Formal Timing Analysis of Digital Circuits

Qurat-ul-Ain and Osman Hasan

School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST)
Islamabad, Pakistan.
{qain.msee15seecs,osman.hasan}@seecs.nust.edu.pk

Abstract. Formal verification provides complete and sound analysis results and has widely been advocated for the functional verification of digital circuits. Besides the functional verification, a very important aspect of digital circuit design process is their timing analysis. However, despite its importance and critical nature, timing analysis is usually performed using traditional techniques, like gate-level simulation or static timing analysis, which provide approximate results due to their in-exhaustive nature and thus may lead to an undesired functional behavior as well. To overcome these issues, we propose a generic framework to conduct the formal timing analysis using the Uppaal model checker in this paper. The first step in the proposed framework is to represent the timing characteristics of the given digital circuit using a state transition diagram in Uppaal. In this model, delays are integrated using the corresponding technology parameters and the information about timing paths is added using Quratus Prime Pro, which is used as a path extracting tool. The Uppaal timing model is then verified through TCTL properties to obtain timing related information, like maximum delay. For illustration purposes, we present the analysis of a number of real-world digital circuits, like Full Adder, 4-Bit Ripple Carry Adder, Shift Registers as well as C17, S27, S208, and S386 benchmark circuits.

Keywords: Timed Automata, Uppaal, Formal Verification, Timing Analysis, Model Checking

1 Introduction

Due to the gradual reduction in transistor sizing governed by the Moore's law and the continuous increase in integrated circuit complexity, modeling and analyzing timing characteristics of digital circuits has become a very challenging task. Timing analysis usually involves determining the timing delays associated with each component of the circuit based on the technology used and its fan-out while considering the circuit variations. The delays of individual components are then used to calculate the overall circuit delay using various analysis techniques, like gate-level simulation [25] or static timing analysis [16]. However, neither of these techniques can ensure an exhaustive analysis due to the complexity of the present-age digital circuits. This kind of an in-exhaustive analysis results in an

incorrect timing analysis, which may in turn lead to a non-optimal design or a functional bug. Digital circuits are increasingly being used in designing safety-critical systems, like the ones used in health-care, transportation and defense related domains. Thus, a non-optimal design or a functional issue may lead to disastrous consequences, like financial losses or even the loss of human life in worst case scenarios.

Formal verification [14] is known to overcome the above-mentioned limitations of traditional analysis approaches, like simulation. It has been extensively used for the functional verification of digital circuits [4, 9, 15, 27]. The main idea in formal verification based analysis is to construct a formal model of the given circuit and formally verify the desired behavior of this model using formal specifications. Model checking [7] is one of the most commonly used formal verification techniques for the functional verification of digital circuits due to its automatic verification and the ability to provide a counterexample in the case of a failing property. It mainly involves modeling the system as a state transition diagram and the verification is done by exhaustively exploring the state space in a push button manner.

Due to the dire need of accurate analysis in the domain of timing analysis, formal verification of timing analysis of digital circuits using model checking got some attention in recent years. The model checker Open-Kronos has been used for the timing analysis of combinational circuits [24]. An abstract model of the given circuit is developed by partitioning the circuit into smaller sub-circuits and reachability graphs are used to model timed automata. A major limitation of this approach is that it uses fixed delay values for all the gates, e.g., the delay of an inverter is assumed to be 0, and thus the technology parameters and process variations are completely ignored in the models. Similarly, Open-Kronos is also used for timing analysis with bi-bounded delay values in [8]. Formal timing verification of digital circuits, including their combinational and sequential components, is performed in [5]. The given circuit is modeled at the macroscopic level where a state transition graph (STG) is modeled as a configuration of inputs while excluding the multiple input transitions. The delays of the components are extracted from SPICE simulations. The timing behavior of the SPSMALL memory architecture is verified using a parametric timed automata based model [12]. This technique derives a set of linear constraints that ensure the correctness of the response times of the memory. Similarly, symbolic timing verification of concurrent systems is proposed in [13]. The complex polyhedra modeling approach is used as the abstraction to represent sets of timed states as a timed transition system. Each event in this model has a symbolic delay defined in an interval $[d_i, D_i]$ where d_i and D_i symbolize minimum and maximum delay values, respectively.

Formal timing analysis of digital circuits has also been done with various other motivations. For example, digital circuits have been formally modeled using propagational delays, which are assumed to take values in an interval $\delta[\tau_{min}, \tau_{max}]$ in the context of testing of circuits in [28]. The model is developed in Uppaal, where delay faults are intentionally inserted into the circuit to gener-

ate counterexamples. These counterexamples are then used for testing of circuits. Similarly, formal timing analysis of combinational circuits has been performed to detect the Hardware Trojans using side channel parameters, like delay and power, in [1]. The main idea in this work is to insert an intrusion in the circuit in the form of logic gates. After intrusion, formal timing verification is performed to generate a counterexample. In this technique, only combinational circuits are formally verified and no sequential circuit is analyzed. Moreover, various circuit paths are identified manually for delay calculations in a circuit in this work [1]. Based on the above-mentioned discussion, we identify the following limitations in the existing literature:

- **Incompleteness:** The existing timing analysis approaches do not consider all the gates and all their possible input transitions.
- **Limited Scope:** The delay models used in formal techniques are usually not based on real delay values.
- **Inviability:** The exhaustive analysis techniques have enormous verification times, which leads to their infeasibility for large circuits.

To overcome the deficiencies of the existing formal timing analysis approaches for digital circuits and thus making the formal timing analysis more accurate, we propose to use the Elmore delay model [30] to compute the delays of both combinational and sequential components of the given circuit. Moreover, instead of using bi-bounded delays, we propose to calculate the value of the delay at every possible input transition of every gate in the design. For example, in the case of a 2 input gate, delays are calculated for all the four possible transitions $\Gamma_{Delay} = [d_{00}, d_{01}, d_{10}, d_{11}]$. Moreover, instead of manually searching of timing paths within a circuit as is the case for all existing formal timing analysis approaches, we propose to use the Quartus Prime Pro software [21] for automatically extracting the paths of the given circuit. This choice allows us to not only automate the timing analysis flow but also reduces the risks of ignoring some timing paths in the design. While using the above-mentioned information, we develop a formal model of the given circuit in the Uppaal model checker and then verify its desired timing properties in Uppaal. To facilitate the modeling and verification process, we provide a generic framework in which by knowing delays of the basic circuit blocks, i.e., NAND, NOR, NOT and a Flip-Flop, we can verify the timing behavior of any digital circuit, such as the clock period of a circuit, the critical paths as well as setup and hold time constraints in a circuit. It is important to note that by using a model checking tool for the timing analysis, our results are based on a rigorous exploration of the state space of the circuit model and thus all the paths and input values are implicitly considered in the analysis.

The remainder of the paper is structured as follows: We present a brief introduction about the Uppaal model checker in Section 2. The proposed methodology is explained in detail in Section 3, followed by Section 4 where we describe case studies and verification results. Finally, Section 5 concludes the paper.

2 Uppaal Model Checker

Uppaal [6] is a free academic model checker for the formal verification of real-time systems. It is based on the timed automata theory [3] and its modeling language offers many additional features, such as bounded integer variables.

2.1 Timed Automata

A timed automaton (TA) is a tuple $TA = (S, s_o, T, \sigma, Y, \beta)$, where:

- S is a set of locations.
- $s_o \in S$ is an initial location.
- T is a set of clocks.
- σ is a set of all defined actions.
- $Y \subseteq S \times \sigma \times B(T) \times 2^T \times S$ is a set of edges between locations.
- $\beta : S \to B(T)$ assigns invariants to locations.

$B(T)$ is the set of conjunctions over simple conditions, i.e., $x - y \bowtie c$ or $x \bowtie c$, where $c \in \mathbb{N}$, $x, y \in T$ and $\bowtie \in \{=, \geq, \leq, >, <\}$. A clock valuation is a function $u : T \to \mathbb{R}_{\geq 0}$ from the set of clocks to the non-negative real values. Thus, writing $u \in \beta(s)$ means that u satisfies $\beta(s)$. Timed automata are finite state automata having states and transitions, enriched with built-in clocks, which evolve at a uniform rate and can be reset to their initial value.

A state is a pair $(S, \alpha$ where α is a valuation of clocks and variables in that particular state. A state (S, α) has a discrete transition t, and system moves to the next state (S', α') if the constraints on t, called guards, are satisfied. The interconnection between two timed automata can be obtained by using synchronization channels. The signal is emitted by one automaton in transition t and received by one or more automata.

2.2 Queries

Verification of a model using the required specifications is a crucial step in modechecking. Similar to a model, properties must be expressed in a formal language. Uppaal uses a simplified version of TCTL (timed computational tree logic) properties. Various path formulae supported by Uppaal are:

- $\exists\Diamond\rho$ (Possibly): There exists a path along which query ρ eventually holds.
- $\exists\Box\rho$ (Potentially always): There exists a path where query ρ always satisfies.
- $\forall\Box\rho$ (Invariantly): For all paths, query ρ always satisfies .
- $\forall\Diamond\rho$ (Eventually): For all paths, query ρ eventually satisfies .
- $\rho \rightsquigarrow \xi$ (Leads-to): Whenever ρ satisfies, query ξ holds always eventually.

3 Proposed Methodology

In this section, we explain the proposed methodology, depicted in Figure 1, for the formal timing analysis of a digital circuit. Our methodology is comprised of three major steps: delay calculation, path extraction and modeling and verification in the Uppaal model checker.

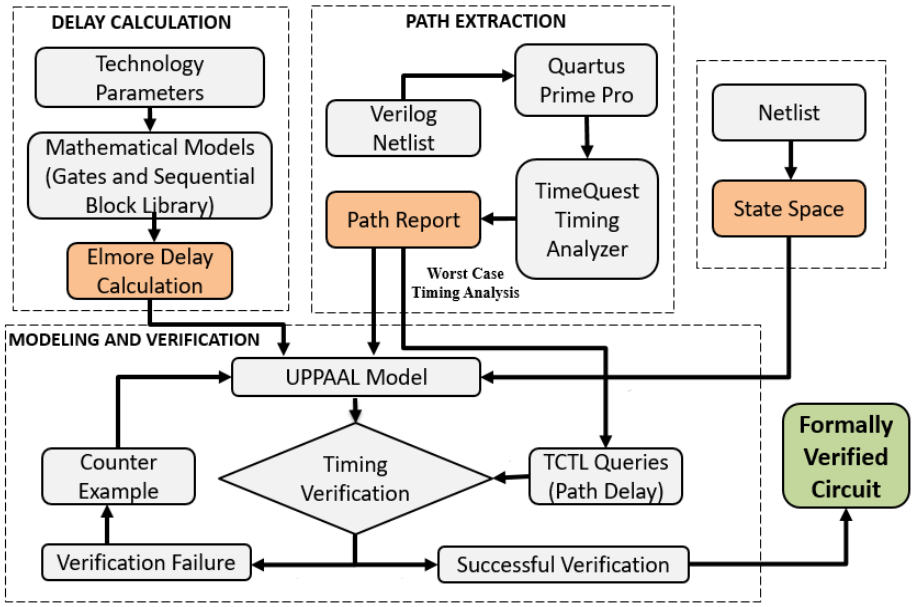


Fig. 1. Proposed Methodology

3.1 Delay Calculation

The individual gate delays are estimated in the proposed methodology based on individual transitions at the gate inputs using the Elmore delay model [1], which computes the delay by representing each circuit in the form of an RC tree. Many timing analysis tools estimate the delay of the component based on the time difference between 50% of the input transition to the 50% of the output transition. However, the Elmore delay model considers the Resistor Capacitor relationships to compute the delay and hence provides a better estimate of the delay compared to the above-mentioned traditional approach. The delay is estimated by the model from a source node to one of the leaf nodes by accumulating the capacitances C_i on each node of the path, multiplied by the effective resistance R_{is} on the shared path from source node to the leaf node.

$$T_e = \sum_i C_i \times R_{is} \tag{1}$$

$$\tau_{delay} = T_e \times ln(2) \tag{2}$$

Using the basic technology parameters, we calculate the capacitance and resistance values for *PMOS* and *NMOS* transistors in an ON state. We propose to develop timing models for the basic circuit components, i.e., NAND, NOR, NOT and a Flip-Flop. These gates are then further used to model complex circuits. Gate capacitances for *PMOS* and *NMOS* [22] are given below:

$$C_{gatenMOS} = C_{gminN} \times fan - out \times WR_{nMOS} \tag{3}$$

$$C_{gatepMOS} = C_{gminP} \times fan - out \times WR_{pMOS} \tag{4}$$

Where C_{gmin} represents the minimum gate capacitance and WR represents the width ratio. C_L is the load capacitance calculated from the addition of gate capacitances of all the gates connected at the output of the considered component.

$$C_L = \sum_{k=1}^{a} C_{gatenMOSk} + \sum_{j=1}^{b} C_{gatepMOSj} \tag{5}$$

Diffusion capacitance C_{Diff} can be calculated from the drain capacitance [30]. The addition of load and diffusion capacitance leads to the total capacitance of a gate C_T, which is used for the calculation of delay.

$$C_T = C_L + C_{Diff} \tag{6}$$

Resistance of a *PMOS* or *NMOS* [23] can be calculated as follows:

$$R_{on} = \frac{1}{WL \times \mu \times Cox \times (V_{GS} - V_{TH})} \tag{7}$$

Using the values of corresponding resistances and capacitances, we can find out the Elmore delays for NAND, NOT, and NOR gates. Delay is calculated by considering all the possible input transitions of a gate. For example, the Elmore delay equations for the NAND gate are shown in Table 1.

Table 1. Nand Gate Delay Equations

Input Transition	Output	Delay Equation
00	1	$ln(2) \times [(C_T \times R_p)/(2 \times WR_{pMOS})]$
01	1	$ln(2) \times [(C_T \times R_p)/WR_{pMOS}]$
10	1	$ln(2) \times [((C_T + C_{ST}) \times R_p)/WR_{pMOS}]$
11	0	$ln(2) \times [(C_T \times 2 \times R_n)/WR_{nMOS}]$

We have used the True Single-Phase Clocked (TSPC) Flip-Flop model [22] to capture the timing behavior of the Flip-Flop as this provides less complexity and less number of transistors to deal with [22]. Setup time, hold time, and clock to Q delay are the three most important timing constraints in a Flip-Flop. In the TSPC Flip-Flop model, the setup time is assigned a delay of one inverter, the hold time is considered to be less than one inverter delay and the propagational delay is considered to be equal to three inverter delays. Similarly, in our model, we consider the hold time to be equal to one inverter delay in the worst case. The delay equations used in our model for setup time, hold time and the clock to Q in a Flip-Flop are given in Table 2.

3.2 Path Extraction

Calculation of a delay in a circuit, which is composed of several gates and Flip-Flops, is done based on its various paths, i.e., from input to a Flip-Flop, between

Table 2. Flip-Flop Delay Equations

Data Input	Output	Delay Equation
Setup Time		
0	0	$ln(2) \times [(C_T \times R_p)/WR_{pMOS})]$
1	1	$ln(2) \times [(C_T \times R_n)/WR_{nMOS})]$
Hold Time		
0	0	$ln(2) \times [(C_T \times R_p)/WR_{pMOS})]$
1	1	$ln(2) \times [(C_T \times R_n)/WR_{nMOS})]$
Clk2Q Delay		
0	0	$ln(2) \times [(3 \times C_T \times R_p)/WR_{pMOS})]$
1	1	$ln(2) \times [(3 \times C_T \times R_n)/WR_{nMOS})]$

Flip-Flops and from a Flip-Flop to an output. The delay of a path is calculated by adding delays of logic elements present in that path. In the case of smaller circuits, we can manually analyze all the paths in a circuit and can calculate the delays of all the paths. But in case of large circuits, it is impossible to analyze the paths manually, therefore we propose to use a software that can provide all the valid paths in a circuit automatically from a given circuit netlist. We found Altera Quartus Prime Pro [21] to be the most relevant tool for this purpose. It not only provides all the possible paths from all input ports to all output ports but can also provide paths from the input port to a Flip-Flop, Flip-Flop to a Flip-Flop, or Flip-Flop to an output port.

In the path extraction phase of the proposed methodology, we have to provide the Verilog code of the circuit that needs to be analyzed. This Verilog file is first analyzed and synthesized. After compilation, we run the TimeQuest Timing Analyzer tool to get the information about the paths in the given circuit. Synopsys design constraint file and a timing netlist is thus created automatically by the Timing Analyzer. After this, we can analyze the paths that are reported by the TimeQuest Timing Analyzer.

3.3 Modeling and Verification in Uppaal Model Checker

Modeling and verification in Uppaal is the most important step in our approach for the timing verification of circuits. Firstly, the given netlist is translated to its corresponding state transition diagram. This state transition diagram along with the delay values of logic elements and path information from TimeQuest Timing Analyzer is used for this purpose in the Uppaal Model Checker. The TCTL properties of path delays have to be given to the Uppaal model checker as well. The state space model is then verified in Uppaal against the identified TCTL properties to judge the circuit performance. We mainly check that the delay in a circuit is less than the required maximum delay. If the delay of the circuit exceeds the maximum delay, then the Uppaal model checker returns a counterexample which provides us the exact trace that caused the timing violation. Thereafter,

it can be investigated if the issues are due to a modeling error or its an actual timing violation.

In order to facilitate the modeling of digital circuits, we developed the formal models of the basic gates, i.e., NAND, NOR, NOT and a Flip-Flop, in Uppaal and these models can be built upon to formalize models of larger complex circuits. For example, The TA of the NOT gate is shown in Figure 2 where *xin_not* is the input and *xout_not* is the output. At the initial state, the selection expression *xin_not: int[0, 1]* allows to assign a boolean value 1 or 0 to the input *xin_not*. The fan-out *fo_not* is updated depending upon how many gates are connected at the output port of the gate. Based on the value of the input, internal resistances, internal capacitances, fan-out, and various technology parameters, the delay *delay_not* is calculated using the Elmore delay equation. The output gets its appropriate value, i.e., the negation of input *out_not:= !(xin_not)*, after the delay has elapsed. Similarly, the models of other basic gates have also been developed and they can be used to formalize any combinational gate-level circuit. Sequential circuits also contain Flip-Flops besides the basic logic gates

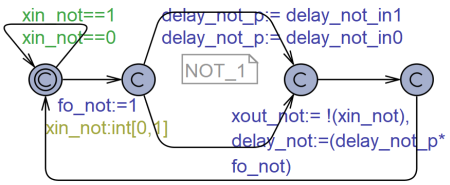
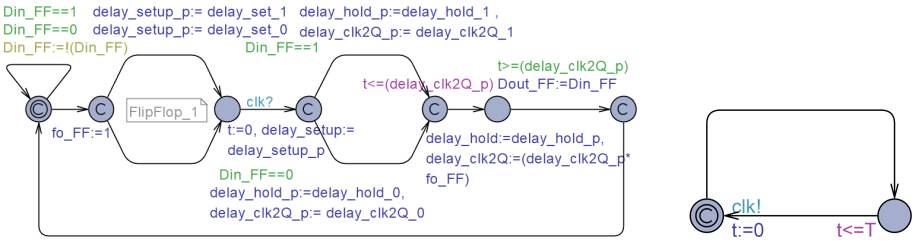


Fig. 2. Timed Automaton of the NOT Gate



(a) Flip-Flop Model (b) Clock Model

Fig. 3. Timed Automata of a Flip-Flop and a Clock

and to formalize their behavior, we also formalized the Flip-Flop. The proposed Flip-Flop model along with the clock is shown in Figure 3. The input signal is updated in the first state. Based on the value of the input, internal resistances, internal capacitances, fan-out, and various technology parameters, the setup time, hold time and clock to Q delay is calculated using the Elmore delay equations.

In the proposed methodology, we can develop the models of more complex circuits by interconnecting the basic gates and Flip-Flop models. Some simple circuits designed from the basic gates, i.e., NAND, NOT, and NOR, are shown in Figure 4.

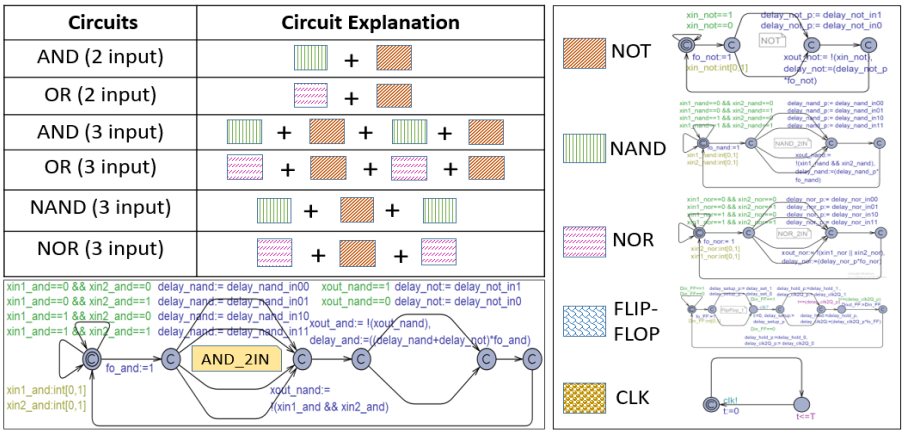


Fig. 4. Designing Some Simple Circuits Using Basic Gates

We propose to verify the following properties.

– Firstly, we check the deadlock property, which ensures that the timed automaton is not stuck at any particular state and thus moves ahead through all the states.

$$\forall \, \Box \;\; not \;\; deadlock$$

– For verifying combinational circuits, we check that the delay, considering all the paths delay in the given combinational model, does not exceed the maximum delay value for the given circuit. If the delay exceeds the maximum value and the property fails then we get a counterexample.

$$\forall \, \Box \;\; !((delay_{gate1} + delay_{gate2} + \cdots\cdots + delay_{gaten}) > D\max_{comb}))$$

Where $D\max_{comb} = \max(delay_{gate1}, delay_{gate2}, \cdots\cdots, delay_{gaten})$ represents the maximum delay in the considered path.

– For verifying sequential circuits, we check the input port to Flip-Flop and Flip-Flop to output port paths just like we check the timing properties of combinational circuits. Moreover, we also need to conduct the Flip-Flop to Flip-Flop path analysis while considering the setup and hold time constraints, which allows us to determine the clock period of the given circuit and avoid metastability. For example, consider a typical sequential circuit scenario, shown in Figure 5, where we have an input port IN, two Flip-Flops $FF1$ and $FF2$ and an output port OUT. There are i gates between input

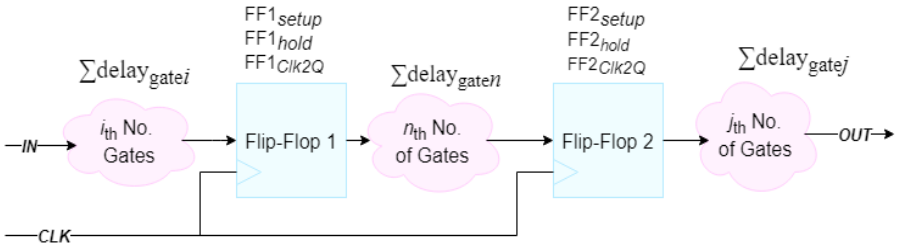


Fig. 5. A Typical Sequential Circuit

and *FF1*, n gates between Flip-Flops, and j gates between *FF2* and output. We propose to verify the following properties in this case.

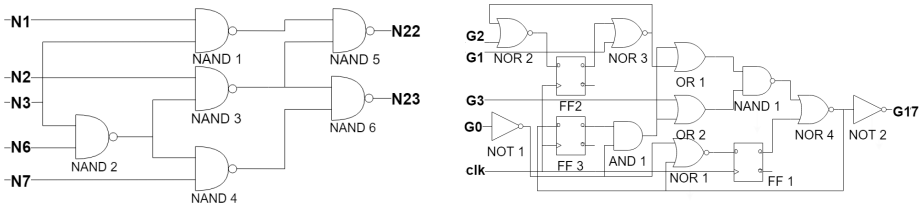
$$\forall \square ((delay_{gate1} + delay_{gate2} + \cdots + delay_{gatei}) \leq D\max_{INtoFF})$$

$$\forall \square (T \geq (FF1_{clk2Q} + delay_{gate1} + delay_{gate2} + \cdots + delay_{gaten}) + FF2_{setup})$$

$$\forall \square ((FF1_{clk2Q} + delay_{gate1} + delay_{gate2} + \cdots + delay_{gaten}) \geq FF2_{hold})$$

$$\forall \square ((FF2_{clk2Q} + delay_{gate1} + delay_{gate2} + \cdots + delay_{gatej}) \leq D\max_{FFtoOUT})$$

4 Case Studies



(a) ISCAS-85 C17 Benchmark Circuit (b) ISCAS-89 S27 Benchmark Circuit

Fig. 6. Benchmark Circuits

For illustration purpose, we present the analysis of C17, and S27 benchmark circuits in this section. Due to the large size of transition diagrams, we only summarize path information and verified properties of these circuits in this section.

4.1 C17 Benchmark

C17, shown in Figure 6(a), is one of the benchmarks from ISCAS-85 that consists of 5 input ports and 2 output ports. The path report of the C17 circuit generated from TimeQuest Timing Analyzer is modeled in Uppaal with the help of a single function in order to reduce the complexity in a state transition diagram.

```
c17_p1 = (delay1_nand_c17+delay5_nand_c17); //N1 --> N22
c17_p2 = (delay3_nand_c17+delay5_nand_c17); //N2 --> N22
c17_p3 = (delay3_nand_c17+delay6_nand_c17); //N2 -->N23
c17_p4 = (delay2_nand_c17+delay3_nand_c17+delay6_nand_c17); //N3 --> N23
c17_p5 = (delay2_nand_c17+delay3_nand_c17+delay5_nand_c17); //N3 --> N22
c17_p6 = (delay2_nand_c17+delay4_nand_c17+delay6_nand_c17); //N6 --> N23
c17_p7 = (delay2_nand_c17+delay3_nand_c17+delay5_nand_c17); //N6 --> N22
c17_p8 = (delay4_nand_c17+delay6_nand_c17); //N7 --> N23
c17_p9 = (delay1_nand_c17+delay5_nand_c17); //N3 --> N22
c17_p10 = (delay2_nand_c17+delay4_nand_c17+delay6_nand_c17); //N3 --> N23
c17_p11 = (delay2_nand_c17+delay3_nand_c17+delay6_nand_c17); //N6 --> N23
```

Some of the properties that are verified against specified paths are shown below, where $Tmax$ represents the maximum value of delay for a particular path. These properties are checked against all the paths and all the states in the model.

- $\forall \Box\ (!(c17_{p1} > Tmax_{c17-p1}))$
- $\forall \Box\ (!(c17_{p2} > Tmax_{c17-p2}))$

We also verified many other properties for this benchmark and the details can be found in [2].

4.2 S27 Benchmark

S27, shown in Figure 6(b), is one of the sequential circuit benchmarks from ISCAS-89 that consists of 4 input ports and 1 output port. All the timing reports of S27, including the paths from input to Flip-Flops, between Flip-Flops, and from Flip-Flops to output, generated from TimeQuest Timing Analyzer, were modeled in Uppaal with the help of three functions, one function for each type, in order to reduce the complexity in the resulting state transition diagram.

```
delay_p1_in = (delay2_or+delay1_nand+delay4_nor); // G3-->FF3
delay_p2_in = (delay3_nor+delay1_or+delay1_nand+delay4_nor); // G1-->FF3
delay_p3_in = (delay1_not+delay2_or+delay1_and+delay1_nand+delay4_nor); // G0-->FF3
delay_p4_in = (delay1_not+delay1_or+delay1_and+delay1_nand+delay4_nor); // G0-->FF3
delay_p5_in = (delay2_nor); // G2-->FF2
delay_p6_in = (delay3_nor+delay2_nor); // G1-->FF2
delay_p7_in = (delay1_not+delay1_nor);  // G0-->FF1
delay_p8_in = (delay1_not+delay1_or+delay1_and+delay1_nand+delay4_nor+
        delay1_nor); // G0-->FF1
delay_p9_in = (delay1_not+delay2_or+delay1_and+delay1_nand+delay4_nor+
        delay1_nor); // G0-->FF1

delay_p1_ff = (delay1_clk2Q+delay4_nor+delay3_setup); // FF1-->FF3
delay_p2_ff = (delay2_clk2Q+delay3_nor+delay1_or+delay1_nand+delay4_nor+
        delay3_setup); // FF2-->FF3
delay_p3_ff = (delay3_clk2Q+delay1_and+delay2_or+delay1_nand+delay4_nor+
        delay3_setup); // FF3-->FF3
delay_p4_ff = (delay3_clk2Q+delay1_and+delay1_or+delay1_nand+delay4_nor+
        delay3_setup); // FF3-->FF3
delay_p5_ff = (delay2_clk2Q+delay3_nor+delay2_nor+delay2_setup);  // FF2-->FF2

delay_p1_h1 = (delay1_clk2Q+delay4_nor); // FF1-->FF3
delay_p2_h2 = (delay2_clk2Q+delay3_nor+delay1_or+delay1_nand+delay4_nor);// FF2-->FF3
delay_p4_h3 = (delay3_clk2Q+delay1_and+delay2_or+delay1_nand+delay4_nor);// FF3-->FF3
delay_p5_h4 = (delay3_clk2Q+delay1_and+delay1_or+delay1_nand+delay4_nor);// FF3-->FF3
delay_p6_h5 = (delay2_clk2Q+delay3_nor+delay2_nor);  // FF2-->FF2

delay_p1_out = (delay2_clk2Q+delay3_nor+delay1_or+delay1_nand+delay4_nor+
        delay2_not); // FF2-->G17
delay_p2_out = (delay1_clk2Q+delay4_nor+delay1_or+delay2_not); // FF1-->G17
delay_p3_out = (delay3_clk2Q+delay1_and+delay2_or+delay1_nand+delay4_nor+
        delay2_not); // FF3-->G17
delay_p4_out = (delay3_clk2Q+delay1_and+delay1_or+delay1_nand+delay4_nor+
        delay2_not); // FF3-->G17
```

Some of the properties which are verified against each specified path are written below. We also verified many other properties for this benchmark and the details can be found in [2]. In these properties, the variable, $Tmax$, represents the maximum delay time and T_{clk}, represents the time period of a clock.

- $\forall \square (delay_{p1-in} \leq Tmax_{p1-in})$
- $\forall \square (T_{clk} \geq (delay_{p1-ff}))$
- $\forall \square (delay_{p-h1} \geq (delay_{3-hold}))$
- $\forall \square (delay_{p1-out} \leq (Tmax_{p1-out}))$
- $\forall \square (delay_{p2-in} \leq Tmax_{p2-in})$
- $\forall \square (T_{clk} \geq (delay_{p2-ff}))$
- $\forall \square (delay_{p-h2} \geq (delay_{3-hold}))$
- $\forall \square (delay_{p2-out} \leq (Tmax_{p2-out}))$

4.3 Verification Results

The considered combinational circuits and their verification statistics are summarized in Table 3 using the information about the total number of gates in the given circuit, its verification time and the memory utilization during the verification phase of corresponding circuit. Modeling and verification details of

Table 3. Result of Combinational Circuits

Circuits	Number of Gates			Verification	
	NAND	NOR	NOT	Time (s)	Memory (MB)
C17 [11]	6	-	-	0.014	7.34
C17 [29]	7	-	-	0.021	7.35
C17 [19]	9	1	2	0.033	11.78
Full Adder [20]	11	-	-	0.032	8.82
Full Adder [17]	10	3	1	0.074	16.06
Full Adder [1]	14	3	1	0.91	18.06
4-bit RCA [17]	40	12	4	63.31	2684

sequential circuits are summarized in Table 4 using the total number of gates and Flip-Flops in the given circuit, its verification time, and the memory utilization. We noticed that the number of explored states during the verification

Table 4. Result of Sequential Circuits

Circuits	Number of Gates			Number of Flip-Flops	Verification	
	NAND	NOR	NOT		Time (s)	Memory (MB)
Flip-Flop [22]	-	-	-	1	0.019	7.85
16-bit SIPO Shift Register [18]	-	-	-	16	0.031	10.38
64-bit SISO Shift Register [18]	-	-	-	64	0.047	16.80
64-bit Ring Counter [26]	-	-	-	64	0.090	21.19
64-bit Johnson Counter [26]	-	-	1	64	0.100	27.29
S27 [10]	2	6	5	3	2.46	43.96
S208 [10]	39	37	90	8	316	8820
S386 [10]	151	36	228	6	3306	29745

significantly increases with an increase in the number of inputs of basic gates, such as NOT, NAND, NOR, AND and OR. For example, the total number of

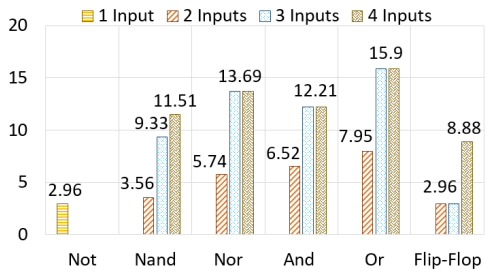
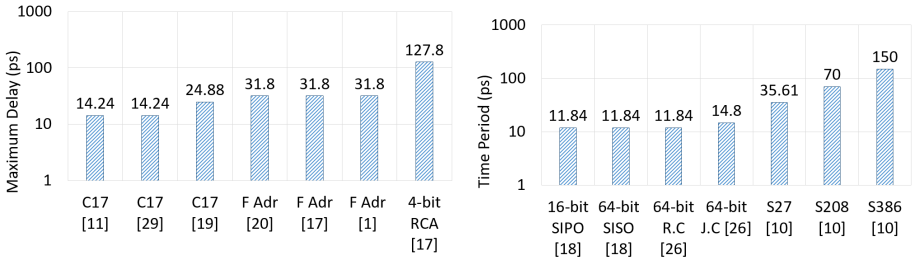


Fig. 7. Maximum Delays of Basic Gates

explored states in the 4-bit RCA is 16316416 whereas 105735463 states were explored while analyzing the S368 circuit.

We calculated the maximum delays of basic gates, such as NOT, NAND, NOR, AND and OR as shown in Figure 7. In case of NOT, AND and OR, the maximum delay for 3 input and 4 input is same since the type and number of logic elements are same in a path. The maximum delays in case of the considered combinational circuits is shown in Figure 8(a), whereas the maximum time periods of the clock in case of sequential circuits is shown Figure 8(b).



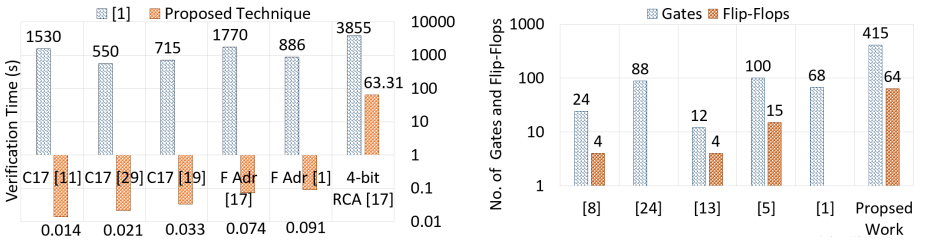
(a) Maximum Delay in Combinational Circuits

(b) Maximum Time Period in Sequential Circuits

Fig. 8. Timing Analysis Results

In comparison with an existing technique, presented in [1], which uses the nuXmv model checker for verifying combinational circuits, we find our results to be acquired in a much faster manner as shown in Figure 9(a). This result is based on the maximum time utilized by the model checker for the property verification. For example, the verification time in [1] and the proposed technique in case of the C17 circuit is 1530 s and 0.014 s, respectively. In [1], real numbers are used for modeling and delay calculations, which causes an enormous increase in the state space. We propose to overcome these limitations by performing major real number calculations manually and then use the final delay values in an integral form in the model checker in order to minimize the state explosion problem.

In comparison with the existing techniques, we also verify circuits with larger number of gates and Flip-Flops, i.e., upto 415 gates and 64 Flip-Flops, as shown in Figure 9(b).



(a) Comparison of Verification Time of [1] and the Proposed Technique

(b) Number of Logic Elements Analyzed by Existing and Proposed Techniques

Fig. 9. Comparison with Existing Techniques

A summary of comparison of the proposed approach with some existing techniques is shown in Table 5. The comparative analysis is mainly based on seven parameters. The first two parameters show the type of a circuit, which is analyzed, i.e, combinational or sequential circuit. Automatic path extraction, depicts whether the existing techniques perform path analysis automatically or not. Next two parameters refer to delay modeling techniques and the model checker used for the formal verification. Finally, the last two parameters show the maximum gates and Flip-Flops verified by the corresponding technique. Our technique is found to be better than existing techniques in the following ways:

- Unlike some existing techniques [24], [1], we perform timing verification of the combinational as well as sequential circuits.
- In order to perform more realistic modeling and verification, we proposed to use the Elmore delay modeling technique [1] instead of assumed delay model as used in [8], [13], [24], [28].
- We proposed to extract the path information automatically using Quartus Prime Pro [21].
- We verify circuits with comparatively larger number of gates and Flip-Flops compared to all the existing formal timing analysis works.

5 Conclusions

This paper presented a model checking based approach for the formal timing analysis of digital circuits. The main idea behind this approach is to use timed automata as a state transitions diagram for formal modeling of the digital circuits and TCTL queries for the formal verification of their timing properties using the Uppaal model checker. We have developed a generic framework to facilitate the

Table 5. Comparison with Existing Techniques

Related Work	Comb cct	Seq cct	Auto. Path	Delay Model	Tool	Max Gates	Max FF
Bozga et al. [8]	✓	✓	x	Assumed delay	Open-Kronos	24	4
Salah et al. [24]	✓	x	x	Assumed delay	Open-Kronos	88	x
Clariso et al. [13]	✓	✓	x	Symbolic delay	Abstract Algorithm	12	4
Bara et al. [5]	✓	✓	x	Spice delay	Kronos/ Uppaal	100	15
Abbasi et al. [1]	✓	x	x	Elmore delay	nuXmv	68	x
Proposed work	✓	✓	✓	Elmore delay	Uppaal	415	64

formal timing analysis by developing the models of the basic components of a digital circuit, i.e., logic gates and Flip-Flops, that can be built upon for the formal modeling of more complex circuits. Moreover, the proposed approach supports the automatic path extraction using Quartus Prime Pro along with the modeling and verification in Uppaal. The proposed approach can be used to formally verify various timing characteristics, such as finding the clock period of a circuit, finding the critical path and the setup and hold time constraints in a circuit. For illustration purposes, we used the proposed approach to conduct the formal timing analysis of a number of real-world digital circuits, such as Adders, Shift Registers, C17, S27, S208, and S386 circuits. In the future, we plan to incorporate routing delays and clock skew in a circuit so that we have a more accurate and realistic timing model.

References

1. Abbasi, I.H., Lodhi, F.K., Kamboh, A.M., Hasan, O.: Formal verification of gate-level multiple side channel parameters to detect hardware trojans. In: Formal Techniques for Safety-Critical Systems. pp. 75–92. CCIS, vol. 694, Springer (2016)
2. ul Ain, Q.: Formal Timing Analysis of Digital Circuits. Available online: http://save.seecs.nust.edu.pk/projects/ftadc/ (2018)
3. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Logic in Computer Science. pp. 414–425. IEEE (1990)
4. Andraus, Z.S., Sakallah, K.A.: Automatic abstraction and verification of verilog models. In: Proceedings. 41st Design Automation Conference, 2004. pp. 218–223 (2004)
5. Bara, A., Bazargan-Sabet, P., Chevallier, R., Ledu, D., Encrenaz, E., Renault, P.: Formal verification of timed VHDL programs. In: Forum on Specification Design Languages. pp. 80–85. IET (2010)
6. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal 4.0 (2006)
7. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and software verification: model-checking techniques and tools. Springer Science & Business Media (2013)
8. Bozga, M., Jianmin, H., Maler, O., Yovine, S.: Verification of asynchronous circuits using timed automata. Electronic Notes in Theoretical Computer Science **65**(6), 47–59 (2002)
9. Braibant, T.: Coquet: a coq library for verifying hardware. In: International Conference on Certified Programs and Proofs. pp. 330–345. Springer (2011)

10. Brglez, F., Bryan, D., Kozminski, K.: Notes on the ISCAS'89 benchmark circuits. North-Carolina State University (1989)
11. Bryan, D.: The ISCAS'85 benchmark circuits and netlist format. North Carolina State University **25** (1985)
12. Chevallier, R., Encrenaz-Tiphene, E., Fribourg, L., Xu, W.: Timed verification of the generic architecture of a memory circuit using parametric timed automata. Formal Methods in System Design **34**(1), 59–81 (2009)
13. Clarisó, R., Cortadella, J.: Verification of timed circuits with symbolic delays. In: Asia and South Pacific Design Automation Conference. pp. 628–633. IEEE (2004)
14. Hasan, O., Tahar, S.: Formal verification methods. In: Encyclopedia of Information Science and Technology, Third Edition, pp. 7162–7170. IGI Global (2015)
15. Irfan, A., Cimatti, A., Griggio, A., Roveri, M., Sebastiani, R.: Verilog2SMV: A tool for word-level verification. In: Design Automation Test in Europe Conference Exhibition. pp. 1156–1159 (2016)
16. Kilts, S.: Static timing analysis. Advanced FPGA Design: Architecture, Implementation, and Optimization pp. 269–278 (2007)
17. Mano, M.M., Kime, C.R.: Logic and computer design fundamentals, vol. 3. Prentice Hall (2008)
18. Maxfield, C.: Bebop to the Boolean boogie: an unconventional guide to electronics. Newnes (2008)
19. Mukhopadhyay, D., Chakraborty, R.S.: Hardware security: Design, threats, and safeguards. Chapman and Hall/CRC (2014)
20. Patterson, D.A., Hennessy, J.L.: Computer organization and design. zadnje izdanje (1994)
21. Quartus prime standard edition handbook (2015)
22. Rabaey, J.M., Chandrakasan, A.P., Nikolic, B.: Digital integrated circuits, vol. 2. Prentice hall Englewood Cliffs (2002)
23. Razavi, B.: Design of Analog CMOS Integrated Circuits. Tata McGraw-Hill Education (2002)
24. Salah, R.B., Bozga, M., Maler, O.: On timing analysis of combinational circuits. In: International Conference on Formal Modeling and Analysis of Timed Systems. pp. 204–218. LNCS, vol. 2791, Springer (2003)
25. Saleh, R., Jou, S.J., Newton, A.R.: Gate-level simulation. In: Mixed-Mode Simulation and Analog Multilevel Simulation, pp. 123–152. Springer (1994)
26. Shift Registers and Counters. Available online: https://computing.ece.vt.edu/LiaB/Microelectronic%20Systems/Lectures/Digital%20Logic/pdf/Shift%20registers.pdf (2014)
27. Shiraz, S., Hasan, O.: A library for combinational circuit verification using the hol theorem prover. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(2), 512–516 (2018)
28. Takan, S., Guler, B., Ayav, T.: Model checker-based delay fault testing of sequential circuits. In: Architecture of Computing Systems. pp. 1–7. VDE (2015)
29. Wei, S., Meguerdichian, S., Potkonjak, M.: Malicious circuitry detection using thermal conditioning. IEEE Transactions on Information Forensics and Security **6**(3), 1136–1145 (2011)
30. Weste, N.H., Harris, D.: CMOS VLSI design: A circuits and systems perspective. Pearson Education India (2015)