REGULAR LAYOUT GENERATION OF LOGICALLY OPTIMIZED DATAPATHS

R.X.T. Nijssen *C.A.J. van Eijk*

Eindhoven University of Technology
Department of Electrical Engineering/ICS EH9.28
P.O. Box 513, 5600MB, Eindhoven, The Netherlands
Email: R.X.T.Nijssen@ele.tue.nl, C.A.J.v.Eijk@ele.tue.nl

ABSTRACT

The inherent distortion of the structural regularity of VLSI datapaths after logic optimization has until now precluded dense regular layouts of optimized datapaths despite their implicit regularity. This paper presents a methodology enabling utilization of datapath regularity for dense layout even after extensive logic optimization. A structural netlist analysis extracts regularity from the initial unoptimized netlist which serves as a partial relative regular preplacement. After each customary iteration of placement, backannotation and logic optimization, functional correspondences between the optimized and the original netlists are identified by a logic correspondence extractor. The functional and structural analyses results are then merged yielding a regular preplacement for the logically optimized design.

1. INTRODUCTION

The logic parts of most VLSI designs are dominated by datapaths which employ bit-wise parallelism for high performance. Two main methodologies are presently used for datapath synthesis, namely datapath compilers (DPCs) and general logic synthesis. In both cases, the netlists containing the datapath instances are inferred from a higher-level description by module generators using parameterized templates. Induced by the repetition of per-bit operators to form datawords referred to as datapath functions, datapath circuitry has an inherently regular interconnect structure. An example of a datapath function is an ALU or a register.

1.1. Logic Synthesis-based Datapath Generation

In the generic approach using logic synthesis, the generated datapath netlists are technology-independent gate netlists, and no notion of regular placement is provided or used. The placement is generated in the conventional way. Hence no advantage can be taken from the netlists's structure because generic placement techniques also lack any notion of regularity, nor will their typical wirelength minimization strategy yield a regular datapath placement even if the input netlist were completely regular. Since it is in no way restricted, logic optimization can use its freedom to perform its techniques which involve structural transformations. These transformations include constant propagation, collapsing, factoring, redundancy removal, technology mapping, gate or transistor sizing, and test logic insertion. The main attractiveness of this approach is its flexibility. It allows complex trade-offs and extensive and specific tuning. For instance, if the constant value 8 is hard-wired to an input of a 16 bit adder, performing constant propagation will remove significant portions of the netlist yielding a smaller and faster, but much less regular circuit. The phase of a net may be inverted, the fanout drive of a cell may be adapted, cells and nets may be removed or added, nets may be connected to other pins, etcetera. Figure 1 illustrates some of these effects on the interconnect structure on a simple example. On the left side

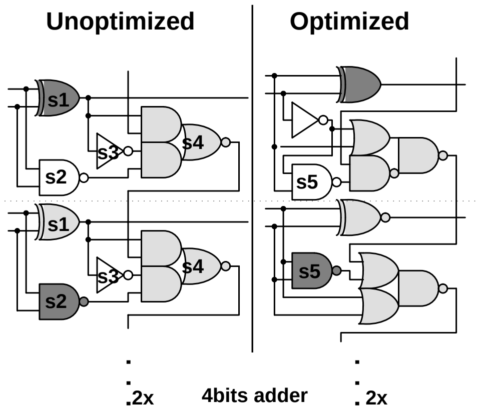

Figure 1. Effect of logic optimization on regularity

2 bitslices of an unoptimized of a carry ripple adder created by a portable module generator are shown. The right side shows the same circuit after logic optimization. We imposed a timing constraint on the carry path which made it more than twice as fast as the original general version. As can be seen, the circuit has been restructured without preservation of the regular interconnect structure: in the unoptimized circuit, all cells are part of the regular structure, but in the optimized circuit, only the two nands form a regular pattern. Subsequent conventional layout methods will thus create a placement as if the circuit were random logic.

1.2. DPC-based datapath generation

In contrast, DPCs exploit datapath regularity by preserving the netlist structure of the datapath functions so that their internal placement which is also created by the module generator can be mapped onto regular layout almost directly. For example, some of the so called designware products use ECO placements for this purpose. The main attractiveness of DPCs is that the strictly imposed and preserved regularity along the entire design flow produces very dense layouts in many cases. Many serious drawbacks of this method come from the entailing rigidity regarding the design flow because DPC tool suites necessarily make up an entirely separate, often technology dependent design subflow. As a result, the integration of datapath blocks with the rest of the design is often far from seamless. Furthermore, within the DPC framework the circuit netlist may not be modified without maintaining consistence with respect to the internal function placement created by the module generator. Since there exist no logic optimization techniques with any notion of interconnect structural regularity, much if not all of the datapath regularity will be lost while the given initial function placements are no longer valid for the modified datapath functions. This drawback almost completely precludes logic optimization on datapaths to satisfy area, timing and power constraints and objectives. Although the DPC generated layouts are more predictable regarding aspect ratios and wire lengths hence timing, a consequence of the impossibility to logically optimize the datapath is that the resulting dense layouts may in fact be suboptimal in many important aspects, while meeting design constraints can be a very tedious if not impracticable task. In general, the fundamental problem of DPCs is that the high extent of structural regularity required for high density layouts can not be provided, while the layout quality decreases rapidly as the circuit is less regular. Common sources of datapath structural irregularities vary from a different number of bits within a datapath to datapath functions which are only partially regular such as carry lookahead logic.

1.3. Regularity Extraction based Datapath Generation

To alleviate this problem, much of the remaining regularity of the interconnect structure can be identified by regularity extraction [5]. The extracted regularity information is then used to compute both horizontal and vertical alignments and orderings for the cells that were identified as being part of a regular circuit. From this information, a *partial* relative placement initialization is obtained for the subsequent cell placement stage in the design flow.

The efficacy of this method depends on the amount of structural regularity that can still be found after logic synthesis. Hence, as the extent of optimization performed by logic synthesis increases, the amount of placement initialization tends to decrease. In netlists generated by logic synthesis, the amount of regularity recognizable by *structural* techniques can drop below the extent where it is still beneficial. Observations from our experiments indicate that placement with structural hints as compared to generic placement pays off less if the extracted fraction of cells drops below 20%. For instance, in figure 1, both slices shown are identical in the unoptimized circuit on the left, so they will all be aligned regularly in the placement initialization. However, the bit slices in the optimized circuit on the right are no longer identical; the remaining structurally regular cells are labeled. The amount of regular placement initialization has clearly dropped significantly, possibly to an such an extent that it is hardly useful.

1.4. Exploiting the Locality of Logic Synthesis

Importantly, most transformations performed by logic synthesis are effectively local. Their impact on the overall structure of the circuit is consequently limited. In figure 1 it can be observed that the unoptimized and optimized circuits have an underlying structure which is largely similar. More specifically, this means that in the netlists before and after logic synthesis, many points can be identified which implement the same function. In fact, it is an intrinsic property of existing logic optimization techniques that the interconnect structure of points in the unoptimized circuit is almost identical to the interconnect structure of their equivalent points in the optimized circuit. In experiments on an RTL description of the minmax benchmark, we observed that even with a high optimization level, a functional equivalence could be found for roughly 50% of the nets in the optimized netlist. These experiments were conducted using the SIS logic optimization system developed at UCB. We exploit this property of similar interconnect structures of functionally equivalent points by inferring a regular placement of the optimized circuit from the regularity information extracted from the unoptimized circuit. The basic idea is that even though these cells are structurally not as regular as in the unoptimized version, they placementwise still form a sufficiently regular pattern to benefit from a regular placement. We therefore use these cells to augment the amount of regular placement initialization.

The reasons why logic synthesis has such a limited effect on the overall structure of the circuit are that it is computationally expensive to collapse a larger circuit, and moreover, this very often yields much worse implementations. Especially for arithmetic circuits, the best results are obtained when only local optimizations are used. Thus, logic synthesis along the design flow implicitly but inherently preserves much of the underlying structure of the circuit. Consequently there is a high extent of implicit structural and functional correspondence left between individual circuit elements in the netlists before and after logic optimization which our method can obtain even if the optimized circuit is no longer strictly regular.

In the work done by Brand et al. [1] on incremental synthesis, the placement of a logically unoptimized circuit is utilized to obtain a similar placement of an optimized version, but there are important differences in approach and objectives. The prime purpose of their method is to preserve as much as possible of a placement after minor changes to the function of the circuit. Importantly, they therefore preserve the placement itself rather than the regular structure, which is the goal of our method. Doing the latter is more flexible for our purpose in that it allows a different regular placement for the optimized circuit subject to the same interconnect structure. Also, they use a feature in their specific logic optimization tool to prevent almost the entire logic netlist from being modified. While such features are not generally available in logic optimizers, it in fact impedes what we want to achieve, namely optimize the circuit. Interestingly, in order to derive which cells should be assigned the same placement, they employ ATPG techniques to find nets in the optimized circuit that can be replaced by nets from the unoptimized circuit without changing the function of the entire circuit. Functional net equivalence is however more suitable for our purpose than circuit level equivalence because the latter may introduce many ambiguities.

2. OVERVIEW OF THE METHODOLOGY

We developed a methodology which combines the advantages of DPCs with the flexibility offered by logic synthesis. It fits into existing design flows by adding a number of extra steps, shown shaded in figure 2. These steps primarily consist of 2 recently introduced algorithms.

First, the input cell netlist is analyzed by a regularity extraction algorithm. It automatically extracts most of the 2-dimensional interconnect structure in the netlist. Next, a relative placement

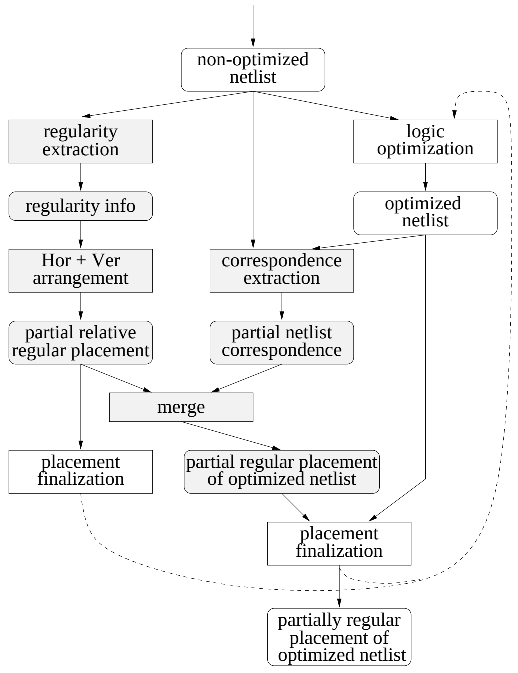
non-optimized netlist
regularity extraction
logic optimization
regularity info
optimized netlist
Hor + Ver arrangement
correspondence extraction
partial relative regular placement
partial netlist correspondence
merge
placement finalization
partial regular placement of optimized netlist
placement finalization
partially regular placement of optimized netlist

Figure 2. The design flow showing the new extensions shaded

is computed for the extracted datapath cells serving as a partial placement initialization for the subsequent placement finalization. The regular placement is then evaluated as usual, and if the design objectives and constraints are not satisfied, the gate netlist is backannotated for a logical optimization step yielding a much less regular netlist.

The second important algorithm in our scheme therefore extracts the functional correspondences between the unoptimized and the logically optimized netlists so that the initially discovered regularity can be merged along these correspondences into the optimized netlist. Finally, after the merging step, a regular placement for the logically optimized netlist is obtained. The remainder of this paper is organized as follows. The next section explains the principle of the datapath regularity analysis step. Section 4 describes the basic concepts of how the functional correspondence extraction is performed. The merging step is outlined in section 5, and experimental results are presented in section 6. The paper ends with concluding remarks and future directions.

3. REGULARITY EXTRACTION

The goal of structural regularity extraction is to identify a 2-dimensional alignment structure of the components of the netlist induced by the repetition of the bit-slices implementing the bitwise parallelism of datapaths. It is based on both the physical properties of the components and their interconnect characteristic categories, namely large hard macros such as memories, structured logic cells like datapaths, and unstructured glue logic. If the regularity of the netlist is not (yet) too severely distorted by optimizations that modify the netlist interconnect structure, the extractor recognizes it allowing even medium non-regularities. Furthermore, circuits are commonly described using the functional hierarchy used throughout the design flow, because the size of indiscriminately flattened netlists is often practically intractable for generic layout generation. At the same time, manual selective flattening will become more impracticable as design complexities increase, especially for circuits automatically generated from behavioral descriptions. The advantages for placement entailing from the locality and restricted problem sizes implied by this hierarchy however do not counterbalance against the disadvantages from the narrow optimization scope in case a physical hierarchy is used, as is done by DPCs. The extraction process selectively flattens the hierarchy on the fly so that datapath member cells end up in the same level. Although this may lead to placement tasks of the datapath cells that would otherwise be too large, we use the extracted regularity information to drastically reduce the placement problem size.

3.1. Datapath Regularity Modeling

The aforementioned alignment structure is constituted by 2 partitionings of the circuit, namely one decomposing it by bit-*slice*, and the other by datapath-*stage*. Components assigned to the same class will be aligned linearly. Figure 3 sketches the two orthogo-

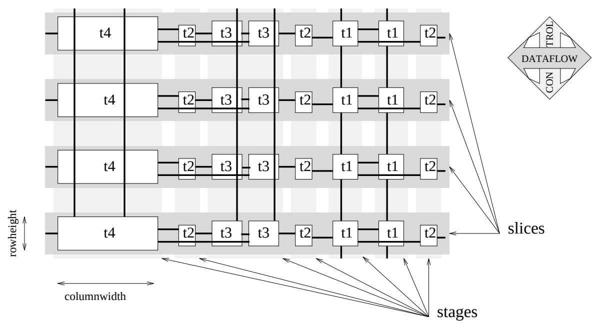

Figure 3. Orthogonal alignment classes

nal partitionings. The cells associated with the same bit are aligned horizontally, coinciding with a row in the common row-based layout style, hence they have about the same height. The cells in a vertical alignment group have the same type, hence the same width. Next to this strong *geometrical* regularity allowing compact placement, there is also a strong intrinsic *interconnect* regularity in that most nets are confined to either one stage or one slice. As a result of the datapath architecture, few nets run almost randomly. These properties allow obtaining a relative placement of the datapath by separately computing a linear arrangement for the slices and the stages. This substantially reduces the size of the solution space of the placement task, enabling larger subcircuits to be placed in one run.

3.2. Regularity Quantification

The extractor identifies regular structures by expanding datapath-wide search-waves through the network. An essential innovation of our method is that it is guided by a relative regularity measure introduced in [5] where it is presented in more detail. An analysis is made of the regularity of the neighborhood of a reference stage, i.e. the set of components that have an incidence to the components of the reference stage. More specifically, the analysis considers the respective incidence *signatures* (RSes) of the adjacent components. An RS consists of at least the terminal-type of the incidence between a net and a cell, which is known in any circuit description. Other attributes such as signal flow direction can also be used if available. The relative regularity measure then computes a value for each signature in the neighborhood of this reference stage, expressing the extent of regularity between the reference stage and the set of adjacent components with that signature. This metric is a weighted sum of the distribution of the degree of the vector of incidences between the slices in the reference stage, their average value and the number of zero-entries. The value of the metric decreases monotonically as the candidate extension is placement-wise considered more regular. Maximum regularity is indicated by the value 0.

For example, the bipartite graph in figure 4 depicts the relation between the bits b_i of a reference stage on the right and the adjacent signatures τ_j on the left. The pin-sets on the edges denote the

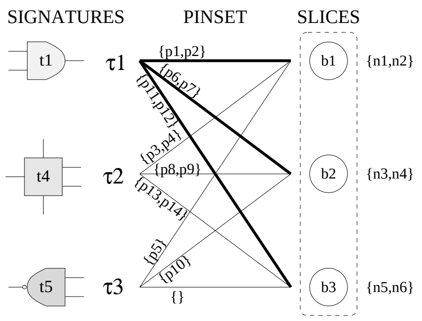

Figure 4. Example regularity analysis

incidences. The incidence vector for τ_1 is $[2, 2, 2]$, which corresponds to a high degree of regularity, since $[1, 1, 1]$ is maximum.

3.3. Extraction Algorithm

Importantly, the above metric imposes a global linear ordering of the candidate extensions by their regularity. This allows comparison of all candidate extensions, from which the most regular extension is selected. Candidates are made up of sets of incidences, one per bit in the reference slice. A new stage is then added to the datapath by assigning all elements in the selected candidate to one new stage, while each element inherits one slice-membership via the incidences.

The search wave is expanded until no more sufficiently regular extensions remain. A new search-wave is started for every *seed*-stage which are automatically recognized by utilizing the datapath characteristic that there are several control-nets that connect to all slices to cells of the same type via the same terminal [6]. After all seeds have been used, the extracted datapath with the best statistics, such as size, width, average regularity etc. is selected, and all overlap with other waves is removed.

Finally, the alignment groups are linearly arranged to obtain a relative placement for the cells that were recognized as being part of regular circuitry. This partial placement then serves as a partial placement initialization. The subsequent placement process pulls many non-initialized cells closely related to the initialized cells into position so that non-initialized positions due to irregularities will still be filled.

4. FUNCTIONAL CORRESPONDENCE EXTRACTION

The objective of functional correspondence extraction is to find a partial correspondence between the cells of the netlists before and after logic optimization. Because the names of cells and nets are typically not preserved during logic optimization, corresponding points do not necessarily have identical names. Therefore, we propose the following approach to extract this partial correspondence. First we identify nets that implement the same Boolean function (modulo negation). With this information, a partial correspondence between the cells of both netlists is constructed.

We restrict the discussion to combinational circuits. In [4], it is explained how the approach can be extended to automatically extract the register correspondence for circuits with identical state encodings. Finding a functional correspondence after sequential optimizations such as retiming is beyond the scope of this paper.

4.1. Detecting equivalent nets

We assume that the correspondence between the primary inputs of both netlists is known. Therefore, we can associate a Boolean function with each net, which expresses the value of that net as a function of the primary inputs. We want to partition the nets into classes of nets with the same function (modulo negation). More precisely, two nets n_i and n_j with functions f_{n_i} and f_{n_j} are called equivalent iff: $(f_{n_i} \equiv f_{n_j}) \vee (f_{n_i} \equiv \overline{f_{n_j}})$. Generally, calculating this partition is a difficult problem, because determining the equivalence of Boolean functions is a co-NP complete problem. However, for circuits with a similar structure, efficient methods are known to calculate an accurate approximation of this partition. We use the method from [3] which was originally developed to formally verify that the netlists before and after logic optimization implement the same functionality. It combines the use of binary decision diagrams (BDDs) [2] with a technique to automatically detect and utilize equivalent internal nets. Because an equivalence check can be performed in constant time on BDDs, the efficiency of a BDD-based verification method is largely determined by the efficiency with which the required BDDs can be constructed. There are types of circuits for which the BDDs become intractably large. To alleviate this problem, the functions of the nets are not expressed over the primary inputs only, but also over extra variables representing equivalent internal nets. A set of equivalent nets for which an extra variable is introduced is called a breakpoint, and the associated variable is called a breakpoint variable. Figure 5 illustrates the meaning of breakpoint variables.

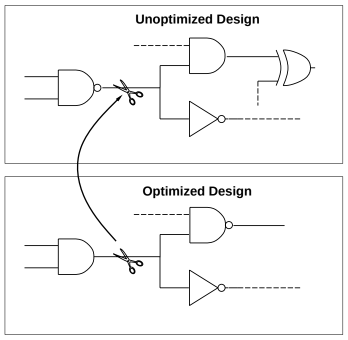

Figure 5. Functionally equivalent nets

The main advantage of introducing breakpoints is that if sufficient equivalences exist between the two circuits, all BDDs remain compact. If for two nets the same BDD is calculated, then these nets are equivalent. If two nets have different BDDs, variable assignments apparently exist for which the functions differ. However, these assignments may not be valid, because the breakpoint variables cannot be assigned values independently. Such invalid assignments are called false negatives. This problem is avoided by replacing each breakpoint variable by the function it represents. It can only be decided that the functions are not equivalent, if the difference expressed over the primary input variables is not zero. Hence, a disadvantage of introducing breakpoints is that the equivalence check may require substitutions, possibly resulting in a large BDD representation for the difference of two nets. However, if the equivalent nets are the result of a similar structure, we expect the difference to be relatively small or even zero, in which case no substitutions are required at all.

Equivalent nets are detected in a single traversal of both netlists. In order to detect and introduce breakpoints on the fly, both circuits have to be traversed simultaneously. To synchronize these traversals, it is necessary to already have a notion of which nets are likely to be equivalent. Therefore, the first step of the functional correspondence extraction is to partition the set of nets into classes of potentially equivalent nets. Simulation with random input vectors is used to compare the functions of the nets for a small part of the input space. It forms a coarse check for the equivalence of two nets without excluding any equivalences. The accuracy of the analysis and the required run time depend on the number of input vectors. If possible, we also try to exploit name equivalences.

After calculating the potentially equivalent nets, the circuits are traversed by repeatedly performing the following steps. First, a set of potentially equivalent nets is selected which is close to the nets that have been calculated thus far. Then BDDs are calculated for these nets, and the functions of the nets are compared. Heuristics are used to decide which nets actually need to be compared. This may result in the detection of one or more classes of functionally equivalent nets. If equivalences are found abundantly, it is not necessary to actually introduce a new variable for each set of equivalent nets. This is decided in a last step. The details of the method can be found in [3].

4.2. Constructing the cell correspondence

When the equivalent nets have been identified, a partial correspondence between the cells of both netlists is constructed. Single-output cells are said to be in correspondence if their outputs drive functionally equivalent nets. Similarly, multiple-output cells are said to be in correspondence if for each output of one cell, there is an output of the other cell which drives an equivalent net. We require that for each cell, there is at most one corresponding cell. Furthermore, we add the extra constraint that the correspondence does not conflict with the topology of both netlists, i.e. if two cells c_{s1} and c_{s2} in the unoptimized netlist correspond with respectively cells c_{i1} and c_{i2} in the optimized netlist, and there is a directed path from c_{s1} to c_{s2}, then there may not be a directed path from c_{i2} to c_{i1}. In practice, this is rarely a problem, and therefore we have a greedy algorithm based on some simple heuristics to construct a cell correspondence which meets these constraints.

5. MERGING STRUCTURAL AND FUNCTIONAL EXTRACTIONS

The merging step in the design flow combines the results of the two extraction steps by folding the regularity extraction results from the unoptimized circuit into the optimized circuit via the extracted functional correspondence information between the two circuits. The cells are member of categories as shown in figure 6. Observe

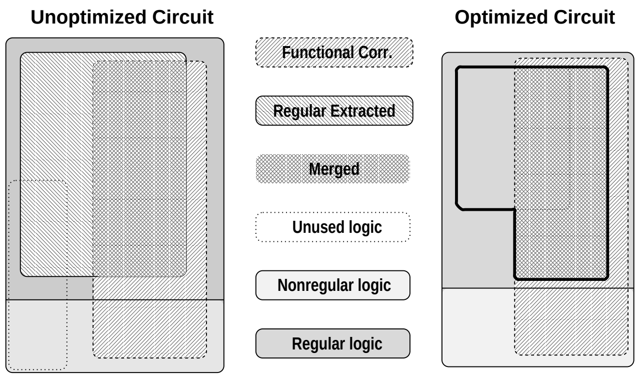

Figure 6. Sets of cells associated with the two extractions

that the extraction results may not be complete, especially in the case of the logically optimized circuit. The regularity extraction result is in general a subset of the set of cells that are part of the structured logic of a circuit. Likewise, the set of cells for which a functional correspondence exists in general does not cover the entire circuit. Note that since unused cells are removed after the optimization step, there can be no overlap between the set of unused cells and the set of cells for which a functionally equivalent cell is found in the optimized network.

Basically, for every cell in the unoptimized circuit with both regularity information and a functional correspondence to a cell in the optimized version, the merging step assigns this regularity information to the latter cell. In the figure, this applies to the area where the two hatched areas overlap, which is crosshatched.

An additional phase of the merging step is to combine the regularity information that could still be found in the optimized circuit with the regularity information that was obtained via the above-mentioned method. In the diagram, the corresponding area is enclosed with a fat line. Our experiments however showed that in practice, the amount of additional regularity thus obtained is small, whereas this step involves many ambiguities that are hard to solve in a robust way. Since the coverage of the regular part of the optimized circuit found is already sufficiently large for the purpose of cell placement, we ignore this additional data.

The run-time complexity of the actual merging step is only $\mathcal{O}(n \log n)$ where n is the number of cells for which a functional correspondence exists. Like the extraction steps, it therefore doesn't noticeably add to the run-time of the design flow.

6. EXPERIMENTAL RESULTS

We performed a number of experiments on four example circuits to evaluate the proposed methodology. These circuits contain only combinational logic cells and can be regarded as pipeline stages of a larger datapath. DP1 contains two 8-bits wide cascaded adder/subtractors. DP2 is a 32-bit datapath made up of various elements including a multiplier, an adder and several multiplexers so that there are many paths with different lengths from the primary inputs to the primary outputs. In contrast, the primary paths in DP3 have almost uniform length. DP4 is a realistic complex 32-bit datapath pipeline stage consisting of an ALU, multiplexers, a shifter and a comparator. It must be noted that cases with a high complexity and size like DP2 are rare in practice.

All circuits were optimized in flat mode to meet realistic global

Table 1. Structural and Functional analyses results

Circuit	Structural						Functional			Merged	
	unoptimized			optimized							
Name	Size	Perc.	Time	Size	Perc.	Time	Corr.	Perc.	Time	Size	Perc.
DP1	138	89%	0.3s	128	0%	0.0s	71	55%	0.4s	67	52%
DP2	1719	94%	2.0s	1591	88%	3.3s	1020	64%	11.7s	955	60%
DP3	421	92%	0.2s	424	0%	0.0s	244	58%	1.1s	235	55%
DP4	1047	92%	1.3s	902	50%	1.0s	477	52%	9.5s	434	51%

timing constraints on the pipeline stage. We observed that through logic optimization large speedups can be obtained which would not have been allowed within a DPC framework. In the case of DP4, the circuit speedup obtained was more than 300%. Table 1 shows the results from structural extraction on both the unoptimized and optimized circuits. Clearly, the regularity extraction works well on the unoptimized netlists. Note that the percentages mentioned are relative to the *entire* circuit as opposed to only its datapath logic. In designs DP1 and DP3 in which almost all paths are critical, hardly any structural regularity remains after logic optimization. In larger more complex circuits like DP2 and DP4, the ratio of critical versus non-critical nets is typically much smaller, which explains why more regularity in those circuits was preserved. The reason why the non-critical parts are not modified is that the structure created by the module generator is generally very good areawise.

Table 1 lists the results from functional correspondence extraction between the unoptimized and optimized circuits. It shows the number of cells in the optimized circuit for which a corresponding cell is found in the unoptimized circuit. The percentages mentioned for the structural extraction results are relative to the entire associated circuit, not only the part of the circuit that is really regular, because the ratio between glue logic and structured logic is not known for these examples. The percentages for the functional analyses are relative to the number of cells in the optimized netlist, as it is the target of the design process. The high percentages confirm that the amount of placement initialization can be augmented utilizing functional correspondences.

Note that the set of cells assigned to an alignment set in the unoptimized netlist does not necessarily fully overlap the set of cells in the same netlist for which a functional correspondence exists. In practice, this overlap turns out to be sufficiently large. Since the run-times of the algorithms are low, clearly our method is applicable for large designs without significant computational effort. The results also confirm that while the degree of distortion of the interconnect structure increases as the circuit size decreases, and that our method provides a way to still place those circuits regularly.

We also experimented with actually generating both regular and non-regular placements for both unoptimized and heavily optimized circuits. These experiments indicate that the improvement obtained when placing an unoptimized circuit regularly as compared to a general placement is also obtained for optimized circuits to a similar extent.

7. CONCLUDING REMARKS

We exploit several properties that are intrinsic to the techniques used for existing VLSI datapath design and synthesis to get rid of the conventional mutual exclusion of dense regular datapath layout generation and logic optimization. Application of functional correspondence extraction between the initial logic netlist and its optimized implementation greatly augments the amount of layout regularity alignment information for the optimized netlist. This data is obtained by performing a structural analysis on the netlist while it is still sufficiently regular, and merging the result with the no longer regular optimized implementation via the correspondences extracted by the functional correspondence analyzer. Thus, a partial regular placement initialization for the logically optimized netlist is obtained so that extensive logic optimizations no longer preclude regular layouts. This approach allows generation of larger layouts of flat netlists as compared to conventional methods at a low computational overhead.

Future work is needed to provide more insight on how the amount of regular placement initialization supplied to the placement tools influences the improvement on the layout quality subject to the placement algorithms used, the target device technology and the involved characteristics of the design. The proposed methodology nonetheless in many cases clearly enables utilization of datapath regularity. In our experiments, practically realistic timing constraints appeared to topologically coincide with the regular structure, hence placement. This observation gives rise to further research the impact of the entailing higher timing predictability during logic synthesis on the convergence of the design flow iteration. Finally, sequential signal correspondence techniques must be added to allow larger networks and more complex restructurings like retiming, so that more comprehensive evaluations of the practical effects of the methodology can be surveyed on a large set of designs.

REFERENCES

[1] D. Brand, et al. *Incremental Synthesis.* Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 14-18, 1994.

[2] R.E. Bryant. *Graph Based Algorithms for Boolean Function Manipulation.* IEEE Trans. on Computers, vol. C-35 no. 8, pp. 677-691, August 1986.

[3] C.A.J. van Eijk and G.L.J.M. Janssen. *Exploiting structural similarities in a BDD-based verification method.* Proc. 2nd Int. Conf. on Theorem Provers in Circuit Design, LNCS vol. 901, pp. 110-125, 1995.

[4] C.A.J. van Eijk and J.A.G. Jess. *Detection of Equivalent State Variables in Finite State Machine Verification.* Workshop notes of the 1995 ACM/IEEE Int. Workshop on Logic Synthesis, pp. 3.35-3.44, 1995.

[5] R.X.T. Nijssen and J.A.G. Jess. *Two-Dimensional Datapath Regularity Extraction.* Workshop notes of the 5th ACM/SIGDA Physical Design Workshop, pp. 110-117, 1996.

[6] G. Odawara, T. Hiraide and O. Nishina. *Partitioning and Placement Technique for CMOS Gate Array.* IEEE Trans. on Computer Aided Design, vol. 6, no. 3, pp. 355-363, 1987.