A Signature Based Approach to Regularity Extraction

Srinivasa R. Arikati*
Mathematical Sciences Department
University of Memphis
Memphis, TN 38152
arikati@cadence.com

Ravi Varadarajan
Cadence Design Systems
555 River Oaks Prkwy
San Jose, CA 95134
rrv@cadence.com

Abstract
Regularity extraction is an important step in the design flow of datapath-dominated circuits. This paper outlines a new method that automatically extracts regular structures from the netlist. The method is general enough to handle two types of designs: designs with structured cluster information for a portion of the datapath components that are identified at the HDL level; and designs with no such structured cluster information. The method analyzes the circuit connectivity and uses signature based approaches to recognize regularity.

1 Introduction
In a traditional design flow the circuit regularity may not be fully extracted. As a result certain gates that are strongly connected to the datapath by buses may be outside the datapath and hence viewed by the floorplanning tool as random logic. This results in inferior place and route results as these buses will have to peel out of the datapath region, connect to the gates in the random logic, and reenter the datapath region. Moreover, the structured placement information of these gates is not captured. In a typical top-down timing-driven HDL design flow for datapath-dominated designs, the datapath components are generated before the rest of the circuit is synthesized and optimized. Logic synthesis then operates on the datapath netlist along with the rest of the design, and synthesizes the control logic. During logic optimization new buffer gates may be introduced to meet timing requirements. Many of these buffer gates are between the datapath functions and should actually belong to the datapath. Moreover in some design flows, through behavioral and RTL synthesis, certain datapath components are not inferred correctly and hence are synthesized as random logic.

The methodology of Odawara, Hiraide, and Nishima [5] was one of the first approaches to regularity extraction. The method is based on the concept of a location macro, which is a group of gates of the same kind that share nets connected to the common input terminal. The location macros are extracted using the design information such as gate type, terminals, and logical hierarchy. Using these macros as placement initializers, the placement is refined by the subsequent standard cell placement. The papers [3, 4, 6] present improvements and modifications to the above method. Hirsch and Siewiorek [3] use flexible data structures and more structural characteristics (such as storage cells) to extract regularity. Nijssen and Jess [4] extend the work of [5] to extract 2-dimensional datapath regularity. Their work expands a search wave through the circuit, stage by stage; the stages are refined location macros. It uses local regularity metric to guide the search wave. Tsay and Lin [6] use different characteristics to extract regularity; they use primary outputs instead of gates to find strongly connected subcircuits called cones.

Compared to general placement, all the above methods lead to some improvements in area, wirelength and run-time. A major drawback of the approaches in [5, 3, 4] is that they all use only local information to extract regularity: they extract one location macro (or stage) at a time. The approach of [6] is not flexible in the sense that it uses only primary outputs to guide the extraction. Also, in all these approaches, the regularity may not be fully extracted. In our experiments, we have found that in most cases, extracting partial regularity produces poorer place and route results than extracting no regularity at all.

This paper outlines a new algorithm that automatically extracts regular structures. The algorithm is very flexible and is based on a generalization of the above-mentioned approaches. The algorithm analyzes the circuit connectivity and uses signature-based approaches to recognize regularity. The algorithm has been integrated into a Cadence HDL-based design flow. Experimental results demonstrate that our approach results in better designs (in terms of area and total wirelength), as compared to the designs obtained without regularity extraction.

Our method is general enough to handle the following types of designs:
(1) Designs in which a portion of the datapath components is identified at the HDL level, and the structured cluster information for these components exists in the netlist.
(2) Designs with no such structured cluster information. In this case, our approach automatically creates such information for datapath components. This is achieved by using hints from bus names and datapath features such as high-fanout control nets.

* This work is fully funded by a research grant from Cadence Design Systems, San Jose, CA.
The rest of the paper is organized as follows. Section 2 presents definitions and other preliminaries. Section 3 presents an algorithm for regularity extraction and a detailed explanation of the algorithm with an example. Some improvements to the algorithm are the topic of Section 4. Section 5 provides experimental results. Section 6 presents the concluding remarks.

2 Preliminaries

We work with the netlist representation of the input circuits. The netlist consists of cell instances of masters from a cell library, and nets. A net is connected to terminals of instances. Figure 2 shows a netlist, as explained in Section 3.2.

Relative placement information captures the structured cluster information of the datapath portion of the input circuit and is stored in tile files. In our flow, the tile files are automatically generated by the datapath generators during datapath synthesis. Each tile file is a group of cell instances that perform a datapath operation, like addition or multiplication. The tile file may be viewed as a column or as a matrix of cells. Each instance in a tile file has a bit number which indicates its relative placement (row number) in the group. In addition, instances in matrix tile files (say, a multiplier) have bit orders which indicate the column numbers.

A datapath function is an instance of a tile file. The size of a datapath function is the number of bits (i.e., rows) it spans. A cell instance is a datapath instance if it belongs to a datapath function; otherwise, it is a random instance.

A regular function is a set of random instances of the same master cell, along with bit numbers for the instances. The major distinction between datapath functions and regular functions is that the datapath functions are part of the input, while regular functions are part of the output.

Given a circuit and possibly its datapath functions, the problem of regularity extraction is to group random instances into regular functions such that the given objective function (area, wirelength, . . .) is optimized. When no input datapath functions are given, our approach automatically creates the initial datapath functions.

We use a signature-based approach to solve the regularity extraction problem. Our definition of signatures is inspired by the work of Ebeling and Zajicek on validating circuit layouts [2]. The signature of a random instance is dictated by its master cell and its connectivity to datapath instances. For example, consider the random instance A shown in Figure 1. It is connected to datapath instance P at terminal x and to datapath instance Q at terminal y. The signature of instance A is given by:

\[
\text{Sign}(A) = f(M(A), h(M(P), x), h(M(Q), y)),
\]

where \( f \) and \( h \) are properly chosen hash functions, and the function \( M \) maps cell instances to their master cells.

![Figure 1: Signature definition.](image)

In general, suppose a random instance \( A \) is connected to datapath instances \( P_1, P_2, \ldots \) at terminals \( x_1, x_2, \ldots \). The signature of instance \( A \) is

\[
f(M(A), h(M(P_1), x_1), h(M(P_2), x_2), \ldots).
\]

3 Regularity Extraction

As the input to the regularity extraction problem, we are given a circuit’s netlist and its datapath functions. Let MaxSize denote the maximum of the sizes of the datapath functions. The data structure for a random instance \( A \) consists of:

- the signature \( \text{Sign}(A) \), and
- array \( \text{CostArray}(A,i), 0 \leq i \leq \text{MaxSize} - 1 \).

The element \( \text{CostArray}(A,i) \) indicates the connectivity cost of assigning the random instance \( A \) to bit position \( i \).

Let random instances \( A_0, A_1, \ldots, A_{k-1} \) form a regular function. Suppose that the bit numbers of the random instances are, respectively, \( b_0, b_1, \ldots, b_{k-1} \). The cost of the regular function is \( \sum_{i=0}^{k-1} \text{CostArray}(A_i, b_i) \).

3.1 The Algorithm

Algorithm 1 presents a solution for the regularity extraction problem. The algorithm takes as input a parameter MinSize that indicates the acceptable minimum regular function size. An example is given below after the explanation of the algorithm.

If no datapath functions are provided as part of the input, the algorithm creates them by using hints from bus names and control nets. Step 1 of the algorithm initializes MaxSize by iterating through the given datapath functions. The computation of \( \text{Sign}(\cdot) \) and \( \text{CostArray}(\cdot, \cdot) \) in Step 2 is explained below.

Computing Signatures: A collision of signatures occurs if two random instances with different connectivity patterns get the same signatures. Proper choice of hash functions can almost always prevent collisions (one in one billion for 32-bit signatures). In practice, the \( \text{Sign}(\cdot) \) is an integer. All the arithmetic is done modulo a large prime number. The modulo arithmetic avoids overflow, and the selection of a large prime number prevents collisions. Our choice for hash functions \( f \) and \( h \), which has not resulted in collisions in all our experiments, is:

\[
f(a, b) = a + b, \quad h(a, b) = a * 2^b.
\]

Computing Connectivity Costs: The computation of \( \text{CostArray}(\cdot, \cdot) \) is flexible in the sense that the connectivity costs can vary depending on our objective. For example, suppose a net connects a random instance \( A \) to an instance \( P \) in a datapath function. Let the bit number of \( P \) be \( b \). The contribution of this net to \( \text{CostArray}(A, i) \) reflects the “vertical distance”
Output: A collection of regular functions.

1. Compute MaxSize: RegularFunctionList := \emptyset.
2. Compute Sign(.) and CostArray(.,.) of each random instance.
3. Partition all random instances into blocks of instances with same Sign(.)
4. For each block \mathcal{A} do
   While \mathcal{A} \neq \emptyset do
     (a) Compute a minimum cost regular function \mathcal{F} of the maximum size.
     (b) If the size of \mathcal{F} \geq \text{MinSize}, add \mathcal{F} to RegularFunctionList.
     (c) Delete the instances of \mathcal{F} from block \mathcal{A}.

Algorithm 1: An algorithm for regularity extraction.

The netlist of Figure 2 has the following components.

Instances: A generic instance \(X_i\) is a cell instance of a library cell \(X\). So, \(P_5, P_1, P_2,\) and \(P_3\) are all instances of a library cell \(P\), and \(C_0\) and \(C_1\) are instances of a cell \(C\). The important terminals are shown as small rectangles inside the instances, i.e., instance \(A_3\) has terminals \(x\) and \(y\).

Datapath Functions: Instances \(P_0, P_1, P_2,\) and \(P_3\) form a datapath function \(F_1\). The bit numbers of these instances are, respectively, 0,1,2,3. Similarly, instances \(Q_0, Q_1, Q_2,\) and \(Q_3\) form a datapath function \(F_2\). These eight instances are datapath instances, and the remaining instances are random instances. \(F_1\) and \(F_2\) might be given as part of the input, or they might be generated by the previous iteration of the algorithm.

During the signature computation, all the instances \(A_0, A_1, A_2,\) and \(A_3\) get the same signature, because they have similar connectivity to datapath instances. Also, all the instances \(B_0-B_5\) get the same signature, which is different from \(A_3\)'s signature. Instances \(C_0\) and \(C_1\) don't get any signature, since they are not connected to datapath instances. So, there are two signature blocks, namely block \(\mathcal{A}\) consisting of all \(A_i\)'s, and block \(\mathcal{B}\) containing all \(B_j\)'s.

In CostArray(.,.) computation, MaxSize = 4. Since \(A_3\) is connected only to bit 3, the element CostArray(\(A_3, 3\)) is 0, and the array CostArray(\(A_3, .\)) = (3,2,1,0). Similarly, CostArray(\(B_2, .\)) = (2,1,0,1), as \(B_2\) is connected only to bit 2.

Consider now the generation of regular functions from the block \(\mathcal{A}\). The minimum cost maximum matching algorithm applied to this block finds one matching: instances \(A_0, A_1, A_2,\) and \(A_3\) are matched to bit positions 0,1,2,3 respectively. So, a new regular function \(F_3 = \{A_0, A_1, A_2, A_3\}\) is generated, and the bit number of \(A_i\) is \(i\). The matching algorithm applied to the block \(\mathcal{B}\) generates two regular functions, namely, \(\mathcal{F}_4 = \{B_0, B_1, B_2, B_3\}\) and \(\mathcal{F}_5 = \{B_4, B_5\}\).

4 Or, \(\mathcal{F}_4 = \{B_0, B_1, B_4, B_5\}\) and \(\mathcal{F}_5 = \{B_2, B_3\}\).
of $B_4$ and $B_5$ are 2 and 3, respectively. This completes the first iteration.

In the second iteration, there are five datapath functions, namely, $F_1$–$F_5$, and two random instances $C_0$ and $C_1$. These random instances are extracted as a regular function $F_6$, and the bit number of $C_i$ is $i$.

The final circuit is shown in Figure 3.

Figure 3: Extracted regular functions from the circuit of Figure 2.

4 Relaxed Signatures

For some circuits, the signature definition given in section 2 may be very restrictive: two random instances, whose connectivities to datapath instances differ only slightly, will get distinct signatures. Consequently, during the generation of regular functions, these two instances will belong to different functions. To overcome this problem, we relax the signature definition.

We may view the previous signature as an instance-based signature. We propose a function-based signature below.

Suppose a random instance $A$ is connected to instances in datapath functions $F_1, F_2, \ldots$. The function-based signature of the instance $A$ is given by:

$$\text{Sign}(A) = f(M(A), F_1, F_2, \ldots).$$

The improved algorithm is shown in Algorithm 2.

Algorithm 2: Improved algorithm for regularity extraction.

1. Apply Algorithm 1 with instance-based signatures.
2. Use function-based signatures and apply Algorithm 1 to all random instances that don’t belong to the regular functions generated in the previous step.

5 Experimental Results

Table 1 shows the area and total wire length results on final (after routing) designs. Tst1, Tst2, and Tst3 are designs without regularity extraction; Tst1Ext, Tst2Ext, and Tst3Ext are obtained from these designs using extraction. Tst1Ext has 21% less wirelength than Tst1. Tst2Ext has 15% less wirelength than Tst2. Tst3Ext has 43% less area and 50% less wirelength than Tst3.

<table>
<thead>
<tr>
<th>design</th>
<th>cells</th>
<th>nets</th>
<th>area</th>
<th>wirelength</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tst1</td>
<td>1318</td>
<td>1126</td>
<td>1190x443</td>
<td>242215</td>
</tr>
<tr>
<td>Tst1Ext</td>
<td>1318</td>
<td>1126</td>
<td>1192x443</td>
<td>189570</td>
</tr>
<tr>
<td>Tst2</td>
<td>883</td>
<td>977</td>
<td>414x707</td>
<td>99953</td>
</tr>
<tr>
<td>Tst2Ext</td>
<td>883</td>
<td>977</td>
<td>431x691</td>
<td>84883</td>
</tr>
<tr>
<td>Tst3</td>
<td>3724</td>
<td>4279</td>
<td>18204x1768</td>
<td>38300000</td>
</tr>
<tr>
<td>Tst3Ext</td>
<td>3724</td>
<td>4279</td>
<td>7512x24804</td>
<td>19200000</td>
</tr>
</tbody>
</table>

Table 1: Experimental Results.

6 Conclusion

We presented a general and flexible approach to extract regularity from datapath-oriented circuits. The approach is signature based and analyzes the circuit connectivity. The experimental results demonstrate that the approach results in better designs, as compared to the designs obtained without regularity extraction.

Acknowledgement: We thank Raymond Nijssen for his valuable suggestions on this paper; he suggested to us to use high-fanout control nets to generate initial datapath functions.

References