VLSI DESIGN
1997, Vol. 5, No. 2, pp. 195–209
Reprints available directly from the publisher
Photocopying permitted by license only

Datapath Optimization Using Layout Information: An Empirical Study

ALLEN C.-H. WU*

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, 30043, R. O. C.

(Received 1 January 1993; In final form 1 December 1993)

Most datapath synthesis approaches use a simple area model to evaluate design area quality. However, using such a simplified model could mislead synthesis tasks into generating inferior designs. This paper presents an extensive experimental study to validate the correlation between the tradition area model, our proposed area model, and the actual layouts. The results show that traditional area quality measures are not good indicators for optimization in datapath synthesis. Moreover, this paper also shows that to provide accurate indications for design tradeoffs in high-level synthesis, the *fidelity* of the estimates is more important than the accuracy.

Keywords: Datapath synthesis, layout model, quality measures, estimation

1. INTRODUCTION

High-level synthesis deals with the transformation of a behavioral description into an RTL design. In general, high-level synthesis performs a number of tasks including scheduling, allocation, and binding [9, 3, 20]. In the first step, operations are scheduled and assigned to control states. In the second step, operations are bound to functional units, variables are bound to storage units, and communication paths (busses and multiplexers) are chosen for data transfer between functional units and storage units. Finally, the synthesis tool generates a structural netlist consisting of a datapath and a control sequence table. The datapath netlist consists of a set of generic register-transfer (RT) components (e.g., functional, interconnect and storage units). The control unit specifies the control signals for executing register transfers in each state, as well as the sequencing for the design's next state.

The process of generating fabrication data for custom or semicustom technologies from a structural design consists of several steps, including technology mapping, module generation, floorplanning, placement and routing. Technology mapping assigns real components from a physical library to the generic components in the structural design. It is usually followed by some optimization procedures to reduce the total area and delay of the design. Module generators perform placement and routing of each module inde-

*Corresponding author. Tel.: 035-715131 ext. 3517. Fax: 011-886-35-723694. E-mail: chunghaw@cs.nthu.edu.tw.

pendently. The floorplanner determines the positions and interconnections of modules and generates the final layout.

In the past, the number and size of functional units, registers, muxes (or equivalent 2-to-1 muxes), mux inputs and connections (or wires) were the commonly-used area measures in high-level synthesis. Consequently, a great deal of effort has been devoted to minimizing the number and size of registers and muxes in high-level synthesis [2, 5, 10, 12, 17, 21, 23, 25, 26, 32]. However, these area measures assume that the layout area is directly proportional to the number and size of RT components and do not take into account layout technology factors, such as layout architectures or styles, component libraries, and the impact of floorplanning, placement and routing. These factors often greatly affect the final layout of the design.

In this paper, we perform an empirical study to validate the correlation between the traditional area model, our proposed area model [35], and the actual layout. We first present the layout area model and an iterative improvement binding algorithm for datapath optimization. Then, we present a series of experiments to compare the traditional area model, our proposed area model, and the actual layouts. We have demonstrated that the quality measures based on the traditional area model are not good indicators for optimization in datapath synthesis. In addition, we have shown that to provide good indications for design tradeoffs during synthesis process, the fidelity of the estimates is more important than the accuracy. The work presented in this paper suggests that a high-fidelity and fast quality measure procedure is desirable in high-level synthesis.

The remainder of this paper is organized in the following manner. Section 2 describes some previous work. Section 3 presents the implementation for the empirical study. Section 4 gives experimental results and discussions. Finally, Section 5 concludes our approach.

2. PREVIOUS WORK

There are two approaches to incorporate physical design information into high-level synthesis, as shown in Figure 1. The first approach (*Path 1*) uses some layout models to estimate design area quality. The drawback of using simplified layout models is low accuracy of obtained estimates. Hence, using such a simplified model could mislead the synthesis tasks into generating inferior designs. The second approach (*Path 2*) feeds back the actual layout information by performing layout synthesis. This approach does provide accurate estimates; however, it is too slow for nontrivial designs.

BUD [18] used layout models to estimate area for each module, and then implemented a floorplanning procedure to estimate the total design area. Chippe [1] fed back layout information obtained from simple layout models to guide scheduling and allocation. Fasolt [13] also fed back layout information to improve scheduling and allocation; however, the layout information is obtained by performing layout generation. 3D-scheduling [34] took into account interconnect delays during the scheduling process by incorporating a floorplanner. Pangrle et al. [22] also addressed placement and routing issues in high-level connectivity synthesis.

McFarland used the BUD system to perform a number of experiments to demonstrate that a simple layout model ignoring interconnects and other layout factors is inadequate to provide accurate estimates during the synthesis process [19] In addition, Parker et al., also performed a series of experiments to show the effects of physical design characteristics on the area-performance tradeoff curve, using both pipelined and nonpipelined design styles [24].

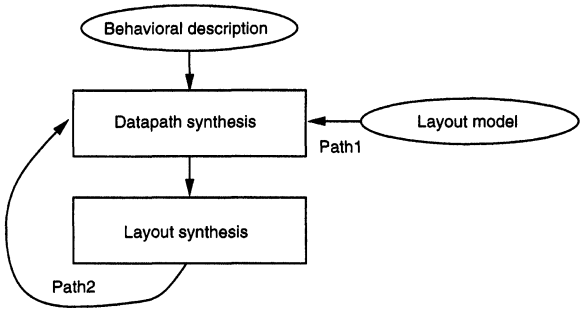

FIGURE 1 Two layout-driven datapath synthesis approaches.

To obtain more accurate quality measures for high-level synthesis, many researchers have focused on developing more realistic layout models and estimation methods. Early studies [6, 8, 11, 31] concentrated on developing estimation models for the prediction of the wiring space.

PLEST [15] and that of Pedram and Preas [27, 28], used analytical models for standard cell area and wire length estimations from gate level netlists. Both models reported a 10% accuracy in predicting standard cell layout areas. Zimmermann [37] used a constructive approach to estimate the area and shape function of custom layouts. Given a design description, it partitions the design into a slicing tree. At the leaf-cell level, the area and shape functions are known precisely. At each level, the wiring area between two blocks is estimated. The chip area and shape can be obtained by adding up through the hierarchy from the leaf-cell level.

LAST [16] and TELE [29] used a combination of analytical and constructive techniques to estimate the area and delay of a netlist of cells. It partitions the circuit repeatedly into a slicing tree in which the level of the slicing tree is specified by the user. The shape function of each leaf cell is then estimated using an analytical model. Because the constructing level is controlled by the user, this approach permits the user to trade off the accuracy of the prediction versus the run time of the predictor.

Recent work such as [4, 30, 35] provides area and delay quality measures for driving high-level synthesis tools. In [4, 35] abstracted layout area and timing models for high-level synthesis were presented. These models considered several layout factors, including layout architectures, placement, and routing, and experimentally shown to accurately and efficiently reflect the effects of the datapath design tradeoffs. In [30] a layout predictive model was proposed to take into account the effects of wiring and floorplanning on the area and performance estimations of RT level designs.

3. IMPLEMENTATION

This section discusses the layout model and binding algorithm in greater detail. In order to study the correlation between the traditional area models, our proposed area models, and the actual layouts, we use an iterative improvement binding algorithm with given resource constraints to allow us explore the design space.

3.1 Layout Architecture and Layout Model

To obtain more realistic quality measures in high-level synthesis, a layout model should take into account most technology factors such as layout architecture, technology mapping, placement, and routing. However, it is difficult to develop a general purpose layout model that will provide accurate quality measures for a variety of layout architectures. Thus, first we have to select a target layout architecture and then focus on this target architecture to develop the layout model.

A datapath consists of a set of regularly structured RT components, such as ALUs, multiplexers, latches, drivers and shifters. Datapath layout is accomplished using a stack of functional and storage units that are placed one above the other and partitioned into bit slices (Figure 2(a)). The stack grows horizontally when the bit width increases; it grows vertically when the number of units increases. Each bit slice of a unit may be a handcrafted custom cell or implemented with one row of connected standard cells. These two layout styles are shown in Figures 2(b) and (c). The difference between these two styles is in the layers used for routing of the control and data wires, in the use of custom or standard cells, and in the routing of data lines over the cells or in a separate channel.

In the first layout architecture shown in Figure 2(b), diffusion strips for P and N transistors are placed horizontally. Power/ground wires and control lines common to different bit slices in each unit run horizontally in the first metal layer. Data lines connecting distinct units in each bit slice run vertically in the second metal layer. In the second layout architecture shown in Figure 2(c), a bit slice of each unit consists of one or more standard cells. P and N diffusion strips are placed vertically. Power and ground wires run vertically in the first metal layer. Data lines

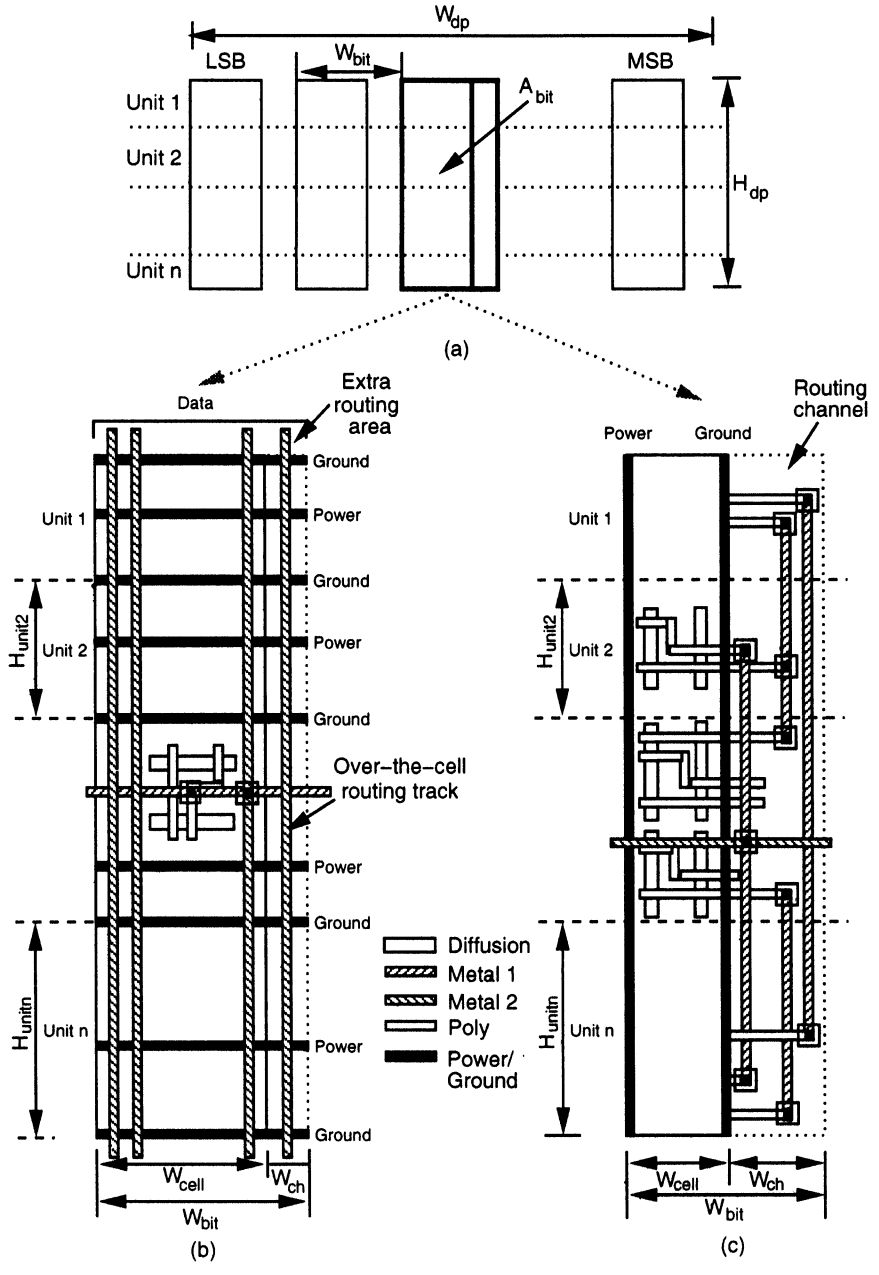

FIGURE 2 The layout architectures and models: (a) datapath stack, (b) custom cell architecture, (c) standard cell architecture.

are placed in the routing channel and run vertically in the first metal or the polysilicon layer.

To compute the height of a bit slice (H_{dp}), we must observe that the height is proportional to the number of transistors in the single row standard-cell layout architecture in Figure 2(c). Each bit slice of a unit consists of several diffusion strips separated by gaps. The transistors on each diffusion strip are separated by metal-diffusion contacts or just by the minimal-allowed poly-to-poly spacing. Thus, the height of each unit (H_{unit}) in Figure (c) can be computed as a product of number of transistors in $unit_i$ (tr_{unit_i}) and

transistor-pitch coefficient α in μm/transistor obtained by averaging α over all units in the library, i.e.,

$$H_{unit} = \alpha \times tr_{unit}. \quad (1)$$

Similarly, the height of the bit-sliced stack of n units is

$$H_{dp} = \alpha \times (\sum_{i=1}^{n} tr_{unit_i}). \quad (2)$$

Similarly, arrangement can be made for layout architecture I shown in Figure 2(b). Although P and N strips are placed horizontally in several rows, the height of the unit slice (H_{unit}) can be computed by Equation 1 using a different transistor-pitch coefficient α'. This is possible since the width of the unit slice (W_{cell}) is a fixed constant. Thus, H_{unit} can reflect the size of the cell in the number of transistors.

The width of a bit slice (W_{bit}) is equal to the sum of the width of the unit slice (W_{cell}) and the width of the routing channel (W_{ch}). W_{cell} for both layout architectures is a constant since all unit slices are predesigned to be the same width. The width of the routing channel (W_{ch}) is calculated as a product of the wire pitch (β) and the difference between the number of estimated routing tracks (Trk_{est}) required to completely connect all nets in one bit slice and the number of available over-the-cell routing tracks (Trk_{top}) as

$$W_{ch} = \begin{cases} 0; & \text{if } Trk_{top} \geq Trk_{est} \\ \beta \times (Trk_{ext} - Trk_{top}); & \text{if } Trk_{top} < Trk_{est} \end{cases} \quad (3)$$

where Trk_{top} in the architecture II is equal to zero, and coefficient β is equal to the sum of minimal wire width and minimal spacing between two metal wires.

Hence, the datapath area (A_{dp}) can be calculated as a product of number of bits (b_w) and the area of one bit slice as shown below:

$$A_{dp} = b_w \times H_{dp} \times (W_{cell} + W_{ch}). \quad (4)$$

3.2 An Iterative Improvement Binding Algorithm

3.2.1 Hypergraph Model

In order to use the proposed layout model, we model the hardware description (RT structural netlist) as a hypergraph that consists of a set of hypernodes V and a set of hyperedges E. Each hypernode represents a physical component, such as a functional unit, an input/output port or a storage unit. Each hyperedge represents a physical wire connecting two hypernodes (two components). For instance, Figures 3(a) and (b) show a structural netlist and its corresponding hypergraph representation. Input/output ports a, b, c, h and g are mapped to hypernodes v_1, v_2, v_3, v_9 and v_{10}, registers *Reg1*, *Reg2* and *Reg3* are mapped to hypernodes v_4, v_5 and v_6, and functional units *Mult* and *Adder* are mapped to hypernodes v_7 and v_8.

Each hypernode contains a set of input/output pins that are either commutable or non-commutable. When an input pin of a hypernode connects to more than one hyperedge, an interconnect unit (e.g., a mux) is needed to choose between the alterative inputs. For instance, since the second input pin (In_2) of the hypernode v_8 in Figure 3(b) connects to hypernodes v_4 and v_6 (i.e., *Reg1* and *Reg3*), a mux (*Mux5*) is needed as shown in Figure 3(a).

3.2.2 Area Cost Function

Since we model the structural netlist as a hypergraph, we can calculate the datapath area from the hypergraph using the layout model described in Section 3.1. Using this model, we need two elements to compute the area: the number of transistors and the number of routing tracks.

The number of transistors of each RT component can be obtained directly from the target component library. To obtain the number of routing tracks required to completely connect all nets in one bit slice, we first implement stack placement using the KLFM [7] algorithm. Then we implement routing track assignments using the left-edge algorithm. Since the stack placement takes pseudo-linear time and the routing-track estimation takes $O(nlogn)$ time where n

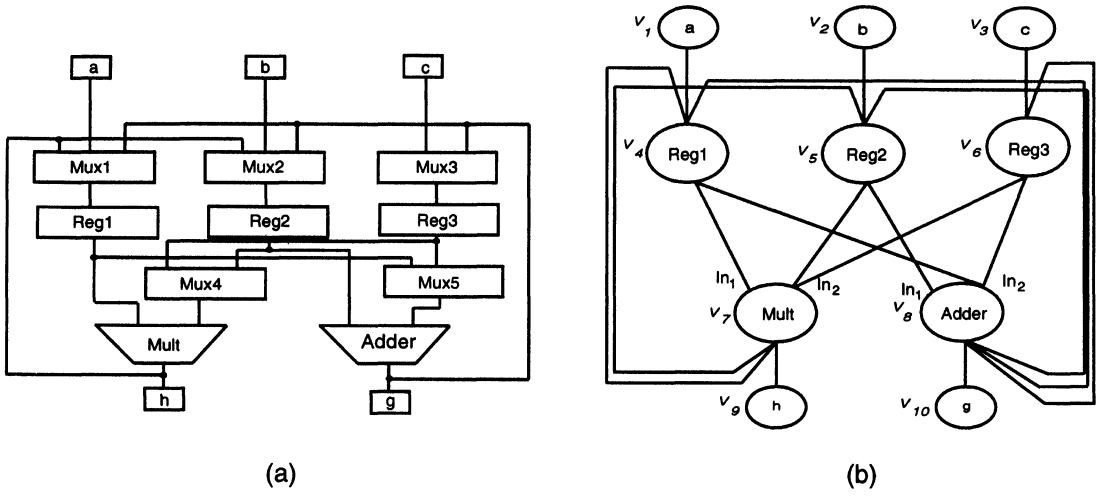

FIGURE 3 Hypergraph formation: (a) a structural netlist, (b) a corresponding hypergraph.

is the number of nets in the RT-netlist, the complexity of the area calculation is approximately $O(nlogn)$.

3.2.3 The Algorithm

The objective of datapath binding is to assign operations to functional units and to ássign variables to storage units so that the total layout area is minimized. Using the hypergraph model, we can formulate the binding problem into a graph-partitioning problem, as follows: Given a data-flow graph, its corresponding schedule, and a set of functional units, partition operations and variables into a set of hypernodes, such that:

1. no two operations in the same control step can be assigned to the same functional-unit hypernode;
2. no variables with overlapping lifetimes can be assigned to the same storage-unit hypernode;
3. the total area of the hypergraph is minimized.

The algorithm consists of two phases: initial assignment and interchange optimization. Initial assignment consists of two steps: hypernode formation and hyperedge formation. In the first step, the algorithm determines the minimum number of registers required to store all variables using the left-edge algorithm [14] and assigns variables to corresponding registers. The algorithm then assigns operations to the given functional units arbitrarily, such that no more than one operation in the same control step will be assigned to the same functional unit. Finally, the input/output ports, registers, and functional units are mapped to a set of hypernodes, and the operations and variables are assigned to their corresponding hypernodes.

For example, Figure 4(a) shows a data flow graph example that is scheduled into five steps. Given an adder and a multiplier, Figure 4(b) shows the operation and variable assignment. Operations *add1*, *add2* and *add3* are assigned to the adder (*Adder*), and operations *mult1* and *mult2* are assigned to the multiplier (*Mult*). For the eight variables a, b, c, d, e, f, g and h, and we need three registers to store them as shown in Figure 4(b). The input/output ports, functional units, and registers are mapped to a set of hypernodes as shown in Figure 4(c). The input port a is mapped to the hypernode v_1, the *Mult* is mapped to the hypernode v_7, and the *Reg1* is mapped to the hypernode v_4. The data flow graph is folded into this set of hypernodes by mapping the operations and variables to their corresponding hypernodes.

In the hyperedge formation step, the algorithm maps dependency edges to hyperedges one at a time. If a dependency edge can share the same path with another hyperedge, then the edges can be merged into a single hyperedge, otherwise a new hyperedge has to be created. Applying the hyperedge formation on the

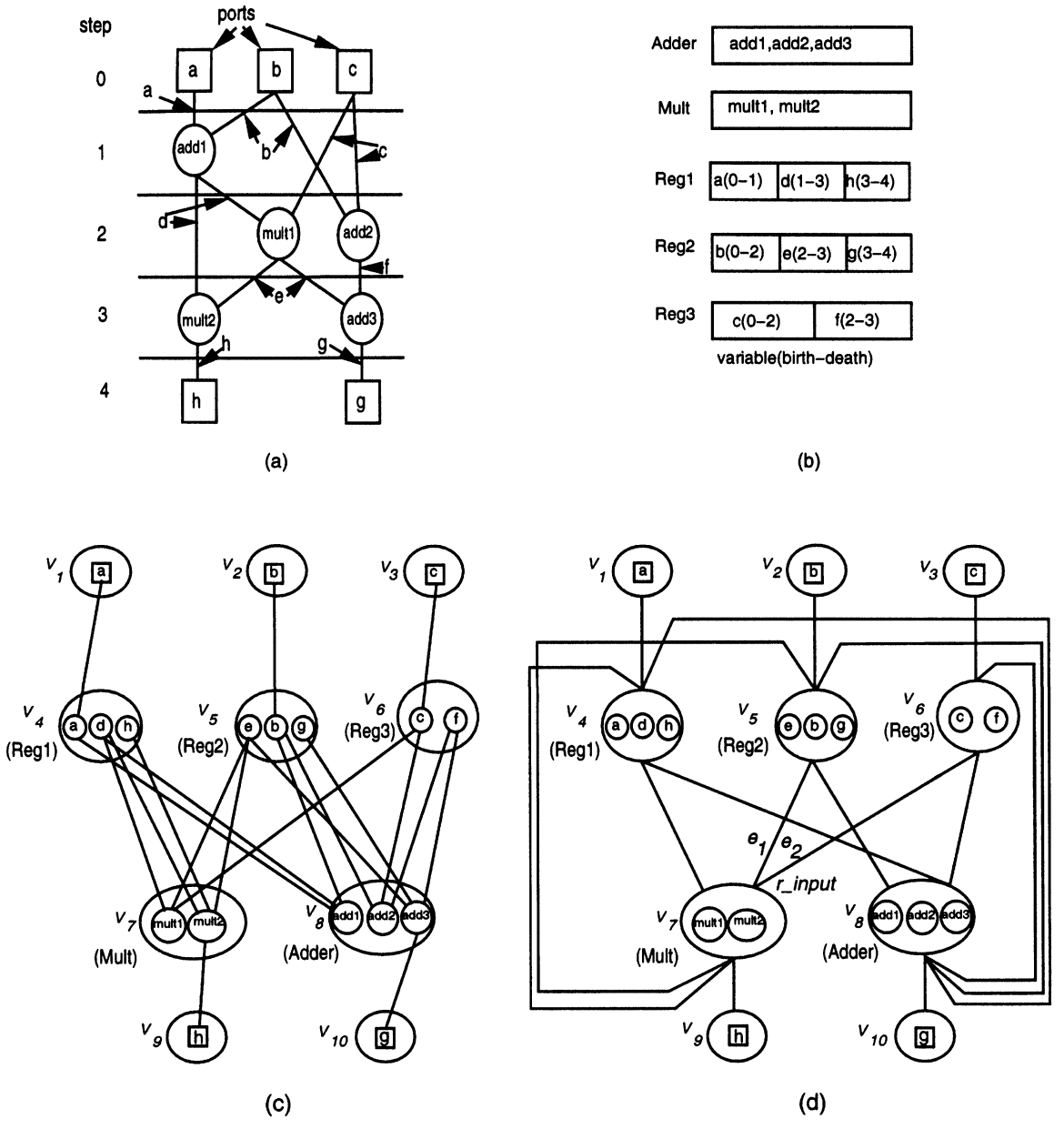

FIGURE 4 Allocation using the hypergraph: (a) a data flow graph and schedule, (b) variable and operation assignments, (c) after folding the data flow graph into a hypergraph, (d) the final hypergraph.

example of Figure 4(c) results in the final hypergraph shown in Figure 4(d).

In the interchange optimization phase, the algorithm performs hyperedge merging by interchanging operations and variables that reside in the hypernodes. The overall binding algorithm is shown in Appendix I [36]. The algorithm first locates the hypernodes such that their inputs connect to more than one

hypernode (*locate_feasible_merging_hyperedge(H)*). The hyperedges connected to the inputs of these hypernodes are called "feasibly-mergeable hyperedges". For example, in Figure 4(d) there are two input hyperedges e_1 and e_2 entering the *r_input* of V_7. e_1 and e_2 are selected as two feasibly-mergeable hyperedges. For each feasibly mergeable hyperedge, the procedure *relocate_node*(e) locates a set of variables or operations associated with this hyperedge. For example, e_1 is associated with variable e in V_5 while e_2 is associated with variable c in V_6. Nodes e and c are selected as the relocate_nodes. The algorithm then tries to rearrange the nodes (e.g., nodes e and c) to reduce the interconnect cost. That is if e and c can be merged by relocating c into V_5 or e into V_6, a 2-to-1 mux in front of the *r_input* of V_7 can be eliminated. The algorithm rearranges the nodes and calculates the layout area as described in the previous section. If a smaller layout area is obtained, then a merging solution has been found. The algorithm begins with the minimum number of registers and performs the binding iteratively by incrementing the number of registers to explore the design space. For each iteration, the algorithm runs repeatedly until no more improvement can be found.

Each inter *while* loop of the algorithm takes $O(prq)$ time, where p denotes the number of hyperedges, r the average number of variables or operations associated with the feasible merging hyperedges, q the number of hypernodes. The detailed complexity analysis can be found in [36]. In our experience, the local optimal state (*no_more_improve*) can be achieved by less than 20 iterations. The run times for all examples described in the Experiments and Results section are less than 15 seconds.

4. EXPERIMENTS AND RESULTS

We have implemented the proposed algorithm using the C programming language on a SUN4 workstation under UNIX. We have experimented on 4 designs with 16 different schedules and resources implementations of the elliptic filter benchmark. Each implementation uses a different number of registers and muxes. The α coefficient was calculated based on the VTI 1.5-μ datapath library [33]. The final layouts were generated using Mentor Graphics GDT tools. The layout architectures used in Figure 5 and Figure 6 correspond to those described in Section 3, in which the Layout Architecture I uses 13 over-the-cell routing tracks for each bit slice. Since the multiplier is treated as a macrocell, its area remains constant throughout all the examples, and is not included in the results. Figure 7 shows that 64 area measures using different combinations of layout architectures, muxes or buses (one tri-state buffer for each mux input). The results show that 90% of the estimates are within 90% accuracy.

We have investigated the "fidelity" of our area estimates. Fidelity is another crucial factor in the quality measure that indicates the degree of the estimated results correspond to the actual results. In other words, fidelity is the deviation from the average error over all design points. If the error over all design points is always of the same magnitude then fidelity is high. For instance, Figure 8 shows two examples, in which the solid line represents the actual results while the dash line represents the estimated results. Figure 8(a) shows that the estimates predict the actual results well; that is, if we have to select the minimum-cost design then design C will be selected since the estimate C' predicts the minimum cost. Thus, the estimates in Figure 8(a) show high fidelity. On the other hand, the estimates in Figure 8(b) show poor fidelity since design B will be selected as the minimum-cost design according to the estimate B'. However, design B has the highest cost.

We compare the "fidelity" of 7 different metrics, namely metric #1, 2, 3, 4, 5, 6, and 7 (Figure 9). The numbers in Figure 9 represent the percent difference between the area of the predicted minimal area implementation and the actual minimal area of the design.

For each design, we first choose the minimum cost implementation according to different metrics. For example, based on metric #5, we choose the implementation with the minimum number of transistors as the best implementation (Figure 5). For design D, the

** Area not including multiplier

Design	# of Reg.	# of Mux. / # Mux Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I: Est. Area (um² / bit)	Layout Architecture I: Actual Area (um² / bit)	Layout Architecture I: Est. / Actual	Layout Architecture II: Est. Area (um² / bit)	Layout Architecture II: Actual Area (um² / bit)	Layout Architecture II: Est. / Actual
A	10	11 / 34	552	27	11(15)	129,680	136,720	0.95	213,625	193,117	1.11
	11	10 / 33	564	27	11(12)	124,080	138,080	0.90	200,216	195,038	1.03
	12	8 / 31	572	27	11(12)	125,840	138,480	0.91	200,796	195,490	1.03
	13	9 / 33	604	28	10(15)	143,653	145,040	0.98	226,625	199,430	1.14
B	10	8 / 30	472	23	10(11)	103,840	113,440	0.92	166,234	156,420	1.03
	11	6 / 28	480	22	9(10)	105,600	113,760	0.93	156,420	151,726	1.03
	12	6 / 28	500	23	9(11)	110,000	117,280	0.94	165,658	156,862	1.06
	13	6 / 29	524	24	9(10)	115,280	122,080	0.94	167,860	163,282	1.03
C	10	7 / 30	480	20	8(11)	105,600	113,536	0.93	160,369	146,464	1.09
	11	5 / 27	480	19	10(12)	105,600	111,456	0.95	161,611	151,836	1.06
	12	5 / 28	504	19	9(11)	110,880	115,296	0.96	162,855	152,934	1.06
	13	6 / 31	540	23	9(11)	118,800	126,736	0.94	179,014	168,235	1.06
D	10	10 / 36	508	23	10(11)	111,760	125,376	0.89	177,093	170,976	1.04
	11	6 / 28	482	20	9(11)	106,040	113,136	0.94	159,804	150,045	1.07
	12	6 / 26	492	21	8(10)	108,240	115,696	0.94	159,082	149,272	1.07
	13	5 / 23	490	21	8(11)	107,800	113,696	0.95	158,595	146,672	1.09

A : 17-step, 3-adder, 2-piped multipliers.
B : 19-step, 2-adder, 2-multiplier.
C : 21-step, 2-adder, 1-multiplier.
D : 19-step, 2-adder, 1-piped multiplier.

FIGURE 5 The datapath area estimates of the elliptic filter example with mux implementation.

implementation with 11 registers, 6 muxes, 28 mux inputs (482 transistors) is chosen as the best implementation in which the areas are 113,136μm^2 and 150,045μm^2 using layout architectures I and II, respectively. On the other hand, the actual minimal areas for design D are 113,136μm^2 and 146,672μm^2. For design D with layout architecture I, the transistor metric (metric #5) accurately predicts the minimum area. Since the percent difference between the area of predicted best implementation and the actual minimum area is 0, the entry for metric #5 and design D with layout architecture I in Figure 9(a) is 0. On the other hand, for design D with layout architecture II, the area of predicted best implementation is 2.3% (150,045μm^2 vs. 146,672μm^2) larger than the actual minimum area of the design. Hence, the number in Figure 9(a) is 2.3.

Since all implementations using layout architecture I use less than 13 actual routing tracks, they do not require any extra routing tracks. Hence, the area of the datapath is solely dependent on the number of transistors. This is the reason why metric #5 can predict the minimum area implementations on all designs using layout architecture I. Metrics #1, 2, 3 and 4 give poor predictions because register and mux counts alone will not accurately predict total number of transistors in the datapath. Metric #6 also gives poor predictions because this metric considers wiring

** Area not including multiplier

Design	# of Reg.	# of Mux. / # Mux Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I: Est. Area (um² / bit)	Layout Architecture I: Actual Area (um² / bit)	Layout Architecture I: Est. / Actual	Layout Architecture II: Est. Area (um² / bit)	Layout Architecture II: Actual Area (um² / bit)	Layout Architecture II: Est. / Actual
A	10	11 / 34	776	26	11(15)	182,888	162,240	1.13	259,600	229,164	1.13
	11	10 / 33	784	28	10(14)	174,526	163,680	1.07	255,750	225,060	1.14
	12	8 / 31	780	28	9(13)	171,600	162,720	1.05	242,063	217,638	1.11
	13	9 / 33	824	29	8(12)	181,280	168,960	1.07	244,976	219,648	1.10
B	10	8 / 30	672	21	8(10)	147,840	136,960	1.08	199,176	172,912	1.15
	11	6 / 28	688	22	7(9)	151,360	136,000	1.11	197,260	171,700	1.15
	12	6 / 28	688	23	9(10)	151,360	139,840	1.08	203,800	187,036	1.09
	13	6 / 29	720	24	8(10)	158,400	146,080	1.08	200,860	189,904	1.06
C	10	7 / 30	672	20	10(10)	147,840	136,896	1.08	199,176	186,816	1.07
	11	5 / 27	656	19	8(8)	144,320	133,536	1.08	184,380	172,464	1.07
	12	5 / 28	688	20	7(8)	151,360	137,376	1.10	192,572	172,446	1.10
	13	6 / 31	744	23	9(8)	163,680	153,216	1.07	209,644	203,652	1.03
D	10	10 / 36	744	21	10(13)	163,680	148,896	1.10	225,099	203,316	1.10
	11	6 / 28	668	19	7(9)	146,960	133,536	1.10	191.706	167,598	1.14
	12	6 / 26	664	20	8(9)	146,080	132,576	1.10	196,824	171,216	1.11
	13	5 / 23	648	20	8(8)	142,560	129,216	1.10	181,324	166,848	1.09

A : 17-step, 3-adder, 2-piped multipliers.
B : 19-step, 2-adder, 2-multiplier.
C : 21-step, 2-adder, 1-multiplier.
D : 19-step, 2-adder, 1-piped multiplier.

FIGURE 6 The datapath area estimates of the elliptic filter example with bus implementation.

area in terms of number of unique nets, which is absent in this case. Our layout model (metric #7) shows accurate predictions except for the design A due to over-estimation in the number of routing tracks caused by our simple linear placement method.

Using layout architecture II, transistors and routing tracks make equal contribution to the total area. Hence, design quality measures which do not consider routing tracks, for example metrics #1, 2, 3, 4 and 5, do not predict layout area well. Metric #6 does not do well because the number of unique nets does not directly indicate the number of routing tracks. As for our layout model, the results show consistent fidelity.

The results also show that neither the design with the minimum number of registers nor the design with the minimum number of muxes can guarantee the minimum area. For instance, The corresponding curve of Figure 5 is shown in Figure 10, which indicates:

(1) The designs with the minimum number of registers do not always produce the minimum area, such as:
   (i) 21-step and architecture I (Figure 10(e)).
   (ii) 19-step and 19-step with 2-adder and 1-piped multiplier, and architecture II (Figure 10(d)(h)).

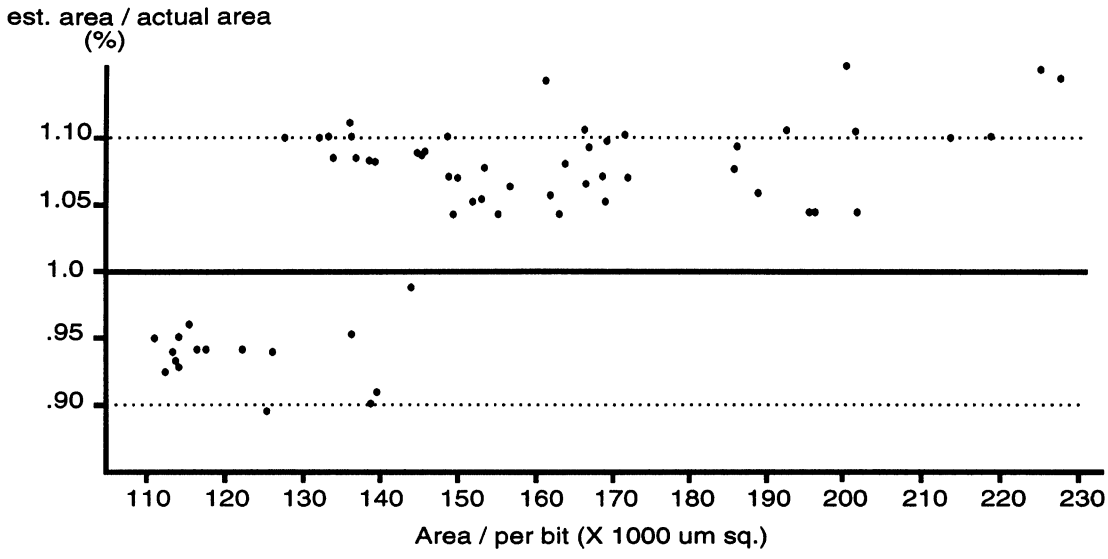

FIGURE 7 The accuracy analysis of the datapath area estimates.

(2) The designs with the minimum number of mux inputs do not always produce the minimum area, such as:
  (i) 19-step with 2-adder and 1-piped multiplier, and architecture I (Figure 10(g)).
  (ii) 21-step and architecture II (Figure 10(f)).
  (iii) 17-step, architecture I and II (Figure 10(a)(b)).

(3) The design that produces the minimum area using layout architecture I does not guarantee the minimum area using layout architecture II. For example, the 21-step design (Figure 10(e)) with 11 registers and 27 mux inputs produces the minimum area using layout architecture I but not layout architecture II (Figure 10(f)).

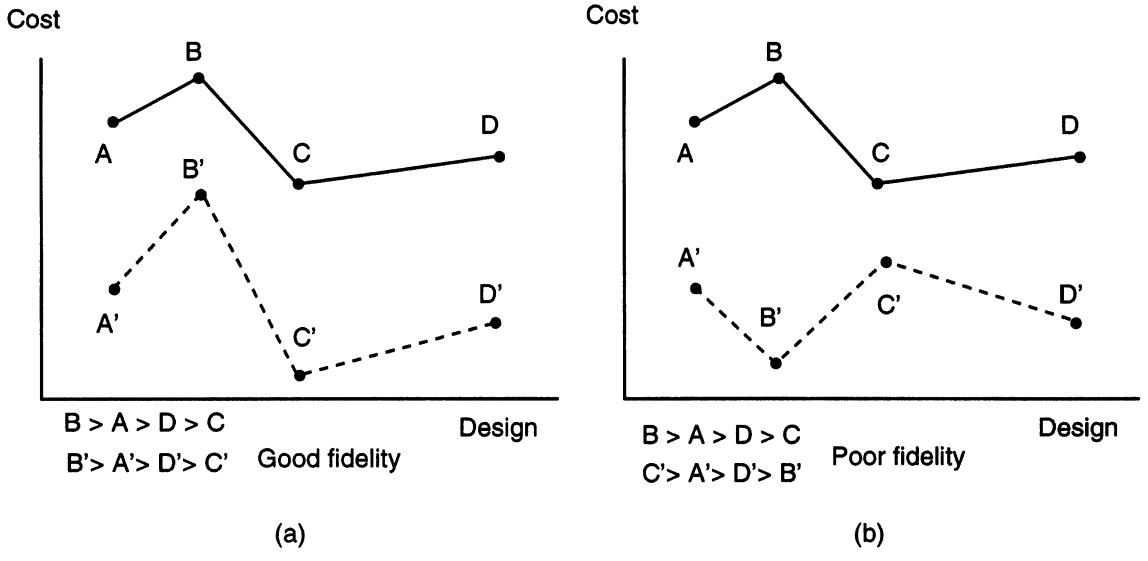

FIGURE 8 The fidelity analysis: (a) good fidelity, (b) poor fidelity.

Metrics	Quality mesures	Percent difference between the predicted best implementation and the actual minimum area of the design							
		Layout architecture I				Layout architecture II			
		A	B	C	D	A	B	C	D
1	# Register	0	0	1.87	10.82	0	3.09	0	16.60
2	# Mux input	1.27	0.28	0	0.50	1.22	0	3.67	0
3	# Equivalent 2:1mux	0	0	0	0.50	0	3.09	3.67	0
4	# Register + # Equivalent 2:1mux	0	0	0	0.50	0	3.09	3.67	0
5	# transistor	0	0	0	0	0	3.09	3.67	2.30
6	# Register+# Mux input + # Unique net	1.29	0.28	0	0.50	1.23	0	3.67	0
7	Our layout model	1.0	0	0	0	1.0	0	0	0

(a)

Metrics	Quality mesures	Percent difference between the predicted best implementation and the actual minimum area of the design							
		Layout architecture I				Layout architecture II			
		A	B	C	D	A	B	C	D
1	# Register	0	0.7	2.5	15.2	5.3	0.7	8.3	21.8
2	# Mux input	0.3	0	0	0	0	0	0.01	0
3	# Equivalent 2:1mux	0.8	0	0	0	5.3	8.9	0.01	0
4	# Register + # Equivalent 2:1mux	0	2.5	2.5	0	5.3	0.7	0.01	0
5	# transistor	0	0	0	0	5.3	0	0.01	0
6	# Register+# Mux input + # Unique net	0	0	0	0	5.3	0.7	0.01	0
7	Our layout model	0.3	0	0	0	0	0	0.01	0

(b)

FIGURE 9 Comparative study of the elliptic filter example with different design quality measures: (a) mux implementation, (b) bus implementation.

5. CONCLUSIONS

In this paper, we presented an extensive empirical study on a set of 16 designs of the elliptic filter benchmark. We have investigated the correlation between the traditional area cost functions, our proposed area model, and the actual layouts. The results show that our proposed area model accurately reflects the effects of datapath area tradeoffs. The results also show that traditional area quality measures are not good indicators for optimization in datapath synthesis.

We also addressed two main factors in quality measures: *accuracy* and *fidelity*. Intuitively, to obtain a high-accuracy layout quality measure in high-level synthesis requires to take into account all aspects of layout generation from a structural design. However, this very time-consuming quality measure procedure

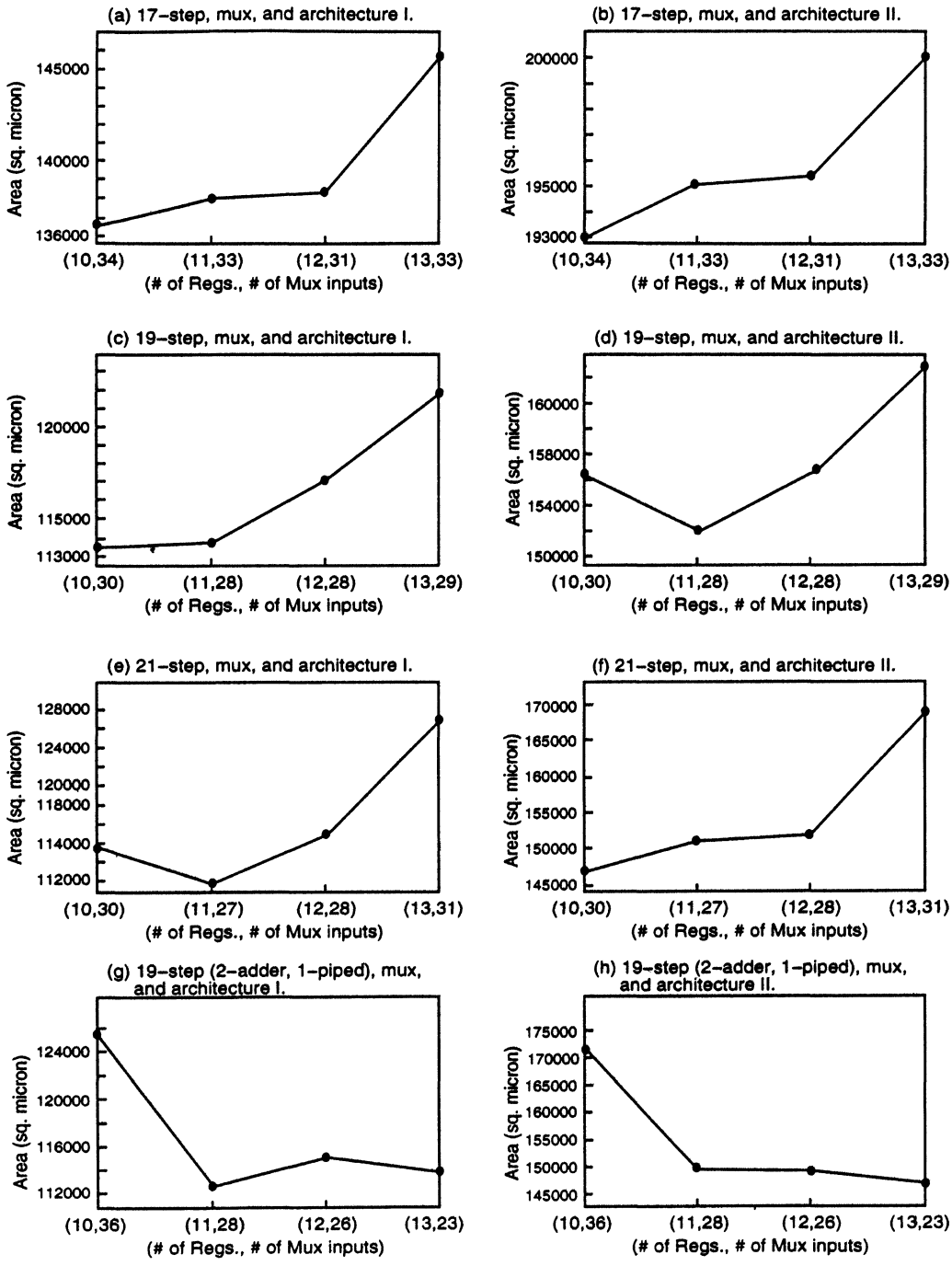

FIGURE 10 The results of the Elliptic Filter example with multiplex implementation.

is not suitable to provide design-tradeoff information during the synthesis process. To support decision-making for design tradeoffs during the synthesis process, we only need the indication about which design is better than others regardless how accurate the quality measure is. Thus, the fidelity of estimates is more important than the accuracy. In addition, to provide a fast design-tradeoff decision during the synthesis process, a simple quality measure model is needed. From above observation, a high-fidelity and effective quality measure is desirable in high-level synthesis.

In this study, we have focused on the area quality measures of datapaths with bit-sliced layouts. Other datapath layout architectures (e.g., multiple-metal routing and macro-cell layouts) and irregular-shaped datapaths are not considered in this study, which should be studied further. In addition, the quality measures in chip-level synthesis, including datapaths, control units, memories, and the impacts of floorplanning, should be studied further. Moreover, more benchmarkings should be done on a wide range of designs in order to validate the correlation between the proposed quality measures and the actual layouts.

Acknowledgments

The author would like to thank Prof. D. D. Gajski and V. Chaiyakul for their invaluable comments throughout the development of this work.

This paper was recommended by Professor F. Kurdahi.

References

[1] F. D. Brewer and D. D. Gajski, "Knowledge Based Control in Micro-Architecture Design," *Proc. 24th DAC*, pp. 203–209, 1987.

[2] R. J. Cloutier and D. G. Thomas, "The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm," *Proc. 27th DAC*, pp. 71–76, 1990.

[3] R. Camposano and W. Wolf, Editors, *High-Level VLSI Synthesis*, Kluwer Academic Publishers, 1991.

[4] V. Chaiyakul, A. C-H Wu, and D. D. Gajski, "Timing Models for High-Level Synthesis," *Proc. Euro-DAC*, pp. 60 65, 1992.

[5] S. Devadas and A. R. Newton, "Algorithms for Hardware Allocation in Data Path Synthesis," *IEEE Trans. on Computer-Aided Design*, vol. CAD-8, no. 7, pp. 768–781, 1989.

[6] A. El Gamal, "Two-Dimensional Stochastic Model for Interconnections in Master Slice Integrated Circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 127–138, 1981.

[7] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. 19th DAC*, pp. 175–181, 1982.

[8] M. Feuer, "Connectivity of Random Logic," *IEEE Trans. Computer*, vol. C-31, pp. 29–33, 1982.

[9] D. D. Gajski, N. D., Dutt, C. H. Wu and Y. L. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[10] B. S. Haroun and M. I. Elmasry, "Architectural Synthesis for DSP Silicon Compiler," *IEEE Trans. on Computer-Aided Design*, vol. 8, no. 4, pp. 431–447, April 1989.

[11] W. Heller et al., "Prediction of Wiring Space Requirement for LSI," *Proc. 14rd DAC*, pp. 20–22, 1977.

[12] C. Y. Huang, Y. S. Chen, Y. L. Lin and Y. C. Hsu, "Data Path Allocation Based on Bipartite Weighted Matching", *Proc. 27th DAC*, pp. 499–504, June, 1990.

[13] D. W. Knapp, "Feedback-Driven Datapath Optimization in Fasolt," *Proc. ICCAD90*, pp. 300–303, 1990.

[14] F. J. Kurdahi and A. C. Parker, "REAL: A Program for Register Allocation," *Proc. 24th DAC*, pp. 210–215, 1987. pp. 110–115, 1989.

[15] F. J. Kurdahi and A. C. Parker, "Techniques for Area Estimation of VLSI Layouts," *IEEE Trans. on Computer-Aided Design*, pp. 81–92, January 1989.

[16] F. J. Kurdahi and C. Ramachandran, "LAST: A Layout Area and Shape Function Estimator for High Level Applications," *Proc. EDAC91*, 351–355, 1991.

[17] T. A. Ly, W. L. Elwood, and E. F. Girczyc, "A Generalized Interconnect Model for Data Path Synthesis," *Proc. 27th DAC*, pp. 168–173, 1990.

[18] M. C. McFarland., "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions," *Proc. 23th DAC*, pp. 472–478, 1986.

[19] M. C. McFarland., "Reevaluating the Design Space for Register-Transfer Hardware Synthesis," *Proc. ICCAD87*, pp. 262–265, 1987.

[20] M. C. McFarland, A. C. Parker, and R. Camposano, "Tutorial on High-Level Synthesis," *Proc. 25th DAC*, pp. 330–336, June, 1988.

[21] A. Mignotte and G. Saucier, "A Generalized Model for Resource Assignment," *Fifth International Workshop on High-Level Synthesis*, pp. 37–43, 1991.

[22] B. M. Pangrle, F. D. Brewer, D. A. Lobo and A. Seawright, "Revevant Issues in High-level Connectivity Synthesis," *Proc. 28th DAC*, pp. 607–610, 1991.

[23] B. M. Pangrle, and D. D. Gajski, "Design Tools for Intelligent Silicon Compilation", *IEEE Trans. on Computer-Aided Design*, vol. CAD-6 no. 6, pp. 1098–1112, Nov. 1987.

[24] A. C. Parker, P. Gupta and A. Hussain, "The Effects of Physical Design Characteristics on the Area-Performance Tradeoff Curve," *Proc. 28rd DAC*, pp. 530–534, 1991.

[25] A. C. Parker, J. Pizarro and M. Mlinar, "MAHA: A Program for Datapath Synthesis," *Proc. 23rd DAC*, pp. 461–466, 1986.

[26] P. G. Paulin, J. P. Knight and E. F. Girczyc, "HAL: A Multi-Paradigm Approach to Automatic Data Path Synthesis," *Proc. 23rd DAC*, 263–270, 1986.

[27] M. Pedram and B. Preas, "Interconnection length estimation for Optimized Standard Cell Layouts," *Proc. ICCAD89*, pp. 390–393, 1989.

[28] M. Pedram and B. Preas, "Accurate Prediction of Physical Design Characteristics for Random Logic," *Proc. ICCD89*, 1989.

[29] C. Ramachandran and F. J. Kurdahi, "Combined Topologic and Functionality Based Delay Estimation Using A Layout-Driven Approach for High Level Applications," *Proc. Euro-DAC92*, 1992.
[30] C. Ramachandran, F. J. Kurdahi, D. D. Gajski, C. H. Wu, and V. Chaiyakul, "Accurate Layout Area and Delay Modeling for System Level Design," *Proc. ICCAD92*, pp. 355–361, 1992.
[31] S. Sastry, "Wireability Analysis of Integrated Circuits," Ph.D. dissertation, University of Southern California, Los Angeles, 1985.
[32] C. J. Tseng and D. P. Siewiorek, "Automated Synthesis of Data Path in Digital Systems," *IEEE Trans. on Computer-Aided Design*, vol. CAD-5, no. 3, pp. 379–395, 1986.
[33] "Data path Library," VLSI Technology, INC., 1988.
[34] J. P. Weng and A. C. Parker, "3D Scheduling: High-Level Synthesis with Floorplanning." *Proc. 28rd DAC*, pp. 668–673, 1991.
[35] A. C-H Wu, V. Chaiyakul and D. D. Gajski, "Layout Area Models for High-Level Synthesis," *Proc. ICCAD91*, pp. 34–37, 1991.
[36] A. C-H Wu and D. D. Gajski, "Layout-Driven Allocation for High Level Synthesis," Technical Report #91-30, Department of Information and Computer Science, University of California, Irvine, 1991.
[37] G. Zimmermann, "A New Area and Shape Function Estimation Technique for VLSI Layouts," *Proc. 25th DAC*, pp. 60–65, 1988.

1. APPENDIX I

Algorithm 1. Layout-driven binding.

Let
G = {V,E} be a data flow graph;
H = {V,E} be a hypergraph;
"count" be a given arbitrary number;
P be a set of feasible merging hyperedges;
F be a set of given functional units;
T be a set of control assignments;
R_x be a set of registers;

Hardware_Allocation(G,F,T,count) {
$R_x = \phi$;
while (count > 0) **do**
H = *Initial_Assignment*(G,R_x);
old_area = *layout_estimation(H)*;
/*Interchange optimization*/
no_more_improve = FALSE;
while (no_more_improve = FALSE) **do**
P = *locate_feasible_merging_hyperedge(H)*;
a_gain_merging = FALSE;
for ($\forall$ feasible_merging_hyperedge $e \in P$) **do**
/*relocate nodes associated with hyperedge e*/
H' = *relocate_node*(e);
new_area = *layout_estimation*(H');
if (new_area < old_area) **do**
$H = H'$;
old_area = new_area;
a_gain_merging = TRUE;
endif
endfor
if (a_gain_merging = FALSE) **then**
no_more_improve = TRUE;
endif
endwhile
/*incrementing one more register for next allocation iteration*/ count = count − 1;
if (count < 0) **then**
$R_x = R_x \cup$ register;
endif
endwhile

}

Author Biography

Allen C-H. Wu (S'84-M'93) received the B.S. degree in electronic engineering from Taiwan Institute of Technology, Taipei, Taiwan, in 1983, the M.S. degree in electrical and computer engineering from University of Arizona, Tucson, in 1985, and the Ph.D. degree in Computer Science from University of California, Irvine, in 1992.

From 1985 to 1988, he was a Research Engineer in the Physiology Department at University of Arizona, Tucson. He is currently an Associate Professor of Computer Science at Tsing Hua University, Hsin-Chu, Taiwan. His research interests include CAD for VLSI and VLSI design methodologies.