Timing Models for High-Level Synthesis

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

Technical Report #91-70
October 31, 1991

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

Abstract

In this paper, we describe a timing model for clock estimation during high-level synthesis. In order to obtain realistic timing estimates, the proposed model considers all delay elements, including datapath, control and wire delays, and several technology factors, such as layout architecture, technology mapping, buffers insertion and loading effects. The experimental results show that this model can provide much better estimates than previous models. This model is well suited for automatic and interactive synthesis as well as feedback-driven synthesis where performance matrices must be rapidly and incrementally calculated.

Contents

List of Figures

1 Introduction

High-level synthesis generates a structural design that implements a given behavioral description of a system and satisfies design constraints such as performance, and area. Quality measures are needed to support high-level synthesis in two ways. First, measures are needed to determine the quality of the final synthesized design. In this way, the measures allow comparisons of the final design with given constraints and identify critical spots in the design. For example, delay measures of all register-transfer operations in the design are used to determine the minimal period of the system clock and pin point the critical register-transfer operations. Secondly, quality measures are needed to guide high-level synthesis tools in the search of the design space. For example, if the clock period of a design is too long, a nonpipelined multiplier must be replaced with a pipelined multiplier.

Traditionally, design performance for high-level synthesis has been characterized by such measures as the clock frequency, the number of instructions or the number of additions/multi plications executed in a second. These individual measures do not characterize the actual design performance with respect to a particular description. Instead, the performance of a design for a particular description is measured by the total time needed to execute that description. Since a description may contain loops without fixed bounds and "if" statements in which "then" and "else" branches require different amounts of time, the execution time of such a description will depend on the input data. Thus, the execution time for any description is equal to the product of the number of control steps needed to execute the description and the duration of the control step (usually a clock period). Hence, the clock period is a good performance measure for a scheduled behavioral description with a fixed number of states.

In high-level synthesis, clock-period measures (or performance measures) are typically performed at two levels: control/data flow graph and RT-structure. Jain et. al. [6, 7] use area/time models to predict design tradeoffs from the data-flow graph. Their models consider only functional units and do not include registers, interconnect, wires and control units. BUD [10] determines the clock period by finding the worst datapath delay. It considers registers, multiplexers, functional units and wire delays. The wire length is obtained from the floorplan and the wiring delay is computed using a simple RC model. SPAID [4] also determines the clock period by the worst datapath delay; however, it does not take into account wiring delay. Chippe [1] estimates clock period by examining control delay (using a PLA model) and datapath delay, but wiring delay is not considered.

In general, none of the previous timing models consider all delay elements, including control delay, datapath delay and wiring delay, nor do they take into account technology factors such as layout architecture and technology mapping. Furthermore, they do not consider the impact of the logic and layout synthesis tools that are used to produce the final layout. Within the logic and layout synthesis tools, various optimization procedures, such as state encoding, logic minimization, and buffers insertion, are used to improve the performance or to minimize the area of the design. In one extreme, these procedures aim to produce designs with the smallest possible area. In the other extreme, their goal is to produce designs with the highest performance by reducing the clock period.

In this paper, we present a simple timing model for estimating the lower bound of the clock period, and the total execution time from the structural design (control-state table and datapath). The lower bound of the clock period is defined as the shortest clock period that can be obtained by applying all performance optimization procedures. The model considers all delay elements and several technology factors, such as layout architecture, technology mapping and loading effect, for delay estimation.

The remainder of this report is organized as follows: In section 2, we describe the elementary delay models. Section 3 introduces definitions of notations that are used in this report. Section 4 describes timing models for each constituent of a chip. From these models, we derived timing models of the system clock in section 5. In section 6, we present results obtained from our experiments using eight different designs and design styles of a high-level synthesis benchmark. Finally, in section 7, we present the conclusions from our research effort, discuss applications of our model and how our model could be improved.

2 Electrical Models

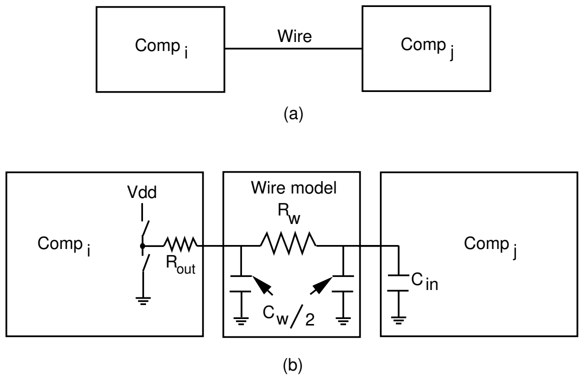


Figure 1: Wire: (a) RT model, (b) equivalent RC delay model.

The lumped RC model, also called the Elmore delay model [PeRu81], is widely used for delay calculation. In the lumped RC model, the propagation delay along a path from the start point to the end point, (t_p(start,end)), is computed as a product of lumping all of the resistances R_j and capacitances C_k along the path, that is,

$$t_p(\text{start}, \text{end}) = \sum_j R_j \times \sum_k C_k. \tag{1}$$

We can use Equation 1 to obtain delay of a connecting wire between two components as shown in Figure 1(a). In CMOS technology we model a component with its input capacitance (C_{in}) and its output resistance (R_{out}), as shown in Figure 1(b). For the connecting wire, we use the well known π-model with input capacitance ($C_w/2$), wire resistance (R_w) and the output capacitance ($C_w/2$). Since a wire is a thin sheet of metal of fixed thickness defined by the fabrication process consisting of rectangular segments, the wire resistance is equal to the product of the sheet resistance (R_s), in Ohm/square, and the ratio of the wire length (L_w) and wire width (W_w), that is,

$$R_w = R_s(\frac{L_w}{W_w}). \tag{2}$$

The wire capacitance (C_w) is equal to the product of the wire area and the ratio of the dielectric constant (ε) and the wire thickness (t), that is,

$$C_w = (L_w W_w)(\frac{\varepsilon}{t}). \tag{3}$$

Using Equations 2 and 3, we can compute the propagation delay ($t_{wire}(net_k)$) of a wire (net_k) used by a component ($comp_i$) to drive load components ($comp_j$, $1 \leq j \leq n$) as

$$t_{p\text{-}wire}(net_k) = (R_{out}(comp_i) + R_w)(C_w + \sum_{j=1}^{n} C_{in}(comp_j)). \tag{4}$$

Thus, the delay for signals from input of $comp_i$ to propagate to input of $comp_j$ via net_k can be formulated as

$$t_p(comp_i, net_k) = t_{p\text{-}gate}(comp_i) + t_{p\text{-}wire}(net_k) \tag{5}$$

where $t_{p\text{-}gate}(comp_i)$ is the internal delay of component $comp_i$ defined by the library of the targeted layout system.

2.1 The Delay of Combinational Circuits

Each combinational circuit consists of one or more components. The delay of a combinational circuit is the propagation delay on the critical path through the circuit, that is, the longest delay from any input to any output. We use the multistage graph algorithm [5] to identify the critical path of a combinational circuit. In short, the netlist of a combinational circuit is first transformed into a directed graph; each component is mapped to a node, a connection between two gates is mapped to a directed edge with the direction corresponds to the signal flows, inputs are mapped to a common source, and outputs are mapped to a common sink. Subsequently, $t_p(comp_i, n_i)$, Equation 5, for each net (n_i) is computed and assigned as a cost to its corresponding edge. Finally, a critical path is determined by finding the longest path from the source to the sink of the graph.

Figure 2 shows an example that illustrates the critical path's identification process of a combinational circuit. The equivalent multi-stage graph of the circuit in Figure 2(a) is shown in Figure 2(b). The highlighted path in Figure 2(b) is the longest path from the source to the sink. This path corresponds to the critical path in the circuit from the input I_3 to the output O_2 that goes through components A, C, E and F and nets n_1, n_4, n_6, n_7 and n_8, Figure 2(c).

Thus, the propagation delay of a combinational circuit, $t_{p\text{-}comb}$, from a set of inputs, I, to a set of outputs, O, can be computed as

$$t_{p\text{-}comb}(I, O) \quad = \sum_{comp_i \in COMP(I_c, O_c)} t_{p\text{-}gate}(comp_i) +$$

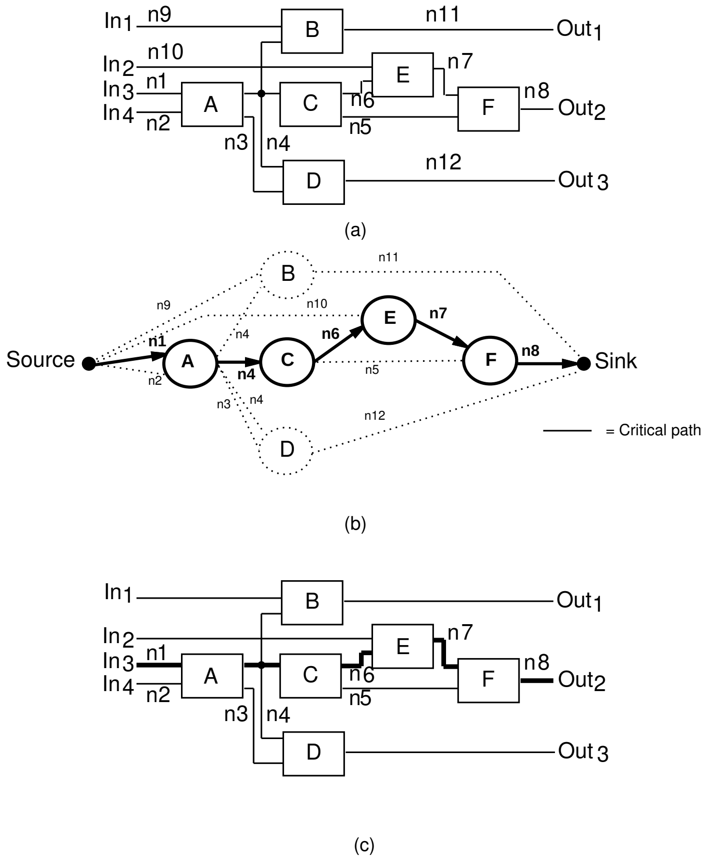


Figure 2: Timing model of the combinational circuit: (a) an example circuit, (b) its equivalent multi-stage graph, and (c) the correspondent critical path.

$$\sum_{net_j \in CONN(I_c,O_c)} t_{p\text{-}wire}(net_j) \tag{6}$$

where,

I_c is an input that lies on the critical path,

O_c is an output that lies on the critical path,

$COMP(I_c, O_c)$ and $CONN(I_c, O_c)$ define all components and all nets on path from I_c to O_c.

2.2 The Delay of Storage Elements

Storage elements are typically composed of latches and flip-flops. In a common implementation of a latch, we can identify two different delay paths: the data-to-output delay (t_{DQ}) and the clock-to-output delay (t_{CQ}), as shown in Figure 3.

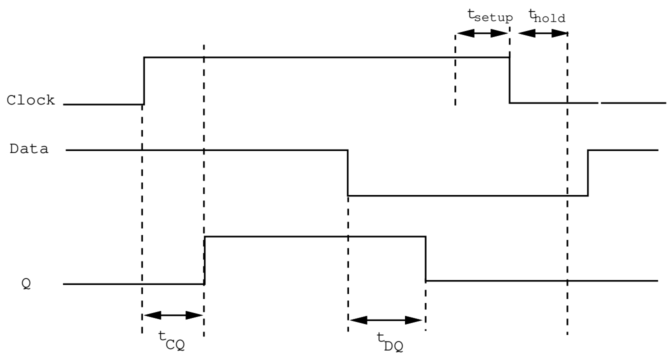


Figure 3: Delays and constraints of a common latch.

In order to ensure proper operation of the latch, two timing constraints are imposed: the setup time (t_{setup}) and hold time (t_{hold}). The setup and hold times specify time intervals before and after clock transitions during which a data signal must be stable to ensure proper operation of the latch (Figure 3). It is obvious from Figure 3 that,

$$t_{DQ} = t_{setup} + MAX(t_{CQ}, t_{hold}). \tag{7}$$

Master-slave (MS) and edge-triggered (ET) flip-flops respond to the transition of the clock signal. In this section, let us assume that in the MS flip-flop, the master latch is active when the clock signal is high while the slave latch is active when the clock signal is low. The output of the MS flip-flop changes after the high-to-low transition of the clock and the propagation delay (t_{CQ}) of the slave latch. Thus, the propagation delay of the MS flip-flop is equal to t_{CQ} of the slave latch. Since the data signal must propagate through the master latch before a high-to-low clock transition and still satisfy t_{setup} of the slave latch,

$$t_{setup}(MSFF) = t_{DQ}(Master\ latch) + t_{setup}(Slave\ latch). \tag{8}$$

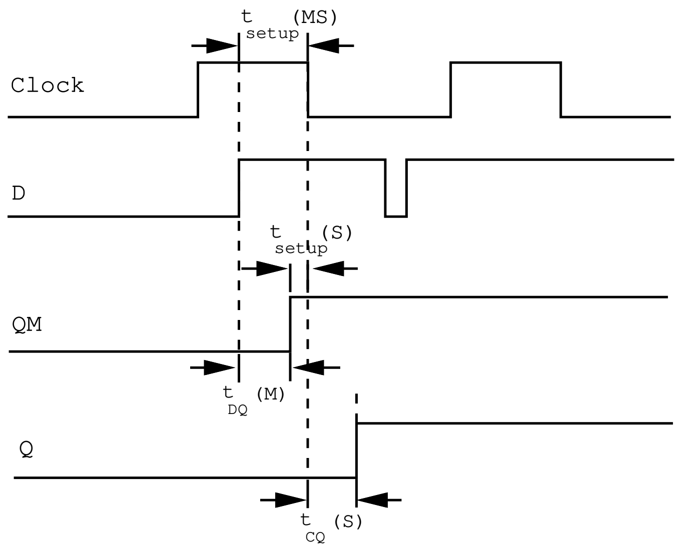


Figure 4: Delay and constraints of a master-slave flip-flop.

Note that the master latch (and hence the MSFF) is not sensitive to changes in the input data while the clock is low. For example, the glitch in the *Data* while clock is low does not affect the output Q of the MSFF (Figure 4). Hence the hold time for the MS flip-flop is equal to the hold time (t_{hold}) of the master latch. We can formulate similar definitions for an ET flip-flop which propagates data on the low-to-high transition of the clock signal.

3 Definitions and Notations

The delay computation is a hierarchical procedure. To compute delay of a signal, we can use Boolean equations or go as far as the physical material level. The more detail we take into account, the more likely we will obtain an accurate result, but with a higher computation complexity. In our layout timing models, the lowest level of timing informations are abstracted from the gate, the wiring resistance, and the wiring capacitance levels. However, if the timing delay of a block of gates, such as macrocells, memories, and functional units are available, we will utilize those informations.

Each gate in the library of the targeted layout system is accompanied by the following informations:

$t_{p\text{-}gate(X)}$ is the internal propagation delay of an X gate, where X is the type of a gate (e.g., 2-input AND, 4-input OR),

$R_{out(X)}$ is the output resistance of an X gate, and

$C_{in(X)}$ is the input capacitance of an input to an X gate.

From the same layout system, we assume that the following wiring informations are provided:

$R_{s(l)}$ is the sheet resistance of layer l routing in Ohm/square,

$C_{s(l)} = \frac{\varepsilon_l}{t_l}$ where,

> $C_{s(l)}$ is the sheet capacitance of layer l routing in pF/square,
>
> ε_l is the dielectric constant of the insulating material between the plates in layer l routing, and
>
> t_l is the insulator thickness for layer l routing.

$Width(l)$ is the nominal width of layer l routing.

Furthermore, throughout the discussion of timing models for control unit, we use the following notations:

S is the number of states in the state table,

C is the number of conditional status signals,

I is the number of inputs to the control unit,

$\mathcal{O}$ is an output of the control unit (i.e., a control or next-state signal), and

$\mathcal{O}_{prod}$ is the number of product terms for the output signal $\mathcal{O}$.

4 Timing models for constituents of a chip

We model the layout of a chip as connected blocks of logic, Figure 5. Each block belongs to one of the following four classes: (1) control logic, (2) datapath, (3) macrocell, and (4) memories. A data path unit consists of a set of regular structured components such as adders/subtracters, ALUs, MUXs, and registers. A control-logic unit consists of a set of random gates or a PLA associated with the datapath to perform required data transfer. Macrocells include some predefined components, such as multiplier and barrel shifter. Memories include register files, RAMs, and ROMs.

Consider a critical path for a signal O that is highlighted in Figure 5. Using Equation 6, the delay of this path can be computed as the sum of the the delay through each logic block and the delay of each inter-block wiring. Thus, in this section we will first introduce timing models for different classes of the logic block, then followed by timing models for the inter-block wiring. Section 3.1 describes timing models for the control logic unit. Section 3.2

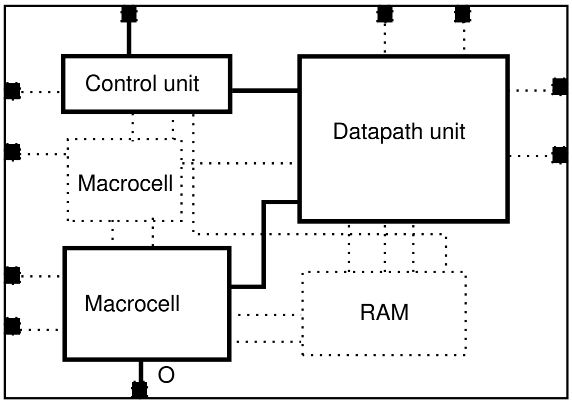


Figure 5: Constituents of a chip.

introduces timing models for units in the datapath and wiring between those units. Macrocells and memories are usually available in a library as predesigned blocks. Hence, timing information for macrocells and memories is assumed to be provided by the library. Subsequently, section 3.3 describeds timing models for inter-block wiring that uses floorplanner to determine the wire length.

4.1 Timing Models for the Control Unit

There are two commonly used layout architectures for a control unit: random logic and PLA. Since the timing information for a PLA is usually provided by its generator, in this section, we will assume the random-logic layout-architecture when describing the timing model for a control unit.

A control unit can be described by a control state-table that specifies next-state and control signals as a function of present states and conditional/status signals. We assume that the present states are encoded as binary values $p_k ... p_1 p_0$, where $k = \lceil \log_2 S \rceil - 1$ and S is the number of states. Similarly, next sta tes are encoded as binary values $n_k ... n_1 n_0$ (Figure 6(a)). Thus, the total number of inputs to the control unit I equals $\lceil \log_2 S \rceil + C$, where C is the number of conditional/status signals.

In reality, a number of optimization procedures, such as state minimization, logic minimization, and buffers insertion, are applied in order to improve the performance of the control logic. We consider the impact of these optimization procedures by deriving a simplified model that is geared toward estimation of the lower bound delay of the control logic. In our model, each next-state and control signal is represented as sum of products of the present-state and conditional/status signals, as shown in Figure 6(b). The product term is implemented with AND gates and the sum with OR gates. However, the target component library will usually provide AND and OR gates with a limited number of inputs. Thus, to

INPUT					OUTPUT			
I1	I2	I3	I4	I5	O1	O2	O3	O4
Present state		Conditional status bits			Next state		Control signals	
ps1	ps0	c0	c1	c2	ns1	ns0	s0	s1
0	1	1	0	0	1	0	0	1
1	0	1	0	1	1	0	1	0
1	0	0	0	1	1	1	1	1
1	1							

(a)

$$O1 = (\overline{I1}\, I2\, I3\, \overline{I4}\, \overline{I5}) \vee (I1\, \overline{I2}\, I3\, \overline{I4}\, I5) \vee (I1\, \overline{I2}\, \overline{I3}\, \overline{I4}\, I5)$$

$$O2 = (I1\, \overline{I2}\, \overline{I3}\, \overline{I4}\, I5)$$

$$O3 = (I1\, \overline{I2}\, I3\, \overline{I4}\, I5) \vee (I1\, \overline{I2}\, \overline{I3}\, \overline{I4}\, I5)$$

$$O4 = (\overline{I1}\, I2\, I3\, \overline{I4}\, \overline{I5}) \vee (I1\, \overline{I2}\, \overline{I3}\, \overline{I4}\, I5)$$

(b)

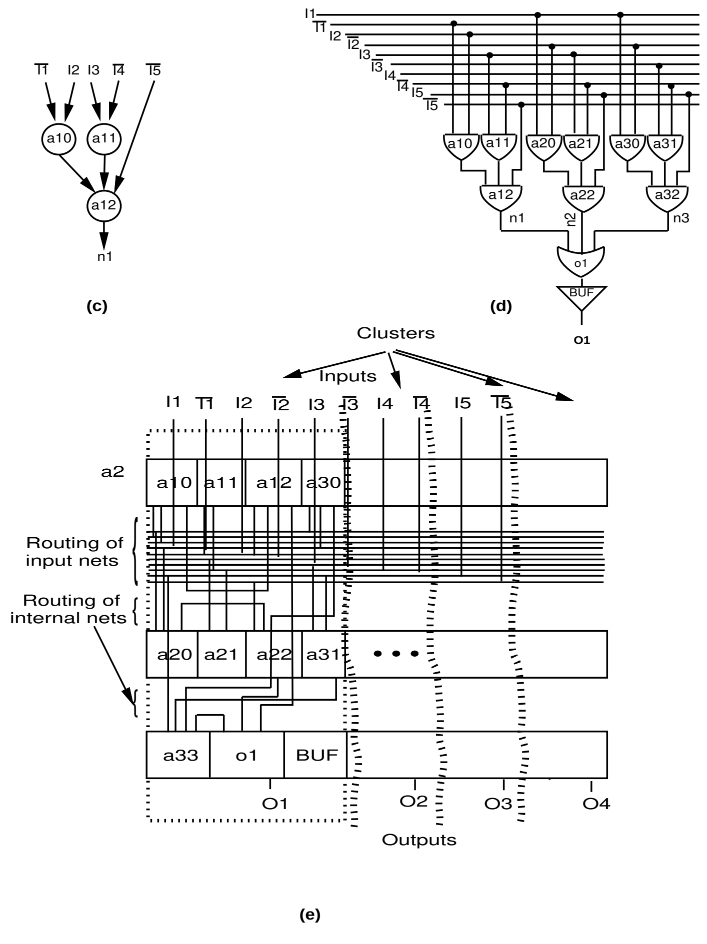


Figure 6: Random-logic model: (a) encoded control state-table, (b) sum-of-product expressions, (c) decomposition of a product term, (d) a multi-level implementation, (e) the layout model.

realize the impact of the technology mapping, the sum and product terms need to be decomposed into a multi-level implementation when the large AND or OR gates are not available in the target library.

The multi-level decomposition aims to produce an implementation with the minimal number of levels. This is guided by the fact that a multi-level implementation of a product term with I number of literals using AND gates with a maximum of n inputs is in the form of an n-ary tree (in this paper, we will refer to this tree as the AND tree). Furthermore, an implementation that has a minimal number of levels is an n-ary tree with the following properties:

1. Each internal node in the tree denotes an AND gate, each leaf denotes a literal, and each edge denotes a net in the gate-level implementation.
2. All nodes in the tree have n children, except internal nodes at the highest level of the tree may have less than n children. But for simplicity, we assume that all nodes in the tree have exactly n children,
3. The height of the tree equals $\lceil \log_n I \rceil$,
4. The critical path of the tree network of AND gates is depicted by the path that defines the height of the AND tree.
5. If the tree height is greater than 0, the critical path comprises $\lceil \log_n I \rceil - 1$ internal nodes. Otherwise there is no internal node on the critical path. For latter reference, let $AND_{I\text{-}node}$ be the number of internal nodes on the critical path of an AND tree.
6. If the tree height is greater than 1, the critical path contains $\lceil \log_n I \rceil - 1$ nets each of which connects two internal nodes. Otherwise, the critical path contains only one node and no net. For latter reference, let $AND_{I\text{-}net}$ be the number of nets on the critical path of an AND tree.

Similarly, the same decomposition scheme can be used to obtain a multi-level OR implementation of the sum term. A pictorial representation of this decomposition scheme is shown in Figure 7.

The capacitive load of each control signal, C_{CUload}, that drives the datapath units is proportional to the size (bit-width) of the datapath. If C_{CUload} is high, buffers are usually inserted to reduce the delay caused by the heavy load. Let us examine the loading effect in our model. If the buffer is not inserted, the last OR gate (i.e., the gate that is represented by the root of the OR tree) has to drive C_{CUload}. Thus, the delay caused by this load equals $R_{out}(OR(m)) \times C_{CUload}$, where m is a maximum number of inputs of an OR gate available in the library. However, if a buffer, BUF, is inserted, the delay caused by the additional buffer and the load equal $(t_{p\text{-}gate(BUF)} + (R_{out}(BUF) \times C_{CUload}))$. Therefore, to realize the influence of buffers insertion, we assume that each output of the control logic is driven by a buffer, BUF, if

$$t_{p\text{-}gate(BUF)} + (R_{out}(BUF) \times C_{CUload}) - (R_{out}(OR(m)) \times C_{CUload}) < 0.$$

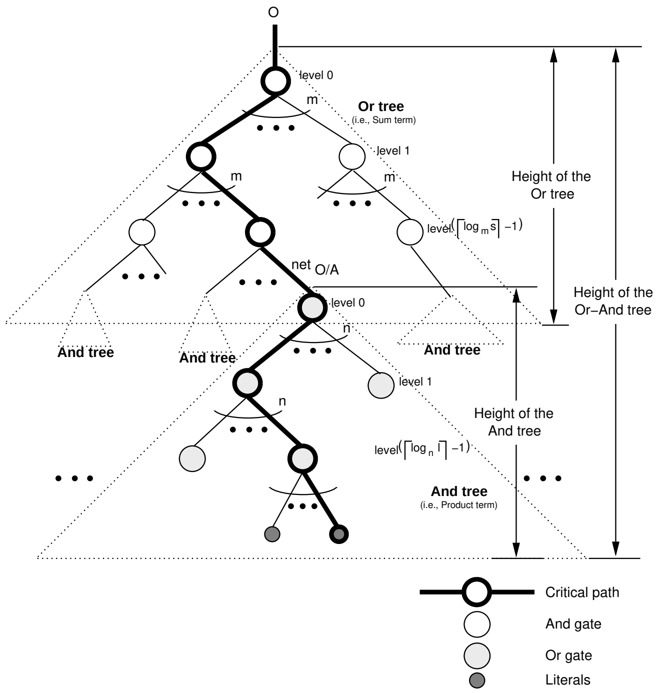


Figure 7: Pictorial representation of the decomposition scheme.

Furthermore, if the targeted library provides variety of buffers with different sizes, the buffer that gives the minimal resultant delay will be selected.

Figure 6 shows an example of a multi-level implementation of a sum-of-products expression. Each product term in the sum-of-products expression, Figure 6(b), requires a 5-input AND gate. If the targeted library provides only AND gates with a maximum of three inputs, all product terms are decomposed into a multi-level implementation, which is represented by a trinary tree shown in Figure 6(c). The equivalent gate implementation of this trinary tree is shown in Figure 6(d).

In our model, we assume that the random-logic is laid out as strips of standard or custom cells with input ports entering at the top and output ports exiting through the bottom. The number of layout strips is predetermined by the floorplanner in such a way that the total chip area is minimized. In addition, we assume that all gates that implement an output signal are placed closely in a cluster, as shown in Figure 6(e). The propagation delay from any input port of the control logic to an output port $\mathcal{O}$ is defined as:

$$t_{p\text{-}CU}(\mathcal{O}) = t_{p\text{-}CU gates}(\mathcal{O}) + t_{p\text{-}CU wire}(\mathcal{O}) \tag{9}$$

where the delay contributed by gates, $t_{p\text{-}CU gates}(\mathcal{O})$, and the delay contributed by wiring, $t_{p\text{-}CU wire}(\mathcal{O})$ are described in more detail below.

Delay Contributed by Gates, $t_{p\text{-}CU gates}(\mathcal{O})$

$t_{p\text{-}CUgates}$ is defined as the delay contributed by gates that lie on the critical path. Using the decomposition scheme described earlier, $t_{p\text{-}CUgate}(\mathcal{O})$, can be formulated as the sum of the propagation delay of gates in the AND tree, the OR tree, and the output buffer, that is,

$$\begin{aligned} t_{p\text{-}CUgates}(\mathcal{O}) &= (AND_{I\text{-}node} \times t_{p\text{-}gate(n\text{-}input\ AND)}) + \\ &\quad (OR_{I\text{-}node} \times t_{p\text{-}gate(m\text{-}input\ OR)}) + t_{p\text{-}gate(BUF)}. \end{aligned} \tag{10}$$

Delay Contributed by Wiring, $t_{CU\text{-}wire}(\mathcal{O})$

$t_{p\text{-}CUwire}(\mathcal{O})$ is defined as the sum of wiring delay of nets that lie on the critical path. From our layout model, wires that connect gates in the same cluster are short. Thus, the wiring resistance and capacitance of these wires are negligible. Hence, the wiring delay, $t_{p\text{-}net}(X, Y)$, of a net that connects the output of a gate of type X to an input of a gate of type Y in the random logic can be derived from Equation 4 with R_w and C_w equal to 0, that is,

$$t_{p\text{-}net}(X, Y) = R_{out(X)} \times C_{in(Y)}. \tag{11}$$

Using properties of the decomposition tree, $t_{p\text{-}CUwire}(\mathcal{O})$ can be formulated as the sum of wiring delay of nets in the AND tree, OR tree, the net that connects the AND and OR tree, and the net that connects the last OR gate (i.e., the OR gate that is represented by the root of OR tree) to the output buffer, that is,

$$\begin{aligned} t_{p\text{-}CUwire}(O) &= (AND_{I\text{-}net} \times t_{p\text{-}net}(n\text{-}input\ AND, n\text{-}input\ AND)) + \\ &\quad (OR_{I\text{-}net} \times t_{p\text{-}net}(m\text{-}input\ OR, m\text{-}input\ OR)) + \\ &\quad t_{p\text{-}net}(n\text{-}input\ AND, m\text{-}input\ OR) + t_{p\text{-}net}(m\text{-}input\ OR, BUF). \end{aligned} \tag{12}$$

4.2 Timing Models for the Datapath Unit

Layout of a datapath is obtained by stacking functional and storage units one above the other. There are two commonly used layout architectures [13]: stack with over-the-cell routing and stack with routing channels (Figures 8(a) and (b)). Each bit slice of a unit may be a handcrafted custom cell, or may be implemented with one row of connected standard cells, as shown in Figures 8(a) and (b), respectively.

In the first layout architecture, power and ground wires run horizontally in the first metal layer. The control lines common to different bit slices in each unit also 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, power and ground wires run vertically in the first metal layer. Control lines run over the standard cells in the second metal layer. Data lines are placed in the routing channel and run vertically in the first metal or the polysilicon layer.

Computation of the propagation delay from one datapath component to another in the same datapath block requires two elements: internal delay of the component(FU), $t_{p\text{-}FU}$,

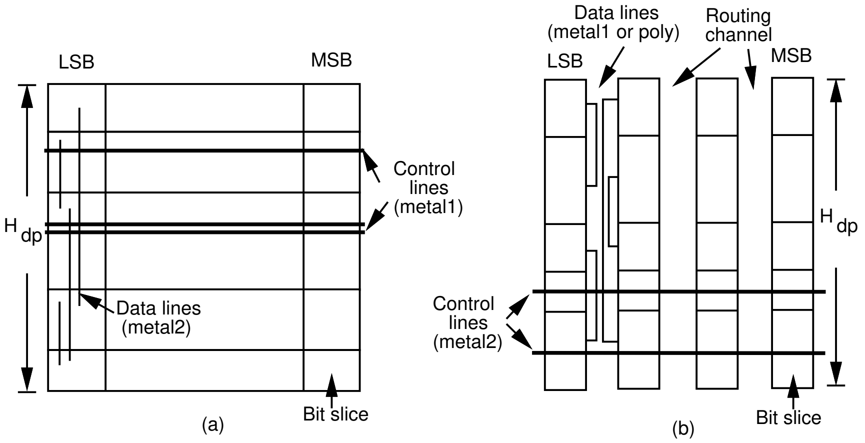


Figure 8: Two data path layout architectures using: (a) over-the-cell routing, (b) routing channel.

and wiring delay of net(s), $t_{p\text{-}FUwire}(s)$, that connects the two components. Typically, $t_{p\text{-}FU}$ for all datapath units are provided by the targeted component library. However, in the case where timing informations are not available, we can obtain them using the following model.

4.2.1 Timing Models for a Datapath Unit FU, $t_{p\text{-}FU}$

Since both architectures construct each datapath unit in a bit-sliced fashion, we first determine the delay for a single bit-sliced unit, $t_{p\text{-}1bitFU}$. Subsequently, if there exists an input of the FU that is an output of the previous bit of FU, $t_{p\text{-}FU}$ is computed as

$$t_{p\text{-}FU} = t_{p\text{-}1bitFU} + ((B-1) \times t_{p\text{-}ripple}) \tag{13}$$

where,

$t_{p\text{-}ripple}$ is the maximum delay of all rippling signals as described in more detail below,

B is the number of bits, and $t_{p\text{-}1bitFU}$ is described below.

Otherwise,

$$t_{p\text{-}FU} = t_{p\text{-}1bitFU}. \tag{14}$$

Delay of a FU Bit-slice, $t_{p\text{-}1bitFU}$

Given a netlist of a single FU bit-sliced, $t_{p\text{-}1bitFU}$ can be computed with Equation 6 as follows:

$$t_{p\text{-}1bitFU} = t_{p\text{-}comb}(I, O) \tag{15}$$

where,

I and O are input and output ports of the FU bit-sliced, respectively.

Since all gates in a FU bit-sliced are placed in a close cluster [13], the wire length of nets that connect these gates are short. Thus, thier wiring resistances, R_ws, and capacitances, C_ws, are negligible.

Maximum Delay of All Rippling Signals, $t_{p\text{-}ripple}$

Rippling signals are output signals that are used as inputs to the next bit slice. Since we are only interested in the delay of a critical path, only the longest delay of all rippling signals are to be used in $t_{p\text{-}FU}$ computation. Hence, the longest delay of all rippling signals, $t_{p\text{-}ripple}$ can be computed using Equation 6 as follows:

$$t_{p\text{-}ripple} = t_{p\text{-}comb}(I, O_r) \tag{16}$$

where,
I are all input ports of a FU bit-slice, and
O_r are rippling signals of a FU bit-slice.

Similar to the computation of $t_{p\text{-}1bitFU}$, all R_ws and C_ws are negligible.

4.2.2 Timing Models for Wiring Delay, $t_{p\text{-}FUwire}(s)$

The actual wire length can be determined only after the completion of computational expensive datapath placement and routing procedures. For simplicity, we assume that the average wire length of a net connecting any two units in the same datapath is equal to half of the datapath height (H_{dp}). In the first layout architecture, H_{dp} is equal to sum of the height of all datapath units. Whereas, in the second layout architecture, H_{dp} is proportional to the number of transistors in a bit slice and the transistor pitch (spacing between transistors) using the layout model similar to one described in [13]. Furthermore, since the stack and the standard cell datapath architectures use different layers to route these nets, we formulate the wiring resistance, R_{DPw}, and capacitance, C_{DPw}, as follow:

$$R_{DPw} = R_{s(l)}\left(\frac{\frac{1}{2}H_{dp}}{Width(l)}\right) \tag{17}$$

$$C_{DPw} = C_{s(l)}\left(\frac{1}{2}H_{dp}\right)(Width(l)) \tag{18}$$

where,
l is the routing layer in second metal for the bit-sliced stack with abutment architecture, and the routing layer in first metal for the bit-sliced standard cells architecture.

Thus, the wiring delay of a signal s that connects output of a component $comp_i$ to units $(u_j, 1 \leq j \leq n)$ in the same datapath can be computed as:

$$t_{p\text{-}FUwire}(s) = (R_{out}(comp_i) + R_{DPw})(C_{DPw} + \sum_{j=1}^{n} C_{in\text{-}unit}(u_j)) \tag{19}$$

where,
$C_{in\text{-}unit}(u_j)$ is the total input capacitance of all components $(comp_k, 1 \leq k \leq m)$ in u_j that are connected to signal s; (i.e., $C_{in\text{-}unit}(u_j) = \sum_{k=1}^{m} C_{in(comp_k)}$)

4.3 Timing Models for Inter-block Wiring delay

Propagation delay of a wire, S, that connects two logic blocks, A and B, on a chip can be computed using Equation 4 as follows:

$$t_{Inter\text{-}block}(S, A, B) = (R_{out}(comp_i) + R_{BLKw}(S, A, B)) \times (C_{BLKw}(S, A, B) + \sum_{j=1}^{n} C_{in}(comp_j)) \tag{20}$$

where,
$comp_i$ is the source driver of S,
$(comp_j, 1 \leq j \leq n)$ are sink components of S, and
$R_{BLKw}(S, A, B)$ and $C_{BLKw}(S, A, B)$ can be obtain from the floorplan as described in more detail below.

4.3.1 Floorplanning

To obtain a chip floorplan, we use a simplied cluster growth algorithm that selects and places blocks constructively one at a time. The cluster is grown from the lower left corner and the algorithm iteratively adds blocks on the top and right. The algorithm determines the order of the block to be placed according to the cost of the resultant area and the connectivity of that block with those already placed. In another word, the objective function selects a block $B_i \in B_{unplace}$ with $max(Area(B_i)) \times \Sigma w(B_i, B_j)$, where $B_j \in B_{place}$ and w is the number of wires. Subsequently, a placement position for the selected block is determined with respect to the overall connectivity and the aspect-ratio constraints. The main reason for using this simplied method is the considerably lower computation effort.

As the result of floorplanning, each block in the chip is centered at a coordinate (x,y). The length of a signal connecting any two given blocks is estimated as the manhattan distance between centers of the two blocks. And depending on the targeted layout system, vertical and horizontal segment of the wire will be routed in different layers. Hence, we formulate the resistance and capacitance of the wiring between blocks as a function of routing layers as follows:

$$R_{BLKw}(S, A, B) = (\frac{|A_x - B_x|}{Width(Ver)} \times R_{s(Hor)}) + (\frac{|A_y - B_y|}{Width(Ver)} \times R_{s(Ver)}) \tag{21}$$

$$\begin{aligned} C_{BLKw}(S, A, B) &= (|A_x - B_x| \times (Width(Hor)) \times C_{s(Hor)}) + \\ &\quad (|A_y - B_y| \times (Width(Ver)) \times C_{s(Ver)}) \end{aligned} \tag{22}$$

where,
$R_{BLKw}(S, A, B)$ and $C_{BLKw}(S, A, B)$ are the total wiring resistance and capacitance of the signal S connecting block A and B, respectively,
Ver is the layer used in vertical routing,
Hor is the layer used in horizontal routing,
A_x and A_y are the x and y coordinates of block A,
B_x and B_y are the x and y coordinates of block B,

5 Timing Models for the System Clock

The clock period is determined as the worst register-to-register delay in the design. Consider a typical register-transfer path shown in Figure 9(a). The register-to-register delay is defined as the delay of the critical path from source registers, $Reg1$ and $Reg2$, to the input of destination register that stores result of the data transfer, $Reg3$.

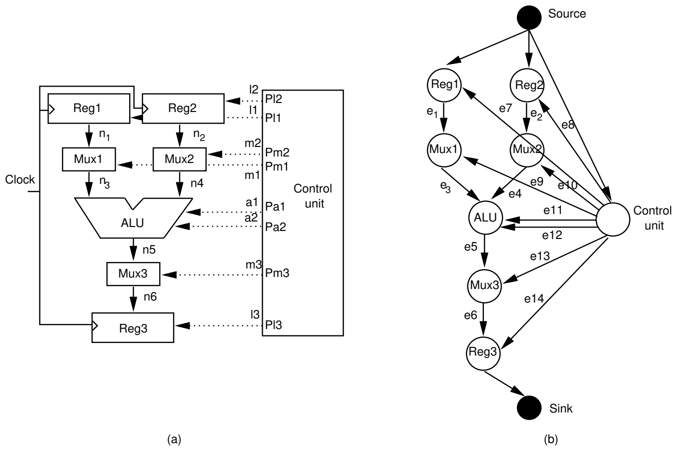


Figure 9: Timing models for system-clock: (a) an example of a register-transfer path, and (b) its equivalent multistage graph representation.

To identify the critical path in a register transfer, we use the multistage graph algorithm. First, we compute propagation delay of all components and wiring in the given register transfer using timing models described in previous sections. For example, the delay of P_{a1}, an output of the control unit, is computed using Equation 9, the delay of ALU is computed using Equation 13 or 14, the delay of signal $m1$ is computed using Equation 20, the delay of signal $n1$ is computed using Equation 19, etc. Then, the register-transfer path is transformed into a multistage graph (similar to the transformation of a combinational circuit to a multistage graph) with each component maps to a node, and each net maps to an edge with direction signifies the signal flow. Furthermore, we introduce a source node with its output edges directed to nodes that represent source registers and the control unit, and a sink node with input edges directed from the destination register, Figure 9(b). Subsequently, the sum of propagation delay of each net, net_i, and the delay of its source node is assigned as a cost to the edge that reprerents the net net_i; except for two special types of net that are assigned a cost of 0, namely, outputs of the source, and inputs of the sink. For example, the sum of the propagation delay of ALU and the wiring delay of net n_5 is assigned as a cost to the edge e_5. Finally, the critical path is identified as the longest path from the source to the sink of the resultant multistage graph. Hence, the propagation delay of a register-transfer, RT, is computed as

$$t_{p\text{-}Rtransfer}(RT) \quad = \sum_{comp_i \in COMP(SR_c, DR_c)} t_{p\text{-}comp}(comp_i) +$$

$$\sum_{net_j \in CONN(I_c,O_c)} t_{p\text{-}wire}(net_j) \quad (23)$$

where,

SR_c and DR_c are the source and destination registers, respectively, on the critical path,

$COMP(SR_c, DR_c)$ and $CONN(SR_c, DR_c)$ define all components and all nets on path from SR_c to DR_c,

$t_{p\text{-}comp}(comp_i)$ is a function that computes delay of the component $comp_i$. $t_{p\text{-}comp}()$ represents Equation 9, 13 or 14 depending on the classification of the logic block for $comp_i$.

$t_{p\text{-}wire}(net_j)$ is a function that computes delay of the net net_j. $t_w()$ represents Equation 19 or 20 depending on the type of net_j.

And thus, the period of the system clock can be compute as

$$t_{system\ clock} = MAX_{1 \leq i \leq n}(t_{p\text{-}Rtransfer}(RT_i)) \quad (24)$$

where,

RT_i is the i-th register transfer in the design and n is the total number of register transfers in the design.

6 Experimental Results and Discussions

We have tested our timing models on the elliptical filter benchmark. The experiment is divided into five parts. In the first part, we evaluate our control-unit timing models by comparing estimates with the simulated delays of four synthesized elliptic-filter designs with the same schedule but different registers and muxes utilization. Using the same set of designs, in the second part of the experiment, we compare our timing models for clock-period estimation against traditional performance measures by comparing estimates with the actual timing. The main distinction between different performance-estimation schemes is the granularity of the underlying model. A realistic timing model should consider all delay constituents of a chip. In section 2, we have identified and provided timing models for each of these constituents. In the third experiment, we determine the percentage contributed by these constituents to show that each of the constituent does in fact contribute delay to the clock period. The amount of delay contributed by each constituent of a chip varies across designs. Even a slight change in the number of bits of datapath units can cause a noticeable difference. This is shown in the fourth part of the experiment as we vary number of bits of the elliptical filter's datapath unit. In the last part of the experiment, we show that estimates from our timing models can be used to guide high-level synthesis tools in the selection of design styles.

In our experiments, the clock period is computed using Equation 24. For simplicity, we divided the delay of the clock period into three parts: *Datapath* delay, *Control unit* delay and *Wire/load* delay. *Datapath* delay includes the delays of wiring, functional, interconnect, and storage units as described in Equation 13, 14 and 19. *Control unit* delay includes the delays of control logic, next-state logic and state register as described in Equation 7 and 9. Wire/load delay takes into account the global wiring delay and the overall driven-load as described in Equation 20. The first, second and third experiments are carried out in 3μm CMOS technology ([3]), while the fourth and fifth experiments use 1.5μm CMOS technology ([12]).

In the first experiment, we have tested our control timing models on four synthesized designs of the elliptical filter benchmark with 2 adders and a 2-stage pipeline multiplier. All four designs are scheduled in 19-control steps but with different utilization of registers and muxes (Figure 10):

- design A contains 10 registers and 36 mux-inputs,
- design B contains 11 registers and 28 mux-inputs,
- design C contains 12 registers and 26 mux-inputs, and
- design D contains 13 registers and 23 mux-inputs.

Figures 10(a) and (b) shows the comparisons between the actual delay and the estimated delay of four control units. Since each control unit has a number of control paths, the data, in Figure 10(a), and data points, in Figure 10(b), represent the worst path delay. Two sets of data are given for the actual delay. The first set is the actual delay obtained from circuits that are created using the same decomposition scheme as assumed by our timing models. The second set is the actual delay of circuits that have been optimized for high performance.

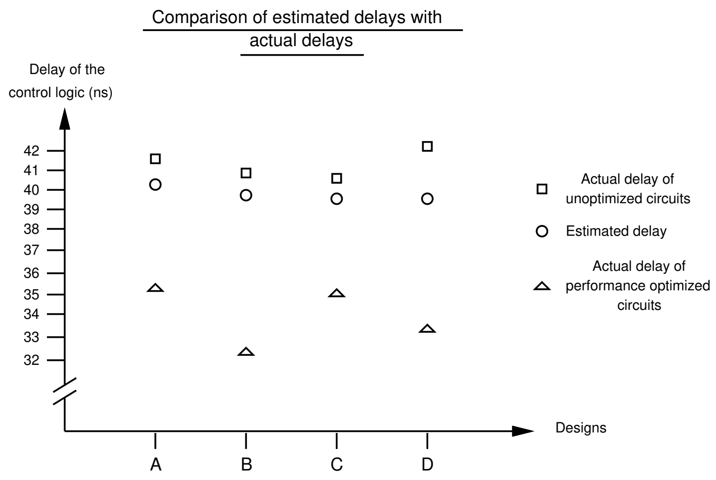
The measure of the first actual delay data-set is carried out in several steps. We first use the control layout model described in Section 2.3 to generate the gate-level netlist from the control-state table. Then, we use the Mentor Graphic GDT tools to generate the layout of the control unit with a 3μm CMOS technology. Finally, we use the GDT simulator, in adept mode, to measure delays for all critical paths and select the worst delay as the control delay. The results in Figure 10(c) shows that our control timing-model can predict the actual delay within an average of 6.4% error.

As we described earlier, our control timing-model aims to estimate the lower bound delay of the control logic, i.e., control logic that is optimized for performance. However, in reality, optimization techniques, such as factorization and transistors sizing, are applied to the logic to improve the performance of the design. To demonstrate the impact of these optimization procedures, we first use GDT optimizers to improve the performance of the logic. Then, we simulate and measure the delay of the optimized designs using GDT's simulator in adept mode; data is shown in the third column of Figures 10(a). **The results (Figure 10(c)) show that our control timing-model can predict the actual delay of the optimized circuit within an average of 14.2% error.**

In the second experiment, the actual clock period of four synthesized designs, which are used in the first experiment, are determined and their estimated values are computed. The delay calculation is based on a 16-bit datapath and a 3μm CMOS technology. Using the layout area model proposed in [13], the elliptic-filter benchmark is laid out in three blocks: a control unit, a datapath, and a 2-stage pipelined multiplier (macrocell), as shown

Elliptical filter designs with 19 control steps, 2 adders and a 2−stage pipe. mult.	Delay of the control logic and its drive load (ns)		
	Estimated delay	Actual delay of unoptimized circuits	Actual delay of performance optimized circuits
A (10reg, 36mux−i/p)	39.6	41.2	34.9
B (11reg, 28mux−i/p)	37.4	40.7	32.1
C (12reg, 26mux−i/p)	38.4	40.7	35.2
D (13reg, 23mux−i/p)	38.4	41.0	32.6

(a)



(b)

Elliptical filter designs with 19 control steps, 2 adders and a 2−stage pipe. mult.	Percentage error of estimated delay as compared to:	
	Actual delay of unoptimized circuits	Actual delay of performance optimized circuits
A (10reg, 36mux−i/p)	4.0%	13.5%
B (11reg, 28mux−i/p)	8.8%	16.5%
C (12reg, 26mux−i/p)	6.0%	9.1%
D (13reg, 23mux−i/p)	6.8%	17.8%

(c)

Figure 10: The evaluation of control-unit timing models: (a) comparison between estimated delay and actual delays (b) comparison in graph (c) percentage error of estimated delay.

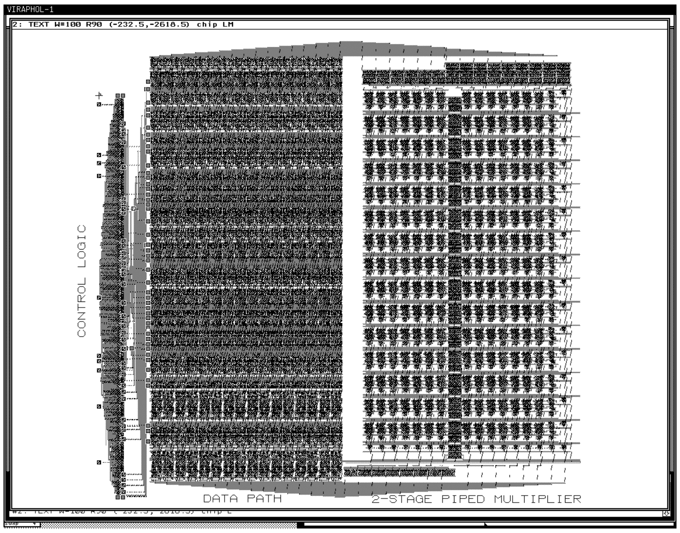


Figure 11: A layout of the elliptical filter benchmark.

in Figure 11. The actual (performance optimized designs) and estimated clock period is shown in Figure 12(a).

Figure 12(b) shows comparison of traditional timing-estimation schemes, our timing models and the actual clock period. And from results in Figure 12(c) we can draw the following observations. Estimators that use only delay of functional units provide estimates with an average error of 31.9% of the actual delay (Figure 12(c)). Estimators that use delay of units in the datapath (i.e., registers, functional units, muxes, etc.) in the clock period estimation provide estimates with an average error of 18.2%. Estimators that obtain clock period estimation by considering datapath and wiring delays give result with an average error of 7.5%. **On the other hand, results obtained with our timing models are within 2.7% of the actual delay.**

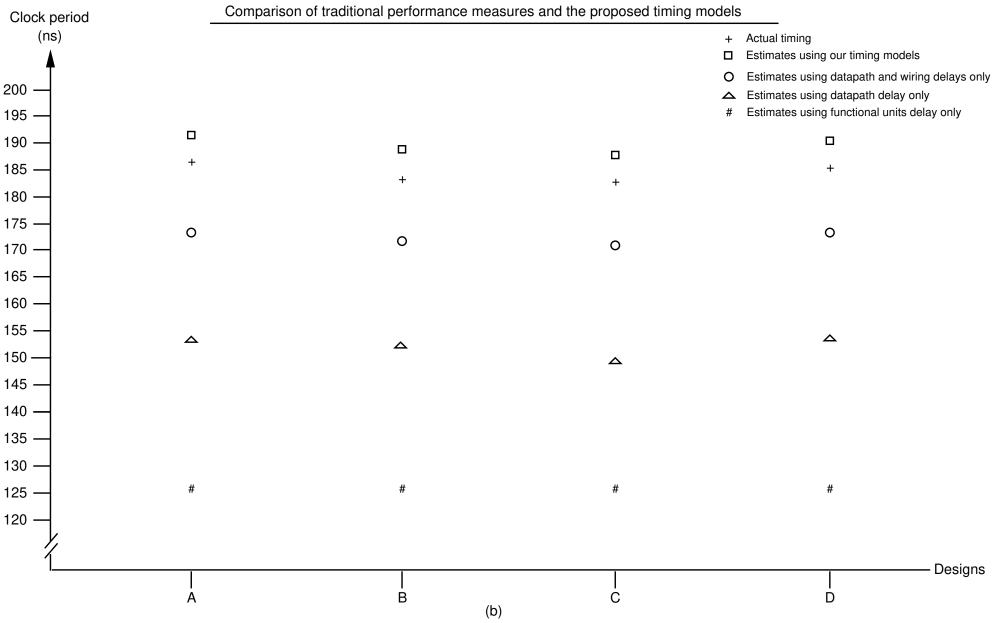
Using data obtained in the second experiment, we derive a distribution bar-chart shown in Figure 13. The chart shows that the clock period comprises of delay contributed by each constituent of the chip, as follows:

- an average of 80% of the clock period is contributed by the delay in the datapath units,
- an average of 10% of the clock period is contributed by the wiring and its driving load, and
- an average of 10% of the clock period is contributed by the control-unit delay.

Because the elliptic-filter benchmark is a datapath dominated design, the main contributor of the the clock period is the datapath delay. However, the amount of contribution by each constituent to the clock may vary from design to design. For example, Figure 14 shows

Elliptical filter designs with 19 control steps, 2 adders and a 2–st pipelined mult.	Datapath delay (ns)	Control delay (ns)		Wire/Load delay (ns)		Clock period (ns)	
		Estimate	Actual	Estimate	Actual	Estimate	Actual
A (10reg,36mux–i/p)	152.0	19.4	14.3	20.2	20.6	191.6	186.9
B (11reg, 28mux–i/p)	151.1	18.8	11.4	19.6	20.8	189.5	183.3
C (12reg, 26mux–i/p)	149.3	18.5	14.1	19.9	21.2	187.7	184.6
D (13reg, 23mux–i/p)	152.5	18.5	12.4	19.9	20.2	190.9	185.1

(a)



(b)

Elliptical filter designs with 19 control steps, 2 adders and a 2–st pipelined mult.	Percent error of clock–period estimates using various performance measures			
	Using only function units delay	Using only datapath delay	Using datapath and wiring delays	Using our timing models
A (10reg,36mux–i/p)	32.6%	18.7%	7.9%	2.5%
B (11reg, 28mux–i/p)	31.3%	17.4%	6.9%	3.4%
C (12reg, 26mux–i/p)	31.2%	19.1%	8.3%	1.7%
D (13reg, 23mux–i/p)	31.9%	17.6%	6.9%	3.1%

(c)

Figure 12: The clock period for four designs of the elliptical filter benchmark: (a) table of data, (b) comparison of different timing estimation schemes, (c) percentage error of each estimation scheme.

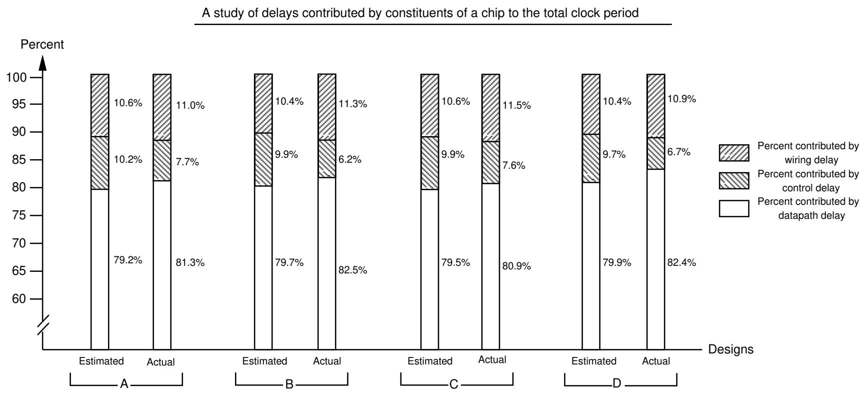


Figure 13: Delay distribution of constituents of a chip.

the distribution of delays as we vary the datapath bit-width of an elliptical filter design. As we increase the datapath bit-width, the capacitive load of each control signal increases, and in response, our model selects a bigger buffer to drive the increasing load. This causes the *wire/load* delay to remain generally constant (Figure 14). However, the control delay increases because of the additional delay contributed by the inserted buffer (Figure 14). Figure 14(b) also shows that the contribution of datapath delay to the clock period increases as the number of bits increases. This experiment shows that all delay constituents of a chip have to be included in order to perform complete analysis of the clock period.

In the last experiment, we have tested our timing models on four synthesized designs of a 16-bit elliptical filter benchmark with four different schedules and design styles (Figure 15): (A) 17 control steps with 3 adders, 2 multipliers, 10 registers and 34 mux-inputs, (B) 19 control steps with 2 adders, 2 multipliers, 11 registers and 28 mux-inputs, (C) 21 control steps with 2 adders, 1 multiplier, 10 registers and 25 mux-inputs, and (D) 19 control steps with 2 adders, 1 2-stage pipelined multiplier, 10 registers and 28 mux-inputs. The delay computation in this experiment is based on a 1.5μm technology [12].

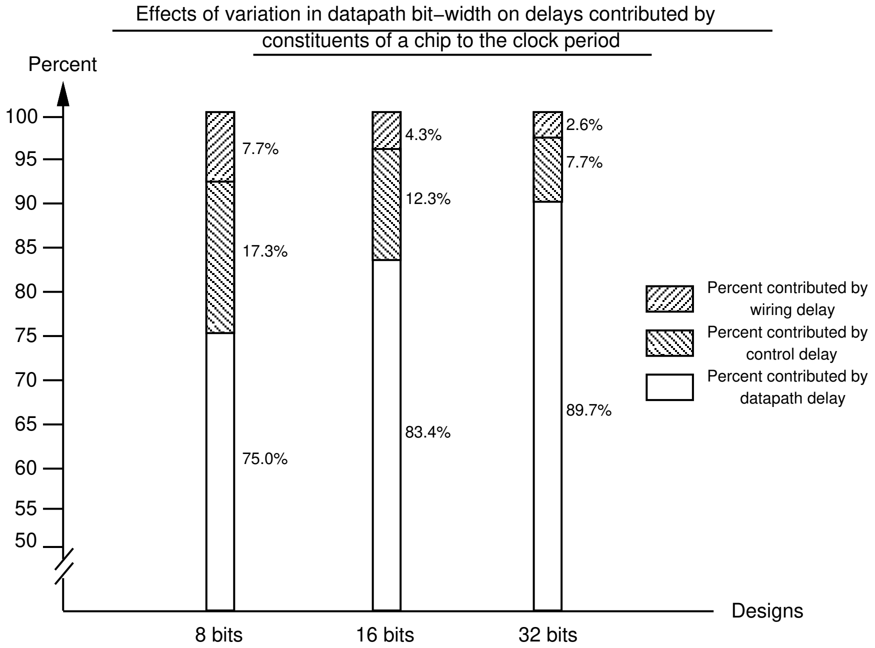
The results in Figure 15 list the total execution time (for one iteration) of four designs. We can use these estimates to guide the selection of designs that will satisfy a given performance constraint. For instance, if the performance constraint is *1000ns*, two designs, *Design A* and *Design B*, can satisfy the constraint. On the other hand, if the performance constraint is *1600ns*, all four designs satisfy the constraint.

7 Conclusions

Performance measure is one of the crucial elements in high-level synthesis. It is used to determine the performance of the final synthesized design and as a guidance in the search of the design space. Without a performance measure, high-level synthesis-tools will not be

Elliptical filter designs with 19 control steps, 2 adders, a 2−st pipe mult., 10 reg, 28 mux−i/p but with different datapath bit−width	Datapath delay (ns)	Control delay (ns)	Wiring/Load delay (ns)	Clock period (ns)
8 bits	63.0	14.5	6.5	84.0
16 bits	126.0	18.5	6.6	151.0
32 bits	252.0	21.5	7.5	281.0

(a)



(b)

Figure 14: The datapath bit-width factor: (a) estimated clock period, (b) delay distribution.

	Estimated delays (ns)				
Design	Datapath	Control	Wire/load	Clock period	Total execution time for one iteration
A(17c.s.)	41.0	16.8	7.0	64.8	1101.6
B(19c.s.)	37.5	15.6	7.0	60.1	1141.9
C(21c.s.)	37.5	18.5	6.0	62.0	1302.0
D(19c.s.)	41.0	19.5	6.0	66.5	1263.0

c.s. : control steps

Figure 15: The estimated clock period and total execution time of four different designs of the elliptical filter benchmark.

able to determine whether the synthesized designs satisfy given timing constraints, nor will they able to distinct a fast design from a slower one. To be useful in high-level synthesis, a performance measure should produce accurate estimates that can depict tradeoff and critical spots in the design efficiently (i.e., fast run time). The quest for a good performance-measure is usually a trade between accuracy and efficiency. In one extreme, a performance measure can produce highly accurate estimates by actually synthesizing the final layout. This type of estimators is not suitable for the use in high-level synthesis because of its slow run time (i.e., it took approximately $2\frac{1}{2}$ days to produce a layout for each example in our experiments). In the other extreme, the performance of a design can be estimated from some abstracted models. This type of estimators produce estimates efficiently, however, their accuracy depends on the underlying models. Commonly used measures that model performance of a chip as functional-units delay, datapath-units delay, or datapath units and wiring delay belong to this category. While these measures are very efficient, results in our experiments have shown that estimates produced by these measures are inaccurate due to its over-simplified models.

To obtain more realistic timing measures in high-level synthesis, we have presented a timing model that takes into account all delay elements, including datapath delay, control delay and wiring delay, and several technology factors, such as layout architecture, technology mapping, buffers insertion and loading effect. Using the popular elliptic filter benchmark, we have shown that the control unit and wire delays contribute 20% of the total clock period even in a datapath dominated design. Though the amount of delay contributed by each constituent of the clock period may vary from design to design, all factors mentioned have to be considered in order to perform complete analysis of the clock period. Furthermore, the preliminary results show that our timing model produces a better estimates on the lower bound of the clock period than previous models.

Using the proposed model, the timing estimates can be computed rapidly (with $n \log n$ complexity, where n is the number of nets in the control/datapath) by implementing a number of fast algorithms, such as the KLFM min-cut algorithm [2] for datapath placement, the left-edge algorithm for datapath routing-track assignments, the cluster-growth algorithm for floorplanning, and the multistage-graph algorithm for identifying critical path in a circuit.

To obtain better estimates of the control delay, more accurate models for control optimization procedures are needed. In order to improve the wire delay estimation, a more complex floorplanning procedure should be implemented.

Timing model described in this report is suitable for the use in high-level synthesis because of its efficiency and accuracy. For the same reason, the model can also be used in interactive synthesis [8] and in feedback-driven synthesis [9] for rapid evaluation of performance.

8 Acknowledgements

This work was supported in part by NSF grant ♯MIP-8922851, and in part by California MICRO grant ♯90-046. The authors would also like to thank Tedd Hadley and L. Ramachandran for their useful discussions.

References

[1] F. Brewer and D. D. Gajski, "Chippe: A System for Constraint Driven Behavioral Synthesis," *IEEE Trans. on Computer-Aided Design*, vol. 9, no 7, pp. 681-695, July 1990.

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

[3] "GDT Database and Language Tools," Silicon Compiler System, Sec. 7, V. 4.0, 1989.

[4] 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.

[5] Ellis Horowitz and Sartaj Sahni, "Fundamentals of Computer Algorithms," *Computer Science Press*, pp.203-208, 1989.

[6] R. Jain, A. C. Parker and N. Park, "Predicting Area-Time Tradeoffs for Pipelined Designs," *Proc. 24th DAC*, 1987.

[7] R. Jain, M. J. Mlinar and A. C. Parker, "Area-Time Model for Synthesis of Non-Pipelined Design," *Proc. ICCAD*, 1988.

[8] D. W. Knapp, "An Interactive Tool for Register-Level Structure Optimization," *Proc. 26th DAC*, 1989.

[9] D. W. Knapp, "Feedback-Driven Datapath Optimization in Fasolt," *Proc. ICCAD90*, 1990.

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

[11] P. Penfield Jr. and J. Rubenstein, "Signal Delay in RC Tree Networks," *Proceedings of the 18th Design Automation Conference*, pp. 613-617, 1981.

[12] "Data path Library," VLSI Technology, INC., 1988.

[13] A. C-H Wu, V. Chaiyakul and D. D. Gajski, "Layout Area Models for High-Level Synthesis," *Proc. ICCAD*, 1991.