Layout Area Models for High-Level Synthesis

Allen C-H. Wu
Viraphol Chaiyakul
Daniel D. Gajski

Technical Report #91-31
November 3, 1991

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717
(714) 856-8059

Abstract

Traditionally, the common cost functions, the number of functional units, registers and selector inputs, are used in high level synthesis as quality measures. However, these traditional design quality measures may not reflect the real physical design. To establish quality measures based on the physical designs, we propose layout estimation models for two commonly used data path and control layout architectures. The results show that quality measures deriving from our models give an accurate prediction of the final layout. The results also show that traditional cost functions are not good indicators for optimization in high level synthesis.

Contents

List of Figures

1 Introduction

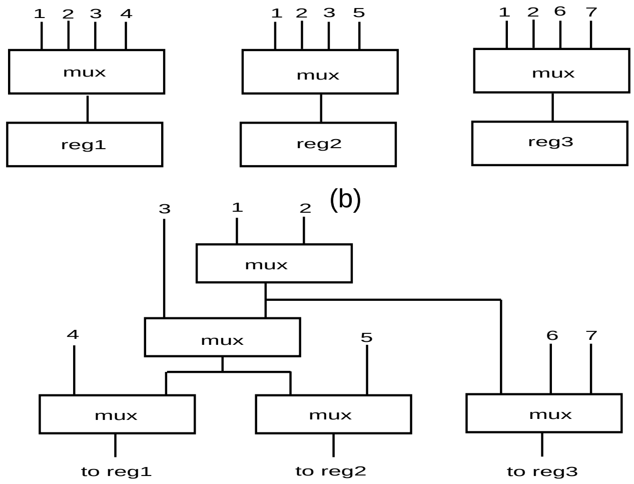
High-level synthesis for digital systems has been an active research topic in recent years. It usually consists of two phases: (i) scheduling and (ii) allocation. In the scheduling phase, operations are scheduled and assigned to control steps to satisfy timing and resource constraints. In the allocation phase, operation nodes are mapped into function units, storage elements are allocated for variables, and connections between function units and storage elements are allocated for performing the required data transfers.

Much research has been reported for solving the scheduling and allocation problems ([3],[5],[6],[9],[10],[11],[15],[18],[20],[21],[22],[23],[24],[27]). Very little synthesis research has taken into account physical design effects ([6],[19]). Most research efforts use a simplified model which assumes that the number of units, registers, and interconnect units are proportional to the layout areas. Since the relationships between the structural design and the physical design have not been established, the traditional design quality measurement may not reflect the real physical design.

Most high-level synthesis systems use the one-level-mux, multi-level-mux, or bus interconnect models ([6],[10],[18],[20],[22],[23],[27]) and measure the design quality with the number of selector (mux) inputs and registers. However, this cost function may not reflect the actual area and delay of the physical designs. To demonstrate this, we used a commercial datapath compiler (bit-sliced stack) [31] which included a set of multiplexers shown in Figure 1(a). Figure 1(b) shows an example with three registers where each register has four inputs. Using one-level-mux model, three 4:1 muxs with 12 mux inputs

mux	height(um)	width(um)	delay(ns) with 1pf load
2:1	80	43.2	7.35
3:1	80	68.8	7.2
4:1	80	80	7.25
5:1	80	102.4	7.85
6:1	80	114	8.9
7:1	80	144	7.9
8:1	80	155.2	7.9

(a)



(c)

Figure 1: Interconnect cost based on physical design

are required. On the other hand, when a three-level-mux model is used by merging the mux inputs, only 11 mux inputs are needed (Figure 1(c)). Based on the mux input count, the three-level-mux design seems to be better than the one-level-mux design. However, by taking into account the mux area and delay information in Figure 1(a), the total area for the one-level-mux design with one bit datapath is 19,200 μm^2 while the delay is 7.25ns. For the three-level-mux design with one bit datapath, the total area is 19,328 μm^2 and the delay is 21.45ns. As a result, the one-level-mux design is better than the three-level-mux design in terms of area and delay even though the one-level-mux design uses more mux inputs than three-level-mux design's. This example demonstrates that the number of mux inputs is not a good quality measures for data path optimization.

As another example considers two designs: (1) using two 2-input multiplexers and (2) using one 4-input multiplexer. Since both designs have four mux inputs, both designs have the same interconnect costs in terms of the mux inputs. However, using the physical information shown in Figure 1(a), the area of the two 2-input multiplexers is 6,912μm^2/bit which is larger than the area of the one 4-input multiplexer (6,400μm^2/bit). Again, the traditional interconnect cost function does not well predict for the layout area.

In this paper, we describe layout models for the data path and control logic. We also define new quality measures, the product of transistor and routing track density for layout area optimization. We demonstrate the superiority of our models and the new cost function by comparing it to the real layout area during design space exploration of the elliptic filter benchmark.

The remainder of this paper is organized as follows: Section 2 describes the data path, control logic, and memory area models. Section 3 describes the experiments and results. Finally, Section 4 summaries our approach.

2 Layout area models

We divide the chip into four parts: (1) Data path, (2) Control logic, (3) Macrocells, and (4) Memories. Data paths consist of regular structural components such as an adder/subtracter, ALU, MUX, or register. Control logic consists of a set of random gates or a PLA associated with the data path to perform required data transfer. Macrocells include some predefined components such as multipliers. Memories include RAMs and ROMs. In this section, we first describe the area estimation models for the data path and the control logic. Then we describe the area estimation models for the multiplier and the memories.

2.1 Data path layout model

There are two common layout architectures ([12],[17],[28],[25],[2],[26]) for data paths: (1) bit slice stack with abutment and (2) bit-sliced macrocells with routing channel where a macrocell represents a bit slice of microarchitecture units. The first and second layout architectures are shown in Figure 2(a) and (b) respectively. The first architecture uses abutment to connect different bit slices, and over-the-cell routing for connecting different units inside one bit slice. Different strips for P and N transistors are laid out horizontally. Data signals run vertically in second metal over the bit slices. Power, ground, and control lines are routed in first metal or poly between the bit slices. The stack grows horizontally when the bit-width increases, and vertically when the number of units increases. In the second layout architecture, bit-sliced macrocells or standard cells of each bit slice are placed vertically and a routing channel is used for connecting different cells inside one bit slice. In this architecture, power and control lines run horizontally in second metal while data lines run vertically in first metal or poly.

We first describe the area cost of the first architecture (bit slice abutment). In this architecture, each bit slice has a fixed number of over-the-cell routing tracks (Figure 3(a)). If the actual number of routing tracks used to connect units is less than or equal to the available routing tracks for a bit slice, then the stack width is equal to the width of the bit

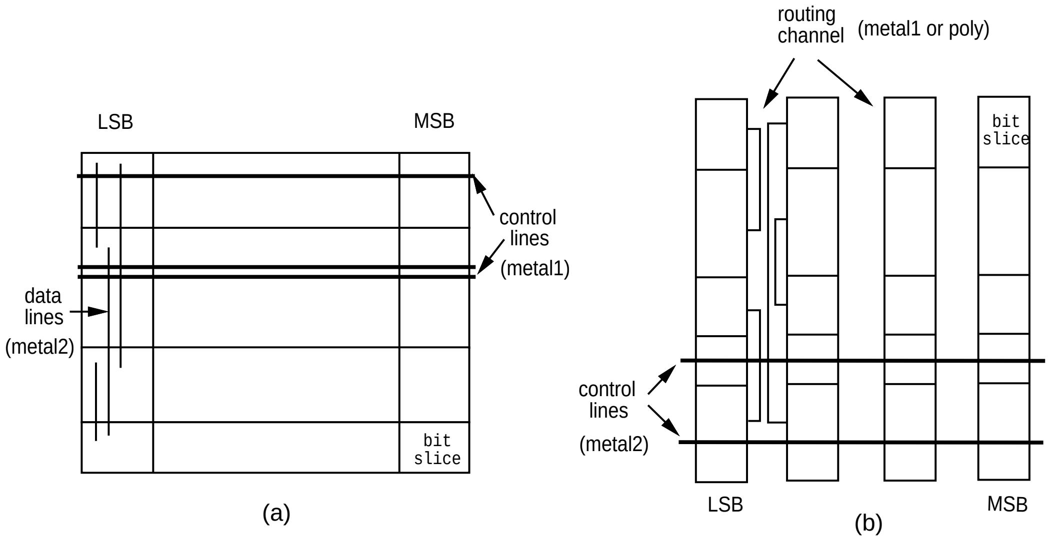


Figure 2: Two data path layout architectures

slice times the number of bit slices. The stack height is equal to the sum of the heights of all units in the stack which include functional units, registers, and interconnect units. On the other hand, if the actual routing track density used to connect units is higher than the number of available routing tracks, then extra routing area is required. Thus, the area cost is calculated as follows:

$$\mathbf{A_{total}} = \begin{cases} \mathbf{w}(\mathbf{A_{FU}} + \mathbf{A_{REG}} + \mathbf{A_{IU}}) & \text{if } \mathbf{Trk_{avail}} \geq \mathbf{Trk_{used}} \\ \mathbf{w}(\mathbf{A_{FU}} + \mathbf{A_{REG}} + \mathbf{A_{IU}} + \mathbf{A_{wire}}) & \text{if } \mathbf{Trk_{avail}} < \mathbf{Trk_{used}} \end{cases}$$

where

$\mathbf{A_{total}}$ is the total area of the data path;
$\mathbf{A_{FU}}$ is the single bit slice area of functional units;
$\mathbf{A_{REG}}$ is the single bit slice area of registers;
$\mathbf{A_{IU}}$ is the single bit slice area of interconnect units, multiplexers or tri-state buffers;
$\mathbf{A_{wire}}$ is the area of a routing channel;
$\mathbf{Trk_{avail}}$ is the available over-the-cell routing tracks of one bit slice;
$\mathbf{Trk_{used}}$ is the actual routing tracks required to connect units in one bit slice.

w is the bit widths of the data path;

For the second architecture (cells with routing channel), the area cost is the same as the described area cost; however, the available over-the-cell tracks in this architecture is equal to zero, that is $\mathbf{Trk_{avail}}$=0 (Figure 3(b)).

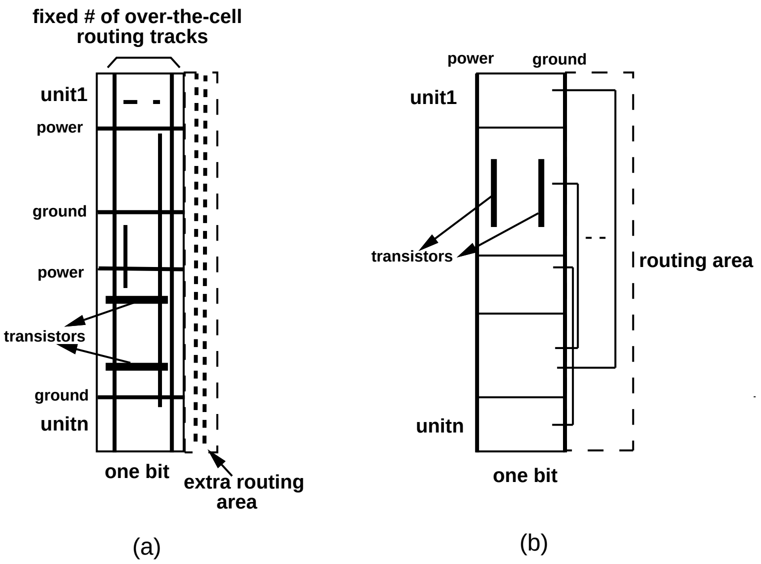


Figure 3: The area models for two layout architectures

The area of a bit slice consists of two parts: (1) the unit area in the bit slice and (2) the wiring area. In this section, we first describe the unit area cost. Then, we discuss the wiring area cost. Finally, we describe the overall area costs for the data paths.

2.1.1 Unit area

As described in the previous section, the unit area of one bit slice consists of three parts: (1) functional unit area, (2) register area, and (3) interconnect unit area. We use the transistor costs as the measures of unit area consumptions. Since the transistor costs of the functional units and registers can be obtained by examining a component library ([31]), we focus on the interconnect unit cost estimations, which consist of two models: (1) multiplexer and (2) bus. Using the bus model, a tri-state buffer is needed for each selector input in the RT design; thus, the transistor cost of the interconnect units is shown as follows:

$$\mathbf{trs(IU)} = \sum_{i=1}^{n} \mathbf{trs(tri\_buffer_i)})$$

where
n is the number of selector inputs in the RT design;
$\mathbf{trs(tri\_buffer_i)}$ is the number of transistors of a tri_buffer.

Using the multiplexer model, the area cost of the interconnect units depends on two factors: (1) area cost of each selector input and (2) area cost of the selector itself. We first describe the multiplexer structures and the correlations between transistor costs and multiplexer inputs. There are two common multiplexer implementations: (1) one select control line for each input (Figure 4(a)) and (2) $\lceil \log_2 \mathbf{n} \rceil$ control lines per $\mathbf{n}$ inputs (Figure 4(b)). The first mux has one optional decoder outside the sliced stack while the other has a decoder in each unit. The multiplexer consists of three parts: (1) the selector, (2) the output drivers, and (3) an optional internal decoder. Assume the selector inputs have the same selection circuit, the number of transistors of a multiplexer $\mathbf{j}$ is shown as follows:

$$\mathbf{trs(mux_j)} = \begin{cases} \mathbf{in(mux_j)} \times \mathbf{tr_1} + \mathbf{tr_2} & \textbf{one control per input} \\ \mathbf{in(mux_j)} \times \mathbf{tr_1} + \mathbf{2}^{\log\lceil \mathbf{in(mux_j)} \rceil} \times \mathbf{tr_3} + \mathbf{tr_2} & \textbf{decoding control inputs} \end{cases}$$

where

$\mathbf{in(mux_j)}$ is the number of inputs of mux $\mathbf{j}$;
$\mathbf{tr_1}$ is the number of transistors of each selector input;
$\mathbf{tr_2}$ is the number of transistors of the output driver.
$\mathbf{tr_3}$ is the number of transistors of each decoder input;

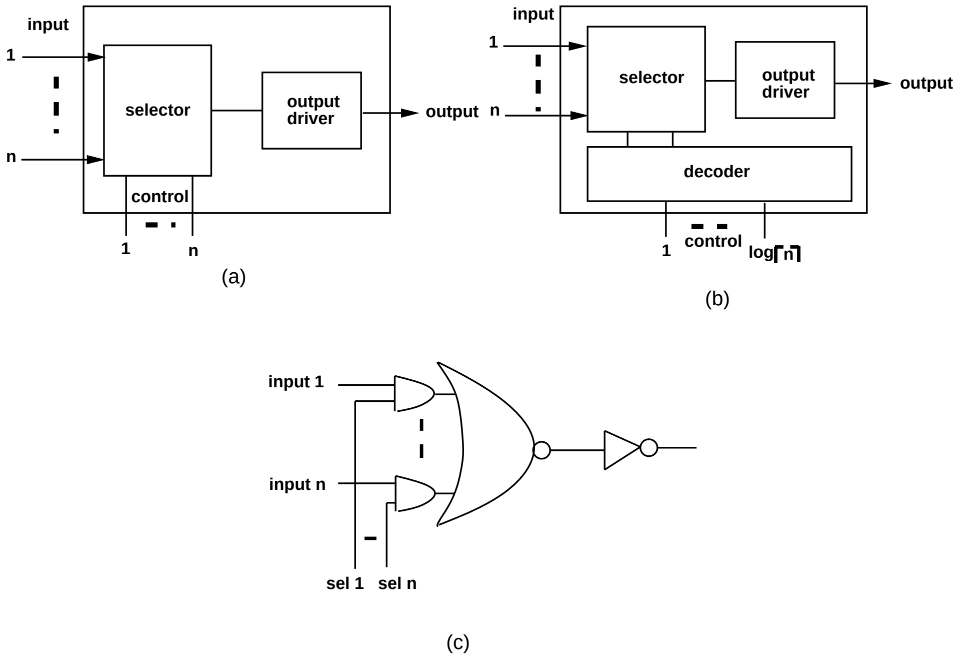


Figure 4: Two multiplexer structures

For example, Figure 4(c) shows a $\mathbf{n}$-input multiplexer with $\mathbf{n}$ control lines. In this multiplexer, there are four transistors for each input and two transistors for each output driver such that $\mathbf{tr_1}$ is equal to 4 and $\mathbf{tr_2}$ is equal to 2. Thus, the transistor cost of a 4-input multiplexer is 18.

Using the multiplexer model, the transistor cost of the interconnect units is shown as follows:

$$\mathbf{trs(IU)} = \sum_{j=1}^{m} \mathbf{trs(mux_j)}$$

where
m is the number of selector in the RT design.

2.1.2 Wiring area

The wiring cost can be described as the routing track density required to completely connect all nets in a bit slice. The actual density can only be determined after bit slice units have been physically placed. However, the placement procedure is often expensive in terms of computation time. Hence, we use an inexpensive wiring estimation method.

In our wiring estimation, the routing density is measured in terms of number of the routing tracks using the linear placement model of two described data path layout architectures. Given a netlist, we place components linearly in a single row. We first use the min-cut partitioning algorithm ([7],[13]) to perform component placement. Then, routing tracks are assigned to nets using the left-edge algorithm which explores the possibility of track sharing. The estimated number of routing tracks for a single bit slice is obtained and the wire cost is calculated as follows:

$$\mathbf{A_{wire}} = \beta \ (\mathbf{Trk_{used}} - \mathbf{Trk_{avail}})$$

where

$\mathbf{Trk_{used}}$ is the number of required routing tracks in a bit slice;
$\mathbf{Trk_{avail}}$ is the number of routing tracks available for over-the-cell routing;
β is the product of wire pitch times the stack height.

β is the area coefficient of a routing track which depends on the wire pitch and routing channel width. The wire pitch depends on the layout technology. For example, using a 3μm technology, the wire width is 3μm while the distance between two wires is 3μm. Therefore, the wire pitch of each routing track is 6μm.

2.1.3 The Overall data path layout model

Using the described area and wiring costs, we can derive the overall area cost of the data path as follows:

$$\mathbf{A_{total}} = \mathbf{w}(\alpha(\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{trs(FU_i)} + \sum_{\mathbf{j=1}}^{\mathbf{m}} \mathbf{trs(REG_j)} + \sum_{\mathbf{k=1}}^{\mathbf{p}} \mathbf{trs(IU_k)}) + \mathbf{A_{wire}})$$

where

$\mathbf{A_{total}}$ is the area cost of the data path;
α is the transistor area coefficient which correlates to the layout technology and the layout system;
$\mathbf{trs(FU_i)}$ is the number of transistors in functional unit $\mathbf{i}$;
$\mathbf{trs(REG_j)}$ is the number of transistors in register $\mathbf{j}$;
$\mathbf{trs(IU_k)}$ is the number of transistors in interconnect unit $\mathbf{k}$;
$\mathbf{n}$ is the number of functional units in one bit slice;
$\mathbf{m}$ is the number of registers in one bit slice;
$\mathbf{p}$ is the number of interconnect units in one bit slice;
$\mathbf{w}$ is the bit widths of the data path;

α is a transistor area coefficient (area/transistor) which is technology dependent and varies with different layout systems. For example, α is 220 μm^2/transistor using the 1.5μm technology component library [31], while α is 1,100 μm^2 using the 3μm technology component library [1]. This area coefficient α is obtained from the empirical studies using the targeted layout system and technology.

2.2 Control layout area model

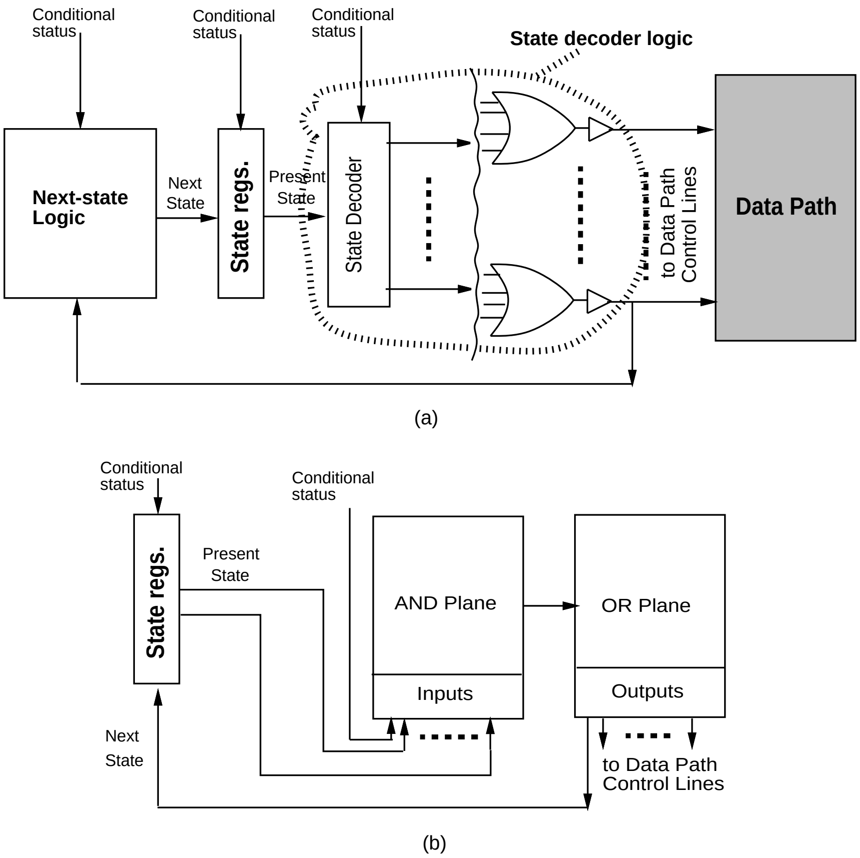


Figure 5: Two control logic layout architectures: (a) Random logic and (b) PLA implementations

In the control logic layout model, we focus on two control logic architectures: (1) Random logic and (2) PLA (Figure 5). Random logic includes gates, flip-flops, decoders, and drivers which are often laid out using standard or custom cells. On the other hand, PLA offers a regular structure for implementing control logic functions. A typical PLA uses an AND-OR structure combined with an input and output circuitries.

One major issue in control logic implementation is the state encoding scheme. Many researches([4],[29]) have focused their efforts to find an encoding scheme which can produce minimal layout area. Results obtained from these schemes vary in a wide range. Hence an accurate general control logic estimation is difficult to achieve. In our control logic layout model, we assume that states are encoded in binary value according to their state number. However, this assumption can be further improved upon the knowledge of encoding scheme uses in the actual layout implementation.

In this section, we first describe the random logic layout model. Then, we describe the PLA layout model. As mentioned earlier, we use the transistor costs as the measurement of area consumptions.

2.2.1 Random logic layout area model

We define the random logic model as a next-state logic, a state registers, and a state decoder logic (Figure 5a). The next-state logic consists of combinational gates which computes the next state from the present state and the conditional status. A set of state registers provides state sequencing effect. The state decoder logic uses the present state and the conditional status to determine the control signals of the data path.

The upper bound of the number of transistors needed in the next-state logic can be

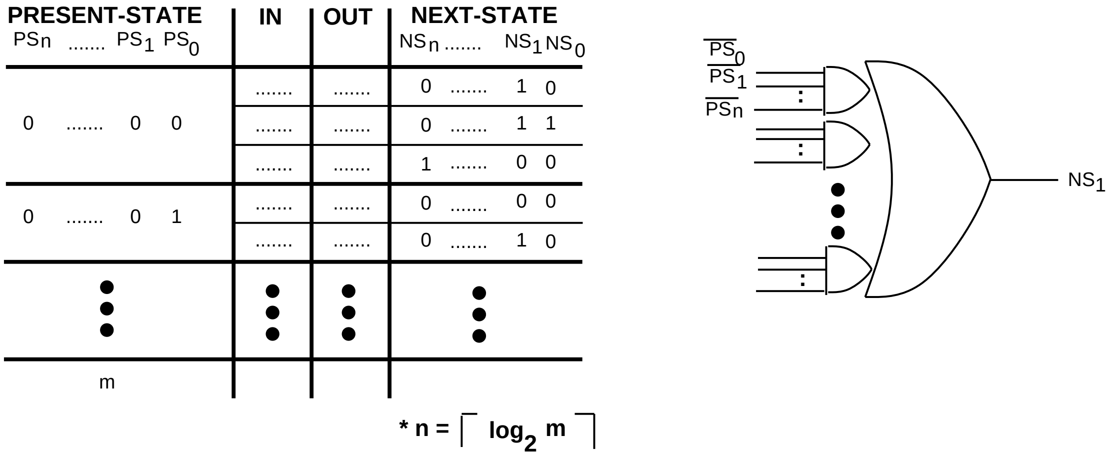


Figure 6: The next-state logic model

determined from the state transition table. Basically, we can express each encoded next-state bit, (NS_i), as a sum of product terms of encoded present-state bits, (PS_i), see Figure 6. To simplify the cost, we do not take into account the conditional status bits. The product terms can be implemented as AND gates while the overall sum can be implemented as an OR gate (Figure **??**). Thus, the next-state logic cost is calculated as follows:

$$\mathbf{trs(ns\_logic)} = \sum_{i=0}^{\lceil log_2 m \rceil} \mathbf{trs(NS_i)}$$

The state decoder logic can be further sub-divided into two parts: (1) a state decoder, and (2) a two-level OR-gate-driver logic. For a data path component with a set of control lines, a **n**-input OR gate is inserted in front of each control line where **n** is the number of time steps in which the particular control is activated(Figure 7(a)). For example, the register in Figure 7(b) loads **n=3** variables **var1, var2,** and **var3** on the time steps 1, 3, and 5 respectively. A **3**-input OR gate takes **3** control inputs for time steps 1, 3, and 5 from the state registers. For an **n**-input MUX (Figure 7(c)), if $\mathbf{n_1}$ variables are selected from mux input1 on $\mathbf{n_1}$ time steps, then a $\mathbf{n_1}$-input OR gate takes $\mathbf{n_1}$ control inputs. Similarly, an **n**-input OR gate takes **n** control inputs to select the add or substract operation of a functional unit (Figure 7(d)).

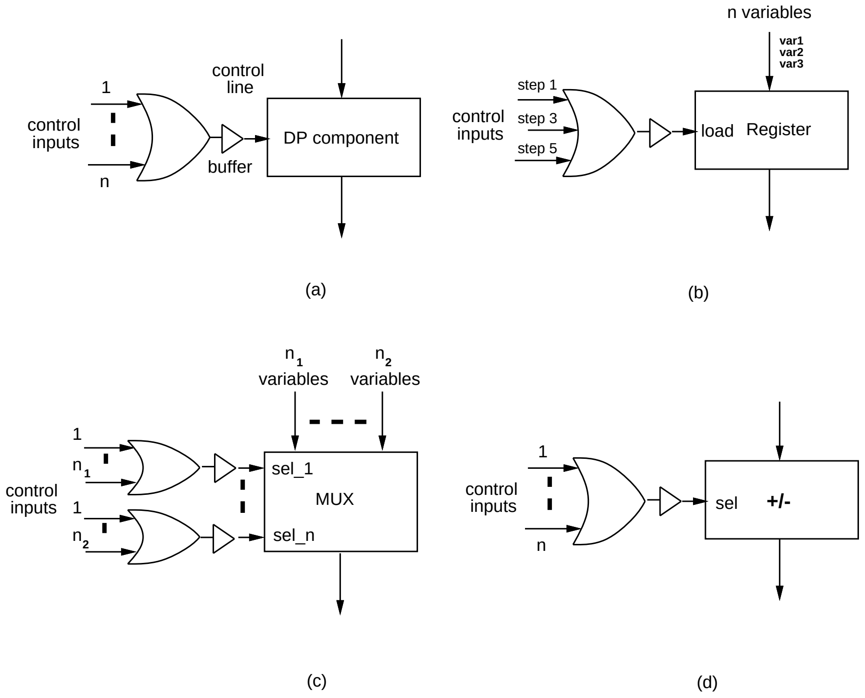


Figure 7: The control driver model

The transistor cost of the random logic implementation consists of five parts: (1) next-state logic costs(**trs(ns_logic)**), (2) state registers costs(**trs(state_reg)**), (3) state decoder costs (**trs(state_decoder)**, (4) OR-gate costs(**trs(OR)**), and (5) driver costs(**trs(driver)**). Where,

$$\mathbf{trs(state\_reg)} = \lceil \mathbf{log_2 m} \rceil \times (\mathbf{trs(register)})$$

$$\mathbf{trs(state\_decoder)} = \mathbf{trs(}\lceil \mathbf{log_2 m} \rceil : \mathbf{m\ DECODER)}$$

$$\mathbf{trs(OR)} = \sum_{i=1}^{n} \mathbf{trs(p\_input\ OR\ gate)}$$

and

$$\mathbf{p} \;\leq\; \begin{cases} \text{\# of control steps in which variables are loaded} \\ \qquad\qquad \text{if } \mathbf{i\ is\ a\ register\ control\ line} \\ \text{\# of control steps in which inputs are selected from mux input i} \\ \qquad\qquad \text{if } \mathbf{i\ is\ a\ mux\ select\ line} \\ \text{\# of control steps in which variables are loaded from the tri\_state buffer input} \\ \qquad\qquad \text{if } \mathbf{i\ is\ a\ tri\_state\ buffer\ control\ line} \\ \text{\# of control steps in which operation i is selected} \\ \qquad\qquad \text{if } \mathbf{i\ is\ a\ functional\ unit\ select\ line} \end{cases}$$

m is the number of control steps;
n is the number of control lines.

Thus, the total transistor cost **trs(RL)** is calculated as follows:

$$\mathbf{trs(RL)} = \mathbf{trs(ns\_logic)} + \mathbf{trs(state\_reg)} + \mathbf{trs(state\_decoder)} + \mathbf{trs(OR)} + \sum_{i=1}^{n} \mathbf{trs(driver)}$$

And the area cost of the random logic is calculated as follows:

$$\mathbf{A_{Random\_Logic}} = \alpha\ \mathbf{trs(RL)}$$

where

α is the transistor area coefficient.

2.2.2 PLA layout model

We use a common PLA layout implementation ([4],[29]) which performs state encoding techniques which yields minimum area in the final implementation. The layout model of a PLA consists of five parts: (1) inputs, (2) AND plane, (3) OR plane, (4) outputs, and (5) state registers which are described as follows:

(1). *Inputs.* For $\mathbf{m}$ control inputs, it requires $\log\lceil\mathbf{m}\rceil$ input drivers and state latches. The transistor cost of the inputs of a PLA is shown as follows:

$$\mathbf{trs(in)} = \lceil\log\mathbf{m}\rceil \times (\mathbf{trs(driver)} + \mathbf{trs(latch)})$$

(2). *AND plane.* The size of AND plane in a PLA is proportional to the number of control inputs. The transistor cost of the AND plane is shown as follow:

$$\mathbf{trs(AND)} = \lceil\log\mathbf{m}\rceil \times \mathbf{m}$$

(3). *OR plane.* The size of OR plane in a PLA is proportional to the number of control inputs as well as the number of control outputs. For $\mathbf{m}$ control inputs and $\mathbf{n}$ control outputs, the transistor cost of the OR plane is shown as follow:

$$\mathbf{trs(OR)} = \mathbf{m} \times \mathbf{n}$$

(4). *Outputs.* For each control output, a driver, a precharging cell, and an optional latch are required. For $\mathbf{n}$ control outputs, the transistor cost of the outputs of a PLA is shown as follow:

$$\mathbf{trs(out) = n \times (trs(driver) + trs(latch) + trs(precharge\_cell))}$$

(5). *State registers.* Since we assumed that states are encoded as binary value according to the state numbers, the transistor cost of the state registers can be defined as follow:

$$\mathbf{trs(state\_register) = \lceil log_2 m \rceil \times (trs(register))}$$

Thus, the total transistor cost of the PLA is:

$$\mathbf{trs(PLA) = trs(in) + trs(AND) + trs(OR) + trs(out) + trs(state\_register)}$$

and the area cost of the PLA is:

$$\mathbf{A_{PLA}} = \ \alpha\ \mathbf{trs(PLA)}$$

where
α is the transistor area coefficient.

2.3 Memory and multiplier layout area models

In this section, we first describe the RAM/ROM layout area model. Then, we discuss the register file layout area model. Finally, we describe the multiplier layout area model.

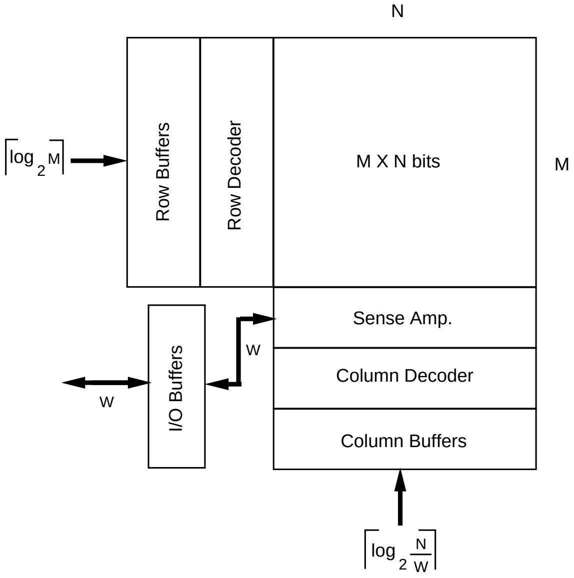


Figure 8: The RAM/ROM layout area model

2.3.1 RAM/ROM layout area model

Figure 8 shows the RAM/ROM model. For a M×N bits RAM/ROM, each row contains N-bit storages which are divided into N/W sections, where W is the word size. The row decoder selects N-bit out of M×N bits of storages. The column decoder selects W-bit out of N-bit of storages. Thus, the transistor cost of RAM/ROM is calculated as follows:

$$\mathbf{trs(Mem)} = \sum_{\mathbf{i=1}}^{\mathbf{M\times N}} (\mathbf{trs(Mem\_cell)}) + \mathbf{trs(row\_decoder)} + \mathbf{trs(column\_decoder)}$$

$$+ \sum_{\mathbf{j=1}}^{\mathbf{N}} \mathbf{trs(sense\_amp)} + \sum_{\mathbf{i=1}}^{\mathbf{q}} \mathbf{trs(buffer)}$$

where

$$\mathbf{q} = \lceil \mathbf{log_2 M} \rceil + \lceil \mathbf{log_2 N/W} \rceil + \mathbf{W}$$

$$\mathbf{Mem} = \{RAM, ROM\}$$

$$\mathbf{trs(row\_decoder)} = \mathbf{trs(\lceil log_2M\rceil : M\ DECODER)}$$

$$\mathbf{trs(column\_decoder)} = \mathbf{trs(\lceil log_2N/W\rceil : N/W\ DECODER)}$$

and the area cost is given as follows:

$$\mathbf{A_{Mem}} = \alpha\ \mathbf{trs(Mem)}$$

2.3.2 Register file layout area model

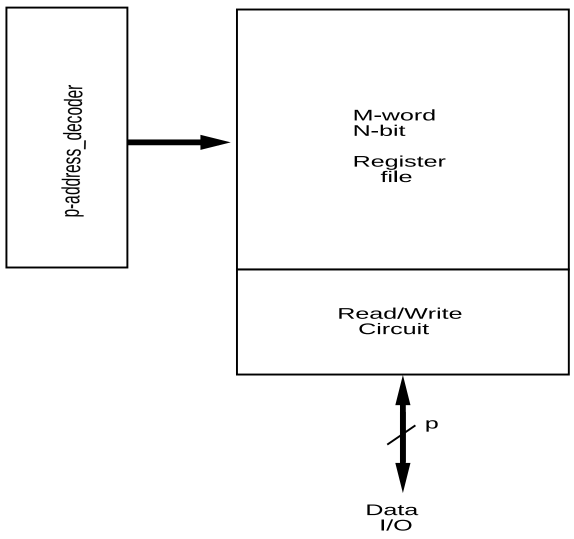


Figure 9: The register file layout area model

Figure 9 shows a p-port M by N bits register file which consists of a register array, a set of address decoders, and a read/write circuitry. For a p-port register file, it requires p sets of decoders to decode p register addresses simultaneously. Thus, the transistor cost of a register file is calculated as follows:

$$\mathbf{trs(RF)} = \sum_{\mathbf{i=1}}^{\mathbf{M\times N}} (\mathbf{trs(Reg\_cell)}) + \sum_{\mathbf{j=1}}^{\mathbf{p}} \mathbf{trs(decoder)} + \sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{trs(r/w\_cell)}$$

where

$$\mathbf{trs(decoder)} = \mathbf{trs}(\lceil \mathbf{log_2 p} \rceil : \mathbf{p\ DECODER})$$

and the area cost is given as follows:

$$\mathbf{A_{RF}} = \alpha\ \mathbf{trs(RF)}$$

2.3.3 Multiplier Layout area model

Area usage of the multiplier is estimated using a M by N S-stage pipeline parallel multiplier model (Figure 10), where M and N is the number of operand bits, and S is the number of pipeline stages.

The core array of the multiplier requires (M $\times$ N) AND gates, (M-1) half adders, and (M-1) $\times$ (N-2) full adders. Hence, the transistor estimation of the core cells array is given as follows:

$$\mathbf{trs(core\_array)} = (\mathbf{M} \times \mathbf{N})\mathbf{trs(AND)} + (\mathbf{M}-\mathbf{1})\mathbf{trs(Half\_adder)}$$

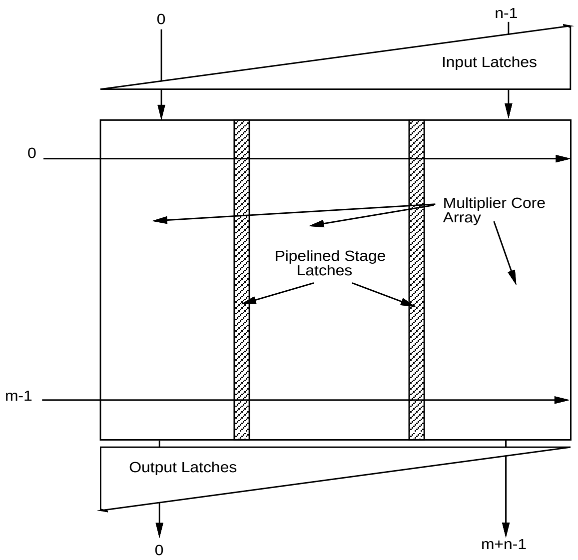


Figure 10: The pipelined multiplier layout area model

$$+(\mathbf{M-1}) \times (\mathbf{N-2})\mathbf{trs}(\mathbf{Full\_adder})$$

If S > 1, (S-1) columns of latches are inserted to store intermediate results of each pipeline stage. For each of the M rows, a total of 3 latches are required for storing the operand, the sum from the adder, and the carry from the adder. In addition, input and output latches are inserted for stage computations. Thus, the required latches is given as follows:

$$\mathbf{trs}(\mathbf{stage\_latch}) = (\mathbf{S-1})(\mathbf{3} \times (\mathbf{M-1}))\mathbf{trs}(\mathbf{latch})$$

$$\mathbf{trs}(\mathbf{input\_latch}) = \begin{cases} \sum_{\mathbf{i=1}}^{(\mathbf{S-1})} \lfloor(\mathbf{M-1})/\mathbf{S}\rfloor \times \mathbf{i} \times \mathbf{trs}(\mathbf{latch}) & \mathbf{if\ S > 1} \\ 0 & \mathbf{otherwise} \end{cases}$$

$$\mathbf{trs(output\_latch)} = \begin{cases} \sum_{\mathbf{i=1}}^{\mathbf{(S-1)}} \lfloor \mathbf{(N+M-1)/S} \rfloor \times \mathbf{i} \times \mathbf{trs(latch)} & \textbf{if } \mathbf{S > 1} \\ 0 & \textbf{otherwise} \end{cases}$$

Thus, the total transistor of the multiplier is calculated as follows:

$$\mathbf{trs(Multiplier) = trs(core\_array) + trs(stage\_latch)}$$

$$\mathbf{+trs(input\_latch) + trs(output\_latch)}$$

and the area cost is given as follows:

$$\mathbf{A_{Multiplier}} = \ \alpha\ \mathbf{trs(Multiplier)}$$

2.4 Complexity analysis

(1). *Data path estimation.* For a structural netlist of n components and m nets, unit area estimation takes $\mathbf{O}(n)$ time. The wiring estimation uses min-cut partitioning ([7],[13]) for unit placement and left edge algorithm for routing track assignment. Both take $\mathbf{O}$(mlogm) time.
(2). *Control estimation.* For the random logic implementation, it takes $\mathbf{O}(p)$ time where p is the number of the control lines. For the PLA implementation, it takes constant time.
(3). *Memory and macros* layout estimations take constant time.

3 Experiments and Results

We have tested our layout models on four different implementations of the elliptic filter benchmark (19-step with 2-adder and 1-piped multiplier, 21-step with 2-adder and 1-multiplier, 19-step with 2-adder 2-multiplier, and 17-step 3-adder and 2-piped multipliers) which were collected from the literature ([6],[10],[23],[24]). We first performed allocation [30] trading off registers and interconnect units on each example. Then, we used GDT [1] tools to generate the layouts. Figure 11 shows the data path of a 16-bit elliptic filter using data path architectures I and II respectively. Figures 12 and 13 show the final layouts of a 16-bit, 19-step, 2-adder, 1-piped multiplier, and 10-register elliptic filter example with PLA and random logic implementations respectively.

In the data path area estimation, we used the single-level interconnect model and performed multiplexer and bus implementations for all cases. The area and transistor information of the bit slices and macro cells are obtained from a VTI data path library [31] which uses a 1.5-μm technology. Figure 14 shows the results of the multiplexer implementation and Figure 15 shows the results of the bus implementation. Since the multiplier is treated as a macrocell, its area is constant throughout all examples. Therefore, the multiplier area is not included. For example, row 1 of Figure 14(a) is described as follows: The 17-step elliptic filter design uses 3-adder and 2-piped multipliers. This design uses 10 registers and 11 selectors with 34 inputs. This design consists of 552 transistors and 27 nets for each bit slice. The final layout of this design contains 11 routing tracks and our track estimation is 15. Using layout architecture I, the estimated area is 129,680μm^2 and the actual area is 136,720μm^2 per bit, and the ratio of estimated and actual areas is 0.95. Using layout architecture II, the estimated area is 213,625μm^2 and the actual area is 193,117μm^2 per bit, and the ratio of estimated and actual areas is 1.11. Figures 16 and 17 (a)-(b), (c)-(d), (e)-(f), and (g)-(h) show the relationships between estimated and actual areas using the multiplexer and bus implementations with layout architectures I and II of 17-step, 19-step (2-adder and 2-multiplier), 21-step, and 19-step (2-adder and 1-piped multipler) respectively.

For the data path architecture I, we use the sliced layout architecture [16] which has 13 over-the-cell routing tracks for each bit slice. Therefore, if the routing tracks are less than or equal to 13, then the area estimations are solely dependent on the number of transistors in the designs. The results in Figure 14 and 15 show that our layout models can predict the actual area with an average of 90% accuracy.

In the total area estimation including data path, control, and multiplier, we have experimented with a 16-bit, 19-step, 2-adder, and 1-piped multiplier elliptic filter example. Using data path architecture I, we implemented two control logic models, PLA and random logic, along with two interconnect models, bus and mux. Since multiplier is treated as a macrocell, the area of multiplier is obtained directly from the component library. Figures 18(a) and (b) show the results of mux and bus implementations respectively. The results show that our layout models can predict the actual area (1) of the data path with 10% error, (2) of the PLA with 18% error, (3) of the random logic with 16% error, and (4) of the total area with 6% error.

Using the layout results, we also investigated the relationships between the structural designs and the physical designs for traditional quality measures. **The results show that neither the design with the minimal registers nor the design with the minimal interconnect units can predict the minimal area** which are described as follows:

(1) The designs with the minimal number of registers do not produce the minimal area,

such as:

(i) 21-step, mux implementation, and architecture I (Figure 16(g)).

(ii) 19-step and 19-step with 2-adder and 1-piped multiplier, mux implementation, and architecture II (Figure 16(d)(h)).

(iii) 21-step and 19-step with 2-adder and 1-piped multiplier, bus implementation, and architecture I (Figure 17(e)(g)).

(iv) 21-step and 19-step with 2-adder and 1-piped multiplier, bus implementation, and architecture II (Figure 17(f)(h)).

(2) The designs with the minimal number of mux inputs do not produce the minimal area, such as:

(i) 19-step with 2-adder and 1-piped multiplier, mux implementation, and architecture I (Figure 16(g)).

(ii) 21-step, mux implementation, and architecture I (Figure 16(f)).

(3) The design which produces the minimal area for one layout architecture does not guarantee the minimal area for another layout architecture. For example, using both of the mux and bus implementations, the 21-step design (Figure 16(e)) with 11 registers and 27 bus inputs produces the minimal area using layout architecture I but not layout architecture II (Figure 16(f)).

The results in Figures 16 and 17 show that using our layout models, the estimated areas do accurately reflect the actual areas. The only two exceptions are the design with 17-step and 19-step with bus and architecture I (Figure 17(a) and (c)). For both cases, the errors are caused by over-estimated routing tracks using our track estimation.

4 Conclusions

In this paper, we demonstrated that the traditional design quality measures, using register and selector input counts, do not well reflect the real layout. We established novel layout area estimation models for data path and control logic. Our models formulate layout area estimation as a function of transistor and wiring costs. Our models are flexible in that we can use different area coefficients α and β to predict layout areas for different layout technologies and layout systems. The results show that our models can accurately predict physical layout. Furthermore, the layout estimation can be computed in a $\mathbf{O(mlogm)}$ time complexity which allows us to explore design space in high level synthesis rapidly and efficiently.

5 Acknowledgements

This work was supported by NSF grant ♯MIP-8922851, California MICRO grant ♯90-046, and contributions from Rockwell International, Western Digital, and Silicon Systems Inc. We are grateful for their support. The authors also like to thank Tedd Hadley and L. Ramachandran for their useful discussions.

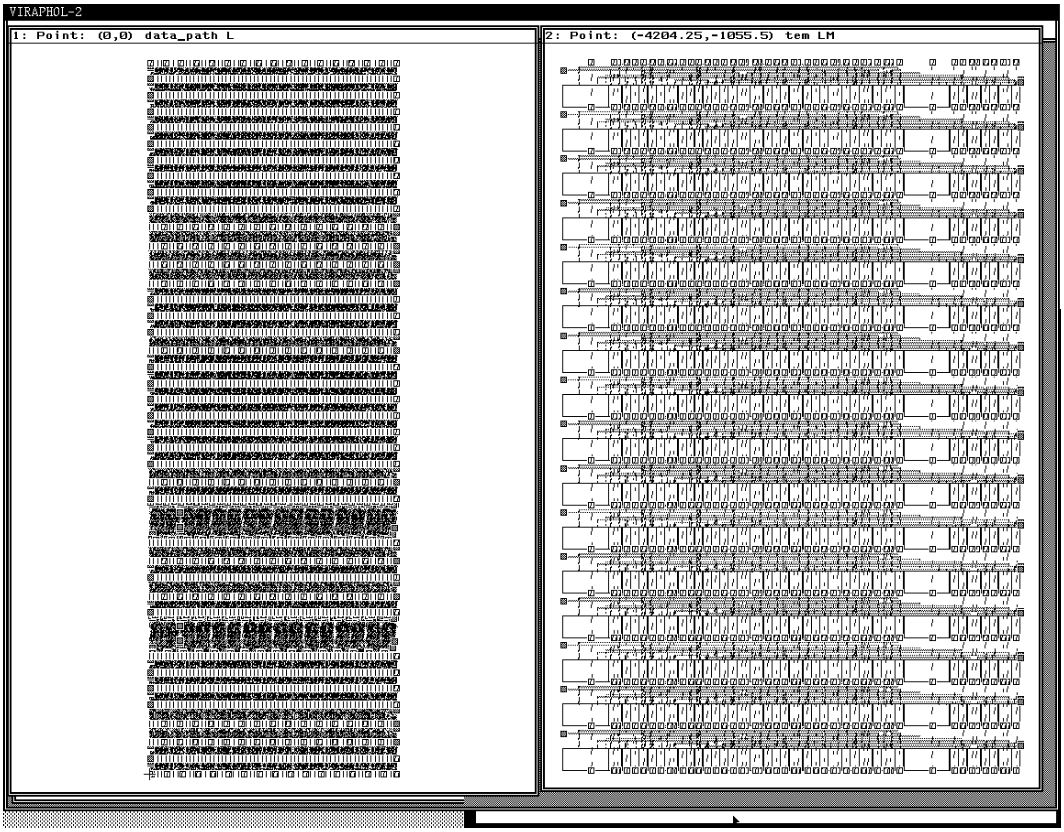


Figure 11: The data path layouts of a 16-bit elliptic filter example: (a) Architecture I and (b) Architecture II (multiplier is not included).

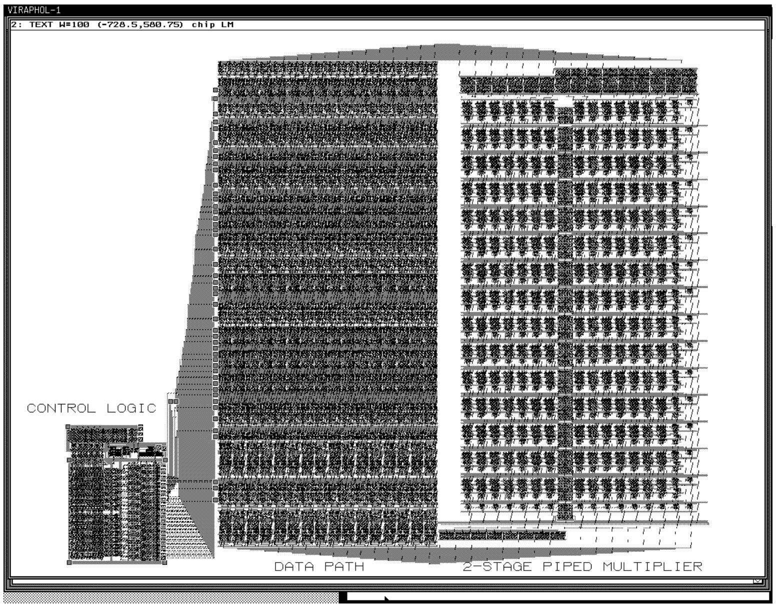


Figure 12: The final layouts of a 16-bit elliptic filter example with PLA implementation.

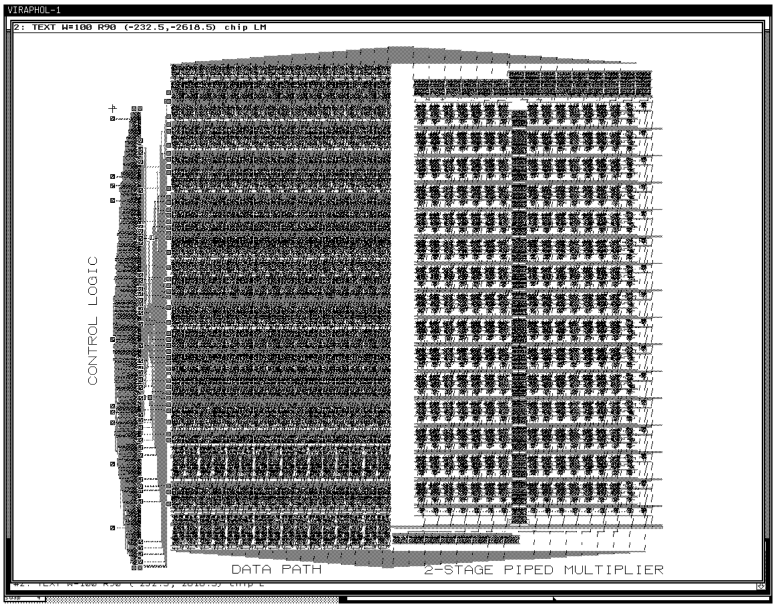


Figure 13: The final layouts of a 16-bit elliptic filter example with random logic implementation.

**** Area not including multiplier**

Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I			Layout Architecture II		
								Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual	Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual
17	3	2-piped	10	11 / 34	552	27	11(15)	129,680	136,720	0.95	213,625	193,117	1.11
17	3	2-piped	11	10 / 33	564	27	11(12)	124,080	138,080	0.90	200,216	195,038	1.03
17	3	2-piped	12	8 / 31	572	27	11(12)	125,840	138,480	0.91	200,796	195,490	1.03
17	3	2-piped	13	9 / 33	604	28	10(15)	143,653	145,040	0.98	226,625	199,430	1.14

(a)

Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I			Layout Architecture II		
								Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual	Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual
19	2	2	10	8 / 30	472	23	10(11)	103,840	113,440	0.92	166,234	156,420	1.03
19	2	2	11	6 / 28	480	22	9(10)	105,600	113,760	0.93	156,420	151,726	1.03
19	2	2	12	6 / 28	500	23	9(11)	110,000	117,280	0.94	165,658	156,862	1.06
19	2	2	13	6 / 29	524	24	9(10)	115,280	122,080	0.94	167,860	163,282	1.03

(b)

Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I			Layout Architecture II		
								Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual	Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual
21	2	1	10	7 / 30	480	20	8(11)	105,600	113,536	0.93	160,369	146,464	1.09
21	2	1	11	5 / 27	480	19	10(12)	105,600	111,456	0.95	161,611	151,836	1.06
21	2	1	12	5 / 28	504	19	9(11)	110,880	115,296	0.96	162,855	152,934	1.06
21	2	1	13	6 / 31	540	23	9(11)	118,800	126,736	0.94	179,014	168,235	1.06

(c)

Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Layout Architecture I			Layout Architecture II		
								Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual	Est. Area (um^2 bit)	Actual Area (um^2 bit)	Est. / Actual
19	2	1-piped	10	10 / 36	508	23	10(11)	111,760	125,376	0.89	177,093	170,976	1.04
19	2	1-piped	11	6 / 28	482	20	9(11)	106,040	113,136	0.94	159,804	150,045	1.07
19	2	1-piped	12	6 / 26	492	21	8(10)	108,240	115,696	0.94	159,082	149,272	1.07
19	2	1-piped	13	5 / 23	490	21	8(11)	107,800	113,696	0.95	160,595	146,672	1.09

(d)

Figure 14: The results of the elliptic filter example with multiplexer implementation.

**** Area not including multiplier**

								Layout Architecture I			Layout Architecture II		
Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual
17	3	2-piped	10	11 / 34	776	26	11(15)	182,888	162,240	1.13	259,600	229,164	1.13
17	3	2-piped	11	10 / 33	784	28	10(14)	174,526	163,680	1.07	255,750	225,060	1.14
17	3	2-piped	12	8 / 31	780	28	9(13)	171,600	162,720	1.05	242,063	217,638	1.11
17	3	2-piped	13	9 / 33	824	29	8(12)	181,280	168,960	1.07	244,976	219,648	1.11

(a)

								Layout Architecture I			Layout Architecture II		
Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual
19	2	2	10	8 / 30	672	21	8(10)	147,840	136,960	1.08	199,176	172,912	1.15
19	2	2	11	6 / 28	688	22	7(9)	151,360	136,000	1.11	197,260	171,700	1.15
19	2	2	12	6 / 28	688	23	9(10)	151,360	139,840	1.08	203,800	187,036	1.09
19	2	2	13	6 / 29	720	24	8(10)	158,400	146,080	1.08	200,860	189,904	1.06

(b)

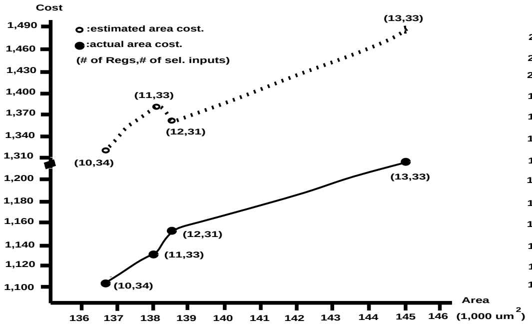
								Layout Architecture I			Layout Architecture II		
Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual
21	2	1	10	7 / 30	672	20	10(10)	147,840	136,896	1.08	199,176	186,816	1.07
21	2	1	11	5 / 27	656	19	8(8)	144,320	133,536	1.08	184,380	172,464	1.07
21	2	1	12	5 / 28	688	20	7(8)	151,360	137,376	1.10	192,572	172,446	1.12
21	2	1	13	6 / 31	744	23	9(8)	163,680	153,216	1.07	209,644	203,652	1.03

(c)

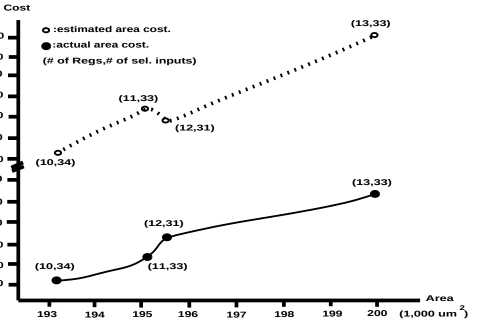
								Layout Architecture I			Layout Architecture II		
Control Steps	# of +	# of *	# of Reg.	# of Sel. / # Sel. Inputs	#trs.	#nets	#trks. Actual (est.)	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual	Est. Area (um² bit)	Actual Area (um² bit)	Est./Actual
19	2	1-piped	10	10 / 36	744	21	10(13)	163,680	148,896	1.10	225,099	203,316	1.11
19	2	1-piped	11	6 / 28	668	19	7(9)	146,960	133,536	1.10	191,706	167,598	1.14
19	2	1-piped	12	6 / 26	664	20	8(9)	146,080	132,576	1.10	196,824	171,216	1.11
19	2	1-piped	13	5 / 23	648	20	8(8)	142,560	129,216	1.10	181,324	166,848	1.09

(d)

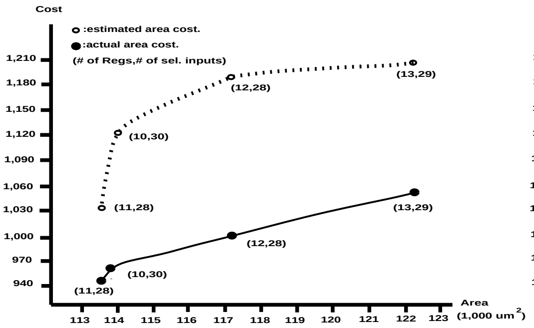
Figure 15: The results of the elliptic filter example with bus implementation.



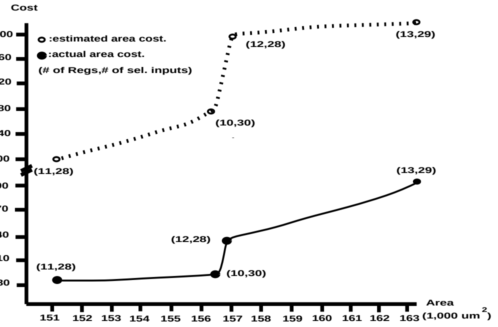
(a). 17-step, mux, and architecture I.



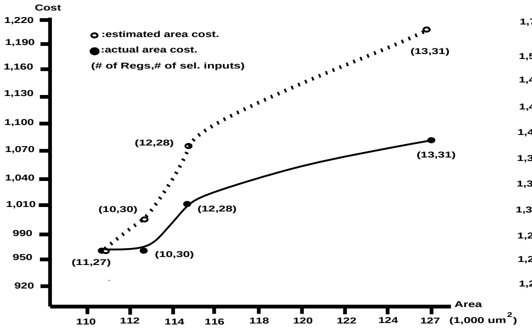
(b). 17-step, mux, and architecture II.



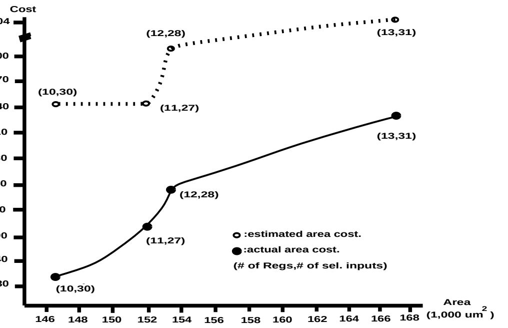
(c). 19-step, mux, and architecture I.



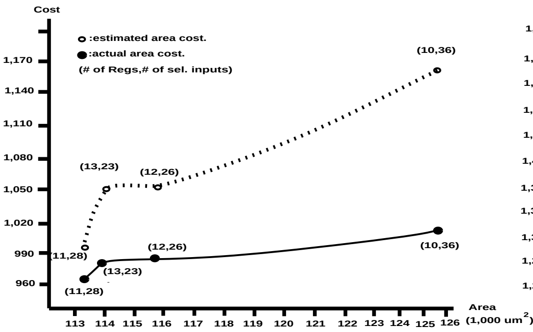
(d). 19-step, mux, and architecture II.



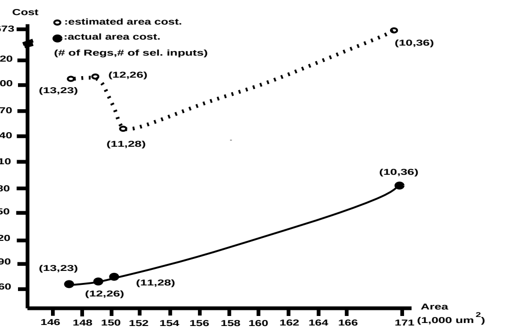
(e). 21-step, mux, and architecture I.



(f). 21-step, mux, and architecture II.



(g). 19-step(2-adder, 1-piped), mux, and architecture I.



(h). 19-step(2-adder, 1-piped), mux, and architecture II.

Figure 16: The relationships between estimated area and actual areas using multiplexer implementation.

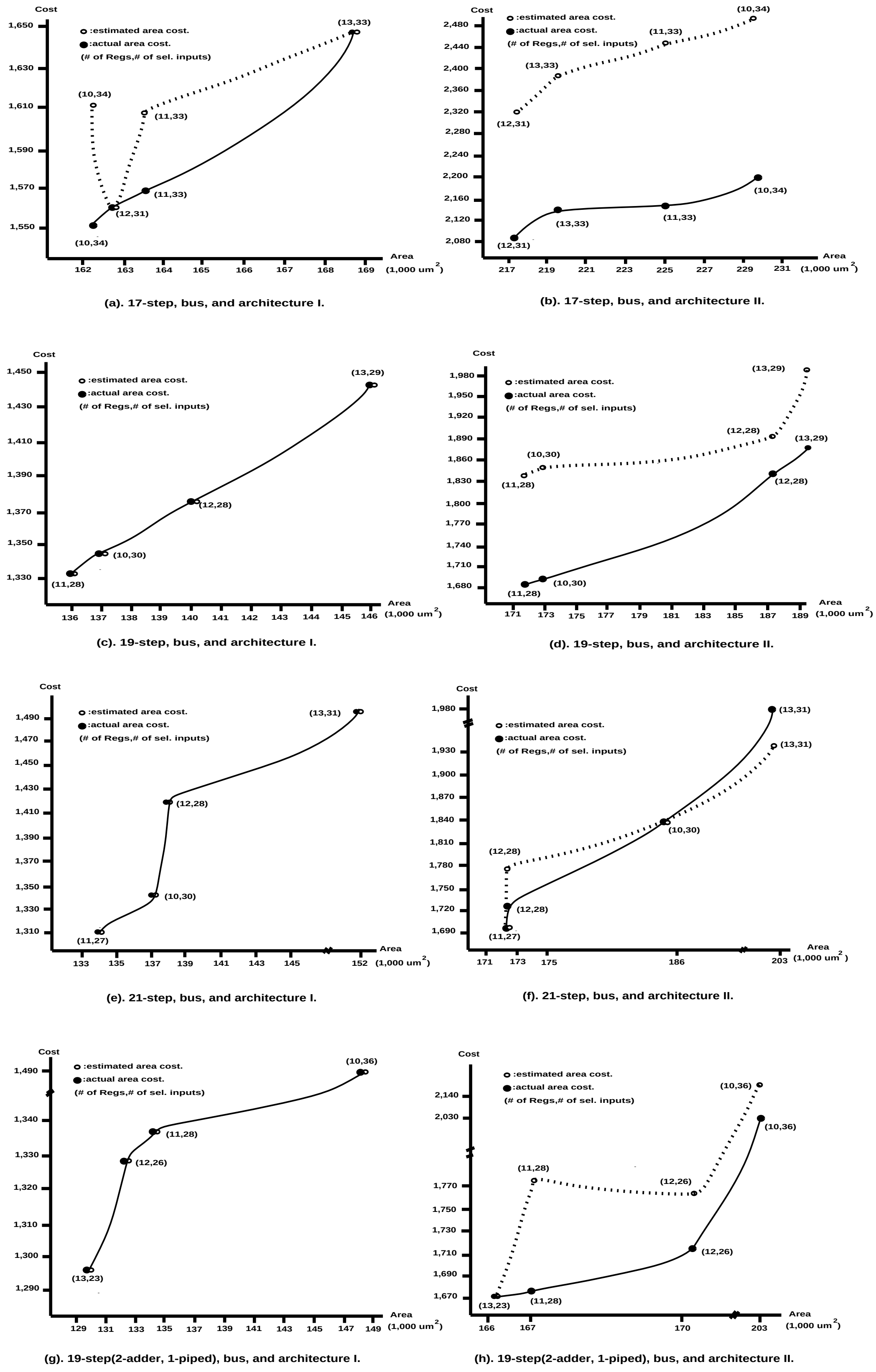


Figure 17: The relationships between estimated area and actual areas using bus implementation.

			Architecture I			Control Logic						Total Area					
# of Reg.	# of Sel. / # Sel. Inputs	Multiplier	Data Path Area (um^2)			PLA Area (um^2)			Random Logic Area (um^2)			PLA Area (um^2)			Random Logic Area (um^2)		
		Area (um^2)	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B
10	10 / 36	2,330,880	1,788,160	2,006,016	0.89	332,600	312,256	1.07	278,400	255,352	1.09	4,667,064	4,649,152	1.01	4,612,864	4,592,248	1.01
11	6 / 28	2,330,880	1,696,640	1,810,176	0.94	315,000	267,540	1.18	248,400	230,082	1.08	4,435,064	4,408,596	1.01	4,368,464	4,371,138	0.99
12	6 / 26	2,330,880	1,731,840	1,851,136	0.94	312,200	266,228	1.17	229,200	196,616	1.16	4,545,544	4,448,244	1.02	4,462,544	4,378,632	1.02
13	5 / 23	2,330,880	1,724,800	1,819,136	0.95	306,600	259,116	1.18	229,200	200,889	1.14	4,559,144	4,409,132	1.04	4,481,744	4,350,905	1.03

(a)

			Architecture I			Control Logic						Total Area					
# of Reg.	# of Sel. / # Sel. Inputs	Multiplier	Data Path Area (um^2)			PLA Area (um^2)			Random Logic Area (um^2)			PLA Area (um^2)			Random Logic Area (um^2)		
		Area (um^2)	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B	Est.(A)	Actual(B)	A:B
10	10 / 36	2,330,880	2,618,880	2,382,336	1.10	332,600	312,256	1.07	278,400	255,352	1.09	5,282,360	5,025,472	1.05	5,228,160	4,968,568	1.05
11	6 / 28	2,330,880	2,351,360	2,136,576	1.10	315,000	267,540	1.18	248,400	230,082	1.08	4,997,240	4,734,996	1.05	4,930,640	4,697,538	1.05
12	6 / 26	2,330,880	2,337,280	2,121,216	1.10	312,200	266,228	1.17	229,200	196,616	1.16	4,980,360	4,718,324	1.06	4,897,360	4,648,712	1.05
13	5 / 23	2,330,880	2,280,960	2,067,456	1.10	306,600	259,116	1.18	229,200	200,889	1.14	4,921,000	4,657,452	1.06	4,843,600	4,599,225	1.05

(b)

Figure 18: The overall area estimation of the elliptic filter example with (a) Mux and (b) Bus implementation.

6 References

[1] M. R. Buric and T. G. Matheson, "Silicon Compilation Environments," Proc. CICC, 1985.

[2] H. Cai, S. Note, P. Six, and H. De Man, "A data Path Layout Assembler for High Performance DSP Circuits," *Proc. 27th DAC.*, pp.306-311, 1990.

[3] 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.

[4] S. Devadas et al, "MUSTANG: State Assignment ofr Finite State Machines for Multi-Level Logic Implementations," *Proc. ICCAD* pp. 16-19, 1987.

[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] E. Dirkes Lagnese and D. E. Thomas, "Architectural Partitioning for System Level Design," *Proc. 26th DAC*, pp. 62-67, 1989.

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

[8] 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.

[9] C. Y. Hitchcock and D. E. Thomas, "A method of Automatic Data Path Synthesis," *Proc. 20th DAC*, 1983.

[10] C. Y. Huang, Y. S. Chen, et. al., "Data Path Allocation Based on Bipartite Weighted Matching", *Proc. 27th DAC*, pp. 499-504, June, 1990.

[11] K. S. Hwang, A. Casavant, et. al., "Scheduling and Hardware Sharing in Pipelined Data Paths", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, Nov. 1989.

[12] Jamier, R. and Jeraya, A., "APOLLON: A Datapath Compiler," *Proc. ICCD*, 1985.

[13] K. H. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell System Technical Journal*, vol. 49, no. 2, pp. 291-307, Februry, 1970.

[14] D. W. Knapp, "Feedback-Driven Datapath Optimization in Fasolt," *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1990.

[15] F. J. Kurdahi and A. C. Parker, "REAL: A Program for REgister ALlocation," *Proc. 24th DAC*, pp. 210-215, 1987.

[16] L. L. Larmore, D. D. Gajaki, and A. C. Wu, "Layout Placement for Sliced Architecture," *IEEE Trans. on Computer-Aided Design*, Oct., 1991.

[17] Luk, W. K. and Dean, A. A., “Multi-Stack Optimization for Data-Path Chip (Microprocessor) Layout,” *Proc. 26th DAC*, pp.110-115, 1989.

[18] T. A. Ly, W. Lloyd Elwood, and E. F. Girczyc, “A Generalized Interconnect Model for Data Path Synthesis,” *Proc. 27th DAC*, pp.168-173, 1990.

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

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

[21] N. Park, and A. Parker, "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications", *IEEE Trans. on Computer-Aided Design*, vol. CAD-7 no. 3, March 1988.

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

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

[24] P. G. Paulin, and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASICs", *IEEE Trans. on Computer-Aided Design*, vol. CAD-8 no. 6, June 1989.

[25] Petersen, B. R., White, B. A., Salomon, D. J. and Elmasry, M. I., “SPIL: A Silicon Compiler with Performance Evaluation,” *Proc. ICCAD*, pp. 500-503, 1986.

[26] M. T. Trick and S. W. Director, “Lassie: Structure to Layout for Behavioral Synthesis Tools,” *Proc. 26th DAC.*, pp.104-109, 1989.

[27] 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.

[28] Varinot P, J., and Courtois B, J., “Principles of The SYCO Compiler,” *Proc. 23rd DAC.*, pp. 715-721, 1986.

[29] D. Varma and E. A. Trachtenberg, “A Fast Algorithm for the Optimal State Assignment of Large Finite State Machines,” *Proc. ICCAD*, pp. 152-155, 1988.

[30] Allen C-H Wu and D. D. Gajski “Layout-Driven Hardware Allocation in Data Path Synthesis,” Tech. Rpt. No. 91-30, ICS Dept., UC Irvine, 1991.

[31] “Data path Library,” VLSI Technology, INC., 1988.