Layout Synthesis for Datapath Designs*

Naveen Buddi, Malgorzata Chrzanowska-Jeske
Dept. of Electrical Engineering
Portland State University, P.O.Box.751
Portland, OR 97207. email:jeske@ee.pdx.edu

Charles L.Saxe, Vince Ast
Tektronix Inc.
P.O.Box 500, M/S 47-536
Beaverton, OR 97077

Abstract

DPLAYOUT is a layout synthesis tool for bit-sliced datapath designs targeting standard-cell libraries. We developed fast and efficient heuristics for placing the cells in a bit-slice, such that the regularity of datapath circuits is preserved and the number of channels occupied by a control signal is minimized. In addition, we propose a novel window-based heuristic for global routing of multi-pin nets. VHDL interface makes DPLAYOUT a general tool which can be easily integrated with any high-level synthesis system. This paper describes the heuristics developed for placement and global routing of a single bit-slice. We compared the area and run-time efficiency of the proposed heuristics with conventional methods and the results show a significant improvement.

Key words: placement, routing, layout, channel, datapath, bit-slice.

1. Introduction

Datapath logic of microprocessors and digital signal processing circuits contributes significantly to the overall chip area. This logic is regular and often implemented by connecting several bit-slices in parallel. In custom circuits the layout of datapaths is generated by replicating the layout of one bit-slice [1]. In cell-based designs, usually *datapath compilers* are used to generate the layouts. To achieve better performance the compilers implement the datapath logic using cells from specialized (datapath) libraries [2,3]. These specialized libraries allow abutment of the leaf cells in a bit-slice, thus resulting in compact layouts. However, the development and maintenance of specialized libraries requires an extra effort and causes cost overhead. In addition, not all vendors supply these specialized libraries. In such cases, the CAD tools use standard-cell library elements to implement the datapath logic. These tools use traditional placement techniques [4] which ignore the regularity present in the datapath designs. Consequently, the resulting layouts are not area efficient. This has necessiated the development of new techniques for datapath module placement in standard-cell based designs.

In the past, different techniques [5-8] have been proposed to solve the datapath layout generation problem. The datapath compiler in LAGER [5] assumes datapath libraries, and uses an extension of the Kernighan-Lin partitioning algorithm to solve the placement problem. Other techniques [6-8] assume standard-cell libraries. Luk and Dean [6] use a multi-stack approach in which the datapath logic cells are partitioned and assigned to several stacks with the objective of minimizing the number of nets crossing a stack. The placement within a stack is determined such that the misalignment of macros and a number of vertical wiring tracks used for routing, are minimized. However, they did not use the regularity present in the datapaths. Wu and Gajski [7] used the regularity property of datapath designs, but concentrated only on partitioning the RTL netlist to generate RTL-component layout based on bit-sliced architecture. In [8], the authors extract similarity among several bit-sliced components to form macros and solve the problem as the placement of the extracted macros. The authors did not effectively use the repetitive nature of bit-sliced datapath designs by considering several bit-slices at a time. Also their cell-matrix approach for placement leaves some empty slots in the matrix. The area wastage due to these empty slots is more significant when the macros have non-uniform width.

In our work, we solved the datapath layout problem using standard-cell libraries by exploiting the bit-sliced nature of datapath designs. While performing the placement and global routing, we consider one bit-slice at a time instead of considering all the bit-slices together. This information is then used to obtain the complete layout of the design. Theoretically, our approach is faster than the recent work reported in [8] because we reduce the size of the problem by considering a single bit-slice at a time. This is more important when the number of leaf cells in a bit-slice is large which is typically the case in

* This work was partially supported by Tektronix Inc. Grant GA455-01.

designs where large number of datapath functional blocks are stacked together. While placing the cells in a bit-slice, we attempt to route each of the control signals in minimum number of channels and preserve the dataflow present in the design. For global routing of multiple nets we propose a *window-based* heuristic which to the best of our knowledge has not been applied to the global routing problem before (see [9] for references). This technique generates an efficient global routing without rip-up-rerouting of nets and can be easily integrated into Spanning/Steiner-tree [9] based global routing algorithms.

The remaining part of the paper is organized as follows. Section 2 gives the problem description and the layout model used in our work. In section 3 we describe the overall approach used in DPLAYOUT. The placement and global routing techniques are explained in sections 4 and 5 respectively. Finally, in section 6 we discuss the performance of DPLAYOUT when applied to a set of datapath designs.

2. Problem description and layout model

We assume that a datapath circuit is designed using standard-cell library components (leaf cells). Given a bit-sliced datapath design, the problem is formulated as placing the leaf cells in the bit-slices and routing them such that no two leaf cells overlap and the area occupied by the final layout is minimized. The layout model used is a single stack of rows with leaf cells placed in rows. We assume that all data signals enter from the top of the stack and leave to the bottom of the stack. The control signals enter from the left side of the stack and leave to the right side of the stack. Fig.1 shows the layout model used in DPLAYOUT.

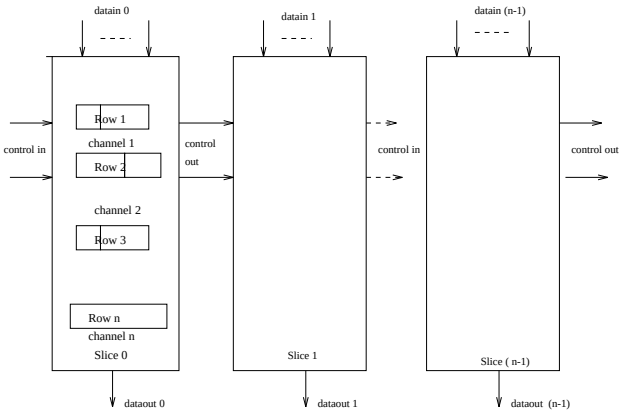


Fig.1. Layout Model of DPLAYOUT

3. Overall approach

The overall approach used in DPLAYOUT is shown in Fig.2. The design is described in the form of a hierarchical VHDL netlist. A bit-slice is described as an *entity* in VHDL, and the datapath is described by instantiating these entities. This interface to VHDL makes DPLAYOUT a general tool which can be integrated with any high-level synthesis system.

Each of the bit-slices is constructed from a set of primitive cells present in the target standard-cell library. We first analyze the netlist and classify the bit-slices into different types. Two bit-slices are treated as different types if they consist of different sets of leaf cells or connectivity among the leaf cells varies, or both. Next, the placement and global routing for each of the bit-slice types is performed.

A datapath is constructed by connecting several bit-slices in parallel. We extract the bit-slice order (the order in which the bit-slices are connected) from the netlist and bit-slices are abutted in that order. Finally, the nets within a channel are routed using greedy channel routing technique [10]. The output of DPLAYOUT is a CIF file containing the layout information.

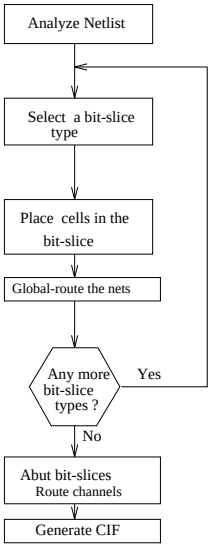


Fig.2. DPLAYOUT overall approach

4. Placement

The area of the datapath layouts is minimized by preserving the dataflow during the placement and by minimizing the number of channels in which a control signal is routed. Usually, in datapath circuits the same control signal is connected to several leaf cells. Therefore, to maintain the dataflow a control signal has to be assigned to more than one routing channel (multiplication of control signals). The routing of a control signal can be limited to minimum number of channels (ideally to one) by placing all the cells connected to that control signal in the same row or in two adjacent rows. However, this may violate the dataflow. Hence while generating the layout, we

need to meet two objectives: preserving the dataflow and minimizing the multiplication of control signals (MCS). The placement heuristic proposed here attempts to meet both objectives, and hence minimize the area. The steps involved in the placement heuristic are described below.

a) Regularity preserving classification of control signals :

In the case of the bit-sliced structure, control signals propagate through several slices. This propagation can be of two types, *direct* and *indirect*, as shown in Fig.3(a). The first type occurs when a control signal is connected to leaf cells belonging to different slices (eg. clock signal). The second type of propagation occurs when an output control signal of one bit-slice is connected as an input control signal to the next bit-slice. For example, in an adder, the carry-out of one bit-slice is connected as carry-in of next bit-slice. We classify the control signals into two categories, *related* (control signals involved with indirect propagation) and *unrelated* (control signals other than *related* control signals). In Fig.3(b) C_{in} and C_{out} of slice-0 are related signals. We identify the *related* control signals within a bit-slice type. These *related* control signals are treated as one signal during initial stages of the placement. DPLAYOUT also allows the designer to specify a set of control signals to be treated as *related.*

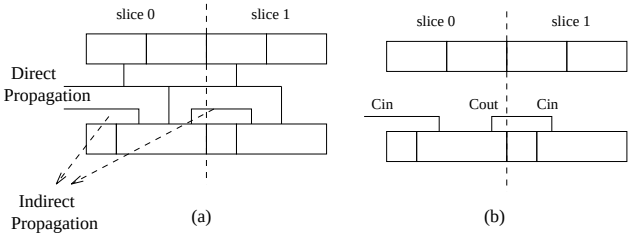


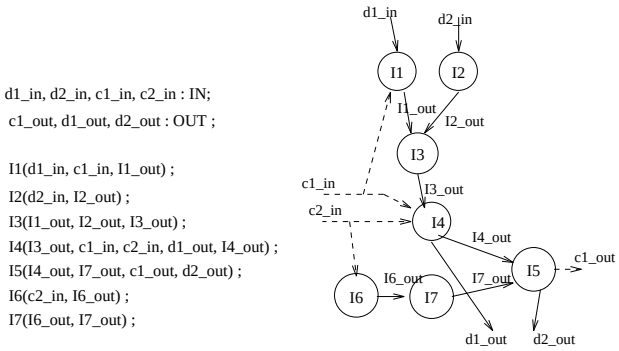
Fig.3. (a) control signal propagation (b) Related Signals C_{in}, C_{out} assigned to same channel

b) Classification of the leaf cells :

The leaf cells in the bit-slice are classified into groups. All leaf cells that are connected to the same control signal or to *related* control signals are assigned to one group. All other leaf-cells are assigned to unique groups. The purpose of this classification is to place leaf cells using the same control signal in the same row or in two adjacent rows. For each of the control signals, we identify all the leaf cells connected to that signal. A leaf cell might appear in more than one group. This occurs when a leaf cell is connected to more than one control signal. The placement of such cells is discussed in the subsequent paragraphs. The classification of leaf cells allows us to assign the *related* control signals to one channel as shown in Fig.3(b), thus minimizing the routing area.

c) Row assignment :

The leaf cells classified into groups are assigned to rows using the following technique. Represent the bit-slice netlist as a directed graph such that nodes represent leaf cells and edges represent the nets (signals). The direction of an edge represents the direction of the signal flow. Fig.4(a) shows a sample netlist and Fig.4(b) its graph representation. In the following description of the proposed heuristic, we use node and leaf cell as synonyms.



cell	row	phase
I1	1	1
I2	1	1
I3	2	1
I4	3	1
I5	4	1
I6	3	2
I7	3	3

(c)

Fig.4. Row Assignment (a) Input netlist (b) Graph representation (c) placement result

The row assignment heuristic involves three phases. In the first phase, the graph is traversed in a breadth-first manner, starting from the leaf cells connected to data-input signals. We assume that the order of data-input signals is specified by the user. The edges representing the control signals are not considered during the traversal. For each of the node (leaf cell) visited, the current level represents the row in which the leaf cell has to be placed. In otherwords, leaf cells connected to data-input signals are assigned to row one (top row). Leaf cells within a group have to be assigned to the same row or to two adjacent rows. Whenever the first leaf cell from a group is encountered during the traversal, then the channel below the row corresponding to the current level is assigned as the channel number of the group. Then all control signals connected to the cells in that group are assigned to that group's channel. If the group the visiting node belongs to has already been assigned to a channel, then the visiting node is placed in a row below that channel if minimization of MCS has more weightage than data flow preservation. If the data flow preservation has more weightage then it is placed in a row corresponding to the current level. When a leaf cell appears in more than one group, the row assigned to that leaf cell is the channel number of

the first group encountered during the graph traversal.

In the second phase, we place all the unplaced leaf cells, which are connected to the control signals. We repeat the following procedure for each of the control signals which have been assigned to a channel. For all the unplaced leaf cells connected to the control signal, assign the row directly below the control signal's channel. In the third phase, we place all the remaining unplaced leaf cells. In this phase, the graph is traversed in depth-first manner starting from the data-output signals. Unplaced leaf cells are assigned to rows using the same row assignment technique as in the first phase. The depth-first traversal backtracks when we encounter any of the input signals (control/data) or placed leaf cells and terminates when all the leaf cells are placed. This phase is required only if there are feedback signals in the given circuit. Fig.4(c) shows the row assignment for the sample netlist in Fig.4(a). The assigned rows are shown in the second column and the row assignment phase is shown in the last column. Analysis of the row assignment heuristic shows that in the first phase all the cells which constitute the dataflow (data-in to data-out) are placed. In the second phase any glue logic associated with the control signals is placed and in the last phase cells in the feedback loop are placed.

This graph based technique preserves the data flow and the grouping of cells minimizes the number of channels used by each control signal, thus the circuit layout area is minimized. The relative position of leaf cells in a row is determined by the order in which the leaf cells are assigned to that row.

d) Row merging :

In this step, some of the rows are merged in order to maintain user specified aspect ratio. However, this step is not trivial because merging of rows may violate the above described minimization of MCS objective. At present we only allow merging of complete rows so that the control signals in the channels adjacent to the merged rows need not be multiplied.

5. Global Routing

After placing the leaf cells of a bit-slice, the nets are routed using the two-stage routing approach. In this paper, we propose a new *window-based* routing technique which has been integrated with Minimum Spanning Tree (MST)[11] based global routing algorithm.

Window-based routing :

Instead of routing a net completely, we route only part of the net and defer the routing of the remaining part. To determine which part of the net to route first, we define a parameter called *window.* A *window* is a rectangular region with constant height and variable width. At the beginning of the global routing, the window width is set to a range 0 - W_s, where W_s is the window width specified by the user. The window height is always fixed and includes all rows of the bit-slice layout. For all the nets which originate in the current window, MST is constructed. Then for each net, only those edges in its MST that terminate in the current window are routed. The window is moved from left to right until all the nets are routed. The advantage of the window-based routing technique is evident from the following example. Fig.5(a) shows the path of a net *N.* When the feed-through assignment for nets routed after *N* results in the cell movement in row 2, the path of the net *N* in Fig.5(a) is disturbed as shown in Fig.5(b). Since we are not rerouting the nets disturbed by the cell movement, the overall global routing solution will not be efficient. The above described *window-based* technique improves the routing quality because the net in Fig.5(a) will not be routed until its end point is visible within the window. Fig.5(b)-(c) show net *N* route after a cell is moved for two different window sizes. The *window-based* technique results in better quality routing than the one obtained without this technique.

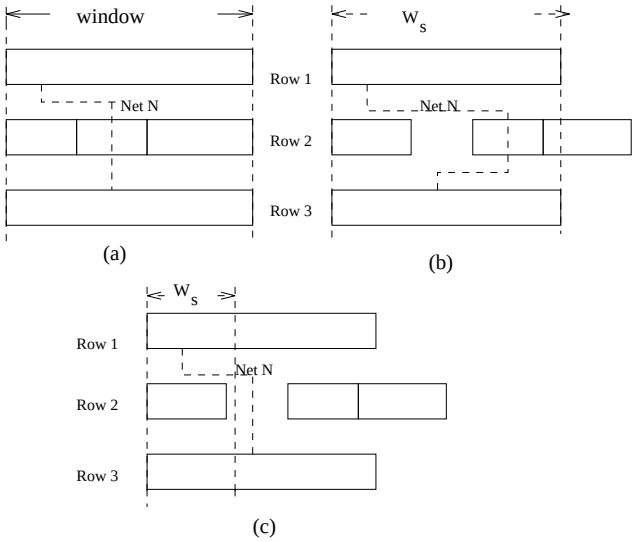


Fig.5. (a) Initial net route (b) Net route after a cell is moved for large window (c) Net route after a cell is moved for small window

6. Results

DPLAYOUT is implemented in C under UNIX environment. We conducted experiments to evaluate the run-time and area efficiency of DPLAYOUT. We compared the results of DPLAYOUT with a standard cell Place and Route tool (*scr*) in the *ALLIANCE CAD* package [12] for a set of data path designs. Tables I and II show the design statistics and results, respectively.

Ex1 is a bit-slice of an *adder-accumulator*. Ex2 and Ex3 are single bit-slices of 8x16 bit *fifo* and 4-bit *ram* circuit, respectively. All the results were obtained using the complete row-merging heuristic. The total CPU time reported is the combined time for input parsing, placement, routing and layout file generation. For all the tested circuits, the area and run time of DPLAYOUT is better than that of *scr*. We also compared the efficiency of the bit-slice based layout generation approach with the non-bit slice based layout generation approach. All the bit-slices of the above 8x16 bit *fifo* and 4-bit *ram* are submitted (Ex4 and Ex5 respectively) to DPLAYOUT and *scr* as one bit-slice. Considering the fact that area of the datapath circuits is proportional to the number of bit-slices, the area of the complete circuit must be close to n-times the area of single bit-slice, where n is the number of bit-slices in the circuit. The same should be true of the total CPU time. However, for both DPLAYOUT and *scr*, the total time and area in Ex4 and Ex5 are more than n-times the time and area of Ex2 and Ex3, respectively. This experiment demonstrates that for datapath circuits, bit-slice based layout generation approach has better area and run-time efficiency over non-bit slice based layout generation approach. The results of Ex4 also show that even when the circuit is not partitioned into bit-slices, DPLAYOUT outperforms *scr* for more regular datapath circuits (*fifos, register files* etc.). However, traditional methods won over our algorithms when the datapath circuits have more random logic associated with them (Ex5).

Conclusions

In this paper we describe an efficient and fast approach for generating layouts of bit-sliced datapath circuits designed using standard-cell libraries. We developed efficient dataflow preserving heuristics for placement. The placement heuristics exploit the regularity characteristic of datapath designs and attempt to route a control signal in minimum number of channels. The *window-based* global routing technique proposed here gives efficient routing without any rip-up rerouting. We also demonstrated that for standard cell based datapath circuits, we can achieve efficient layouts, when the circuits are partitioned into bit-slices and the bit-slices are handled separately. Currently, we are working on improving the row-merging algorithm. As a future work we would like to investigate the possibility of using our approach to solve the Register-Transfer level component layout generation problem inorder to achieve better datapath layouts.

Acknowledgements

We would like to thank the *ALLIANCE CAD* team, for their public domain software. We also would like to thank the anonymous reviewers for their critical comments.

References :

[1]. K. Usami et.al., "Hierarchical Symbolic Design Methodology for Large-Scale Data Paths", in *IEEE Journal of Solid-State Circuits,* Vol. 26, No. 3, pp. 381-385, March 1991.

[2]. R. C. Mason and M. T. Fertsch, "A Bit-modular Cell Library Optimized for Datapath Applications", in *Proc. of ISCAS*, pp. 10-14, 1986.

[3]. Y. Tsujihashi et.al., "A High-Density Data-path Generator with Stretchable Cells", in *IEEE Journal of Solid-State Circuits*, vol. 29, No. 1, pp. 2-7, Jan.1994.

[4]. C. Sechen and A. Sangiovanni-Vincentelli, "The Timber-Wolf Placement and Routing Package", *IEEE J.Solid-State Circuits*, vol. 20, pp. 510, April 1985.

[5]. C. B. Shung et.al., "An Integrated CAD System for Algorithm-Specific IC Design", in *IEEE Trans. on CAD*, vol. 10, no. 4, pp. 447-462, April 1991.

[6]. W. K. Luk and A. A. Dean, "Multistack Optimization for Datapath Chip Layout", in *IEEE Trans. on CAD*, vol. 10, no. 1, pp. 116-129, Jan. 1991.

[7]. A. C. H. Wu and D. D. Gajski, "Partitioning Algorithms for Layout Synthesis from Register-Transfer Netlists", in *IEEE Trans. on CAD*, vol. 11, no. 1, pp. 453-463, April 1992.

[8]. C. E. Cheng and C. Ho, "SEFOP: A Novel Approach To Datapath Module Placement", in *Proc. of ICCAD*, pp. 178-181, Nov. 1993.

[9]. W. Swartz and C. Sechen, "A New Generalized Row-Based Global Router", in *Proc. of ICCAD*, pp. 491-498, 1993.

[10]. R. L. Rivest & C. M. Fiduccia, "A Greedy Channel Router", in *Proc. of Design Automation Conf.*, pp. 256-262, 1982.

[11]. N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall International Inc. 1974.

[12]. *ALLIANCE CAD system-2.0*, Laboratoire MASI/CAO-VLSI, Universite Pierre et Marie Curie, PARIS, FRANCE.

Table-I Summary of Design Statistics

design	#cells	#nets
Ex1	9	17
Ex2	16	35
Ex3	58	107
Ex4	128	154
Ex5	232	284

Table-II Summary of DPLAYOUT results

	DPLAYOUT			*scr*		
design	*place time*[&] (sec.)	*total time*[&] (sec.)	area (sq.mm.)	*place time*[&] (sec.)	*total time*[&] (sec.)	area (sq.mm.)
Ex1	0.01	0.32	0.02	0.5	1.87	0.03
Ex2	0.01	0.6	0.057	0.59	3.0	0.08
Ex3	0.03	7.1	0.18	2.4	7.75	0.25
Ex4	0.03	12.8	0.6	5.3	46	1.35
Ex5	0.07	94	1.45	10.5	54.25	1.37

[&] CPU Time measured on SUN SPARC-2 workstation.