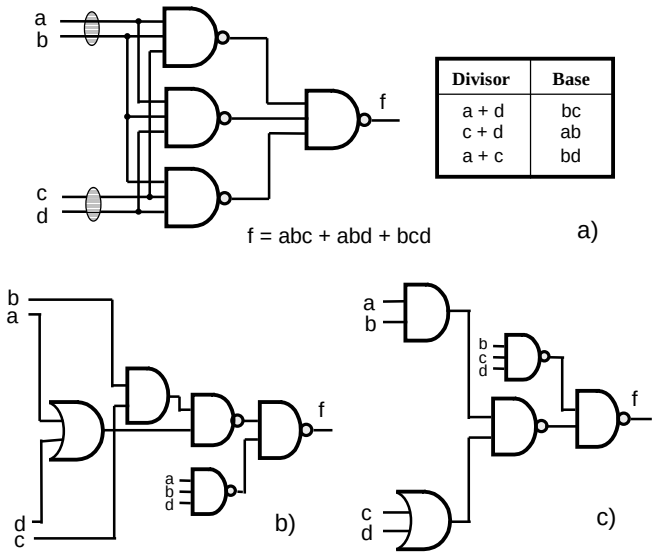

Figure 2: a) Original circuit, b) Generic decomposition
c) Layout driven decomposition

close to each other, respectively. Extracting the kernel $a+d$ (or $a+c$), as shown in Figure 2b, creates a local area of high congestion and increases wire lengths because the grouped signals (a,d) and (b,c) originate from opposite ends of the chip. In contrast, if we choose the kernel $c+d$ as shown in Figure 2c, wire length is decreased and congestion is distributed more evenly because the source signals of the extracted kernel $c+d$ and respective base (or co-kernel) ab are located close to each other, allowing the extracted gate to move towards its sources.

Furthermore, if a common expression can be extracted from different nodes in the network, it might be beneficial not to share the expression under certain conditions, that is, if the nodes are far apart and/or the path under consideration is timing critical. Consider the example shown in Figure 3. Assume that the common expression ab can be shared by nodes n_1, n_2 and n_3 as depicted in Figure 3a. Depending on the location of the nodes, the delay on the path to node n_3 can be improved by not sharing expression ab with nodes n_1 and n_2. Despite the fact that this procedure slightly increases *global* wire length, it improves the delay on the path along n_3 by reducing the maximum length of a wire and the fanout of the node subject to extraction. This can be achieved by considering the delay at each of the nodes that share a common expression, taking into account the wiring delay from the shared expression to the subject node.

Accurate decomposition is particularly important as any decision made at this point is in most cases only reversible at significant cost in later design stages.

Problem Formulation:

Based on the previous observations, we can formulate the problem of extracting common subexpressions from a subject network under consideration of layout information as follows. We define the layout cost of extracting a kernel as the sum of the Manhattan distances between the signal origins of the literals in the divisor and the remaining base, also referred to as co-kernel, respectively. Considering the example in Figure 2, the layout cost of extracting the double-cube divisor $c+d$ is the sum of the L_1 distances between the sources of signals c and d, and the co-kernel, a and b, respectively. Consequently,

we seek to minimize the total cost of grouping while concurrently minimizing literal count. As shown in [6], the weight of a double-cube divisor W_{dc}, that is, the literal saving if extracted, is defined as follows:

$$W_{dc} = d(r-1) - r + \sum_{i=1}^{r} |b_i| + C \qquad (1)$$

In the above equation, d denotes the size of the double cube divisor, r is the number of times the divisor is being shared, i.e. the number of different bases, while b_i is the number of literals in base i, and C is the number of literals that can be saved by using the complement of the double-cube divisor. Similarly, the weight of a two literal single cube divisor W_{sc} is calculated as:

$$W_{sc} = C(i, j) - 2 \qquad (2)$$

$C(i,j)$ denotes the coincidence of a single cube divisor with literals i and j, thus, the number of cubes which contain both i and j, minus 2 to implement the extracted divisor itself.

To choose the best possible divisor considering both layout information and literal saving, we combine the two cost functions. Consequently, we select the divisor with the minimum layout cost within a certain weight, i.e. literal saving range:

$$Min\left(\sum_i (L_1(d_i) + L_1(b_i))\right)$$
$$s.t. \quad Min(W_{dc,i}, W_{sc,i}) > (W_{\max,i} - w) \qquad (3)$$

In the above equation, $L_1(d_i)$ *and* $L_1(b_i)$ denote the Manhattan distances between the signal origins of the divisor d_i and the base b_i in iteration i of the decomposition process. Only single- and double cube divisors with a weight larger than a certain weight are considered, that is, the maximum weight of any divisor minus a given window of size w.

Implementation:

Due to the fact that extracting one divisor can affect the weight and layout cost of other divisors, an optimal solution of the formulated problem becomes infeasible. Therefore, similarly to the original decomposition algorithm, a greedy approach is chosen. Instead of minimizing the total layout cost, the divisor with the smallest cost is selected from a set of candidate divisors having weights larger than a given lower bound, during each iteration separately. The number of candidate divisors is further reduced by the fact that only *dependent* candidate cube divisors affect each other's extraction.

Definition 1*: Two different candidate cube divisors are considered to be* ***dependent*** *on each other if they have at least one cube of some function f in common.*

In the example of Figure 2, all three divisors $a+d$, $c+d$ and $a+c$ are dependent on each other because they each share a common cube. As a result, extracting a candidate divisor in one iteration prevents the extraction of its dependent divisors subsequently. It follows that candidate divisors that share no common cube are *independent* of each other. All dependent divisors and their relations can be represented as a *dependency graph* $G = (D, R)$ with cube divisors $d \in D$ as vertices and their respective dependency relations $r \in R$ as edges.

In Figure 4, the dependency graph for the three cube divisors from Figure 2 is shown in ①. Because all divisors are dependent on each other, the graph is a clique.

The extraction of a divisor does not prevent the subsequent extraction of its independent divisors. This observation is useful for the selection of candidate divisors with minimum layout cost because only divisors that belong to the same dependency graph need to be considered. In other words, a better choice for a

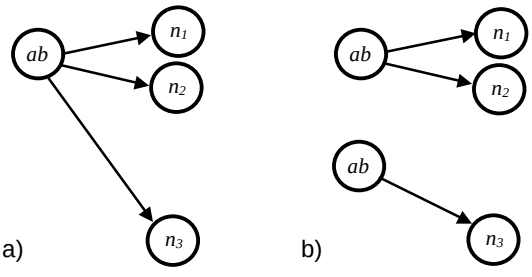

Figure 3: Sharing of an Expression between different Nodes a) Generic, b) with Layout Consideration

divisor with an undesired high layout cost can only be found within the set of divisors of its dependency graph. Figure 4 illustrates the selection process. C_l denotes the layout cost of extracting a cube divisor, and W_{sc} and W_{dc} are the weights of single- and double-cube divisors, respectively. Dependency relations between the candidate divisors are shown as solid lines. In every iteration, the candidate with the maximum weight (literal saving) is chosen. Following, all cube divisors within its dependency graph that fall within the width of window *w*, shown as shaded region, are concurrently considered for extraction, and the divisor with minimum layout cost is selected. After each extraction, affected cube divisors are updated. Generally, we choose a window size of one literal, which is our lower bound for *w*. However, at least one other divisor in the dependency graph, other than the subject divisor, should be considered. Therefore, *w* can be increased up to a fixed upper bound, such that at least one alternative candidate divisor from the dependency graph will be considered.

In addition, when a cube divisor is shared between different functions, we only share it between nodes whose timing is estimated to be non-critical, as previously illustrated by means of the example in Figure 3.

Each newly created gate in the decomposition process has to be assigned a placement location. An exact solution can be determined by solving a quadratic placement problem formulated within the boundary box of the affected objects. However, for efficiency, placing only the extracted expression at the center of gravity of its input and output connections can be chosen.

In conclusion, common expressions are extracted based on their layout cost and associated weight, the estimated literal saving. Consequently, overall wiring length and local congestion is reduced.

4 Layout Driven Synthesis Flow

Subsequently, we illustrate how the concepts outlined in the previous sections are combined in a layout driven design flow. The overall flow is shown in Figure 5. We start with a technology independent, unoptimized net list of the design and run initial logic optimization transformations. Following, we create an initial placement using a quadratic placement algorithm with quadrisection to place the technology-independent net list. The physical properties of the technology-independent cells, i.e. size, shape and pin locations, are created based on the actual technology library that is going to be used for technology mapping. Furthermore, the placement algorithm uses a density function that allows the circuit to be spread across the layout image evenly. After each cell has been assigned a placement location, layout aware extraction of common subexpressions with congestion consideration as described before is performed. Following, we apply our layout aware technology decomposition

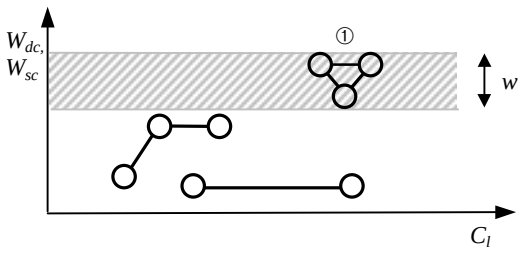

Figure 4: Candidate Cube Divisor Selection

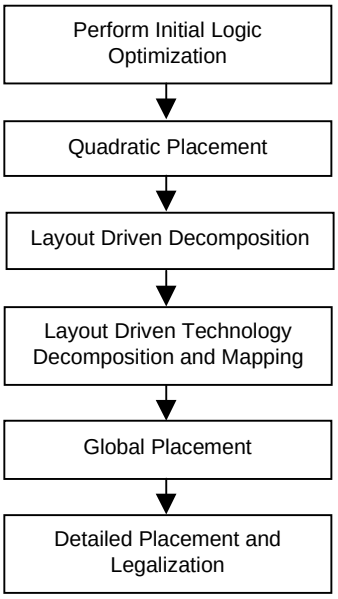

Figure 5: Layout Driven Synthesis Flow

and mapping algorithms as described in [9]. However, for the purpose of the experimental results described in the following section, only the layout driven decomposition described in this paper was used. Placement information is retained by means of assigning a layout location to each newly created cell. We continue global placement on the technology-mapped net list using our quadratic placement algorithm. Finally, we perform detailed placement and legalization.

5 Experiments and Results

The proposed layout driven synthesis flow has been implemented in C++ within the framework of the logic synthesis tool BooleDozer™ [10]. We employ the illustrated layout aware decomposition and extract cube divisors based on literal and layout cost. In addition, we share common expressions between different nodes dependent on their delay. Table 1 presents results of the layout driven optimization process for a number of industrial ASIC designs. *Lit fx* and *Lit lds* show the amount of literals after performing logic minimization using the standard implementation of the *fast extract* and our layout aware extraction algorithm, respectively. *Area fx* and *Area lds* represent the active cell area after all optimization and final placement. On average, optimization using the layout aware approach produces about 7% more literals compared to the generic extraction due to the fact that cubes are extracted in a different order and sharing between different nodes is reduced. However, the cell area after technology mapping and placement is only about 1% larger on average. This is due to the fact that smaller sized gates are chosen and less buffers need to be inserted to meet the timing requirement. *Delay lds* represents the reduction of the delay of the longest path during layout driven optimization due to the fact that wire length has been reduced. It represents the actual delay after physical design, using *RC* extraction to calculate wire delays. In addition, *Avg Cong* and *Max Cong* show the average and maximum horizontal and vertical wiring congestion of the layout driven and generic extraction runs. Due to the reduced

Circuit	*Lit fx*	*Lit lds*	*Area fx*	*Area lds*	*Delay lds*	*Avg Cong fx*	*Avg Cong lds*	*Max Cong fx*	*Max Cong lds*
chip1	142521	154922	373214	375152	-0.15 ns	42.2%	38.5%	94.7%	82.0%
chip2	115432	122517	317661	324424	-0.21 ns	48.5%	44.2%	144.1%	121.4%
chip3	232484	247190	689458	709799	-0.26 ns	62.4%	59.2%	108.3%	91.4%
chip4	303207	321802	792388	799012	-0.04 ns	52.2%	50.7%	151.0%	102.8%
chip5	196108	206421	523652	525010	-0.27 ns	72.2%	68.5%	114.7%	91.7%
chip6	112358	119708	342481	339032	-0.08 ns	51.9%	48.3%	88.5%	73.9%

Table 1: Optimization Results

amount of wire length and by preventing areas of high local congestion, which is achieved by extracting signals of close proximity, we can significantly improve congestion compared to the standard flow. Of particular importance is the maximum congestion because it determines the routability of a chip, which is consistently reduced. By creating a layout aware structure of the network, wiring is decreased and better distributed, and we are able to significantly reduce delay and congestion at virtually no cost in area increase. In conclusion, by improving critical delay and routability, we are able to efficiently reach design closure.

6 Conclusions and Future Work

The presented layout driven concurrent decomposition algorithm successfully integrates the use of layout information during logic restructuring. By extracting common expressions depending on the proximity of the input and output signals, we are able to create a layout aware structure of the net list, resulting in improved path delay and reduced routing congestion. In addition, we consider the dependency relation of different cube divisors to improve the extraction process.

Future work will address a more seamless integration of layout driven synthesis and placement algorithms.

References

[1] W. Donath, P. Kudva, P. Villarrubia, L. Stok et. al., „Transformational Placement and Synthesis“, In *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 194-201, 2000

[2] G. Stenz et. al., „Timing Driven Placement in Interaction with Netlist Transformations“, In *Proc. of the International Symposium on Physical Design, 1997*

[3] R. K. Brayton and C. Mc Mullen, „The Decomposition and Factorization of Boolean Expressions“, In *Proc. of Int. Symp. of Circuits and Systems*, 1982, pp. 49-54

[4] R.K. Brayton and C. McMullen, „Synthesis and Optimization of Multistage Logic“, In *Proceedings of Int. Conference on Computer Design*, 1984, pp. 23-28

[5] J. Rajski and J. Vasudevamurthy, „A Method for Concurrent Decomposition and Factorization of Boolean Expressions“, In *Proc. of the Int. Conference on Computer-Aided Design*, 1990, pp. 510-513

[6] J. Rajski and J. Vasudevamurthy, „The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Expressions“, In *IEEE Trans. on CAD, Vol. 11, No. 6, June 1992*, pp. 778-793

[7] M. Pedram and N. Bhat, „Layout Driven Technology Mapping“, In *Proceedings of the 28th Design Automation Conference*, pp. 99-105, 1991

[8] M. Pedram and N. Bhat, "Layout Driven Logic Restructuring/ Decomposition", In *Proc. of the Int. Conf. on Computer-Aided Design*, pp.134-137, 1991

[9] T. Kutzschebauch and L. Stok, "Congestion Aware Layout Driven Logic Synthesis", In *Proc. of the Int. Conference on Computer Aided Design*, Nov. 2001, pp. 216-223

[10] L. Stok, D. Brand, D. Kung et. al., "Booledozer: Logic synthesis for ASICs", In *IBM Journal of Research and Development*, July 1996, pp. 515-547

[11] J. Vygen, "Algorithms for Large-Scale Flat Place- ment", In *Proceedings of the 34th Design Automation Conference*, 1997, pp. 746-751