Timing Models for High-Level Synthesis

Viraphol Chaiyakul  Allen C-H Wu  Daniel D. Gajski
Department of Information and Computer Science
University of California
Irvine, CA, 92717

Abstract

In this paper, we describe a timing model for clock estimation in high-level synthesis. In order to obtain realistic timing estimates, the proposed model considers 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 can be rapidly and incrementally calculated.

1 Introduction

High-level synthesis generates a structural design that implements the given behavior and satisfies design constraints, such as performance and area. Quality measures are needed to support high-level synthesis in two ways: to determine the quality of the final synthesized design and to guide high-level synthesis tools in the selection of design style and target architecture.

In high-level synthesis, performance measures (or timing 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. SPAID [5] 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 mentioned 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 performance optimization procedures.

The remainder of this paper is organized as follows: In section 2, we describe the delay models. In section 3, results on a set of eight different designs of a high-level synthesis benchmark are presented. Finally, in section 4, we present the conclusions from our research effort, discuss applications of our model and how our model could be improved.

2 Delay Models

We model the chip layout as a set of connected blocks that include control units, datapaths, macrocells and memories, as shown in Figure 1. Typically, a datapath is implemented using a bit-sliced stack, standard cells or macrocells. A control unit is implemented using a PLA or standard cells. Macrocells are some predefined components such as multipliers and barrel shifters. Memories include register files, RAMs and ROMs.

In this section, we first describe the electrical models for wiring and component delays. Second, we describe the delay model for each constituent of a chip shown in Figure 1. Since macrocells and memories are usually available in a library as predesigned blocks, we assume that timing information for macrocells and memories is provided by the library. Hence, we only describe the delay models for the datapath and the
control unit. Third, we present the inter-block wiring delay model. Finally, we describe the clock period model.

2.1 Electrical Models

The lumped RC model, also called the Elmore delay model [11], is widely used for delay calculation. In this 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,

$$t_p(start, end) = \sum_j R_j \times \sum_k C_k. \quad (1)$$

We can use Equation 1 to obtain the delay of a connecting wire between two components, or between two blocks. In CMOS technology we model a component as having input capacitance ($C_{in}$) and output resistance ($R_{out}$). The connecting wire is modeled using the well known $\pi$-model with wire capacitance $C_w$ and resistance $R_w$.

The propagation delay of a wire $n$ ($t_p(n)$) used by a component ($comp_i$) to drive load components ($comp_j, 1 \leq j \leq n$) can be computed as

$$t_p(n) = (R_{out}(comp_j) + R_w)(C_w + \sum_{j=1}^{n} C_{in}(comp_j)). \quad (2)$$

Thus, the delay for signals to propagate from the input of $comp_i$ through a wire $n$, to one of $comp_j$'s driven components, $comp_p$, is

$$t_p(comp_i, n, comp_p) = t_p(comp_i) + t_p(n). \quad (3)$$

where $t_p(comp_i)$ is the internal delay of component $comp_i$.

2.2 Datapath Delay Model

A datapath consists of a set of regularly structured RT components, such as ALUs, multiplexers, latches, drivers and shifters. Datapath layout is accomplished with a stack of functional and storage units that are placed one above the other. There are two commonly used layout architectures [13]: stack with over-the-cell routing and stack with routing channels (Figures 2(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 2(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 and wiring delay. Typically, the internal delays of components are provided by the targeted component library.

The actual wiring length can be determined only after the completion of computationally 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) [13]. Thus, the average wiring resistance and capacitance are calculated as: $R_w(DP) = R_s((1/2H_{dp})/W_w)$ and $C_w(DP) = C_s((1/2H_{dp})/W_w)$, where $W_w$, $R_s$ and $C_s$ are the width the sheet resistance and capacitance of the routing layer for data lines, respectively. The propagation delay between two datapath components ($comp_i$ and $comp_j$) via a net ($net_k$) is computed using Equations 2 and 3.
2.3 Control Delay Model

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 and the next states are encoded as binary values $p_1\ldots p_1 p_0$ and $n_k\ldots n_1 n_0$, where $k = \lceil \log_2 S \rceil - 1$ and $S$ is the number of states, respectively. 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.

![Random-logic model: (a) decomposition of a product term, (b) a multi-level implementation, and (c) the layout model.](image)

Figure 3: Random-logic model: (a) decomposition of a product term, (b) a multi-level implementation, and (c) the layout model.

We consider the impact of 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. The product term is implemented with AND gates and the sum with OR gates. Since, the target component library will usually provide AND and OR gates with a limited number of inputs, 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, where 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. The height of the tree equals $\lceil \log_n I \rceil$, and the critical path of the product terms is denoted by the path that defines the height of the tree. For latter reference, let $AND_{I\text{-node}}$ and $AND_{I\text{-net}}$ be the number of internal nodes and nets on the critical path, respectively.

Similar decomposition scheme can be used to obtain a multi-level OR implementation of the sum term.

The capacitive load of each control signal, $C_{\text{Cloud}}$, that drives the datapath units is proportional to the size (bit-width) of the datapath. If $C_{\text{Cloud}}$ 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 $C_{\text{Cloud}}$. Thus, the delay caused by this load equals $(R_{\text{out}}(\text{OR}(m)) \times C_{\text{Cloud}})$, where $m$ is a maximum number of inputs of an OR gate available in the library. However, if a buffer, $B$, is inserted, the delay caused by the additional buffer and the load equal $(t_p(B) + (R_{\text{out}}(B) \times C_{\text{Cloud}}))$, where $t_p(B)$ is the propagational delay of the inserted buffer. Therefore, we assume that each output of the control logic is driven by a buffer, $B$, if

$$t_p(B) + (R_{\text{out}}(B) \times C_{\text{Cloud}}) - (R_{\text{out}}(\text{OR}(m)) \times C_{\text{Cloud}}) < 0.$$

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

Figure 3 shows an example of a multi-level implementation of a sum-of-products expression with maximum three inputs AND gates. The multi-level implementation is represented by a trinary tree shown in Figure 3(a) and its equivalent gate implementation is shown in Figure 3(b).

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 3(c). The propagation delay from any input port of the control logic to an output port $O_i$ consists of two elements: the gate delay ($t_p(\text{gate}(O_i))$) and the wire delay ($t_p(\text{net}(O_i))$).
Gate delay is defined as the sum of delays of gates along the critical path. Using the decomposition scheme described earlier, the gate delay \( t_p(gate(O_i)) \) can be formulated as the sum of the delay of gates in the AND tree, OR tree, and the output buffer, \( B \), that is,
\[
t_p(gate(O_i)) = (AND_{tree-node} \times t_p(AND(n))) + (OR_{tree-node} \times t_p(OR(m))) + t_p(B).
\]
where \( t_p(AND(n)) \) is the propagation delay of an \( n \)-input AND gate, and \( t_p(OR(m)) \) is the propagation delay of an \( m \)-input OR gate.

Wire delay, \( t_p(net(O_i)) \), is defined as the sum of delay of wires on the critical path. Using our layout model, wires that connect gates in the same cluster are relatively short. Thus, the wiring resistance and capacitance of these nets are negligible. Hence, \( t_p(net(O_i)) \) can be formulated as the sum of the wiring delays of nets in the AND tree, OR tree, 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,
\[
t_p(net(O_i)) = (AND_{net} \times t_p(net(AND(n), AND(n)))) + (OR_{net} \times t_p(net(OR(m), OR(m)))) + t_p(net(AND(n), OR(m))) + t_p(net(OR(m), B))(5)
\]

Thus, the propagation delay from any input port to an output port \( O_i \) is
\[
t_p(CU(O_i)) = t_p(gate(O_i)) + t_p(net(O_i)). 
\]

2.4 Inter-Block Wiring Delay Model

We can use Equation 2 to compute the propagation delay of a net \( n \) that connects two blocks \( A \) and \( B \) on a chip. To obtain the wire length of the net, we use a simple cluster growth algorithm to form the chip floorplan [2].

As a result of the floorplan, each block in the chip is centered at a coordinate \((x, y)\). The length of a net connecting any two blocks \( A \) and \( B \) is estimated as the manhattan distance between the centers of the two blocks. Thus, the inter-block wiring resistance and capacitance are
\[
R_w(n) = \frac{|A_x - B_x|}{W_w} R_s + \frac{|A_y - B_y|}{W_w} R_s \quad (7)
\]
\[
C_w(n) = \frac{|A_x - B_x|}{W_w} C_s + \frac{|A_y - B_y|}{W_w} C_s \quad (8)
\]

where \( W_w \) is the width of the routing wire, and \( (A_x, A_y) \) and \( (B_x, B_y) \) are \( x \) and \( y \) coordinates of block \( A \) and \( B \), respectively.

The propagation delay of \( n \) is then computed using Equation 2.

2.5 Clock Cycle Model

The clock cycle is determined by the longest register-to-register delay that includes the propagation delays in the control unit, in the datapath unit and between blocks. In our implementation, the clock computation is based on a simple Finite-State-Machine Datapath model (FSMD) [4], as shown in Figure 4 and described below.

In Figure 4, the critical path (Path1) is from the State register, through the Control logic, the Datapath, and the Next-state logic, and back to the State register. Thus, the clock period is the sum of the propagation delays of the State register \( (t_p(ST)) \), the Control logic \( (t_p(CL)) \), the Datapath \( (t_p(DP)) \), the Next-state logic \( (t_p(NS)) \), and the set-up delay of the State register \( (t_u(ST)) \), that is,
\[
t_{clk} = t_p(ST) + t_p(CL) + t_p(DP) + t_p(NS) + t_u(ST) 
\]

\[t_p(CL)\) and \( t_p(NS)\) are computed using Equation 6. \( t_p(DP)\) is determined by the longest register-to-register delay in the datapath. For example, a typical register-to-register delay in a datapath is shown in Figure 5. It includes the delay through the source storage unit \( (MAX(t_p(R1), t_p(R2))) \), functional unit \( (t_p(FU)) \), connections \( (MAX(t_p(n1), t_p(n2)) \) and \( t_p(n3)) \), and the setup time of the destination storage unit \( (t_u(R3)) \). Since the access time to a RAM is slow and often takes several clock cycles, we consider the storage units, \( R1, R2 \) and \( R3 \), as registers or a register file. Connections \( n_1, n_2 \) and \( n_3 \) are implemented as wires or interconnect units such as a muxes.
or a bus. Thus, for a single-cycle operation \( op_i \), or a single-cycle chaining operation, the register-to-register delay of operation \( op_i \) is computed as

\[
t_p(op_i) = \text{MAX}(t_p(R1), t_p(R2)) + t_p(FU) + \text{MAX}(t_p(n_1), t_p(n_2)) + t_p(n_3) + hll(R3)
\]

(10)

And, the longest register-to-register delay for each clock-cycle among all operations is

\[
t_p(DP) = \text{MAX}(V(t_p(op_i))).
\]

(11)

3 Experiments and Discussions

We have tested our timing models on the elliptical filter benchmark. The experiment is divided into four 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. In the third experiment, we determine the contribution of delay from each constituent of a chip, which is discussed in section 2, to the clock period. The amount of delay contributed by each constituent 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 our experiments, the clock period is computed using Equation 9. 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 10. Control unit delay includes the delays of control logic, next-state logic and state register as described in Equation 6. Wire/load delay takes into account the global wiring delay and the overall driven-load. The first, second and third experiments are carried out in 3μm CMOS technology [3], while the fourth experiment uses 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 pipelined multiplier. All four designs are scheduled in 19-control steps but with different utilization of registers and muxes (Figure 6). Figures 6 shows the comparisons between the actual delay and the estimated delay of four control units. Since each control unit has many control paths, data in Figure 6 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.

<table>
<thead>
<tr>
<th>Designs</th>
<th>Delay of the control logic and its drivers (ns)</th>
<th>Delay of functional units (ns)</th>
<th>Delay of interconnect units (ns)</th>
<th>Percentage error of estimated delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (10.96)</td>
<td>29.6</td>
<td>41.2</td>
<td>24.9</td>
<td>4.5%</td>
</tr>
<tr>
<td>B (11.36)</td>
<td>24.7</td>
<td>40.7</td>
<td>23.1</td>
<td>8.4%</td>
</tr>
<tr>
<td>C (15.26)</td>
<td>28.4</td>
<td>40.7</td>
<td>35.2</td>
<td>6.5%</td>
</tr>
<tr>
<td>D (13.22)</td>
<td>28.4</td>
<td>41.0</td>
<td>32.6</td>
<td>8.8%</td>
</tr>
</tbody>
</table>

*Elliptical filter designs with 19 control steps, 2 adders and 2-stage pipelined multiplier with different number of registers and mux inputs.*

Figure 6: The evaluation of control-unit timing models.

Experimental results, Figure 6, show that our control timing-model can predict the actual delay of unoptimized circuits within an average of 6.4% error and the actual delay of the optimized circuit within an average of 14.2% error.

Figure 7: A layout of the elliptical filter benchmark.

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. Using the layout area model proposed in [12], the elliptic-filter benchmark is laid out in three blocks: a control unit, a datapath, and a 2-stage pipelined multiplier (macrocell), as shown in Figure 7, using GDT and Mentor Graphics tools [3]. The actual (performance optimized designs) and estimated clock period is shown in Figure 8.

Figure 8 also shows comparison of traditional timing-estimation schemes, our timing models and the actual clock period. Results show that estimators that use only delay of functional units provide estimates with an average of 31.9% error. Estimators that use delay of units in the datapath (i.e., registers, functional units, muxes etc.) provide estimates with an average of 18.2% error. Estimators that consider datapath and wiring delays give result with an average of
Figure 8: Comparison of the clock period estimates using different measures.

7.5% error. On the other hand, results obtained with our timing models are within 2.7% error.

Data obtained in the second experiment also show that the clock period comprises of delay contributed by each constituent of the chip, as follows: an average of 80% is contributed by the delay in the datapath units, an average of 10% is contributed by the wiring and its driving load, and an average of 10% is contributed by the control-unit delay.

Because the elliptic-filter benchmark is a datapath dominated design, the main contributor of the clock period is the datapath delay. However, the amount of contribution by each constituent may vary from design to design. For example, Figure 9 shows the distribution of delays as we vary the datapath bit-width of an elliptical filter design.

As the datapath bit-width increases, 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 9). However, the control delay increases because of the additional delay contributed by the inserted buffer. Results also show that the contribution of datapath delay to the clock period increases as the number of bits increases.

4 Conclusions

The quest for a good performance-measure is usually a tradeoff between accuracy and efficiency. In one extreme, highly accurate estimates can be obtained by actually synthesizing the final layout. In the other extreme, fast estimates can be obtained from some abstracted models. This approach produces estimates efficiently, however, the 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 approach. While these measures are very efficient, results from our experiments have shown that estimates produced by these measures are inaccurate due to their over-simplified models.

To obtain more realistic timing measures in high-level synthesis, we have presented a timing model that takes into account datapath delay, control delay and wiring delay, and several technology factors, such as layout architecture, technology mapping, buffers insertion and loading effect. Estimates can be computed from the proposed model rapidly with $O(n \log n)$ time complexity, where $n$ is the number of nets in the control/datapath. The preliminary results show that our timing model produces better estimates on the lower bound of the clock period than previous models. Nevertheless, more accurate models for control optimization procedures are needed to improve estimates of the control delay.

The proposed timing model 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 quality of designs.

5 References