Interface Timing Verification Drives System Design

Ajay J. Daga
Peter R. Suaris
Interconnectix, a Mentor Graphics Business
10220 S.W. Nimbus Avenue, Bldg. K4
Portland, OR, 97223
(503) 684-6641
{ajay, peter}@icx.com

Abstract

System design, i.e. the design of board-level circuits and systems-on-a-chip, focuses on the integration of off-the-shelf and application-specific VLSI components. A key aspect of system design is to ensure the satisfaction of component interface timing requirements. This is necessary for the correct exchange of information among components on a system. We present a methodology for the interface timing verification and subsequent timing-driven floorplanning of systems. We present results on the application of this methodology to real-world circuits.

I. Introduction

Increasing circuit complexities have progressively raised the level of abstraction at which design is performed. Also, shorter design cycles have placed a premium on design reuse. Board-level circuits are routinely composed of dozens of complex VLSI components, many of which are available off-the-shelf, while other components are application specific. Similarly, IC design is approaching the systems-on-a-chip era, where reusable intellectual property is assembled along with synthesizable blocks to yield desired functionality. We refer to the design of both systems-on-a-chip and board-level circuits as system design.

System design commences with the specification of a circuit as composed of interconnected components. At this stage, the required behavior of each component is well documented. Subsequently, components are implemented either by reusing off-the-shelf intellectual property, or through synthesis. A key design validation task for the system is to ensure that the interaction among components conforms to specifications. This task, called interface timing validation, takes component and interconnect delay into account to determine if timing constraints associated with the information exchange among components are satisfied. Interface timing validation is particularly crucial given the high speeds at which circuits operate; bus-speeds of 100 MHz are not uncommon on today's board-level circuits. At such high speeds, managing component and interconnect delay is central to successful system design.

Simulation as an approach to interface timing validation is cumbersome, time consuming and resource intensive. Besides, the extent to which timing problems are discovered using simulation is entirely dependent on the stimuli provided by a design engineer. To obtain complete results in an efficient manner, design engineers have increasingly turned toward verification technology. Verification approaches accept as input a circuit description, requirements on circuit behavior, and automatically, i.e. without user-supplied stimulus, determine if requirements are satisfied. Since design engineers do not provide any stimulus, they do not impact the completeness of verification results.

Static timing analysis [8] is the most commonly available commercial solution to the timing verification problem. Static analysis was developed to address the gate-level timing verification problem, where a circuit is composed of combinational logic blocks (CLBs), flip-flops and latches, and the objective is to verify the satisfaction of set-up and hold time requirements. When modeling component behavior, static analysis ignores functional information (e.g., the Boolean function computed by a CLB output), while capturing temporal information (timing delays from input to output, set-up/hold times, etc.).

By ignoring functional information, static analysis is able to efficiently analyze the timing characteristics of realistic gate-level circuits. However, for the same reason, static analysis reports timing violations, called false violations, that represent fictitious problems. False violations result from traversing false paths [4] during static analysis. False violations are problematic because they require design engineers to do one of two things:

1) Manually sift through the list of violations and determine which are false. This task is cumbersome and potentially error-prone.

2) Assume that most violations are true. This results in conservative designs that do not push performance limits, and wasted effort in unnecessarily changing error-free portions of a design.

In this paper we present a methodology for the interface timing verification of systems using commercially available tools, Tau and IS Floorplanner from Interconnectix. This methodology provides accurate timing verification results that are used to drive the physical design of systems.

Tau is based on a verification technology called symbolic timing analysis [3]. Symbolic analysis takes as input a circuit description, and for each component in the circuit a bus-functional timing model. In addition to capturing timing delay and constraint information, bus-functional models encapsulate functional aspects of component interface behavior (e.g., how read and write transactions are performed). Symbolic analysis automatically enumerates the different temporal behaviors on a circuit, and the conditions under which these behaviors occur. Timing constraints that apply to each temporal behavior are verified. Symbolic Boolean manipulation is used to maintain consistency during analysis, i.e. to prevent analyzing the impact of two behaviors that are mutually inconsistent.

Under the assumption that bus-functional models accurately capture component interface behavior, symbolic analysis identifies all timing violations on a circuit without reporting any false violations. Also, symbolic analysis accepts and verifies asynchronous behavior and constraints. Symbolic analysis is efficient because it separately analyzes only those behaviors that are temporally unique. Temporally equivalent behaviors are grouped together. In this manner, an implicit enumeration of the space of possible circuit behaviors is performed, i.e. exhaustive results are provided without separately, or explicitly, examining all individual circuit behaviors.

Permission to make digital/hard copy of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
In addition to the benefits resulting from the use of symbolic analysis, \( \tau \), in integration with the IS Floorplanner provides design engineers with a complete timing-driven physical design solution. Design engineers use \( \tau \) to automatically identify the critical paths on a circuit and the slack available for interconnect delay consumption. This information is obtained well before physical design commences. As \( \tau \) takes functional information into account, false critical paths are not reported, thereby allowing design engineers to focus on the real physical design challenges posed by their circuits. Information on the critical interconnect paths of a circuit are exported from \( \tau \) to the IS Floorplanner in the form of relative path-delay equations.

Relative path-delay equations constrain the permissible delay variation between a maximum interconnect delay path and a minimum interconnect delay path (e.g., delay along a maximum path subtracted from the delay along a minimum path should be less than \( x \)). These equations provide more versatility in interconnect delay management, than afforded by an absolute path-delay constraint (e.g., delay along a maximum path should be less than \( x \)), or a pin-pair delay constraint (e.g., delay from an output pin to an input pin should be less than \( x \)).

The IS Floorplanner is used to interactively converge on a floorplan that satisfies the delay equations established by \( \tau \). Interconnect delays are computed in real-time, using a Manhattan estimate for pin-pair delay, for each change in floorplan (e.g., if component placement or net topology is changed). This, in turn, allows a real-time update on the satisfaction of path-delay equations. Once floorplanning is complete, the path-delay equations are transformed to pin-pair constraints through a slack allocation process. Pin-pair constraints are used to drive the subsequent routing process. \( \tau \)'s integration with the IS Floorplanner significantly reduces the number of iterations required to obtain a physical design that is consistent with the timing requirements of a circuit.

The rest of this paper is organized as follows. In section II we describe how symbolic analysis is used to perform accurate and efficient timing verification of systems. In section III we describe a timing-driven floorplanning process that guides engineers toward a correct physical implementation. In Section IV we present experimental results on the application of this methodology to actual design scenarios. Section V summarizes the contributions of this paper. For a discussion of the theoretical foundations of this work we refer the reader to the following papers [2, 3].

II. Interface Timing Verification

Consider the simple system shown in Figure 2.1. It consists of a bus controller that communicates with two memory banks through some glue circuitry (decoder, wait-state generator, etc.). In addition to the circuit netlist in Figure 2.1, interface timing verification requires bus-functional timing models for each component in the system.

2.1 Bus-Functional Timing Models

There are two aspects to the information contained in a bus-functional timing model: temporal and functional. Temporal information defines the propagation delay in placing values on output signals and timing constraints on input behavior. Functional information describes, through a finite state machine (FSM), when a component changes values on its output signals and latches values on its input signals.

Figure 2.1: Example system.

Temporal Information

Figure 2.2 illustrates temporal information for the bus controller. Temporal information is specified through delay and constraint relationships between signals on a component. Delay relationships establish lower and upper bounds on the time taken to propagate a signal value to an output, in response to a transition on an input signal. The lower bound on a delay \( x \) is denoted \( x^- \), while the upper bound is denoted \( x^+ \). Lower and upper bounds reflect the impact of process variations and operating conditions on component delay. For example, timing link \( tDZ \) specifies the delay in driving \( DATA \) to high-impedance relative to the rising-edge of \( CLK \); \( tDZ^- \) is 8 ns and \( tDZ^+ \) is 12 ns. Similarly, \( tDS \) specifies the delay in placing a stable value on \( DATA \) relative to a transition on \( CLK \).

Figure 2.2: Timing specification for the bus controller.

Typically, delays within a component, i.e. within the same physical package, tend to track each other, i.e. are correlated. Consequently, when a propagation delay on a component is assumed to exhibit its upper bound delay value, it is unrealistic to assume that another propagation delay on the same component will exhibit its lower bound delay value. Delay correlation is a simplistic (non-stochastic) means of specifying the expected variation between delay values on a component.

Figure 2.3 illustrates the delay correlation specification for the bus controller. The output signals on the bus controller are grouped into two correlation classes: \( DATA \) and \( control \). The \( DATA \) class includes signals \( ADDR \) and \( DATA \), while the \( control \) class includes signals \( SEL \) and \( WR \). Delay correlation within each class is specified to be 0.4 ns, while correlation between classes is specified to be 1.2 ns. Delay correlation between classes \( a \) and \( b \) is denoted \( f(a, b) \). The delay correlation specification in Figure 2.3 specifies, for example, that if the delay, \( tWR \), on \( WR \) is at its upper bound value, \( tWR^+ \), then the delay, \( tDZ \) on \( DATA \) can be no lower than \( tDZ^+ \). Note that in the absence of this specification a timing verifier could assume that the delay on \( DATA \) is \( tDZ^- \) (8 ns) when the delay on \( WR \) is \( tWR^+ \).

Figure 2.3: Delay correlation spec. for the bus controller.
Constraint relationships establish requirements on the time-separation between changes in input signal values. For example, timing link t\_SH in Figure 2.4 specifies, for an SRAM, the set-up and hold requirements on IO relative to the rising-edge of \( \overline{WE} \). These times define the minimum duration before and after a rising-edge transition on \( \overline{WE} \) for which the value on IO must be stable to be correctly latched by an SRAM. In addition to set-up and hold requirements, pulse-width constraints establish the minimum duration that must elapse between transitions on the same input signal (\( t_{PW} \) in Figure 2.4 is a pulse-width constraint on the address signal of an SRAM).

**Figure 2.4: Timing specification for an SRAM.**

**Functional Information**

Temporal information merely establishes the propagation delays and timing constraints on a component. To further describe the interface behavior of a component it is also necessary to know when output signal values are changed, when input signal values are latched, and how output values are a function of input values. Interface Finite State Machines (IFSMs) [2] are used to describe this information. While conventional FSM formalisms describe behavior in terms of output values, IFSMs focus on the switching characteristics of a component, i.e. the output signals that transition and the input signals that are latched at each state transition. These are called relevant signals. IFSMs are uniformly applicable for the description of synchronous as well as asynchronous behavior.

Figure 2.5 illustrates a portion of the IFSM behavior for the bus controller in Figure 2.1. The bus controller has three states: read, write and idle. The state-transition from idle to write takes place when input signal \( \overline{I} \) is low and \( \overline{W} \) is high (the bus controller is synchronous and all state-transitions take place at the rising edge of \( \overline{CLK} \)). For each state-transition the relevant signal column lists output signals whose values are changed and input signals whose values are latched. For example, at the idle to write transition on the bus controller, output signal values on ADDR, DATA and SEL are changed, while values on input signals \( \overline{I} \), \( \overline{W} \), ADDR\_IN and DATA\_IN are latched. Note that the value latched on ADDR\_IN is, in turn, driven onto ADDR, and the value latched on DATA\_IN is driven onto DATA.

![IFSM description for the bus controller.](image)

**Figure 2.5: IFSM description for the bus controller.**

Now, consider the description of the asynchronous SRAM IFSM shown in Figure 2.6. The SRAM also has three states, read, write and idle. As the SRAM is asynchronous the state-transition from write to idle, for example, takes place at the instant when the transition condition is satisfied. As shown in Figure 2.6, when the SRAM transitions from write to idle the value on IO is latched.

![IFSM description for the SRAM.](image)

**Figure 2.6: IFSM description for the SRAM.**

The functional and temporal information described by a bus-functional timing model complement each other to provide an adequate description of component interface behavior for timing verification purposes. The functional information contained in an IFSM specifies when output signals transition and values on input signals are latched by a component. The timing information, in turn, specifies the delay associated with an output signal transition, and the set-up/hold constraints that must be satisfied when the value on an input signal is latched.

**2.2 Symbolic Timing Analysis**

Given the netlist for a system, bus-functional timing models for each component in the system, and a description of the clock waveforms on the system, symbolic timing analysis [3] automatically identifies any, and all, timing violations on a circuit. Systems often have clocks that are completely asynchronous to each other, or operate at different frequencies. Figure 2.7 describes the clock waveforms for the circuit in Figure 2.1. This system has two clocks that operate at 100 MHz with a \( \pm 0.5 \) ns skew between their rising edges.

![Clock specification.](image)

**Figure 2.7: Clock specification.**

Symbolic analysis commences by first identifying the master components in a circuit. A master component is one whose behavior from its initial state is only a function of primary input values. Next, all components are placed in their initial state. Then, starting with a master component the different behaviors on the output signals of the component are enumerated. The condition when each of these behaviors occurs is captured symbolically, and is used to ensure consistency during analysis. After the behaviors on a component \( x \) have been identified the behaviors on components connected to \( x \) and in response to \( x \) are determined. During this process timing constraints are verified. This process continues from component to component, until all components on a system have returned to their initial state. This approach is, in effect, a form of symbolic simulation [1], where many behaviors on a circuit are simultaneously analyzed without any user-supplied stimulus.

![Example behavior on SRAM1 Inputs.](image)

**Figure 2.8: Example behavior on SRAM1 Inputs.**

Consider analysis of the memory subsystem SRAM1 for the behavior shown in Figure 2.8, where the stable value on CS is ADDR\_IN\[19\]. There are two possible responses on SRAM1. If CS is high, SRAM1 remains in the idle state for the entire
Where $\text{difference in the actual time separation (}\tau_x)$ violates a timing constraint is satisfied, while negative slack indicates a timing violation. In Figure 2.10, the hold-time requirement of 1 ns, the slack equals $\tau_x - s_x$. The lower bound time of a transition $x$ occurs relative to the rising-edge of $\text{WE}$. Consider verification of the hold-time constraint on $\text{IO}$.

**Relative Path-Delay Equations**

We have shown how the slack, $s(x, y)$, associated with a timing constraint between transitions $x$ and $y$ is computed. Note that $s(x, y)$ is computed under the assumption that there is zero delay contribution from the interconnect-delay constituents of $x$ and $y$. To drive physical design we constrain the permissible delay on the interconnect delay constituents of $x$ and $y$, so that the available slack $s(x, y)$ is not entirely consumed when physical design is completed. In this context, let $y^+_i$ denote the upper bound interconnect delay contribution on $x$, and $x^+_i$ the lower bound interconnect delay contribution on $x$. For the timing constraint between $x$ and $y$ to remain satisfied after physical design is completed it must be the case that $y^+_i - x^+_i \leq s(x, y)$. In effect, the slack $s(x, y)$ is consumed by the interconnect delay constituent of $y$, $y^+_i$. If $y^+_i$ exceeds $s(x, y)$ then the timing constraint between $x$ and $y$ to remain satisfied, the interconnect delay contribution of $x$, $x^+_i$ must also increase to offset the increase along $y^+_i$.

The relative path-delay equation generated for the hold-time constraint on $\text{IO}$ relative to $\text{WE}$ is shown in Figure 2.11a. Figure 2.11b shows the delay equation for the set-up time constraint on $\text{IO}$ relative to $\text{WE}$.

To compute $t_d(x, y)$ between transitions $x$ and $y$, when one or more delay constituents on $x$ are correlated to those on $y$ it is necessary to compute the lower bound transition time on $x$, denoted $x^-$, and the upper bound transition time on $y$, denoted $y^+$, taking delay correlation into account. Then, $t_d(x, y) = \text{Min}(x^-, y^+)$. When computing $x^-$, if a delay constituent $a$ on $x$ is correlated to a delay constituent $b$ on $y$, and the correlation value is $\text{Max}(a^+, b^+)$, then the delay contribution of $a$ is not $a^+$ but $\text{Max}(a^+, f(a, b), a^+)$. This is because when delay values $a$ and $b$ are correlated, and $b$ is at its upper bound value $b^+$, then $a$ can be no less than $a^+$ - $f(a, b)$. Similarly, when computing $y^+$, if a delay constituent $a$ on $y$ is correlated to a delay constituent $b$ on $x$, and the correlation value is $f(a, b)$, then the delay contribution of $a$ is not $a^+$ but $\text{Min}(a^+, f(a, b), a^+)$. For the transitions in Figure 2.10, $\text{WE}^+$ is 26.2 ns; rather than use 5 ns as the delay contribution from $\text{controller} / \text{WR} : \text{tWR}$, 3.2 ns (2 + 1.2) is used. Also, $\text{IO}^+$ is 30.8 ns; rather than using 8 ns as the delay contribution from $\text{controller} / \text{DATA} : \text{tDZ}$, 10.8 ns (12 - 1.2) is used. As a result, $t_d(\text{IO}^+, \text{WE}^+) = \text{Max}(28 - 26.2, 30.8 - 28) = 1.8$ ns. Once delay correlation is taken into account the slack $s(\text{IO}, \text{WE})$ is 0.8 ns and the hold-time constraint on $\text{IO}$ is satisfied.
III. Timing-Driven Floorplanning

The relative path-delay equations generated by Tau are imported by the IS Floorplanner. For each change in the floorplan interconnect delays are computed and updated if they are constrained by these equations. Delay computations are made assuming manhattan estimates for the routes. During the initial stages of floorplanning, delays are estimated using pre-characterized formulas for RLC lines. Once a final floorplan is reached, the delays are estimated using a more accurate transmission line simulator.

Bus delays are abstracted from wire delays by establishing the worst-case delay value over all wires. For example, the bus delay (controller/DATA $\rightarrow$ SRAM1/I0)$^+$ is the maximum upper bound delay over each wire (e.g., (controller/DATA[0] $\rightarrow$ SRAM1/I0[0])$^+$).

Once a floorplan is complete, the design is ready to be routed. Before routing can be performed, relative path-delay equations must be partitioned into bus delay constraints, which in turn yield individual wire delay constraints that drive the router. The problem of breaking path-delay equations into wire delay constraints is called the constraint allocation problem. Existing solutions [5, 7, 9] to the constraint allocation problem consider the issue of establishing upper bound delay restrictions on wires. Our formulation, however, requires satisfying relative path-delay equations. Such equations may be satisfied not only by restricting upper bound delays along maximum delay paths, but by increasing lower bound delay requirements on minimum delay paths. This enhances the flexibility available to a router when attempting to meet timing requirements. Increasing lower bound delay requirements is equivalent to directing a router to add extra interconnect on a minimum delay path. For example, the equation in Figure 2.11b can be satisfied by either minimizing delay along the data path, which is 32 bits wide, or by adding interconnect to a single-bit wide control (WE) path. In some situations it is easier to do the latter. The next section describes our formulation and solution of the constraint allocation problem.

3.1 Constraint Allocation

Constraint allocation is the process of computing lower and upper bounds, called minimum and maximum delay allocations, within which bus delays must reside for a system of relative path-delay equations to be satisfied. Delay allocations depend on the estimated propagation delay on busses. For example, the minimum allocation has to be greater than the estimated minimum delay. Also a single bus can be constrained by a number of delay equations, requiring a simultaneous solution of all related equations to compute its allocations.

The total set of timing equations is denoted $E$. Also, $A^+(p)$ and $A^-(p)$ denote the maximum and minimum allocations for a path. The system of relative path-delay equations is:

$$A^+(p) - A^-(p) \leq \text{slack}, \forall i \in E \tag{1}$$

The interconnect delay constituents of a path $p$ are bus delays $a_j$, $j \in p$. Let $a^+_j$, $a^-_j$ denote the maximum and minimum allocations on a bus $j$. Then, a path-delay equation is expressed as follows:

$$\sum_{j \in p} a^+_j - \sum_{k \in p} a^-_k \leq \text{slack} \tag{2}$$

For an allocation to be feasible, we require that the minimum allocation on any bus be at least as large as the estimated minimum delay. Let the estimated maximum and minimum delay of a bus $j$ be denoted $d^+_j$, $d^-_j$, and the set of all constrained busses be $B$. The feasibility requirement is:

$$a^-_j \geq d^-_j \forall j \in B \tag{3}$$

Also, the maximum allocation must be greater than the minimum allocation:

$$a^+_j \geq d^+_j \forall j \in B \tag{4}$$

A solution to equations (2), (3) and (4) will provide a set of bus allocations. However, the amount of extra interconnect added must be minimized, as it can result in routing problems. Hence, equations (2), (3) and (4) must be solved subject to the following objective function:

$$\text{Minimize } \sum_{j \in B} (a^-_j - d^-_j) \tag{5}$$

The above formulation can be solved as a linear programming problem. However, such an approach is prohibitively expensive in terms of computation time. Instead, a heuristic approach was taken, consisting of the following steps:

1) Compute minimum and maximum allocations for paths constrained by path-delay equations.

2) Compute minimum allocations for bus with violations violating the maximum allocations for paths computed in Step 1.

3) Compute maximum allocations for busses.

**Computing Path Allocations (Step 1)**

During this step the values $A^+(p)$ and $A^-(p)$ are obtained for all constrained paths $p$. The computation is performed byanchoring the maximum allocation for a path $p$ to the maximum estimated delay along $p$. Then, the minimum allocation for all paths that are related to $p$ by a path-delay equation are computed based on the maximum allocation for $p$. This can sometimes result in an infeasible minimum path allocation that exceeds the assigned maximum allocation for a path. Hence, an iterative step must be performed until convergence. The algorithm is as follows:

1) For each allocation $A^+(p) = D^+(p)$, i.e. the maximum allocation is equal to the maximum estimated delay for each path.

2) Next, until we achieve convergence the following steps are performed for each path:

a) The minimum allocation for a path $p$ is computed by subtracting the maximum allocation for other paths in a delay equation containing $p$ from the slack for the equation. This is done over all equations that contain the path $p$.

b) As the minimum allocation could now exceed the maximum allocation, the maximum allocation is updated to be greater than or equal to the minimum allocation for a path. If a change is made to the maximum allocation then step 2 is performed again.

**Computing Bus Allocations (Steps 2 and 3)**

Once path allocations have been computed, bus allocations are computed. Previous work in this area [5, 7, 9] concentrates on the maximum delay allocation problem. Computing minimum delay allocations is not addressed. In [5] the maximum delay allocation for a bus is computed by determining a weight for the bus relative to other constrained busses on a path. The maximum delay allocation for a bus is its weight multiplied by the slack for the path. We use the same approach for determining maximum bus delay allocations.

Minimum bus allocations are computed so that their sum over all busses on a path exceeds the minimum allocation for the path. The difference between the assigned minimum allocation for a path $p$ (computed in Step 1) and the summation of minimum bus allocations for all busses belonging to $p$ is called the unassigned path allocation. We iteratively establish the minimum bus allocation by partitioning the unassigned path allocation into bus allocations using a weight (computed separately) for each bus on
a path. Iterations are required to ensure that minimum bus allocations do not exceed maximum bus allocations.

Illustration of Constraint Allocation

Consider the interconnect delay equations shown in Figure 2.11. Let $P1$ denote the path $controller/WR \rightarrow buffer/IN + buffer/OUT \rightarrow SRAM1/WE$ and $P2$ denote the path $controller/DATA \rightarrow SRAM1/IO$. The allocation equations for these paths are:

$\Delta t(P1) - \Delta t(P2) \leq 0.8$

$\Delta t(P2) - \Delta t(P1) \leq 16$

Let us assume that $\Delta t(P1) = 2, \Delta t(P1) = 1.8, \Delta t(P2) = 1$, and $\Delta t(P2) = 0.8$. Applying step 1 we get $\Delta t(P1) = 2$ and $\Delta t(P2) = 1$. Applying step 2a we get $\Delta t(P2) = \Delta t(P1) - 0.8 = 1.2$. Also, $\Delta t(P1) = \Delta t(P2) - 16 = -1.5$. Readjusting in step 2b we get $\Delta t(P2) = 1.2$. As $\Delta t(P2)$ is 1.2 ns and $\Delta t(P2)$ is 0.8 ns, the allocations imply that 0.4 ns of delay must be added in the form of extra interconnect on path $P2$.

IV. Experimental Results

Experimental results on the application of the Tau and IS Floorplanner tools to board-level circuits are presented in Figure 4.1. We provide a short description of the circuits that these tools have been used on:

1) The design $accelerator$ has an application-specific bus controller, similar to the TURBOchannel bus. It communicates with four memory banks containing eight SRAM chips each.

2) The design $galaxy$ has dual 68000 microprocessors that communicate with four SRAM chips each. All glue circuitry is integrated into a single ASIC. A separate ASIC compares values on like signals on each 68000.

3) The design $pwrpc$ has a PowerPC 603 microprocessor that communicates with two SRAM chips.

4) The design $snif$ has an application-specific controller that communicates with 8 memory banks containing 12 SRAM chips each. The controller performs read and write accesses simultaneously to different halves of the memory subsystem. The read and write accesses are performed relative to asynchronous clocks that operate at different frequencies.

5) The design $truefire$ contains the memory controllers PMC and DBX for the Pentium Pro microprocessor. These chips communicate with four DRAM Dual In-Line Memory Modules.

6) The design $khw$ contains a PowerPC 603 microprocessor that communicates with level two cache. It also contains an Ethernet and ATM controller communicating with a PCI bus, and other application-specific circuitry.

<table>
<thead>
<tr>
<th>Name</th>
<th># Comp.</th>
<th>TV Time</th>
<th># Equations</th>
<th>CA Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>accel</td>
<td>41</td>
<td>4.65</td>
<td>137</td>
<td>61.2</td>
</tr>
<tr>
<td>galaxy</td>
<td>20</td>
<td>10.64</td>
<td>88</td>
<td>39.44</td>
</tr>
<tr>
<td>pwrpc</td>
<td>16</td>
<td>12.78</td>
<td>316</td>
<td>13.07</td>
</tr>
<tr>
<td>snif</td>
<td>119</td>
<td>51.34</td>
<td>178</td>
<td>113.41</td>
</tr>
<tr>
<td>truefire</td>
<td>6</td>
<td>4.83</td>
<td>96</td>
<td>8.9</td>
</tr>
<tr>
<td>khw</td>
<td>251</td>
<td>312.3</td>
<td>433</td>
<td>†</td>
</tr>
</tbody>
</table>

Figure 4.1: Experimental results.

As a measure of circuit complexity we have listed, in Figure 4.1, the number of components on each design (under the column # Comp.). For each circuit we have listed the timing verification time in seconds in Tau (under the column TV Time), as well as the time, in seconds, taken to perform constraint allocation in the IS Floorplanner (under the column CA Time). All times were obtained on a Sparc 10 platform operating at 41 MHz and with 64 Mbytes of RAM. For each circuit we have listed the number of delay equations generated. This is a measure of the behavioral complexity of a circuit as it reflects the number of timing constraints that must be satisfied. As may be observed from Figure 4.1, timing verification and constraint allocation times were on the order of a few minutes or less. These results are encouraging given that these systems are representative of typical board-level circuits.

V. Summary

We have described a design methodology that offers the following value to design engineers:

1) A reduction in both the time and resources required to perform system design by using a timing-verification tool, Tau, that does not report false violations. Also, by not reporting false violations Tau allows engineers to push the performance limits of their designs.

2) By driving the physical-design process, the iterations required to achieve a circuit implementation that meets timing requirements are significantly reduced. The IS Floorplanner uses the relative path-delay equations generated by Tau to guide engineers toward a physical implementation that is consistent with system timing requirements.

Experimental results demonstrate the viability of this methodology when applied to a broad range of board-level circuits.

Acknowledgements

The authors thank Greg Aldrich for his useful feedback on the contents of this paper.

References


† The $khw$ design had yet to be floorplanned when this paper was written.