# False Path Exclusion in Delay Analysis of RTL-Based Datapath–Controller Designs[†]

Mehrdad Nourani[‡] Christos Papachristou[§]

## Abstract

*In this paper, we present an accurate delay estimation algorithm at the register transfer level. We introduce "resource binding" as an important source of false paths in a register transfer level structure. "Path mismatching" between two path segments may create another type of false paths when the datapath and controller interact. The existence and creation of such paths and their effect in delay analysis are discussed. We also introduce the Propagation Delay Graph (PDG), whose traversal, for delay analysis, is equivalent to the traversal of sensitizable paths in the datapath.*

## 1. Introduction

### 1.1 Motivation

High level synthesis (HLS) fills the gap between the behavioral level and the layout level by automatically generating a register transfer level (RTL) realization from a behavioral description. The actual circuit layout can be generated later from the datapath using a silicon compiler. A common characteristic of most HLS systems is the lack of consideration to the logic and layout synthesis aspects during the high level synthesis process. The designs generated by these systems may or may not satisfy the initial design requirements (e.g. area, performance) and thus the design may need to be changed and modified. One method to address this deficiency is to use estimation during HLS, specifically for time delay and layout area [1].

Estimation at the RT level can be done faster than at other levels, such as logic, because of dealing with fewer components and easier access to the control information. The delay estimators at the RT level, in particular, can be used for different purposes such as: *performance analysis* (e.g. finding the upper bound speed) and *performance improvement* (e.g. using pipelined ALUs to reduce critical path delay).

False path detection at the logic level has been investigated in depth. However, the notion of false paths at the RT level did not raise much attention due to: 1) ambiguity in sources that may create false paths; and 2) lack of a clear understanding as to how time analysis at three design levels, RTL, logic and layout, relate to each other. The nature of false paths at the RT level is, in general, different from false paths at the logic level. The false path problem at the logic level is a *data-oriented* phenomenon. In spite of this difference in nature, the effect of false paths at both levels is the same, that is, the accuracy of the time estimators may be heavily affected by the existence of such paths.

In this paper, we propose an accurate delay estimation algorithm for cell-based designs at the RT level. The main contribution of our work is twofold. First, we introduce false paths at the RTL caused by *resource binding* (in datapath) and signal *path mismatching* (in datapath–controller interaction) and their effect in delay estimation, i.e. register-to-register and multi-cycle critical paths. Second, we present an efficient delay estimation process based on a Propagation Delay Graph (PDG) model with annotated information from the structural level (RTL). The estimator computes the structure-based (static) and behavioral-based (multi-cycle) critical delay of sensitizable paths and determines the minimum period of the system clock.

### 1.2 Related Works

Time prediction techniques based on *false path analysis* are widely used at the logic design level (logic synthesis). Works by [2] and [3], for example, discuss the use of false path detection in time analysis of logic circuits. Works presented in [4] and [5] are two attempts to expand the notion of false path to the RT level by identifying the false paths through the sequences of conditional operations, and false loops through resource sharing, respectively.

The works presented in [6] and [7] are attempts to couple floorplanning and high level synthesis by taking wire delay into account when making the synthesis decisions. A timing model for clock estimation is proposed in [8] to integrate control and wire delays and several technology factors. Work presented in [9] combines topological and functionality based delay estimation using a layout-driven approach.

Reference [10] used area-time models to predict design tradeoffs from the DFG graph. However, their model considers only functional units at the scheduling level and does not include RTL components such as registers, multiplexers and ALUs. SPAID [11] also determines the clock period by considering delay of components and finding the worst datapath delay, but it does not take into account the wiring delay. In BUD system [12] the wire lengths are obtained from a floorplanner and the wiring delay is computed using a simple RC model. CHIPPE [13] estimates the clock period by examining control delay using a PLA model, but wiring delay is ignored.

This paper is organized as follows: The existence and creation of false paths, in register-to-register delay estimation, caused by resource sharing are discussed in section 2. Section 3 presents the multi-cycle false paths caused by resource binding. Timing model for the control unit and the interaction of datapath and controller in terms of delay are discussed in section 4. In section 5, we describe the delay estimation algorithm in which we prohibit the possible inclusion of false paths during delay estimation. The experimental results are presented in section 6. Finally, concluding remarks are in section 7.

## 2. False Paths in Register-to-Register Delay Analysis

### 2.1 The Importance of Detecting False Paths

If we search a datapath purely based on the delay of components and connections, we may erroneously identify a false path as the critical path and inaccurately determine the performance. This is due to the fact that to achieve higher speed most delay analyzers perform a value-independent search by simply checking all possibilities in a node and picking the worst.

VLSI manufacturing technology scale factor ($\lambda$) of 0.8 micron is quite normal these days. This implicitly means the propagation delay of components, including wires, are getting closer and closer to each other. The closeness of delays shows

[†]This work was partially supported by the Semiconductor Research Corporation (SRC) under Contract No. DJ-527.

[‡]Department of Electrical and Computer Engineering, University of Tehran, Tehran, Iran.

[§]Department of Computer Engineering, Case Western Reserve Univ., Cleveland, Ohio.

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00 © 1996 IEEE

that wrong inclusion or exclusion of components and wires changes the result of performance analysis dramatically.

### 2.2 False Paths Caused by Resource Binding

An RTL datapath realizes a behavior through register-to register data transfer. The sequence of these transfers is determined by a CDFG. Fig. 1 shows a typical register-to-register path. The data pass through two interconnection networks, before and after the functional unit. If some of the components in these two interconnection networks (e.g. multiplexers, buses and wires) are shared, there is a high possibility of creating false paths. Assume two values should be read from registers $D_1$ and $D_2$, manipulated by $FU_1$ and $FU_2$ and finally stored in output registers $O_1$ and $O_2$, respectively. Fig. 2 shows two scenarios for these data transfers. In Fig. 2(a), there are only two paths and both are sensitizable since there are no shared components between $N_{11}, N_{12}$ in $\{D_1 \rightarrow N_{11} \rightarrow FU_1 \rightarrow N_{12} \rightarrow O_1\}$ path and their counterparts $N_{21}, N_{22}$ in $\{D_2 \rightarrow N_{21} \rightarrow FU_2 \rightarrow N_{22} \rightarrow O_2\}$ path. In Fig. 2(b), however, there are some shared components (shaded area). During analysis of register-to-register delays of Fig. 2(b), we confront 8 paths, out of which only two are sensitizable. Note that only two false paths (broken and dotted lines) are shown in this figure.

In what follows, we present six situations under which false paths may be created. Each case is explained by a small example. Let $T^{func}_{comp}$ denote the propagation delay of component *comp* when it performs function *func*. For example, $T^{+}_{(+->)}$ is the propagation delay of $(+- >)$ (i.e. a multi-function ALU) when it performs addition and $T^{s}_{R5}$ ($T^{e}_{R5}$) shows the setup (transition) time of register $R5$. All delay values of components used in these examples are based on a 4-bit implementation of cells using a 0.8 micron CMOS library [14]. Note that for simplicity, we have ignored the wire delays in these examples. Also, we use $P_f$ and $P_s$ to refer to false and sensitizable paths, respectively.

To show that these cases can happen in practice, when it was possible, we have selected (or created using the same architectural style) some examples reported in the literature. Due to lack of information about components or constraints which resulted in those datapaths, certain assumptions will be made to show the falsehood of a specific path.

#### 2.2.1 Example 1: Certain Binding Decisions

Most HLS systems use heuristics for binding in the allocation phase to optimize component and/or interconnection cost. Some of these decisions may create false paths. Fig. 3 is a datapath for HAL benchmark, selected from [19]. The multiplier is a two stage pipelined multiplier. Assume register $R*$, has been installed between two stages with delay of $\frac{T^{*}_{(*)}}{2}$ each. Suppose for the multi-function ALU we have: $T^{-}_{(+->)} > T^{+}_{(+->)} > T^{>}_{(+->)} > \frac{T^{*}_{(*)}}{2}$. Assuming the propagation delays of all registers are the same, the critical register-to-register path is as follows:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{e}_{R*} + \frac{T^{*}_{(*)}}{2} + T_{BUS1} + T^{-}_{(+->)} + T^{s}_{R2} \\ \text{Sensitizable Path:} & T(P_s) = T^{e}_{R1} + T_{MUX3} + T^{-}_{(+->)} + T^{s}_{R5} \\ \text{Error:} & \Delta T = |T(P_f) - T(P_s)| = |\frac{T^{*}_{(*)}}{2} + T_{BUS1} - T_{MUX3}| \approx 5.48ns \end{cases}$$

#### 2.2.2 Example 2: Multi-Function ALU's

Fig. 4 is a datapath for HAL benchmark, selected from [20]. Assume that we use a fast two stage pipelined multiplier and a slow adder/subtracter such that: $T^{-}_{(+-)} > T^{+}_{(+-)} > \frac{T^{*}_{(*)}}{2}$. Let all registers have the same propagation delay and $R5$ provide data for subtraction only through BUS4. Considering the fact that there is a register between the two stages of the multiplier, the critical register-to-register path occurs through $(+-)$:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{e}_{R3} + T_{BUS2} + T_{BUS4} + T^{-}_{+-} + T^{s}_{R6} \\ \text{Sensitizable Path:} & T(P_s) = T^{e}_{R3} + T_{BUS2} + T_{BUS4} + T^{+}_{+-} + T^{s}_{R6} \\ \text{Error:} & \Delta T = |T(P_f) - T(P_s)| = |T^{-}_{(+-)} - T^{+}_{(+-)}| \approx 1.52ns \end{cases}$$

#### 2.2.3 Example 3: Testable Datapaths

There are many applications in which we need to use different type of registers in a design. Two of such applications are: 1) using a combination of *normal* (latch-based) and *master-slave* registers for *Level Sensitive Scan Design* (LSSD) [21]; and 2) different registers in a testable design, such as *Built In Self Testing* (BIST). Depending on the test methodology, there are different type of test registers, such as: BILBO, TPGR and MISR [21].

As an example, we selected the datapath of the FACET example (introduced first in [22]) presented in [23]. The schedule and datapath are shown in Fig. 5(a) and (b), respectively. Note that the data for division come from $R5$ and $R7$. Consider two design styles: In the first, registers $R1, R3, R4, R5$ are master-slave and $R2, R6, R7$ are normal. In the second style, we have a testable design based on the BIST methodology. Note that registers $R2$ and $R5$ are normal, $R6$ and $R7$ are TPGR, $R3$ is MISR and $R1$ and $R4$ are BILBO. The setup and transition time for different type of registers, given in Fig. 5, are based on COMPASS implementations. Assume that the critical register-to-register delay occurs through divider:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{e}_{R6} + T_{MUX3} + T^{/}_{(/)} + T_{MUX4} + T^{s}_{R4} \\ \text{Sensitizable Path:} & T(P_s) = T^{e}_{R5} + T_{MUX3} + T^{/}_{(/)} + T_{MUX4} + T^{s}_{R4} \\ \text{Error (style 1):} & \Delta T = |T(P_f) - T(P_s)| = |T^{e}_{R6} - T^{e}_{R5}| \approx 2.58ns \\ \text{Error (style 2):} & \Delta T = |T(P_f) - T(P_s)| = |T^{e}_{R6} - T^{e}_{R5}| \approx 1.21ns \end{cases}$$

#### 2.2.4 Example 4: Chaining

Consider the datapath shown in Fig. 6(b). The scheduled CDFG is shown in Fig. 6(a). By examining this schedule, in which chaining of addition and subtraction is allowed, we see that the path shown by broken lines in Fig. 6(b) is false since the result of multiplication from the first fan-out branch ($m_1$) is never followed by the subtraction ($s_1$). The two longest sensitizable paths are shown in Fig. 6(b) using dotted lines. Depending on the length of wires between components (estimated or measured in the physical layout), one of them will be the sensitizable critical path. Assuming all registers have identical structures, the register-to-register delays of paths are:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{e}_{R3} + T^{*}_{(*)} + T_{MUX3} + T^{-}_{(-)} + T^{s}_{R2} \\ \text{Sensitizable Path 1:} & T(P_{s1}) = T^{e}_{R4} + T^{*}_{(*)} + T_{MUX3} + T^{s}_{R1} \\ \text{Sensitizable Path 2:} & T(P_{s2}) = \\ & T^{e}_{R5} + T_{MUX1} + T^{+}_{(+)} + T_{MUX3} + T^{-}_{(-)} + T^{s}_{R2} \\ \text{Error (Path 1):} & \Delta T_1 = |T(P_f) - T(P_{s1})| = |T^{-}_{(-)}| \approx 3.51ns \\ \text{Error (Path 2):} & \Delta T_2 = |T(P_f) - T(P_{s2})| = \\ & |T^{*}_{(*)} - T^{+}_{(+)} - T_{MUX1}| \approx 3.86ns \end{cases}$$

#### 2.2.5 Example 5: Redundant Components

Heuristics for binding may introduce redundant components into a design. Fig. 7 shows such an example. Assume we use an iterative binding scheme which assigns operations and data to the hardware components sequentially (e.g. one functional unit at a time). In the first time step, $b$ is passed through *MUX2* and fed to both ALUs. So, the connection $W_1$ is considered. In the second step, no wire sharing to pass $b$ and $c$ through *MUX2* is possible and so a new connection ($W_2$) has been made to pass $b$ to the multiplier. The datapath is pictured in Fig. 7(b). Assuming the delay of registers are the same and $T^{*}_{(*)} > T^{+}_{(+)}$, we have:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{e}_{R3} + T_{MUX2} + T_{MUX3} + T^{*}_{(*)} + T^{s}_{R6} \\ \text{Sensitizable Path:} & T(P_s) = T^{e}_{R3} + T_{MUX3} + T^{*}_{(*)} + T^{s}_{R6} \\ \text{Error:} & \Delta T = |T(P_f) - T(P_s)| = |T_{MUX3}| \approx 1.11ns \end{cases}$$

Of course, a refinement process can remove the redundant gate (MUX3) to generate the datapath of Fig. 7(c). However, in reality the designs are very complex and such refinement can be very expensive without guaranteeing the optimality of the result.

### 2.3 Exclusion of False Paths Caused by Resource Binding

To prohibit the possible inclusion of false paths during delay estimation of RTL datapath, we use CDFG whose edges are augmented by weighting values, annotating the delay information of datapath and CDFG simultaneously. For brevity, we call the augmented CDFG *Propagation Delay Graph* (PDG). By weighting the edges in CDFG, we bind data flow information in CDFG with the component/connection delay information in the datapath. Figure 6(a) is the CDFG corresponding to the datapath in Fig. 6(b) explained before. Fig. 8(a) shows the corresponding PDG. $RR_i$ denotes register-to-register delay between two CDFG node. Fig. 8(b) computes this delay for $RR_5$. We will show that the PDG traversal for delay estimation, e.g. to find critical paths, is equivalent to the traversal of sensitizable paths in the datapath. We never traverse the false paths caused by resource binding since by the PDG traversal, we follow only the paths which actually transfer data.

## 3. False Paths in Multi-Cycle Analysis

For some applications such as testable designs and non synthesized datapaths in addition to the longest register-to-register delay, it can be very useful to have *multi-cycle* critical sensitizable paths. By definition, a multi-cycle critical path is a sequence of register-to-register transfers from primary inputs to the primary outputs, derived by analysis of a datapath and scheduled DFG simultaneously. Our work is really an extension of multi-cycle false path detection at the logic level introduced in [16]. in which the authors have shown that the notion of false paths, traditionally defined for combinational logic circuits, can be extended to the sequential context by considering the operation of the circuit over multiple clock cycles.

### 3.1 Structure-Based Versus CDFG-Based Analysis

Depending on user's objectives, for delay estimation there are two choices. In the *structure-based* (static) delay estimation, the design is swept from inputs to outputs and the longest input-to-output delay is found without considering any feedback or activity sequence of components. In contrast to the static delay estimation, a *CDFG-based* (multi-cycle) delay estimation evaluates the overall execution time, at the RTL, logic or gate level by considering the activity sequence (e.g. control steps in a scheduled DFG) of components.

Static and multi-cycle delay estimations are important for evaluation and modification of a design, especially at the lower levels. For example, the placement of the cells may be modified or long wires may be broken based on static delay estimation to achieve higher performance. Multi-cycle delay estimation, in particular, can provide useful guidelines for performance-driven synthesis tools (e.g. layout generators) to balance the cycle time and improve the overall performance.

### 3.2 Multi-Cycle False Paths Caused by Binding

Using the method presented in section 2.3 for register-to-register delay analysis, we can avoid the false paths caused by resource binding when doing multi-cycle delay analysis.

#### 3.2.1 Example 6: Resource Binding

Fig. 9 is a datapath for HAL benchmark, selected from [19]. Let all registers have the same delay and ignore wire delays. In the search of critical multi-cycle path in the datapath, we have:

$$\begin{cases} \text{False Path:} & T(P_f) = T^{*}_{ALU2} + T_{R1} + T^{-}_{ALU4} + T_{R2} + T_{BUS3} + \\ & T^{*}_{ALU1} + T_{R3} + T_{BUS1} + T^{+}_{ALU3} + T_{R5} + T^{>}_{ALU5} \\ \text{Sens. Path:} & T(P_s) = T_{BUS3} + T^{*}_{ALU1} + T_{R3} + T_{MUX2} + T^{*}_{ALU2} + \\ & T_{R1} + T^{-}_{ALU4} + T_{R2} + T_{BUS3} + T^{-}_{ALU4} + T_{R2} \\ \text{Error:} & \Delta T = |T(P_f) - T(P_s)| = T^{+}_{ALU3} + T^{>}_{ALU5} + T_{BUS1} - \\ & T^{-}_{ALU4} - T_{MUX2} - T_{BUS3}| \approx 3.39ns \end{cases}$$

### 3.3 Multi-Cycle False Paths Caused by Conditions

The work presented in [4] described a heuristic algorithm for the detection and elimination of false paths during path-based scheduling. However, as explained in [4], this heuristic has shortcomings when used in delay estimation. Due to lack of space, we do not pursue this issue here.

## 4. Timing Model for the Control Unit

The control unit commands the components in datapath what to do (according to their functionality) and when to do (according to the scheduled DFG). The control signals can be described as a control-state table, state diagram or a FSM chart that specifies the next-state and control signals (outputs of controller) as a function of present states and conditional/status signals (inputs of controller). In practice a number of optimization procedures, such as state minimization, state assignment (encoding) and finite state machine partitioning are applied in order to improve the performance of the control logic. Some works such as [8] consider a fix simplified model (e.g. random logic)and estimate the delay of the control unit. The main shortcoming of this strategy to estimate the controller delay is that it's simply too restrictive. The timing model depends heavily on the design optimization procedures, in general, and chosen architecture, in particular, and may not be generalized.

### 4.1 A Unified Timing Model for Datapath and Controller

As explained before, the basic tool of delay estimation in our approach is the PDG traversal. To consider the control delay in PDG, we need a unified timing model for datapath and controller. Fig. 10 shows how we join these two delays together. For brevity, we have shown only one component ($M$) of the datapath.

Let's consider the delay of this unit drawn again in Fig. 11(a). Although the propagation delay of this component (see $T_p$ in Fig. 11(b)) is available in the library, it does not include the delay of control lines. In fact, $T_p$ has been computed with the assumption that input lines $x$ and control lines $c$ are ready simultaneously. In reality, we have to differentiate between these signals ($x$ and $c$) since they are generated by different circuits (i.e. datapath and controller) through different paths and Fig. 11(b) is an oversimplified model hiding the important role of control delay. A more realistic model is pictured in Fig. 11(c) showing that the delay of a component with control lines is $T_p + T_c$ where $T_c$ is the delay of the circuit (controller) which generates control signals.

The clock period $T_{clock}$ should be determined by considering $T_c$ and $T_p$ simultaneously. In a scheduled DFG all operations are synchronized by clock cycles (time step). The duration of this clock not only depends on propagation delay of the components contributing in a specific time step but also depends on the controller delay. The important advantage of using PDG in conjunction with a delay analyzer of controller is that we don't *pessimisticly* estimate $T_{clock}$ as $MAX\{T_{c_i}\} + MAX\{T_{p_i}\}$ over different time steps $i$. Instead, we use the estimate of control path delay at step $i$ ($T_{c_i}$) obtained by a gate level analyzer, to modify overall delay of components (augmented edges in PDG). This realistic view leads to $MAX\{T_{c_i} + T_{p_i}\}$ over different time steps $i$ as more accurate estimate of $T_{clock}$. $T_{p_i}$ is obtained from data sheets (library of components) and $T_{c_i}$ can be obtained accurately (by analyzing the controller for the corresponding path(s)) or roughly (by considering the critical path in controller).

#### 4.1.1 Example 7: Path Mismatching

As we have shown the datapath and controller both contribute to the overall delay estimation of the system clock period ($T_{clock}$). However, the datapath and controller interact with each other in a specific order based on the CDFG flow and their structures. In other words, their interaction follows

a specific ordered pair of matching path segments which includes *components/connections* in the datapath and the controller. Arbitrary selection of pair of segments (or selection based on the individual critical path) may lead to an impossible ordering and path mismatching, (a false interaction path) and consequently lead to an unrealistic delay analysis.

Fig. 12 shows an example. Because of the multiplication (slowest operation) in Fig. 12(a) we have to find the frequency bound based on the activities in step 5. Random logic implementation of the controller by COMPASS toolset [15] shows a mismatch between the paths. Specifically, the critical delays of generating *R2_load* and *R3_load* are 0.62 and 1.92 nanosecond, respectively. Intuitively, the decoding part to generate control signals for adder and it's surrounding components (MUX1, MUX2, R3 and R4) is larger (more logic gates/levels) than the multiplier and it's surrounding component (R2). Thus, considering only the activities at time step 5 and also assuming the delay of registers are the same and $T^{*}_{(*)} > T^{+}_{(+)}$, we have:

$$\begin{cases} \text{False Path:} & T_{clock}(P_f) = T^{e}_{R1} + T_{c_{R3}\_load} + T^{*}_{(*)} + T^{s}_{R2} \\ \text{Sensitizable Path:} & T_{clock}(P_s) = T^{e}_{R1} + T_{c_{R2}\_load} + T^{*}_{(*)} + T^{s}_{R2} \\ \text{Error:} & \Delta T = |T_{clock}(P_f) - T_{clock}(P_s)| = \\ & |T_{c_{R3}\_load} - T_{c_{R2}\_load}| \approx 1.30ns \end{cases}$$

Note that this means almost 15% error in estimating the $T_{clock}$ lower bound.

## 5. Delay Estimation Process at RT Level

◦ **Inputs:** The required data for the RTL delay estimation process are: DFG/CDFG, datapath, connection matrix $C$ showing the connection length between components and technology parameters, including: 1) parasitic factors for components, i.e. $C_{in}$, $R_{out}$ and $T_{comp}$; and 2) wire parameters, i.e. $R_s$, $\epsilon$, $W$ and $t$.

◦ **Outputs:** The output of the algorithm is the sensitizable critical paths (register-to-register, static and multi-cycle) and their corresponding delay values.

The delay estimation algorithm analyzes the RTL datapath in the following four steps:

- *Step 1*: Construct the PDG (Propagation Delay Graph). PDG is really CDFG augmented by adding register-to-register delay to its edges. We add a *source* node, connected to all primary inputs (i.e. nodes with no predecessors), and a *sink* node, connected to all primary outputs (i.e. nodes with no successors). The delay value associated to an edge in CDFG is the summation of propagation delays of components and wires through which the signal passes between two operation nodes.
- *Step 2*: Identify the critical path in datapath.
  To find the critical path (register-to-register, static and multi-cycle), we apply the *Depth First Search (DFS)* algorithm to the PDG graph, using the delays found in the previous step.
- *Step 3*: Compute the frequency bound for the system clock.
  The clock period is determined as the worst register-to-register (RR) delay in a design. As we mentioned before, in a PDG we augment every edge by a weight equal to the delay of the path between two registers. Let $T_{RR_k}$ denote the propagation delay of a typical RR node with the source $src_k$ and destination $dest_k$:

$$\begin{cases} (T_{clock})_{min} = Max\{[(1/c_k) * T_{RR_k}^{src_k \rightarrow dest_k}] + T_{c_k}\} \;\; ; \;\; 1 \le k \le m \\ (f_{clock})_{max} = \frac{1}{(T_{clock})_{min}} \end{cases}$$

## 6. Experimental Results

The delay estimation algorithm described in the previous sections has been implemented in C on a SUN SPARC-IPC workstation. The delay estimation results, in nanoseconds, for different experiments are tabulated in Tables 1, 2 and 3. All datapaths for these examples, except the first two in Table 1, have been produced by our synthesis tool, *SYNTEST* [17]. Note that the datapaths in the two tables are produced with different set of options and are not structurally equivalent. We used our own layout estimator at the RT level to estimate the wire lengths between the components [18].

The examples are HAL [19], FACET [22] and four digital filters including the *fifth order elliptical filter* chosen as a benchmark for the 1988 High-Level Synthesis Workshop. They have been widely used in the literature as benchmarks. The CPU time for running the delay estimator is less than four seconds for these examples. The wall clock time is about ten seconds. For comparison, we have implemented these designs (4-bit width) based on the 0.8 micron CMOS library [14] within COMPASS [15] on a SPARC-IPC workstation with 32M RAM. The ASIC synthesizer tool in COMPASS takes a datapath circuit description in VHDL and generates the complete layout, for these benchmarks, in a few minutes.

To show the importance of false path exclusion in delay estimation, we have selected datapaths which contain register-to-register false paths and are traceable by hand, shown in Table 1. The first two examples are selected from [19] and [23], respectively. The other two examples have been generated by us. This table highlights the importance of false paths in delay estimation. By excluding the false paths (the 6th column in Table 1), the estimated results are *realistic*, not *pessimistic*. That's why the estimated delays are usually less when we exclude shortly. For quality comparison, the worst case timing behavior was obtained by running *QTV*, the timing verifier in COMPASS on the layout. The results reported by QTV are assumed to be very close to the actual delay of the designs implemented in silicon. QTV can neither detect the false paths nor consider the number (or sequence) of component activities and thus can not compute the overall execution time directly. To be fair in our comparison, we compare the results of register-to-register and static delays separately. Table 2 compares the result of register-to-register and static (critical path) delays provided by the estimator and QTV. Estimated delays are within 14% accuracy of those reported by QTV.

In Table 2, we also present the *multi-cycle* delay analysis. In this analysis, the number of times that a component or a wire is active, and also the sequence of execution of operations, are considered. The PDG graph used in our delay estimator considers all of these factors. Multi-cycle analysis by QTV is not possible because every loop has to be broken and every register-to-register path has to be searched manually to compute the cumulative delay.

The experiments for datapath–controller interactions is tabulated in Table 3. The last column of this table ($T_{DP\&C}$) shows the most accurate and realistic estimate of register-to register delay (lower bound of clock period) by considering the sensitizable critical path of datapath and controller for all possible path mismatching when the datapath and controller interact. To show the importance of false paths and path mismatching in delay estimation we have performed additional analysis. The third column ($T_{DP}$) shows the critical path in the datapath only while the fourth column shows the pessimistic analysis of critical path without considering false paths and path mismatching. As these two examples show realistic values (last column) are 10-15% less than pessimistic estimation.

## 7. Conclusion

An efficient RTL delay estimation algorithm has been presented, to be used after the generation of a datapath circuit.

Table 1: Register-to-register (R-R) delay estimation

| Design Name | Sch. Step | # of RTL Comp. ALUs | MUXs | REGs | With Excl. | Without Excl. |
|---|---|---|---|---|---|---|
| HAL | 8 | 2 | 4 | 9 | 16.29 | 20.29 |
| FACET | 4 | 3 | 4 | 7 | 16.75 | 18.94 |
| AR Filter | 13 | 4 | 7 | 37 | 14.29 | 19.20 |
| Wave Filter | 17 | 6 | 10 | 28 | 14.58 | 20.68 |

The cornerstone of our work is the PDG graph annotating the timing information of schedule and datapath in its nodes. To achieve higher accuracy, RTL false paths caused by resource binding are avoided during register-to-register, static and multi-cycle delay estimation. We have also incorporated the controller delay in the estimation process by careful examination of the datapath–controller interaction and avoiding path mismatching.

## References


[1] D. Knapp, "Manual Rescheduling and Incremental Repair of Register Level Datapaths," in Proc. ICCAD-89.

[2] J. Silva, K. Sakallah and L. Vigidal, "FPD: An Environment for Exact Timing Analysis," in Proc. ICCAD-91.

[3] S. Perremans, L. Claesen and H. Man, "Static Timing Analysis of Dynamically Sensitizable Paths," in Proc. 29th DAC, June 1992.

[4] R. Bergamaschi, "The Effect of False Paths in High Level Synthesis" in Proc. ICCAD-91, Nov. 1991.

[5] L. Stok, "False Loops through Resource Sharing" in Proc. ICCAD-92, Nov. 1992.

[6] J. Weng and A. Parker "3D Scheduling: High-Level Synthesis with Floorplanning," in Proc. 28th DAC, 1991.

[7] D. Knapp, "Datapath Optimization Using Feedback," in Proc. EDAC-91, Feb. 1991.

[8] V. Chaiyakul, A. Wu and D. Gajski, "Timing Models for High Level Synthesis," in Proc. EURO-DAC 92.

[9] C. Ramachandran and F. Kurdahi, "Combined Topological and Functionality Based Delay Estimation Using a Layout-Driven Approach for High Level Applications," in Proc. EURO-DAC 92, Sept. 1992.

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

[11] B. Haroun and M. Elmasry, "Architectural Synthesis for DSP Silicon Compiler," IEEE Trans. on CAD, April 1989.

[12] M. McFarland, "Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Description," in Proc. 23rd DAC, July 1986.

[13] F. Brewer and D. Gajski, "Chippe: A System for Constraint Driven Behavioral Synthesis," IEEE Trans. on CAD, July 1990.

[14] VLSI Technology, "0.8-Micron CMOS VSC450 Portable Library," VLSI Technology, Inc., 1993.

[15] Compass Design Automation, "User Manuals for COMPASS VLSI V8R4.4," Compass Inc., 1993.

[16] P. Ashar, S. Dey and S. Malik, "Exploiting Multi-Cycle False Paths in the Performance Optimization of Sequential Circuits," in Proc. ICCAD-92, 1992.

[17] H. Harmanani, C. Papachristou, S. Chiu and M. Nourani, "SYNTEST: An Environment for System-Level Design for Test," in Proc. EURO-DAC 92, Sept. 1992.

[18] M. Nourani and C. Papachristou, "A Layout Estimation Algorithm for RTL Datapaths," in Proc. 30th DAC.

[19] P. Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," IEEE Trans. on CAD, June 1989.

[20] F. Brewer and D. Gajski, "Knowledge Based Control in Micro-Architecture Design," in Proc. 24th DAC.

[21] M. Abramovici, M. Breuer, A. Friedman, Digital Systems Testing and Testable Designs, Computer Science, 1990.

[22] C. Tseng and D. Siewiorek, "FACET: A Procedure for the Automated Synthesis of Digital Systems," in Proc. 20th DAC, June 1983.

[23] D. Gajski and D. Thomas, "Introduction to Silicon Compilation," in Silicon Compilation, D. Gajski (Ed.) Addison-Wesley, 1988.


Table 2: Comparison with QTV (COMPASS time analyzer)

| Design Name | Sch. Step | # of (A,M,R) | Estimated Crit. Path | | | QTV Crit. Path | |
|---|---|---|---|---|---|---|---|
| | | | R-R | Static | Multi | R-R | Static |
| HAL | 4 | (7,4,17) | 13.40 | 36.21 | 33.79 | 14.30 | 31.12 |
| Biquad | 10 | (7,14,23) | 14.29 | 88.94 | 61.26 | 15.05 | 83.61 |
| AR | 13 | (3,6,39) | 17.20 | 55.52 | 106.80 | 16.53 | 57.27 |
| BP | 13 | (3,6,33) | 19.29 | 54.46 | 71.38 | 17.60 | 47.53 |
| Wave | 21 | (4,8,32) | 14.78 | 64.31 | 165.21 | 13.67 | 70.18 |

Table 3: The effect of datapath–controller interaction in register-to-register delay estimation

| Design Name | # of (A,M,R) | $T_{DP}$ (Datapath) | $T_{DP\&C}$ (Pessimistic) | $T_{DP\&C}$ (Realistic) |
|---|---|---|---|---|
| Biquad Filter | (7,8,14) | 11.98 | 20.51 | 18.54 |
| AR Filter | (9,18,11) | 13.20 | 21.68 | 19.46 |
| BP Filter | (8,12,12) | 12.29 | 21.10 | 18.36 |
| Wave Filter | (3,6,18) | 13.07 | 25.49 | 22.21 |

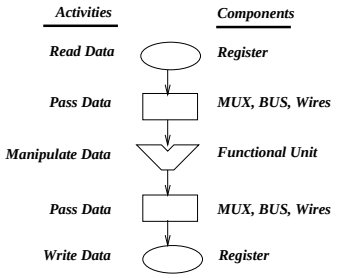


Figure 1: Register-to-register transfer path

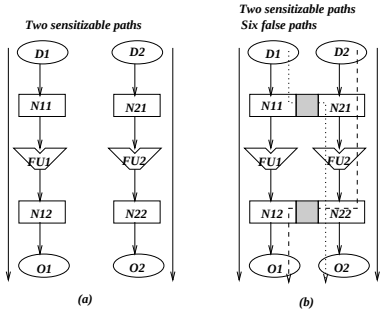


Figure 2: Creation of false paths: (a) Independent paths, (b) Shared components

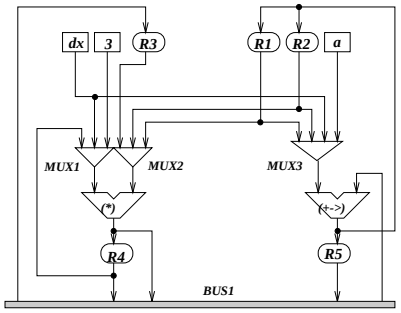


Figure 3: Creation of false paths because of binding decisions

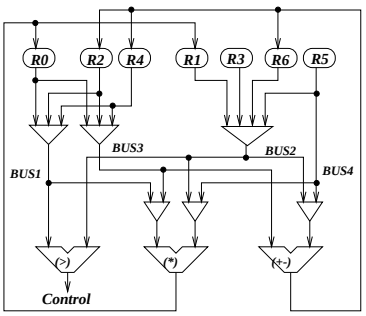


Figure 4: Creation of false paths because of multi-function ALU

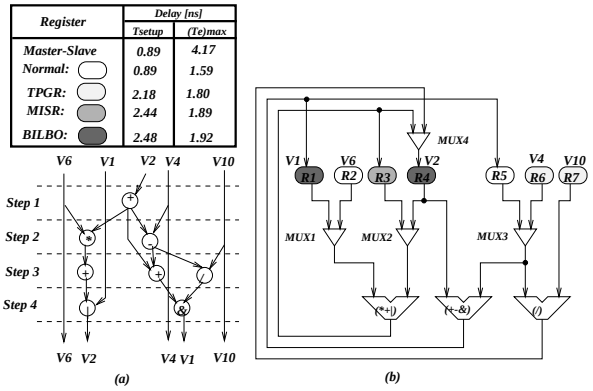


Figure 5: Creation of false paths because of testability insertion: (a) Schedule, (b) Testable datapath

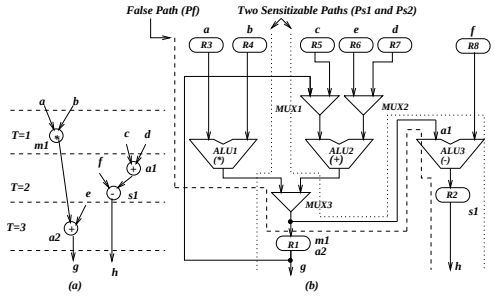


Figure 6: Creation of false path because of chaining: (a) Scheduled DFG, (b) Datapath

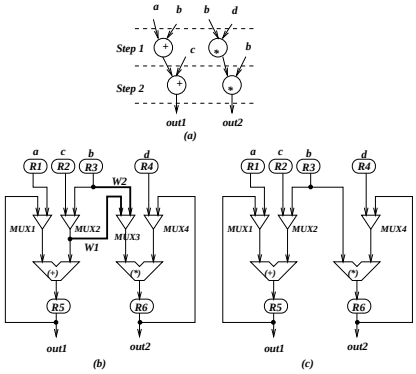


Figure 7: Creation of false paths because of redundant component: (a) Datapath, (b) After removing redundant component

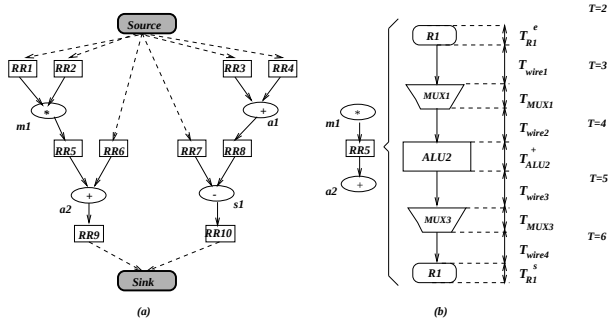


Figure 8: Propagation Delay Graph: (a) Complete PDG, (b) Delay of a typical node

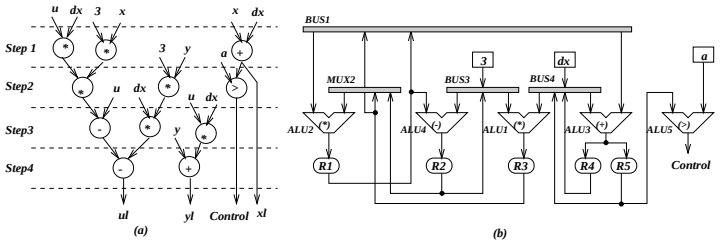


Figure 9: Multi-cycle false paths in static and multi-cycle analysis: (a) CDFG (c) Datapath

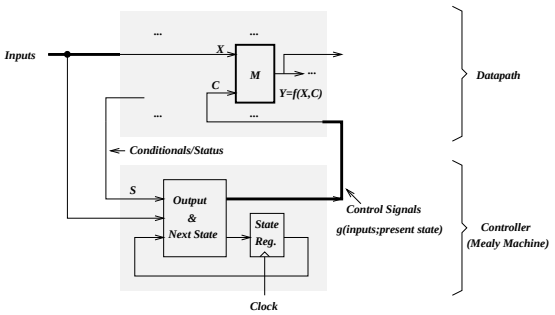


Figure 10: A unified timing model for datapath and controller

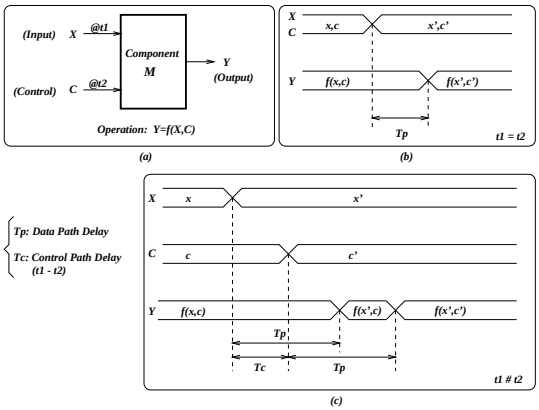


Figure 11: The effect of delay on control lines (controller delay)

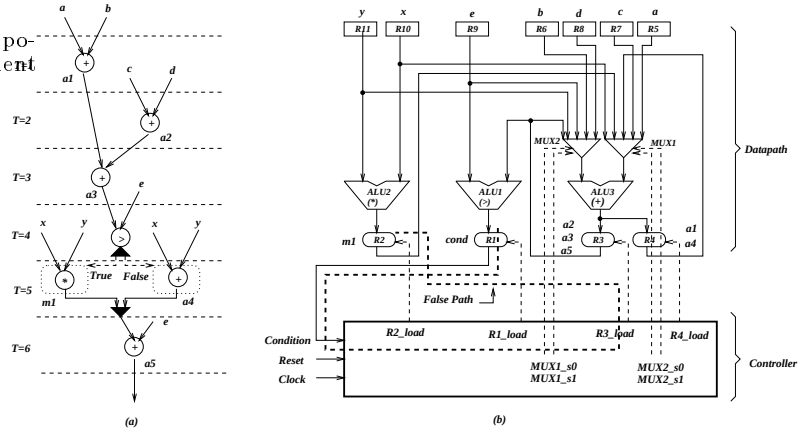


Figure 12: An example of false paths caused by path mismatching