

Regularity Driven Logic Synthesis

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

Abstract

We present a new and innovative logic synthesis approach using regularity information of a design to selectively apply transformations and globally guide the synthesis process.

Since traditional logic synthesis applies transformations without consideration of global design characteristics such as regularity and dataflow, it destroys a substantial amount of regular structures. In addition, due to the non-incremental nature of most logic transformations, synthesis relies vastly on the computationally expensive concept of trial and error application of transformations, a time-consuming process in the synthesis of large designs.

The proposed approach addresses both shortcomings of traditional logic synthesis and describes a mechanism to speed up logic synthesis and preserve regularity. It selectively applies transformations to places with similar characteristics and to the same stage of a regular structure, introducing a notion of dataflow-aware synthesis.

Preservation of regular structures has tremendous advantages to the following physical design stages. It yields high-density layouts, shorter wiring length and improved delay. In addition, the layout becomes more predictable at an earlier design stage.

1 Introduction

The increasing complexity of microelectronic designs and the continuous development of smaller sized fabrication processes presents new challenges to existing design automation tools. It is widely acknowledged that future electronic design automation methodologies must be able to handle the challenges and opportunities imposed by very large designs. This particularly applies to the field of logic synthesis, as existing design sizes already present a considerable challenge to current synthesis tools. Future synthesis tools are expected to handle millions of gates in a reasonable amount of time. However, it is questionable whether traditional logic synthesis is able to satisfy these increasing demands. Due to the amount of available topological and functional transformations and their non-incremental nature, it is generally difficult and computationally expensive to select the best suitable subset

of transformations and their respective sequence. Therefore, current synthesis methodologies mostly rely on the concept of applying trial and error sequences of transformations. This process, however, produces a tradeoff between efficiency and quality and generally does not yield optimal results. In addition, the achieved results vary depending on the actual designs, hence the overall optimization is left to the experienced designer.

In the logic synthesis of large designs, it is very important to efficiently find places where to apply transformations to the design. When a transform succeeds at a certain place, it would be desirable to be able to apply it quickly at other places in the design with the same characteristics.

Another important challenge of logic synthesis is the preservation of regular design structures during the optimization process. Maintaining regularity has significant advantages to the subsequent physical design stages. Regular structures can be placed in rows and columns, which yields a much denser layout, tremendously decreases wire length and delay, and furthermore simplifies the overall placement task. In addition, the layout of a regular design is more predictable at an early design stage. The extraction of regularity and its usage has been exploited and used thoroughly to obtain high-density layouts in placement [1, 8, 9, 10, 11]. Special placement techniques, such as datapath placement [5, 9], have been developed to take advantage of regular structures.

While it is widely acknowledged that generic logic synthesis destroys a substantial amount of structural regularity, particularly during logic minimization and technology mapping [2, 3, 7], previously published approaches only focus on the extraction of regularity. The problem of maintaining regularity throughout the design flow has rarely been addressed [6, 7], and to date, no solutions have been presented. A commonly used concept to avoid destruction of regularity is to skip logic minimization and map the technology by manual assignment of library cells [4]. This process does not only demand a substantial amount of manual work, it is also unable to benefit from potential improvements by logic minimization algorithms and is therefore generally undesirable.

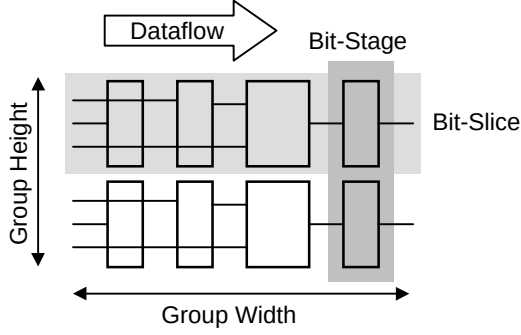


Figure 1: Illustration of a regular group

This paper presents solutions to the mentioned shortcomings of conventional logic synthesis and describes a mechanism that is able to speed up logic synthesis while maintaining regular structures. Since conventional logic synthesis is generally unable to understand and use any notion of regularity, transformations are applied without any notion of the dataflow or other global design information. In contrast, the presented methodology extracts regularity information of the design and selectively applies suitable transformations to places with similar regularity signatures. Through reapplication of successful transformations, a faster and more complex identification of a suitable subset and sequence of logic transformations is possible. In addition, more regularity is preserved, since identical transformations are applied to the same stage within a regular structure. In contrast to generic logic synthesis, the resulting process is aware of global design characteristics, such as regularity and the dataflow.

The remainder of this paper is organized as follows. Section 2 introduces the general concept of drivers and transforms and their interaction. In section 3, we outline global and local regularity metrics and explain their usage. The general framework of a regularity driven logic synthesis methodology is elucidated in section 4. Results on several benchmark designs are presented in section 5, followed by conclusions and future work in section 6.

2 Concept of Drivers and Transforms

The proposed synthesis algorithm utilizes a concept of *drivers* and *transforms* within the framework of an electronic design automation tool [12].

A *driver* is defined as the part of an algorithm which decides where and how to apply an action in a design. In contrast, we define a *transform* as the part of an algorithm which applies an action. A driver iterates through the design, determines a beneficial sequence of transformations and selectively applies them to a set of logic gates. This concept builds the base foundation for controlling the synthesis flow and applying actions

dependent upon global design characteristics such as regularity, symmetry and dataflow.

We use two primary groups of drivers, *generic drivers* and *timing drivers*. *Generic drivers* iterate through the design and apply a set of transformations to all gates or nets in the network. In some cases, it is desirable to process gates or nets in a specific order. To address this need, *levelized drivers* are used to process gates or nets from inputs to outputs (in left to right order) or vice versa. Generic drivers are mainly used during technology-independent optimization and technology mapping.

The second group of drivers, *timing drivers*, uses timing information of the design to improve cost functions like critical delay, area or power consumption. In general, complicated delay rules reduce the ability of an algorithm to estimate the effect of changes to the network on the delay. Thus, *timing drivers* apply a number of transformations, collect cost and benefit data, and undo the transformations. Drivers may try different transformations and places in the network before determining the most effective transformation and its place of application. In general, this trial and error sequence of applying transforms is computationally expensive.

We use two different types of timing drivers, *critical* and *non-critical drivers*. The *critical driver* applies a set of transformations to the critical path in the network to reduce the critical delay of the design. In contrast, the goal of the *non-critical driver* is to optimize cost functions along non-critical paths, for example area and power reduction. Timing drivers are applied during technology dependent optimization such as delay and area reduction.

3 Regularity Modeling

The proposed regularity driven synthesis methodology utilizes *global* and *local regularity metrics* to drive the synthesis process, which will be detailed in the following sections.

3.1 Global Regularity Metric

Global regularity refers to regular structures that improve cost functions during the physical design process. Most modern VLSI designs are characterized by a large amount of datapath circuitry to achieve high performance through parallelisation and contain a high degree of regularity. As we determined through extensive experiments, it is essential to identify and maintain structural regularity at an early design stage, as particularly technology independent optimization and technology mapping destroy a substantial amount of regularity. In fact, the area and delay optimization criteria used during these optimization steps are often inaccurate since no physical and structural design information is utilized. In many cases, the actual placement area and delay of a highly

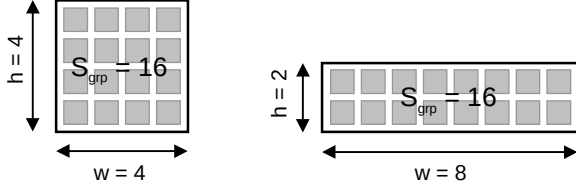


Figure 2: Shape and boundary length of regular groups

regular design can be significantly smaller since placement of regular structures in rows and columns yields a high packing density and short wiring. Therefore, the use of structural regularity as an additional optimization criteria during logic synthesis improves and yields a more accurate prediction of the final physical layout.

Given a design D , consisting of a set G of gates, a set N of nets and pins $P \subset G \times N$, we use our highly efficient regularity extraction algorithm as described in [7] to identify *regular groups*. A *regular group*, as shown in Figure 1, is a rectangular block containing a number of bit-slices and bit-stages, which determine *group height* h and *group width* w , respectively. Consequently, we define S_{grp} , the product of height h and width w , as the *size of a regular group*. The fact that rows and columns are spaced in the final layout, and as result, the actual size of a regular group is effectively larger, is negligible at this point, as it applies to all gates in a design.

To measure the amount of structural regularity that effectively improves area and delay cost functions in the final physical layout, we introduce the *physical regularity index* RI_p as follows:

$$RI_p = \frac{1}{n_{grp} + n_{nr}} \cdot \left(n_{nr} + \sum_{i=1}^n \left(S_{grp_i} \cdot \frac{2 \cdot \sqrt{S_{grp_i}}}{h_i + w_i} \right) \right) - 1$$

In the above equation, n_{grp} is the number of regular groups, while n_{nr} is the number of gates which are not within a regular group, i.e. the number of objects with group size 1. Hence, the first part of the equation measures the total number of objects in a design. The second part of the equation represents the size of all non-regular objects, i.e. all gates not within a regular group plus the size of all regular groups, multiplied by a factor that considers the shape of each regular group. In fact, the shape of a group, i.e. its height versus its width is very important during placement. It is obvious that a group which is either substantially wider than higher, or vice versa, is more difficult to place in the layout. Most placement algorithms employ vertical and horizontal cuts to move objects in the placement area. The larger the height or width of an object, the more likely it will become frozen at an early placement

stage, often stuck in a less than optimal position. In Fig. 2, while both groups have the same size, the sum of height and width of the right group is significantly larger than that of the left group, and in conclusion, it is more difficult to place. For a given area, a square has the minimum sum of height and width $2 \cdot \sqrt{S_{grp}}$ of any rectangle. For simplicity, we assume the size of each gate to be identical. In general, the physical size of a logic gate g can either be defined by the area footprint of its library cell, or alternatively, by the number of literals for an unmapped gate.

The higher the regularity index, the higher is the estimated benefit during physical design phases. A value of zero indicates that no usable regularity exists.

In addition, the order of stages in a group determines the dataflow, as shown in Figure 1. The described structural regularity and dataflow information is utilized to drive the overall synthesis process as detailed in section 4.

3.2 Local Regularity Metric

The local regularity metric utilizes the concept of regularity signatures which serves two primary purposes. First of all, regularity signatures are used to speed up the synthesis process by applying transformations to places with similar characteristics, and secondly, to observe and control local changes made by transformations during synthesis to maintain global regularity.

3.2.1 Regularity Signature Definition

Given a design D , consisting of a set G of gates, a set N of nets and pins $P \subset G \times N$, each gate g of design D has a *gate type* γ . A gate type is defined by its logic function, area and/or power consumption. Similarly, each pin p of a gate g is characterized by a *terminal type* τ which is described by the function of p , i.e. input or output for simple combinatorial gates, or special functions for more complex or sequential gates, and additional characteristics, for example the required arrival time of pin p . Please note that a port, denoted by a solid circle in Figure 3, is treated as a special pin that is not assigned to any gate. In addition, we define a *net type* v for each net n . A net type is characterized by timing, load and capacitance of n , among others. The basic concept is shown in Figure 3.

For a better understanding of subsequent parts, we introduce the following definitions.

Def. 3-1:

Let γ , v , and τ be a gate, net and terminal type, respectively. A labeled directed graph $LG = (V, E, \Gamma, M, \Pi)$ consists of a set V of vertices, a disjoint set $E \subseteq V \times V$ of edges, and mappings Γ of V to γ , M of E to v , and Π of E to τ .

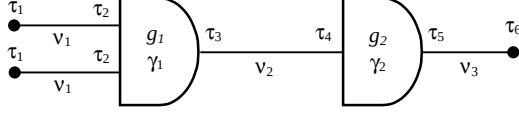


Figure 3: Relation of gate, terminal and net type

We define the *regularity signature* **RS** of net n as the labeled directed graph LG consisting of the source vertex v , its *immediate predecessors* and associated edges. Hence, a regularity signature essentially represents the two logic levels which compute the value of net n . Regularity signatures are defined equivalent if their labeled directed graphs are identical.

A regularity signature considers the two preceding logic levels of a net to determine the effect of a logic transformation since many transforms employ two-level optimization. In general, this definition can be extended to include n logic levels.

The application of a regularity signature, among others, is to control and compare the changes made by a transformation to the network. Therefore, a more complex signature that includes all logic levels affected by a particular optimization algorithm, called an *adaptive regularity signature*, is defined. It is dependent on the particular transform T and is defined as follows.

Def. 3-2:

Let LG_0 be the labeled directed graph of the original design, and LG_1 be the labeled directed graph after a transformation T . Then, an *adaptive regularity signature* RS_a is defined as the symmetric difference of LG_0 and LG_1 :

$$RS_a = LG_0 / LG_1 = (LG_0 - LG_1) \cup (LG_1 - LG_0).$$

Hence, an adaptive regularity signature is an exact measure of the effect of transform T on a particular design. By including additional information into terminal, gate and net type, we differentiate the following types of regularity signatures: *logic*, *time*, *area* and *power signatures*. Signature types are combined to satisfy the specific requirements and goals of the optimization process. Additional signature types can be defined, depending on the particular requirements.

3.2.1.1 Logic Signatures

A *logic signature* **LS** of a net n represents the logic function f of a regularity signature. The labeled directed graph representing a logic signature contains the logic function within gate type γ , and the pin function of a pin within terminal type τ .

In addition, a logic signature **LS** can be extended by information about the bit-stage of each gate, as identified in section 3.1, to utilize the dataflow.

Logic signatures are applied during technology dependent and independent transformations, they are the basic building block of every regularity signature.

3.2.1.2 Time Signatures

A *time signature* **TS** contains information about the arrival and required arrival times at the pins p of gates g .

The required arrival time *RAT* is associated with a pin, and is therefore modeled in the terminal type τ . To obtain identical signatures for identical structures, we model a single wiring delay per net, hence the arrival time *AT* is identical for all input pins attached to a net. Therefore, the arrival time is modeled in the net type v .

In addition, time signatures may contain information about the slew and load at the source and sink, respectively. Time signatures are equivalent by definition, if all its timing characteristics vary within defined boundaries. Time signatures are usually used during technology dependent optimization, e.g. to identify similar places in critical paths.

3.2.1.3 Area Signatures

Similarly to time signatures, an *area signature* **AS** is calculated by considering the area **A** of all gates g within the regularity signature.

Area signatures are considered equivalent, if the area of its gates g , modeled in gate type γ , vary within defined boundaries. They are applied during technology dependent optimization.

3.2.1.4 Power Signatures

A power signature **PS** describes the power properties of a net n and is characterized by

- the power consumption **P** of gate g ,
- the net capacitance **c**,
- the switching factor **s**.

The power consumption **P** is modeled within gate type γ , the net capacitance **c** and switching factor **s** within net type τ .

Similarly to time and area signatures, power signatures are defined as equivalent, if all of its characteristics vary within a defined limit.

4 Regularity Driven Synthesis

This section describes how the previously described global and local regularity metrics are applied within a regularity driven logic synthesis methodology.

Within the framework of drivers and transformations as described in section 2, we introduce a *generic regularity driver*, and a *timing regularity driver*, to guide the synthesis process.

4.1 Generic Regularity Drive

The generic regularity driver is applied during the first step of logic synthesis, boolean minimization and technology mapping. Our main objective at this design stage is to maintain global regularity. To achieve this goal, we employ the following algorithm:

- Step 1) Extract structural (global) regularity information
- Step 2) Select an unvisited regular group
- Step 3) Process gates within regular groups stage by stage, according to the dataflow
- Step 4) Apply transform to all gates within a stage
- Step 5) Calculate and compare local regularity signatures for all gates within a stage
- Step 6) Undo transform, if signatures are unequivalent
- Step 7) Goto Step 2, until all groups have been processed
- Step 8) Process all remaining gates not within a regular group

At first, we identify global regularity using our efficient regularity extraction algorithm as described in [7]. Gates within regular groups are processed stage by stage, in the order of the dataflow, applying a transform to all gates within a stage. Thereafter, adaptive regularity signatures, based only on logic signatures since we employ logic independent optimization, are calculated for all gates within the stage. The signatures also contain information about the bit-stage of each gate, as explained in section 3.2.1.1. To maintain existing regularity, signatures only include gates that are part of a regular structure, since changes to the network outside a regular group are allowed. If signatures of gates within the same stage are not identical, the transform will be undone. This process continues until all regular groups have been processed. Thereafter, all remaining gates are processed.

Since the synthesis process identifies regularity at an early stage, applies transforms according to the dataflow simultaneously to gates within one stage and ensures that regularity signatures remain identical, it effectively preserves more regular structures than a traditional synthesis process which applies transforms uncontrolled and in arbitrary order.

4.2 Timing Regularity Driver

Technology dependent optimization such as delay and area reduction is, compared to logic independent optimization, a computationally significantly more expensive process. Due to complex delay rules, transformations are repeatedly applied and undone utilizing a trial and error concept, until the most beneficial place and transformation has been identified. In general,

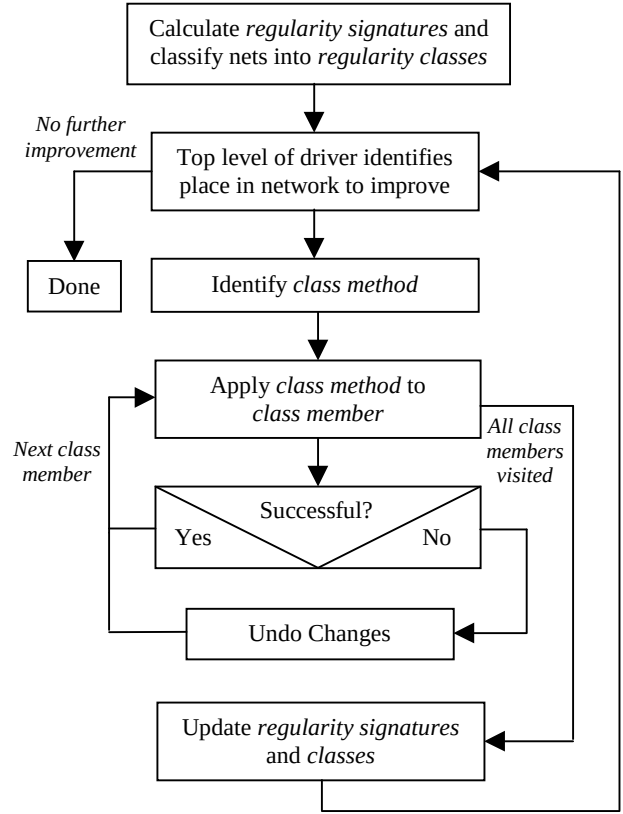


Figure 4: Timing Regularity Driven Synthesis Flow

this procedure yields unsatisfactorily high turn-around-times, particularly for very large designs.

The proposed regularity driven synthesis methodology uses local regularity signatures to identify places with similar characteristics in the design, and selectively applies transforms. The overall procedure is detailed in Figure 4.

First, regularity signatures, i.e. time-logic signatures, and depending on the optimization goal, power and area signatures, are calculated for all nets n in the design. Using the signatures, the set of nets is partitioned into equivalence classes, referred to as *signature classes*. The top level of the optimization process determines the particular part of the design to work on, e.g. the critical path to optimize delay, or area reduction on a non-critical path. A sequence of transforms is applied to a place in the network, until a set of transforms which achieves a favorable or the best solution, called a *class method*, has been identified. This *class method* is applied to each *class member*, and, if unsuccessful, any changes in the network are undone. Thereafter, the top level continues optimization on the next design part, for example the newly identified critical path. This procedure continues until a particular optimization goal has been achieved.

For example, regularity driven optimization of the critical delay in the network would first identify and optimize the most critical path. Assuming the most critical

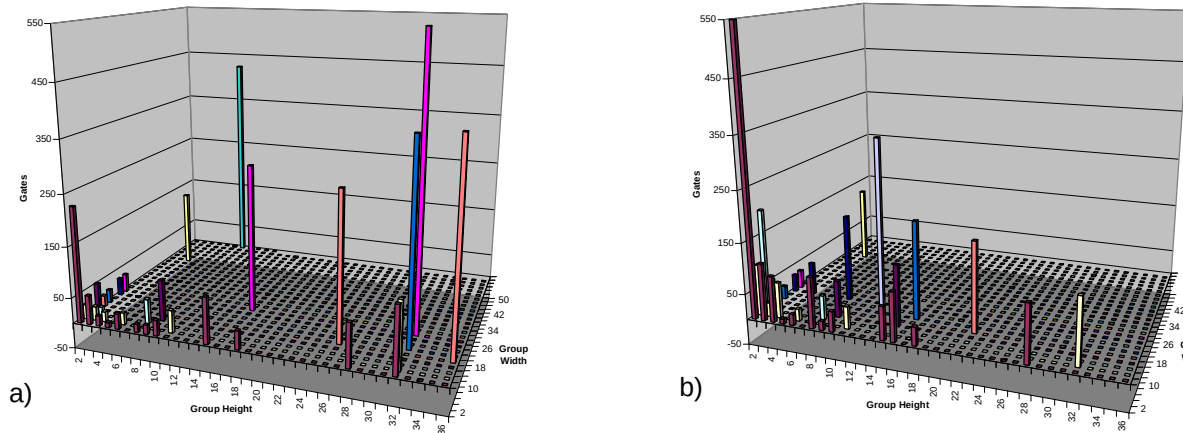


Figure 5: Distribution of Regular Groups for design *Bluechip*, a) *Regularity Driven*, b) *Conventional Optimization*

path is part of a datapath, all other paths contained in the same datapath are member of the same class, and will be optimized using the same set of transformations.

The application of transformations to similar parts of the design speeds up the overall optimization process. In addition, since transformations are applied to all class members, regularity is widely preserved.

5 Experimental Results

The presented regularity driven synthesis methodology was implemented in C++ within the framework of the logic synthesis tool BooleDozer™ [12].

Table 1 presents results of the technology independent optimization which performs logic restructuring on several benchmark and actual production designs with a high amount of datapath circuitry. *Bluechip*, *Micro* and *Stealth* are microprocessor kernels, *Solar* and *Max2* are encryption circuits, *Redox* and *Pipe* are communication controllers. Shown are the previously introduced physical regularity index RI_p , the amount of gates within regular groups (Reg), and the literal count (Lit) of the original unoptimized (*orig*), conventionally optimized (*conv*) and the regularity driven optimized (*reg*) designs. The latter uses the generic regularity driver described in section 4.1.

It is evident, that the amount of physical regularity measured by RI_p is significantly improved, ranging from 8.2% for the *Redox* chip to 207% for the *Micro* design. On average, an improvement of approximately 57% has been shown. At the same time, an almost identical literal count has been achieved, increased only by a negligible amount of 0.4%. Furthermore, run times are increased by roughly 8% on average, as regularity extraction and signature calculation is performed in addition to logic optimization.

Interestingly, the percentage of regular gates during optimization only decreases by a small amount, while the actual physical regularity measure often decreases substantially. This can be explained by the fact that large regular groups are split into several small ones during logic optimization.

Furthermore, the physical regularity index after logic optimization is in some cases higher than the original one. This is due to the fact that logic outside regular groups is minimized to a higher extent than regular logic because non-regular transformations to regular groups are being rejected.

Figure 5 shows the distribution of regular groups by group height and width for design *Bluechip* after conventional and regularity driven logic optimization. It can be clearly seen, that the regularity optimized design

Design	Gates	RI_p orig	Reg orig	RI_p conv	Reg conv	RI_p reg	Reg reg	Lit orig	Lit conv	Lit reg
S1423	653	0.41	32.0 %	0.34	30.7 %	0.52	35.4 %	1262	1262	1204
S38417	11890	3.08	84.0 %	1.11	63.1 %	1.99	82.8 %	23496	21215	21320
S38584	14489	0.74	53.2 %	0.69	47.3 %	0.83	49.7 %	30136	25155	25480
Bluechip	4430	1.39	63.5 %	0.83	52.3 %	1.73	72.4 %	15905	15817	15847
Solar	21728	1.02	55.6 %	0.98	51.2 %	1.12	54.9 %	54223	56143	57098
Max2	10047	0.97	56.4 %	0.62	46.9 %	0.84	54.1 %	27619	28432	28501
Pipe	84292	1.31	69.1 %	1.02	56.2 %	1.35	70.1 %	309315	308266	309373
Micro	2575	1.27	62.1 %	0.88	54.7 %	2.70	75.4 %	8851	8515	8235
Stealth	5876	1.20	60.3 %	1.10	58.7 %	1.23	60.8 %	17753	17054	17040
Redox	77215	0.67	70.8 %	0.61	68.2 %	0.66	70.2 %	195412	192713	193192

Table 1: Technology Independent Optimization Results

Design	Gates	Delay conv	Delay reg	RI_p conv	RI_p reg	Lit conv	Lit reg	CPU reg
Bluechip	4669	4.15	4.15	1.28	1.32	16022	16010	-17 %
Solar	21838	1.32	1.32	1.03	1.05	55102	54811	-15 %
Max2	10453	2.20	2.21	0.80	0.82	27499	27380	-23 %
Pipe	100223	4.57	4.57	1.14	1.18	322499	322831	-29 %
Micro	3015	2.11	2.08	1.06	1.12	8419	8522	-19 %
Stealth	5992	3.52	3.52	1.02	1.02	18653	18701	-11 %
Redox	90982	6.33	6.30	0.62	0.65	210549	211431	-32 %

Table 2: Delay Optimization Results

contains several large regular groups, while the conventionally optimized design only features smaller and less useful regular groups.

Table 2 summarizes the results for the delay optimization of the technology mapped designs. Shown are the critical delay, physical regularity index and literal count of the conventionally and regularity driven optimization. The main goal of regularity driven optimization at this step is the improvement of run-time by reapplication of successful transformations to points with similar characteristics. On average, a run-time improvement of approximately 21 % has been shown, at identical or better delay and literal count. Figure 6 shows the final placement of design *Bluechip* utilizing the available regularity during datapath placement. All non-regular gates have been placed with a conventional placement algorithm. The placement of the regularity optimized design shows a significant amount of structural regularity that has been placed in rows and columns to achieve a higher density layout and shorter wiring length. In contrast, the conventionally optimized design cannot benefit from the same amount of regularity, as most of it has been destroyed during aggressive logic restructuring algorithms. The regularity optimized design, shown in Figure 6a, shows an improvement of more than 8% in critical delay over the conventionally optimized design, resulting from shorter wiring length. In addition, the packing density of the regularity driven optimized design is significantly higher, as shown in the final layout.

6 Conclusions and Future Work

The proposed logic synthesis methodology combines regularity information of the design with a driver-transform concept utilizing global design information to drive local transformations in the synthesis process. It has been shown that regularity driven synthesis is able to speed up the synthesis process and improve synthesis quality while maintaining design regularity. Due to the re-application of previously identified transforms to identical places in the network, more attention can be given to the careful selection of a good set of transforms to achieve higher optimization results. In addition, reuse of transforms yields a faster optimization process. Therefore, it is particularly suitable for the challenges imposed by the synthesis of very large designs. In addition, the preservation of regularity improves the final layout significantly.

Future work will focus on the addition of symmetry information to the synthesis process. The identification of symmetric decomposition functions in a design appears to be a promising addition to structural regularity extraction.

References

- [1] S.R. Arikati and R. Varadarajan. A signature based approach to regularity extraction. In *Proceedings of the International Conference on Computer Aided Design*, p. 542-545, November 1997.

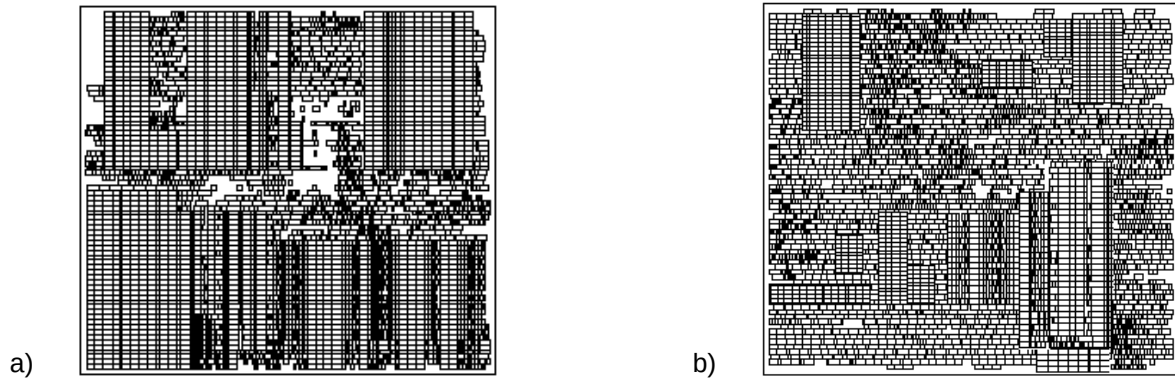


Figure 6: Actual placement of design *Bluechip*, a) *Regularity Driven*, b) *Conventional Optimization*

- [2] T. Chan et. al. Challenges of CAD Development for Datapath Design, In *Intel Technical Journal*, 01/1999
- [3] A. Chowdhary et.al. A general approach for regularity extraction in datapath circuits. In *Proceedings of the International Conference on Computer Aided Design*, p. 332-338, November 1998.
- [4] R. F. Damiano, A. Drumm et. al. Method for mapping in logic synthesis by logic classification, *U.S. Patent No. 5,537,330*
- [5] S.-T. Hui and D.M. Wong, Method for regular placement of data path components in VLSI circuits, *U.S. Patent No. 5,359,538*
- [6] V.N.Kravets and K.A.Sakallah , M32: A Constructive Multilevel Logic Synthesis System, In *Proceedings of the Design Automation Conference*, p. 336-341, June 1998
- [7] T. Kutzschebauch. Logic optimization using regularity extraction. In *Proceedings of the Internat. Workshop on Logic Synthesis*, p. 264-270, June 1999
- [8] Marshburn et. al. Datapath: A CMOS datapath silicon assembler. In *Proceedings of the Design Automation Conference*, p. 722-12, 1986
- [9] R.X.T. Nijssen and J.A.G. Jess, Two-dimensional datapath regularity extraction. In *Proceedings of the 5th ACM/IEEE Physical Design Workshop*, Reston, Virginia, 1996.
- [10] R.X.T. Nijssen and C.A.J. van Eijk. Regular layout generation of logically optimized datapaths. In *Proceedings of the International Symposium on Physical Design*, p. 42-47, 1997.
- [11] D.S. Rao and F.J. Kurdahi. On clustering for maximal regularity extraction. In *IEEE Transactions on Computer Aided Design*, p. 1198-1208, Aug. 1993
- [12] L. Stok, D. Brand, D. Kung et. al. Booleadozer: Logic synthesis for ASICs. In *IBM Journal of Research and Development*, 40(4), p. 515-547, July 1996