PHYSICAL SYNTHESIS FOR ASIC DATAPATH CIRCUITS

Terry Tao Ye, †Samit Chaudhuri, †Felix Huang, †Hamid Savoj, Giovanni De Micheli

Computer Systems Laboratory, Stanford University, USA {taoye, nanni}@stanford.edu
†Magma Design Automation, Cupertino, CA 95014 †{samit,fhuang,hamid}@magma-da.com

Abstract

This paper presents a physical synthesis methodology for ASIC datapath modules. It exploits the regularity information of datapath circuits and integrates the synthesis and placement processes together. This work is distinctive in the following aspects: (1) It works very well with datapath designs implemented in ASIC, where the datapath circuits are mostly semi-regular, therefore cannot be placed in a strict bit-sliced fashion. (2) it integrates physical planning with the synthesis process, so relative locations of the cells follow the bit order and the dataflow order. Experiments performed within a commercial framework show that this methodology improves the quality of datapath placements, and produces better post-route layouts for the datapath modules. This methodology narrows the performance gap between automatically generated ASIC datapath modules and manually constructed high-performance datapath circuits.

1. INTRODUCTION

As more complicated data processing operations are implemented on silicon, datapath is becoming the critical component of today's ASIC designs. However, manually constructed datapath circuits in custom ICs usually outperform the automatically constructed datapath circuits in ASICs. Bridging this performance gap is crucial for high-performance datapath designs.

In custom IC designs, the datapath circuits are normally partitioned into dedicated blocks on the floorplan. The designers manually construct each datapath circuit and place it in a bit-sliced style. This design approach reduces the timing skews between different bits, and accurately predicts the loading of individual nets [1][2][3]. Compared with the custom datapath design approach, ASIC datapath design flow has the following two limitations:

1) Datapath synthesis and placement are isolated. The datapath synthesis tool has no knowledge of how the circuit might be placed, so the load and timing of the interconnect cannot be estimated accurately. The placement process has no knowledge on the regularity information from the datapath circuit, so the layouts of datapath modules are often sub-optimal.

2) Datapath is not fully regular. Datapath design implemented in ASIC often contains circuits that are not fully regular. This will make the datapath physical synthesis problem even more complicated because such datapath circuits do not have a strictly bit-sliced structure, and cannot be placed into a regular array of rows and columns.

Previous work on datapath synthesis and generation focus either on the datapath synthesis process (pre netlist generation), or on the placement process (post netlist generation), namely in the following areas:

1) Datapath layout generation [4][5][6]: Datapath is optimized at transistor level, and cannot be linked directly to standard-cell based ASIC design processes.

2) Datapath placement optimization: Research has been done on reduction of track density [7][8][9], or minimization of congestion and total wire length [10][11] of datapath circuits. All of them are stand-alone procedures which operate on a post-synthesis netlist where regularity information has been partially lost.

3) Datapath placement with regularity awareness [12]: This methodology works best on a fully regular datapath circuit, which can be abstracted into one-bit abstraction model and further optimized for wire length and congestion minimization. However, as analyzed later in this paper, most ASIC datapath circuits are not strictly regular, where bit-sliced structures are often hard to extract.

This paper addresses the issues of ASIC datapath physical design, and presents a novel physical synthesis methodology where datapath synthesis and placement are tightly coupled. This methodology uses higher-level dataflow information that is present only at the register-transfer level (RTL) but not present in post-synthesis netlists. Furthermore, this technique is robust and can handle semi-regular datapaths that more commonly occur in ASIC designs.

The paper is organized as follows: Section 2 discusses the sources of irregularities in a semi-regular datapath. A broader notion of regularity, called semi-regularity, is introduced in Section 3. Section 4 introduces the idea of clusters, and section 5 describes how clusters generation and placement can be coupled with datapath synthesis process. Experimental results are presented in Section 6, and the summary in Section 7.

2. SEMI-REGULARITY OF DATAPATH CIRCUITS

In ASIC design flows, datapath circuits are seldom fully regular. Most often, they are semi-regular and contain some irregularities caused by the following sources:

1) Operational Irregularities When input operands are not of the same width, or datapath width varies as operations proceed, the bit sliced structure can no longer be preserved for every bit and the layout becomes non-rectangular.

2) Architectural Irregularities Some datapath architectures are not regular, such as Wallace trees, or certain carry look-ahead trees. These architectures do not have a bit-sliced structure, and cannot be placed regularly.

3) In/Out Interface Irregularities

When input bits have skewed arrival time, or output bits have different loads, cells performing the same operation on different bits may get sized and optimized differently. This also destroys the uniformity of the bit-sliced structure.

3. DEALING WITH SEMI-REGULARITY

Above examples show why fully regular datapaths are so rare in ASIC designs. ASIC datapaths are only semi-regular and cannot be placed in a strictly bit-sliced fashion. As a matter of fact, designers have observed that strictly bit-sliced layouts are not necessarily the goal of datapath placement:

- Cells are not required to be placed in a straight row or column to reduce the routing congestion or timing skews between different bits.
- Local cell movements do not hurt the timing if the moving distance is small.
- Local cell movements often increase utilization and produce more compact layouts.

0-7803-7448-7/02/$17.00 ©2002 IEEE

The above observations indicate that datapath layouts do not have to be the strictly regular arrangement of bit slices. Since fully regular datapaths are so rare, ASIC datapath placement approaches should adopt a broader notion of regularity:

- The bit ordering of each word is consistent throughout the entire dataflow (Fig. 1). This minimizes the wire intersection between different bits, as well as routing congestion and interconnect delays.
- The cells operating on the same bit are placed relative to each other according to the dataflow order across the entire datapath (Fig. 1). This minimizes the timing skews between different bits.
- The datapath placement needs not specify a fixed location for each cell. Rather it specifies relative placement constraints that can be used by the incremental and detailed placers to locally adjust the cell positions.

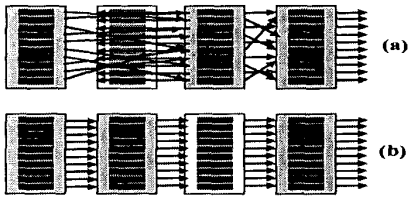

Fig. 1. (a) Datapath placement that violates dataflow order and bit order, (b) Improved placement that preserves dataflow order and bit order

We now summarize the characteristics of fully regular and semi-regular datapaths as follows:

1) Fully Regular Datapaths:

- Cells are placed in bit-sliced fashion
- Bit-slices are replicated for all bits
- Datapath circuits are modularized

2) Semi-Regular Datapaths:

- Cells are placed in bit-ordered fashion that is consistent throughout the dataflow
- Bit slices are not replicated, but dataflow order is preserved
- Datapath circuits are localized

Fig. 2 illustrates the difference between fully regular and semi-regular datapaths.

This broader notion of semi regularity in datapaths fits better with the automated ASIC design flows. Although it does not necessarily produce bit-sliced layouts, it minimizes wire-length, congestion, and timing skews among different bits, and produces more predictable timing.

4. CLUSTER-BASED PLACEMENT OF SEMI-REGULAR DATAPATHS

We have developed algorithms and methodology for placing semi-regular datapaths. This approach uses *clusters* to capture the notion of semi-regularity. A cluster is a set of cells, that are usually placed close to each other in a certain pattern. For example, the full-adder cells of a ripple-carry adder can be grouped in a cluster that is placed as a column in the datapath (Fig. 3).

Clusters can be hierarchical. A hierarchical cluster contains cells and subclusters while a leaf cluster contains only cells (Fig. 3). The topmost cluster in the hierarchy corresponds to the entire datapath module.

A cluster only defines the relative locations of its cells and its subclusters, and uses this information as placement constraints rather than fixed positions. With clusters, the datapath physical synthesis problem becomes a hierarchical cluster placement problem that is formulated and solved in the following order:

1. Create cluster hierarchy
2. Place cluster hierarchy
3. Release cluster boundaries
4. Perform incremental/detailed placement of the leaf cells.

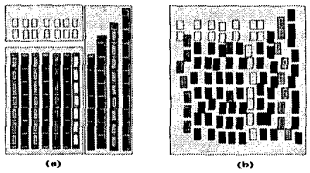

Fig. 2. Layouts of fully regular and semi-regular datapaths

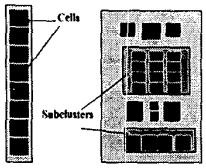

Fig. 3. Clusters and cluster hierarchy

Cluster creation and placement need to be tightly coupled with the datapath synthesis process, therefore the regularity information can be preserved in the cluster hierarchy while the datapath compilation proceeds. The cluster creation and placement steps are described in detail in the following sections. The remaining two steps can be performed with any physical design tools.

5. DATAPATH PHYSICAL SYNTHESIS

5.1. Cluster Hierarchy Generation

The cluster hierarchy is created during datapath synthesis, and not by post-processing the synthesized netlist. Datapath synthesis follows the dataflow order and bit order, and places a cell or cluster inside its parent cluster as soon as it creates them. The physical placement process that follows will preserve this information. This is a bottom-up procedure, and is shown in the following pseudo code:

```
create a cluster CL0;
for each operation; // follow dataflow order
create a cluster CL1 and insert it into CL0;
  for each bit; // follow bit order
    implement bit operation with cell C1;
    insert C1 into CL1;
  done;
done;
```

5.2. Cluster Placement Problem

Datapath cluster placement is different from general macro placement, or macro floor-planning. While placement and orientation of macros are driven by connectivity, placement and orientation of clusters can exploit higher-level information to prune out inferior solutions:

1. The relative locations of the clusters should follow the dataflow order.
2. The relative orientations of the clusters should follow the bit order, and the same bit order should be preserved throughout the dataflow.

These observations indicate that for a horizontal dataflow, the horizontal order and orientation of the clusters are constrained by the dataflow order, and the vertical orientation of the clusters are constrained by the bit order (Fig. 4). A methodology that can utilize this information does not need to formulate a general 2-dimensional problem to orient and place the clusters. Instead it should formulate and solve a smaller scale problem. We refer to this smaller problem as a 1.5-dimensional problem. In contrast, macro placement or floor-planning is a full-blown 2-dimensional problem.

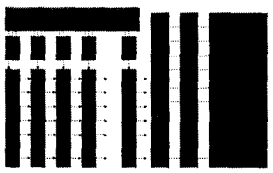

Fig. 4. Cluster placement problem is a 1.5 dimensional problem

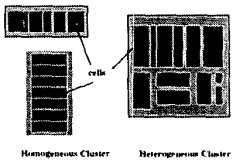

Fig. 5. Homogeneous and Heterogeneous Clusters

5.3. Homogeneous and Heterogeneous Clusters

The 1.5 dimensional placement problem is still intractable. To further simplify this problem, we classify the clusters into two types, *homogeneous clusters* and *heterogeneous clusters*. A homogeneous cluster places all its cells and subclusters either in a single column or in a single row. Any other cluster is a heterogeneous cluster and has multiple rows and columns (Fig. 5).

In datapath cluster structure, heterogeneous cluster can actually be partitioned into a hierarchy of homogeneous clusters. Therefore, we can simplify any datapath cluster structure into a hierarchy consists of homogeneous clusters only.

The cluster generation procedure of Section 5 produces homogeneous clusters only. Since a homogeneous cluster is placed in a single row or column, at each level of the hierarchy the cluster placement problem is a linear (1-dimensional) placement problem. Thus the placement of a hierarchy of homogeneous clusters can be performed by a hierarchical linear placement procedure that solves a sequence of linear (1-dimensional) placement problems in a top-down fashion.

5.4. Top-down Procedure

The top-down procedure places the entire cluster hierarchy. It starts at the topmost cluster, uses the top-level pin locations as placement constraints, and works recursively down the hierarchy. At each level of the hierarchy, it uses the current pin constraints and solves a linear placement problem to place its cells and subclusters, and determines the pin constraints for its subclusters. These pin constraints can then be used to place the subclusters in later iterations. This procedure works down the hierarchy until it reaches the leaf cells. Fig. 6 illustrates this top-down procedure. The linear placement problem that is solved in the inner loop of the top-down procedure is formulated in the next section.

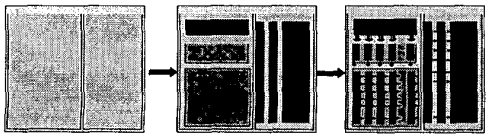

Fig. 6. Cluster placement can be simplified into a hierarchical linear placement procedure

5.5. Linear Placement Problem Formation

A cluster contains a set of cells and subclusters, $V = \{v_1, v_2, ..., v_m\}$, and a set of interconnects $N = \{n_1, n_2, ..., n_n\}$. Each cell or subcluster $v_i \in V$ connects to a subset of interconnects $N_{v_i} \subset N$, and each interconnect $n_j \in N$ connects to a subset of cells and subclusters $V_{n_j} \subset V$. The relative locations of some cells and subclusters $V_f \subset V$ are fixed by external constraints. The external constraints are either relative-placement constraints or pin-order constraints. The problem is to find a linear placement of V, that satisfies all the constraints and minimizes the total interconnect length.

Several existing heuristic algorithms address this problem ([13] [14] [10]). We solve the linear placement problem by minimizing a quadratic objective function. The quadratic objective function is constructed from the connectivity information as discussed below:

Quadratic Objective Function

Given n blocks $v_i \in V, 1 \leq i \leq n$, with locations on x_1, x_2, x_n, the objective function represents the total weighted square wire length as

$$\Phi(x) = \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 = \mathbf{x}^T\mathbf{Q}\mathbf{x} \qquad (1)$$

where blocks v_i and v_j are located at x_i and x_j, and have a weight factor w_{ij} representing the strength of their connection. The objective function can be more concisely described using a vector $\mathbf{x}$ of all cell locations, and a matrix $\mathbf{Q}$ of the weight factors.

The weight factor w_{ij} between blocks v_i and v_j is 0 when the blocks are not connected; otherwise it is equal to the strength $force(i,j)$ of the connectivity between the blocks. The strength $force(i,j)$ is a function of the number of connections between two blocks. The more connections between a pair of blocks, the bigger value $force(i,j)$ will be, and the closer the two blocks will be placed in the linear placement. When i and j are identical, the diagonal elements $w_{11}, w_{22}, ...w_{nn}$ of the matrix $\mathbf{Q}$ are equal to the negative sums of all the elements on the respective rows. This is summarized in the following equation:

$$w_{ij} = \begin{cases} 0 & i \neq j \text{ and no interconnect between } v_i \text{ and } v_j \\ force(i,j) & i \neq j \text{ and } v_i, v_j \text{ are connected} \\ -\sum_{k=1,k\neq i}^{n} w_{ik} & i = j \text{ (The sum of } w_{ik} \text{ in the row i)} \end{cases}$$

The objective function can be modified to capture placement constraints. When some blocks are fixed by the placement constraints, the location vector is split into two parts: one represents the fixed locations $\mathbf{x}_f \subset \mathbf{x}$, and the other represents the movable locations $\mathbf{x}_c \subset \mathbf{x}$. The objective function is then re-written as:

$$\Phi(x) = (\mathbf{x_c} \quad \mathbf{x_f}) \begin{pmatrix} Q_{cc} & Q_{cf} \\ Q_{fc} & Q_{ff} \end{pmatrix} (\mathbf{x_c} \quad \mathbf{x_f})^T \qquad (2)$$

We can compute the zeroes of the derivative of this objective function with a technique similar to [15], and get:

$$\mathbf{Q_{cc}x_c} = -\mathbf{Q_{cf}x_f} \qquad (3)$$

The quadratic objective function is very flexible, because the relative locations of the blocks can easily be adjusted by selecting different connectivity strengths between them. The objective function is also sensitive to long wires, and can reduce long-interconnect problems.

6. EXPERIMENTS

This paper focuses on generating optimal cell placement during automated synthesis of ASIC datapath modules. It creates a compact layout of the datapath module which can be integrated as a macro for whole chip design planning. This macro can be placed and routed with other parts of the design using macro placement or regional placement techniques. The integration of datapath modules with non-datapath logic circuits is still a challenging issue, as addressed in the next section.

Our methodology has been implemented with tightly-coupled synthesis and placement engines. Datapath is described at Register Transfer Level (RTL), and is synthesized into a netlist. Simultaneously, datapath cells are grouped into a cluster hierarchy and are relatively placed according to the dataflow order and the bit order. The datapath module is then routed and physically verified.

Experiments have been performed on 3 datapath modules using 0.18μm technology libraries. The results are compared with a standard design flow where the synthesized netlist is placed using a commercial physical design tool, which contains a quadratic placement engine. The design constraints are timing critical. The comparison is based on 5 metrics: total wire length, average wire length per net, total cell count, cell area, and timing slack. All of these metrics are measured after final routing which is also performed with the same physical design tool. Total wire length represents the summation of all routing wires. The following three paragraphs summarize the results of the three experiments.

Table 1 shows the results for a 56-bit array multiplier design that contains approximately 10K cells (40K gates). Cluster-based placement reduces total wire length by 8.0% and average wire length per net by 5.5%. The reduced wirelength also reduces the amount of buffering of the wires, which consequently reduces the number of cells by 10.7% and total cell area by 11.6%, and causes less congestion and fewer routing violations. Cluster-based placement also improves the final timing slack.

Table 1. Results for a 56-bit Array Multiplier

	total wire length(m)	average wire length(μm)	total cell count	cell area (mm^2)	timing slack(ps)
with clusters	0.973	86	9622	0.926	2012
w/o clusters	1.058	91	10777	1.047	552
improvement	8.0%	5.5%	10.7%	11.6%	1460

Table 2 shows the results for a 32-bit multiplier-accumulator (MAC) design commonly used in DSP applications. It contains about 2600 cells (11K gates). Cluster-based placement produces 9% improvement of total wire length and average wire length per net. It also reduces the cell count by 150, and improves timing slack by 177ps.

Table 2. Results for a 32-bit MAC

	total wire length(m)	average wire length(μm)	total cell count	cell area (mm^2)	timing slack(ps)
with clusters	0.245	68	2474	0.217	24
w/o clusters	0.270	75	2624	0.251	-153
improvement	9.3%	9.3%	5.7%	13.5%	177

Table 3 shows the results for a fast 56-bit floating point multiplier with Booth encoding and Wallace Tree architecture. It consists of 7.2K cells (30K gates). This experiment also produces similar improvements in wire length, cell count, and timing slack.

Table 3. Results of a Fast Floating Point Multiplier

	total wire length(m)	average wire length(μm)	total cell count	cell area (mm^2)	timing slack(ps)
with clusters	0.440	53	7196	0.420	1616
w/o clusters	0.507	60	7274	0.424	722
improvement	13.2%	11.6%	1.1%	1.0%	894

The effect of cluster-based placement is illustrated in the following layout snapshots of the 56-bit fast multiplier. Fig. 7 shows the layout produced by quadratic based standard-cell placement where cells are packed in rows. Fig. 8 shows the layout produced by cluster-based placement where the cells are aligned in columns on the left hand side, and as the operations proceed, the columns gradually merge into the irregular carry-lookahead adder on the right hand side (in this experiment, we did not perform regular placement on the carry-lookahead adder, so it looks irregular).

7. SUMMARY AND FUTURE WORK

This paper has described a new methodology for physical synthesis of ASIC datapath modules. It introduced a broader notion of regularity that is more appropriate for semi-regular datapaths common in ASIC designs. Using clusters, placement of semi-regular datapath is simplified into a hierarchical linear placement problem. Clusters are generated in the dataflow order during synthesis, and are placed by recursively minimizing a quadratic objective function. This methodology fits very well inside an automated design flow, and improves the placement quality of the experimental datapath modules.

Although we now have a methodology for producing compact layouts for semi-regular datapath modules, there are future challenges. In particular, a full design contains a varying mix of datapath and irregular random logic. To improve the placement of the whole design, we need a methodology that smoothly combines the strengths of datapath placement and standard-cell placement algorithms. Future research needs to address this problem before automated design flows can produce designs that are comparable in quality with manual designs.

8. REFERENCES

[1] D.G. Chinnery, K. Keutzer ; *Closing the Gap Between ASIC and Custom: An ASIC Perspective* Design Automation Conference, 2000. Proceedings., 37th ACM/IEEE , 2000 , Page(s): 637 - 642

[2] William J. Dally, Andrew Chang ; *The Role of Custom Design in ASIC Chips* Design Automation Conference, 2000. Proceedings., 37th ACM/IEEE , 2000 , Page(s): 643 - 647

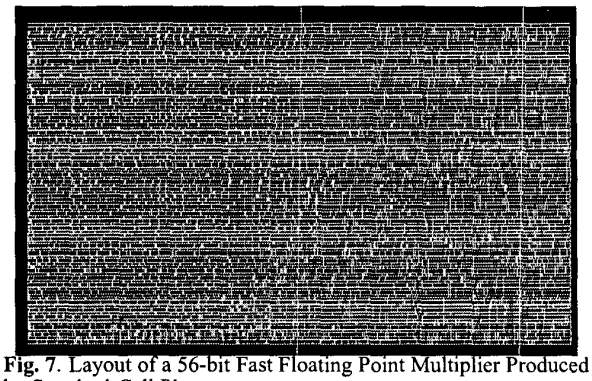

Fig. 7. Layout of a 56-bit Fast Floating Point Multiplier Produced by Standard-Cell Placement.

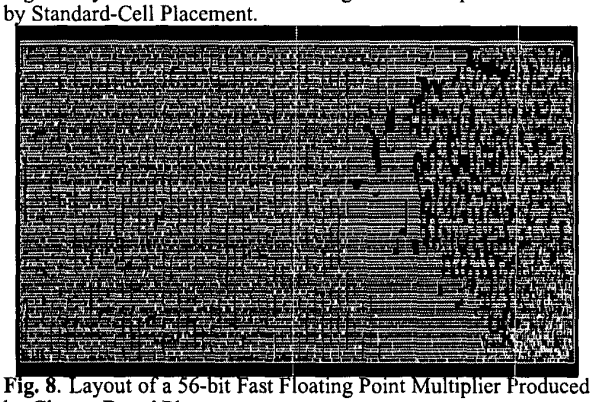

Fig. 8. Layout of a 56-bit Fast Floating Point Multiplier Produced by Cluster-Based Placement.

[3] Leveugle, R.; Safinia, C.; Magarshack, P.; Sponga, L. ; *Data path implementation: bit-slice structure versus standard cells* Euro ASIC '92, Proceedings. , 1992 , Page(s): 83 -88

[4] Askar, S.; Ciesielski, M. ; *Analytical approach to custom datapath design* Proc.. ICCAD 1999 , Pg(s): 98 -101

[5] Serdar, T.; Sechen, C. ; *AKORD: transistor level and mixed transistor/gate level placement tool for digital data paths* Digest of Technical Papers. ICCAD 1999 , Page(s): 91 -97

[6] Kim, J.; Kang, S.M. ; *A timing-driven data path layout synthesis with integer programming* Proc.. ICCAD 1995 , Pg(s): 716-719

[7] Luk, W.K.; Dean, A.A. ; *Multistack optimization for data-path chip layout* CAD of Integrated Circuits and Systems, IEEE Transactions on Volume: 10 1 , Jan. 1991 , Page(s): 116 -129

[8] Cai, H.; Note, S.; Six, P.; de Man, H. ; *A data path layout assembler for high performance DSP circuits* Proc., 27th ACM/IEEE Design Automation Conference, 1990 , Page(s): 306 -311

[9] Buddi, N.; Chrzanowska-Jeske, M.; Saxe, C.L. ; *Layout synthesis for data-path designs* Design Automation Conference, 1995, with EURO-VHDL, Proc., EURO-DAC '95, Page(s): 86 -90

[10] Yim, J.-S.; Kyung, C.-M. ; *Data path layout optimization using genetic algorithm and simulated annealing* Computers and Digital Techniques, IEE Proc, March 1998 , Page(s): 135 -141

[11] Nakao, H.; Kitada, O.; Hayashikoshi, M.; Okazaki, K.;Tsujihashi, Y. ; *A high density data path layout generation method under path delay constraints.* CICC 1993, Proceedings, Page(s): 9.5.1 -9.5.5

[12] Tao Ye, T.; De Micheli, G.; *Data path placement with regularity* Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on , 2000 , Page(s): 264 -270

[13] Kang, S. ; *Linear Ordering and Application to Placement* Proc. 20th Design Automation Conf., 1983, pp. 457-464

[14] Sung-Woo Hur; Lillis, J. ; *Relaxation and clustering in a local search framework: application to linear placement* Proceedings, 36th Design Automation Conference, 1999 , Page(s): 360 -366

[15] Alpert, C.J.; Chan, T.; Huang, D.J.-H.; Markov, I.; Yan, K. *Quadratic Placement Revisited* Design Automation Conference, 1997. Proceedings of the 34th , Page(s): 752 -757