14

DataPath regularity extraction

R.X.T. Nijssen and J.A.G. Jess
Eindhoven University of Technology, Dept. of Electrical Engineering, Design Automation Section/ES, P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
R.X.T.Nijssen@ele.tue.nl

Abstract

An efficient method is introduced to extract datapath regularity from circuit netlists so as to speed up and improve placement of datapaths. A search-wave traverses the circuit constructing the datapath by choosing the most regular extensions as indicated by a local regularity index measuring the uniformness of the neighborhood of known datapath stages.

Keywords

Regularity Extraction, Datapath Placement, Layout Generation

1 INTRODUCTION

The inherent regularity of data-path circuitry can be exploited to obtain high density layouts. This can be achieved by aligning, ie. placing linearly, cells associated with the same bit-slices in one direction, and by aligning the identical (or highly similar) bit slices perpendicularly. This regularity comes from the use of bit-wise parallel operators at the architectural design level. These operators are implemented by repeating single-bit operator cells to form a multi-bit module. By interconnecting the corresponding bit-cells of different modules, a datapath is formed consisting of a number of highly similar bit*slices*. The cells within the same multi-bit module comprising a *stage* are identical and can hence be aligned. By imposing geometrical regularity upon the size of one side of the bit-cells in a slice, eg. they span the same number of rows in all stages, alignment in the other direction is also provided for. Figure 1a shows an abstracted representation of an ideal datapath aligned in 2-directions. Furthermore, datapath regularity is made up by the fact that the majority of nets in datapath circuitry is local to either one single slice or one single stage, so there are no or just a few nets that are connected to cells which are connected to both multiple slices and stages.

Combined exploitation of these two characteristics, alignability of cells in two perpendicular directions and coinciding perpendicular locality of nets, yields dense placements. The information on alignability consists of an assignment to one stage and, at the same time, one slice for every component. With this information, the task of generating a good relative placement for datapath circuitry in terms of density and wire length is decomposed from a general 2-dimensional placement problem of all cells of the datapath to one linear arrangement of the stages of only one single slice, and one linear arrangement of the slices of only one stage. These two orderings can be solved using eg. [GCT77][NKH+93] or [Asa82]. This complexity reduction follows directly from the properties of the connectivity and geometry of stages and slices. In particular, the arrangement of the stages within a slice does not depend

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

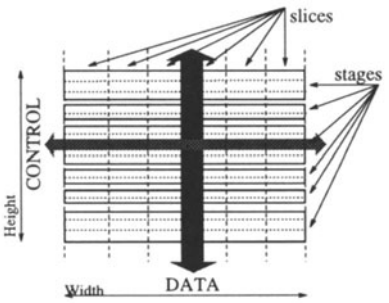

Figure 1 ideally aligned datapath

on the arrangement of the slices within a stage, hence these two tasks can be carried out separately. In addition, the latter ordering task is trivial in most cases. The total solution space of the problem is thereby reduced by many orders of a magnitude, allowing for the size of a datapath circuit to be placed in one run to be much larger than with conventional methods which have no explicit notion of regularity. The paradigms they are based on will fail to implicitly work towards perpendicular alignment of datapath cells, despite long run-times.

Special-purpose CAD environments, often called *data-path compilers* (DPCs), have been developed that do take advantage of these characteristics. Like generic tools, DPCs are incapable of extracting information from a circuit on its regularity. Thus there is a need for a general datapath regularity extraction algorithm.

The problem formulation is then as follows: Given a network, the problem is defined as tagging as many network entities as possible with both a slice tag and a stage tag. The goal is to determine this labeling such that these slices and stages describe a suitable bit-cell alignment in both directions. In other words, the individual slices should be as similar as possible.

Previous work on datapath regularity extraction is based on a regularity guided partitioning method presented in [OHN87], like [HS88][CH93]. None of these methods however are designed to extract complete datapaths, they rather form a number of datapath chunks called location macros, ie. clusters to be used as preplacement seed units for conventional placement tools. They do not determine slice and stage assignments to the contained cells. Only a few datapath regularity characteristics are used, and the extraction procedures wrongly identify certain common regularity aberrations as regular.

2 REGULARITY CHARACTERISTICS

Three basic categories of data path regularity characteristics are distinguished, namely *structural* ie. the minimal representation of the circuit entities and their connectivity, *functional* ie. annotations to circuit entities regarding their function or usage, eg. signal flow direction, the types or type equivalences of terminals and their purpose, eg. clock, etc. *explicit* ie. data (partially) specifying the data-path, eg. a group of nets tagged as a bus. In most cases, the basic structural data already provides sufficient information to accurately quantify the regularity of most data

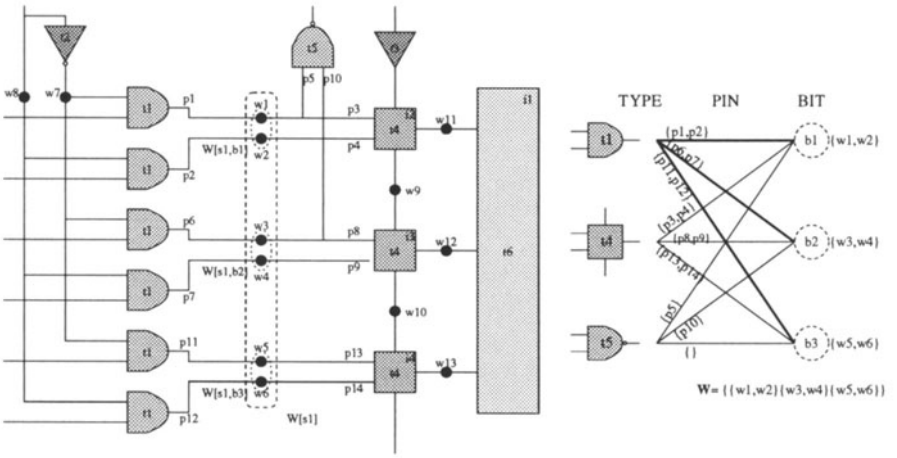

Figure 2 a)Part of a datapath b) Regularity Graph of a datapath

paths. This type of information is always available for any circuit, therefore we henceforth focus on it. The other categories are optional, but may be used to enhance the amount of detail so as to be able to distinguish between cases that are structurally identical. The structure of a circuit is completely defined by a set of module instances, the type of a module instance, a set of nets, and a set of pins, ie. the connectivity between modules and nets through specified terminals. The type of a wire is taken to be its degree.

The key idea of our modeling is that we observe the amount of regularity in the neighborhood of one already known datapath stage, the *vantage* stage. We call this *local relative datapath regularity*. A stage that consist of wires (instances) will henceforth be referred to as a *Wstage* (*Istage*). Consider figure 2b showing a part of an example datapath circuit. In this example, it is assumed that the wires $\{w_1, w_2\}$, $\{w_3, w_4\}$, and $\{w_5, w_6\}$ are each in a distinct slice, and that together, they form one stage s_1, denoted as $\{\{w_1, w_2\}, \{w_3, w_4\}, \{w_5, w_6\}\}$. Wstage s_1 is outlined in the figure. A candidate Istage, say s_2, is formed by the cells of type t_4.

The structural regularity characteristics used as criteria to measure the extent of regularity are the presence of:

W-T *multi-slice module types* like $T(i_2) = T(i_3) = T(i_4) = t_4$ relative to vantage Wstage s_1. Such module types occur across the entire width of the datapath, as seen from a certain vantage Wstage like s_1.

W-I *multi-slice modules* like instance i_1 if the vantage Wstage is $\{\{w_{11}\}, \{w_{12}\}, \{w_{13}\}\} = s_3$ These are modules that are adjacent to all slices of the vantage Wstage. Instances like i_1 may be a non-expandable block or a block possibly even containing another datapath.

I-T *multi-slice wire types* like $T(w_{11}) = T(w_{12}) = T(w_{13}) = 2$ relative to vantage Istage s_2. This is a wire type attached to all slices of the vantage stage.

I-W *multi-slice wires* like w_7. Such wires are common for all slices of the vantage Istage. This category mainly consists of control signals for all bits, like enable signals, function selectors etc, largely constitute this category.

For conciseness, we will henceforth focus on the first criterion W-T, viz. the presence of multi-slice module types. This explanation can straightforwardly be generalized to the other criteria.

3 DEFINITIONS

Let I denote the set of *instances*, W the set of *wires*, and $P \subset I \times W$ denote the set of *pins*. The bipartite graph $C = (I, W, P)$ is the usual network graph describing the structure of any circuit. Let $E = I \cup W$ be the set of circuit entities. The function $I : P \rightarrow I$ maps each pin to its connected instance, and $P : E \rightarrow 2^P$ maps each wire or instance to the set of pins through which it is connected. Note that the *function* is denoted as $I()$ unlike the *set* I, etc., and that for brevity of the notation, a function name may be overloaded unambiguously depending on its argument. Further, let T denote the set of circuit entity types. The function $T : E \rightarrow T$ maps each entity to its type. We define the type of a wire $w \in W$ as the number of its terminals, $T(w) = |P(w)|$.

A *stage* is a set of entities of the same type, and that are to be aligned in one direction. A *slice* is a set of entities that are to be aligned in a direction perpendicular to the stages. Ideally, a slice contains all nets corresponding with one bit of the datapath, and nets crossing slices are contained in one stage.

The assignment of circuit entities to stages and slices is denoted by labels. All entities associated to the same stage have the same *stage-label* being one of $S = \{s_1, s_2, ..., s_{|S|-1}, s_\infty\}$. Entities for which stage label is not known (yet) are labeled s_∞. Entities of the same slice have the same *slice-label* being one of $B = \{b_1, b_2, ..., b_{|B|-1}, b_\infty\}$. Entities for which a slice label is not known (yet) have slice label b_∞.

We use following functions mapping entities to stages or slices: The function $\varsigma : E \rightarrow S$ maps entities to stage labels. Note that by this definition of ς, an entity can be member of at most one single slice. Also recall that by definition of a stage, the types of all entities in the same stage are the same. The function $\beta : E \rightarrow B$ maps entities to slice labels. In the example of figure 2, $\varsigma(w_1) = s_1$ and $\beta(w_3) = b_2$.

With these definitions, the alignment of the components of a datapath can be fully described by β and ς. The goal of our regularity extraction method boils down to constructing a β and ς mapping defined for as many circuit entities as possible.

The functions ς and β are not injective, so they have no inverse. Instead, we define the function $\varsigma^{\leftarrow} : S \rightarrow 2^B$ of ς returning the set of slices of stage $s \in S$ as

$$\varsigma^{\leftarrow}(s) = \{b \in B | \exists\, e : e \in E : \varsigma(e) = s \wedge \beta(e) = b\} \tag{1}$$

Function $\beta^{\leftarrow} : S \times B \rightarrow 2^E$ returns the set of entities of which a slice $b \in B$ of stage $s \in S$ consists:

$$\beta^{\leftarrow}(s, b) = \{e \in E | \varsigma(e) = s \wedge \beta(e) = b\} \tag{2}$$

For example, in figure 2, wireslice $\beta^{\leftarrow}(s_1, b_1) = \{w_1, w_2\}$ is encircled by a dotted oval, and wirestage $\varsigma^{\leftarrow}(s_1) = \bigcup_i \beta^{\leftarrow}(s_1, b_i) = \{\{w_1, w_2\}, \{w_3, w_4\}, \{w_5, w_6\}\}$ is enclosed by the dashed frame.

We define the notion of local relative datapath regularity as the extent to which the appropriate criteria like W-T are present between a known stage s and one of its adjacent candidate stages. An *adjacent candidate stage* $s' \in S$ of s consists of a number of entities of the same type of which at least one is adjacent to an entity of s. We represent this adjacency with a *regularity graph*.

4 REGULARITY GRAPH

For every criterion, there is a relation between it and its stage s reflected by a complete bipartite graph:

$$\Gamma_{W-T}(s) = (\varsigma^{\leftarrow}(s), T, B \times T) \tag{3}$$

$\Gamma_{W-T}(s)$ has a vertex for every slice of stage s, and a vertex for every instance type of any adjacent instance. The type-vertices of graph $\Gamma_{W-T}(s)$ are obtained from C by removing all instances from it that are not adjacent to s, then collapsing all remaining instances of the same type onto one type-vertex. The edges $\epsilon \in B \times T$ of $\Gamma_{W-T}(s)$ are tagged with a set of pins realizing the respective adjacencies between the elements in a slice of s and a criterion. The attribute function $\gamma_{W-T}(s) : B \times T \rightarrow 2^P$ returns the set of pins between stage $b \in B$ and type $t \in T$:

$$\gamma_{WT}(s, \epsilon) = \{p \in P(w) | \epsilon = (b, t) \wedge w \in \beta^{\leftarrow}(s, b) \wedge T(I(p)) = t\} \tag{4}$$

Figure 2 shows the wirestage-to-celltype relation graph $\Gamma_{W-T}(s_1)$ for the stage s_1 of figure 2. All cell types of cells that are not adjacent to any wire in s_1 are omitted from the graph. Note that the fact that b_3 has no connection to a cell of type t_5 is represented by edge (b_3, t_5) having an empty associated set of pins.

5 RELATIVE REGULARITY METRIC

The proposed *relative local regularity metric* $\rho_{W-T}(s)$ is based on a statistical analysis of the cardinalities of the pin sets on the edges of every celltype node of $\Gamma_{W-T}(s)$. This metric is inversely proportional to the extent of regularity from wirestage s with respect to the types of its adjacent cells.

We denote the score vector made up of the cardinalities of the pin sets on the edges of node $t \in T$ of $\Gamma_{W-T}(s)$ as $X_{W-T}(s, t)$. Thus, the elements $x_{W-T}(s, t)[i]$ of $X_{W-T}(s, t)$ are defined as

$$x_{WT}(s, t)[i] = |\gamma(s, (b_i, t))| \tag{5}$$

In figure 2, the sizes of the pin sets on the edges of node t_1 are $X_{W-T}(s, t_1) = [|\{p_1, p_2\}|, |\{p_6, p_7\}|, |\{p_{11}, p_{12}\}|] = [2, 2, 2]$. The extent of regularity between vantage stage s and adjacent instance type t is obtained by interpretation specific statistical properties of $X_{W-T}(s, t)$. Ideally, $X_{W-T}(s, t)$ has a uniform distribution, corresponding with a maximally regular candidate stage of type t.

To quantify the extent to which the distribution of $X_{W-T}(s, t)$ matches a uniform one, we compute the *range* and the *symmetry* of X_{W-T}. If the range, defined as the difference between the largest and the smallest values of X_{W-T}, is 0, then X_{W-T} is uniformly distributed. If it is larger, but not too large, we look at the the skewness of the distribution of X_{W-T}, expressing the symmetry of $X_{W-T}(s, t)$. The skewness is given by the *third absolute central moment* μ_3 about the average of the vector. The n-th absolute central moment of a distribution about its average is defined as $\mu_n = \mathrm{E}[|X - \overline{X}|^n]$This measure increases as the symmetry decreases. A completely regular environment of s is reflected by $\mu_3 = 0$. The relative local regularity $\rho_{W-T} : S \times T \rightarrow \mathbf{R}$ from stage $s \in S$ relative to cell type $t \in T$ is defined as

$$\rho_{W-T}(s, t) = R(X(s, t))(\mu_3(X(s, t)) \tag{6}$$

This function is a suitable metric because $\rho_{W-T}(s) = 0$ implies maximal regularity, $\rho_{W-T}(s) = \infty$ maximally

irregular while the value of $\rho_{W-T}(s)$ increases monotonically as the regularity decreases. These properties allow comparison of the extent of regularity at different stages of the datapath and with respect to different types.

6 REGULARITY EXTRACTION ALGORITHM

The objective of regularity extraction is to maximize the number of circuit entities that have a stage label. Our regularity extraction algorithm works by expanding a search-wave through the network. The search-wave consists of stages that have already been discovered. The search-wave is initialized with one or more *seed* stages. In practice, these can be found easily eg. from attributes at the circuit's primary terminals, or they may be specified explicitly.

The search-wave is extended from one of its stages s with one suitable not yet discovered adjacent stage s', over its full width at a time. Extensions are chosen from a set of candidate extensions. This set is ordered by the value of the regularity metric associated with the respective each candidate extensions. This process is iterated until there are no more suitable wave extensions left. With every extension, the newly added stage is tagged with a new stage label from S. The slices of the new stage s' inherit the slice labels of the slices of s via the pins that realize the adjacency of s' and s. In case of maximally regular datapath circuitry, all entities have a unique stage label $\varsigma^{\leftarrow}(s_\infty) = \emptyset$ and the slices of every stage $s \in S$ have the same number of elements $\forall\, b : b, b' \in \varsigma^{\leftarrow}(s) : |\beta^{\leftarrow}(s,b)| = |\beta^{\leftarrow}(s,b')|$. The optimization goal f_c can thus be defined as "minimize $f_c = |\varsigma^{\leftarrow}(s_\infty)|$". The algorithm described here tries to achieve this by greedily selecting the most regular candidate extension known at any moment. Particularly when the circuit is highly regular many choices will be equivalent because they have the same regularity value.

In the algorithm outlined below, the wirestage seed of the search-wave is given as SEED. Priority queues are used for efficiency. Two different threshold values further control the expansion process. An upper limit ρ_{tr} filters candidate extensions leaving those that are sufficiently regular. A lower bound w_{tr} prevents the algorithm from selecting a candidate extension that is regular, but too narrow in terms of the number of slices. If the thresholds are set very tight, say $\rho_{tr} = 0$ and $\mathrm{w}_{tr} = |\varsigma^{\leftarrow}(\mathrm{SEED})|$, the algorithm will only find slices that are completely identical.

```
Stage s := s_0; Queue Q_{W-T} := {0, SEED}, Q_{I-T} = ∅;
ρ_{W,min}, P_W := Q_{W-T}.front(); ρ_{I,min}, P_I := Q_{I-T}.front();

analyze_wire_stage(s) { /* fill γ_{W-T} */
    γ_{W-T}[s,] := ∅;
    forall b ∈ ς^{←}(s) do
        forall e ∈ β^{←}(s,b) do
            forall p ∈ P(e) do
                γ_{W-T}[T(I(p)), b]∪ = {p};
    /* compute ρ_{W-T}(s,t) for all types */
    forall t ∈ T do
        forall b do
            P(s,b) := M[t, W(s,b)];
        if |P(s)| ≥ w_{tr} →
            ρ_{W-T}(s,t) := μ_3(P(s,b));
            if ρ_{W-T}(s,t) < ρ_{tr} → Q_{W-T}∪ = {ρ_{W-T}(s,t), P(s)} }

do{ s := new S;
    if ρ_{W,min} < ρ_{I,min} → mark_W(P_W, s); analyze_wire_stage(s);
    else mark_W(P_I, s); analyze_inst_stage(s);
    ρ_{W,min}, P_W := Q_{W-T}.front(); ρ_{I,min}, P_I := Q_{I-T}.front(); }
while (|Q_{W-T}| > 0) ∨ (|Q_{I-T}| > 0);
post_process();
```

The function **analyze_inst_stage** is analogous to **analyze_wire_stage**. The tagging procedure mark assigns the new stage label s, and the slice labels to the entities of s inherited via the pins in P. The main loop in the algorithm ends if there are no wave continuations with sufficient regularity left from sets of wire-strands or sets of instance-strands. The post-processing phase resolves undefined stage and slice labels for entities for which, looking back, a clear choice can now be made. This means that if for a module a slice and stage label is induced by its environment with limited ambiguity, and if the conditions for it being part of a datapath are met, it will be added to the datapath.

The algorithm has a run-time complexity of $\mathcal{O}(|P|)$ since every pin is analyzed at most twice. In practice, only pins in regular areas are considered, so actual run times are smaller.

7 RESULTS AND CONCLUSIONS

We implemented the algorithm in C++. The current implementation uses a more or less greedy strategy in that it favors wirestage-to-celltype wave expansions over others. The expansion process ends if all candidate expansions have been exhausted. Our results, which could not be included in this paper due to space limitations, show that the algorithm correctly extracts slices and stages from a number of circuits of different sizes and moderate irregularity. The results were evaluated by comparing the output of the extraction with explicit regularity information provided by module generators. The extraction performance is then given by the percentage of entities of which the extractor found the same stage/slice membership. The method has been applied to a number of large examples from various circuit synthesis environments such as a 8048 microprocessor core. In all cases, the datapath in the netlist was regained within a few seconds of CPU time. The threshold values proved to be an effective means to control the susceptibility for irregularity.

The datapath extraction algorithm presented in this paper enables datapath placement of circuits that were generated by general synthesis frontends. The algorithm is suitable for large circuits which are at least partially regular only to a some extent.

REFERENCES

Asa82] T. Asano. An optimum gate placement algorithm for mos one-dimensional arrays. *Journal of Design Systems*, 6(1):1–27, 1982.

CH93] C.E. Cheng and C.-Y. Ho. Sefop: A novel approach to data path module placement. In *Proceedings of the International Conference on Computer Aided Design*, pages 178–181. IEEE, Nov 1993.

GCT77] S. Goto, I. Cederbaum, and B.S. Ting. Suboptimum solution of the back-board ordering with channel capacity constraint. *IEEE Transactions on Circuits And Systems*, 24(11):645–652, Nov 1977.

HS88] M. Hirsch and D. Siewiorek. Automatically extracting structure from a logical design. In *Proceedings of the International Conference on Computer Aided Design*, pages 456–459. IEEE, 1988.

NKH+93] H. Nakao, O. Kitada, M. Hayashikoshi, K. Okazaki, and Y. Tsujihashi. A high density datapath layout generation method under path delay constraints. In *Proceedings of the Custom Integrated Circuits Conference*, pages 9.5.1–9.5.5. IEEE, 1993.

OHN87] G. Odawara, T. Hiraide, and O. Nishina: Partitioning and placement technique for cmos gate arrays. *IEEE Transactions on Computer Aided Design*, CAD-6(3):355–363, May 1987.