Multiple Clock Domain Synchronization in a QBF-based Verification Environment

Djordje Maksimovic[1], Bao Le[1], Andreas Veneris[1,2]
[1]Department of Electrical and Computer Engineering, University of Toronto
[2]Department of Computer Science, University of Toronto
{djordje, bao, veneris}@eecg.toronto.edu

***Abstract*—Modern designs are growing in size and complexity, becoming increasingly harder to verify. Today, they are architected to include multiple clock domains as a measure to reduce power consumption. Verifying them proves to be a computationally intensive and challenging task as it requires their clocks to be synchronized. To achieve synchronization, existing Boolean satisfiability-based methodologies add hardware to combine the clock domains before transforming them into their iterative logic array representation (ILA). As a consequence, this results in the addition of redundant time-frames adding overhead during verification. This paper introduces a novel framework to verify designs with multiple clocks using Quantified Boolean Formula satisfiability (QBF). We first present a formulation that models an ILA representation with symbolic universal quantification to achieve synchronization. This is later extended with the use of a clock divider to overcome inefficiencies. The net effect is the reduction in the number of redundant time-frames. Furthermore, the usage of QBF results in significant memory savings when compared to traditional methods. Experiments on bounded model checking demonstrate memory reductions of 76% on average with competitive run-time performance.**

I. Introduction

Verification has become a major bottleneck in the digital circuit design process [1]. To reduce power consumption, modern designs are architected with multiple clock domains, a practice that has introduced new challenges during verification [2]. To overcome this challenge, recent research has proposed a clock domain unification scheme that generates a single clock domain [3]. This technique allows one to verify the design without having to worry explicitly about the synchronization of different domains. While the unification technique overcomes the synchronization issue it can introduce redundant time-frames. In essence, these redundant frames are copies of the transition relation where no state change occurs. As a result, additional time and memory are devoted to verify these unnecessary time-frames.

In more detail, Bounded Model Checking [4] (BMC) verifies a multiple clock domain design by synchronizing the clocks [3]. It achieves synchronization by converting state elements into functionally equivalent components clocked by a global clock. Conceptually, the global clock acts as a data probe where its positive edge indicates a state transition of some clock domain in the design. Following this, an iterative logic array (ILA) is generated [5], an action also knows as "circuit unrolling". However, in the presence of multiple clocks, a clock may have to wait for computation from another clock. This wait time will manifest itself as a redundant time-frame(s), where no state changes occur. With large circuits, the memory capacity becomes a concern because ILA memory size scales with the number of time-frames. Over many cycles the memory requirements may become substantial [5]. This realization prompts into developing memory-efficient methodologies.

In this paper, we present a novel verification methodology for multiple clock domain designs that tackles key inherent challenges of the problem. In an introductory formulation we explicitly model the ILA representation with symbolic universal quantification. We employ this to compute the transition relation of each clock only when there is clock activity. Furthermore, unlike the ILA construction that requires multiple time-frames, the proposed methodology synchronizes all clocks using a single copy of the transition relation. Later, we extend the original formulation with the use of a divider circuitry that saves on time and memory when compared to the original one. An additional contribution of the paper is that it presents these two new verification schemes in the context of Quantified Boolean Formula (QBF) satisfiability. By using QBF, it avoids the memory intensive ILA representation.

Experiments on BMC demonstrate the efficiency of the proposed QBF-based multiple clock verification framework. Specifically, when compared to traditional SAT approaches, the method achieves memory reduction on average by 76% and in many cases, it is able to complete verification where SAT fails to do so. With all these benefits aside, the computation time still remains comparable to SAT. Evidently, advances in QBF solvers, a fast growing field, will further harness the benefits of the proposed methodology.

The remainder of this paper is outlined as follows. Section II contains background material including prior art in multiple clock domain verification. Section III describes the basic clock domain synchronization approach while Section IV extends it with the use of a divider to synchronize the different clocks. Section V presents the experiments and Section VI concludes this work.

II. Background

A. Quantified Boolean Formula Satisfiability and Notation

Given a propositional logic formula Φ with sets of mutually disjoint quantified Boolean variables $V_1, V_2, \ldots, V_n$, the QBF problem asks whether Φ can be satisfied through a proper assignment to the variables [5]. A satisfying assignment (SAT) to Φ yields $\Phi = 1$, while an unsatisfying one (UNSAT) yields $\Phi = 0$. We allow a variable quantifier to take from two values, existential ($\exists$) and universal ($\forall$). As such, a QBF instance in *prenex normal form* is written as follows:

$$Q_1 V_1 Q_2 V_2 \ldots Q_n V_n \mid \Phi \tag{1}$$

where $Q_i \in \{ \exists, \forall \}$ is a scope, $Q_i \neq Q_{i+1}$, and Φ is a logic formula in conjunctive normal form (CNF). In this notation, the leftmost (rightmost) Q_i is called the outer (inner) scope, and variables are assigned values moving from the outer scope to the inner one.

The following notation is used throughout this paper. Assume a design with n clock domains, where c_i, $1 \leq i \leq n$, corresponds to the i-th distinct clock domain. For each domain c_i, we use $s_{c_i}^0, \ldots, s_{c_i}^{m_{c_i}}$ to represent state element values at cycle $0, \ldots, m_{c_i}$ respectively, where m_{c_i} is the maximum cycle that clock c_i can reach. We assume every state element is always clocked by the same clock domain. Predicate $B(s_{c_i}^k)$ is the set of bad states which should not be reachable in k cycles of c_i. We assume all clock domains have a fixed frequency, they all start at the positive edge of cycle 0, and state changes occur at the positive edge of the clock. Similar to the work in [3], we also generate a global clock $gclk$ with a frequency that is the lowest common multiple of all the original n clock domains. For example, if there exist two clock domains of 2 and 3 MHz each, the global clock is at 6 MHz. We denote as t_g^i the i-th cycle of this clock, $i = 0, 1, 2, \ldots, m_g$ where m_g is the maximum cycle $gclk$ can reach. Intuitively, the positive edge of t_g^i indicates a state transition in at least one of the original clock domains. Finally, $t_{c_i}^j$ is the cycle number of the global clock $gclk$ when clock domain c_i is at the positive edge of its cycle j.

B. Prior Work

In [6], the authors present a methodology for modelling the relations between clock domains, such that designs can be verified under any formal technique. This is done by translating clock constraints into state machines representing these relations. In our methodology, we encode the relationship between clock domains using clock constraints specified by the engineer. The work of [7] presents a method to abstract phase relations between clock domains. It assumes that clocks are generated within the netlist of the design and the first task is to find the clocks by matching nets with repeating bit patterns. They employ heuristics to decide on the number of phases they wish the design to have, and use this information to reduce the number of state variables. Our methodology assumes that clocks are generated from the same source and phase differences are negligible.

Of more interest to our work is the methodology in [3] that unifies clock domains with the use of a single global clock. The authors use embedded flip-flops clocked by a global clock, which oscillates at a frequency where the positive edge indicates a state change in some clock domain. A positive edge detector is used to detect an edge of the original clock. When this occurs the flip-flops take the new value, otherwise the previous value is held at the output. By replacing all the state elements with this circuitry, a single clock domain design is generated that retains the functionality of the original design.

After unifying the domains, the ILA representation is generated. This involves duplicating the combinational part of a sequential circuit into k time-frames, such that the next-state variables of the current time-frame are connected to the current-state variables of the next time-frame. Fig. 1(a) displays a sequential logic circuit and Fig. 1(b) displays its ILA representation, where a, b, c (a^i, b^i, c^i) are input (at time-frame i), and x, y, z (x^i, y^i, z^i) are output (at time-frame i). Due to the process of unifying the clock domains and the generation of a global clock, redundant time-frames (*i.e.,* copies of the transition relation) where no state changes occur can be generated in the ILA. Redundant time-frames pose bottlenecks in traditional SAT instances of multiple clock domain BMC as they require extra time and memory to compute. Two types of redundant time-frames exist: *extra frames*, and *waiting frames*.

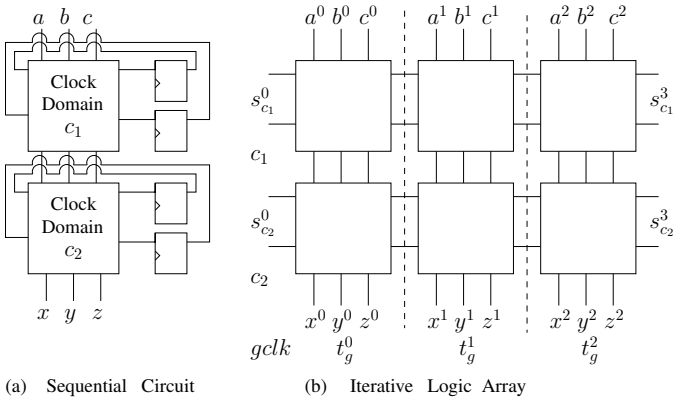
Extra frames are instantiated by the global clock due to state changes in other clock domains. On a positive edge of the global clock, there is no way to distinguish whether an edge occurs in two or more clock domains. All that it indicates is that at least one clock domain is transitioning states. In order to ensure correct functionality of the design, a new frame must be created for every clock domain. For clock domains that are not transitioning states, this results in duplicate computations of their next state assignment. For example, if clock domain c_1 is two times faster than c_2, then in the ILA representation of Fig. 1(b) the transition relation of c_2 during t_g^1 is an extra frame as it corresponds to the second half of c_2's period. The second clock domain would not be on a positive edge until `gclk` reached t_g^2.

On the other hand, waiting frames arise when one clock domain, is waiting for an acknowledge from another clock domain. Assuming inputs are held constant, suppose that in Fig. 1(b), domain c_1 is waiting for data from c_2, this means that during t_g^0 and t_g^1, c_1 may have had no state changes, because it is waiting for an acknowledge from c_2 that would appear on t_g^2. Consequently, any assignment to the transition relation during t_g^0 is redundant because it is identical to the assignment during t_g^1. Waiting frames are not generated by clock unification, but may appear in the original multiple domain design.

III. QBF-based Clock Synchronization

In this section we present a novel clock synchronization formulation that eliminates redundant time-frames. This is possible by tailoring the solution around a QBF implementation. For the sake of simplicity, we build the presentation for a two clock domain design. This can be easily generalized for a variable number of clock domains.

The framework is depicted in Fig. 2. Inside the dashed boxes, a robust QBF-based ILA encoding is presented [5]. Combinational logic clocked by clock i is represented by the transition relation $T_{c_i}(s_{c_i}, s'_{c_i}, input_{c_i}, output_{c_i}, CDC_{c_i,c_j})$, where $s_{c_i} \in \{s^0_{c_i}, s^1_{c_i}, s^2_{c_i}, \ldots, s^{m_{c_i}-1}_{c_i}\}$ ($s'_{c_i} \in \{s^1_{c_i}, s^2_{c_i}, \ldots, s^{m_{c_i}}_{c_i}\}$) is the current



(a) Sequential Circuit (b) Iterative Logic Array

Fig. 1. Sequential design and its Iterative Logic Array representation

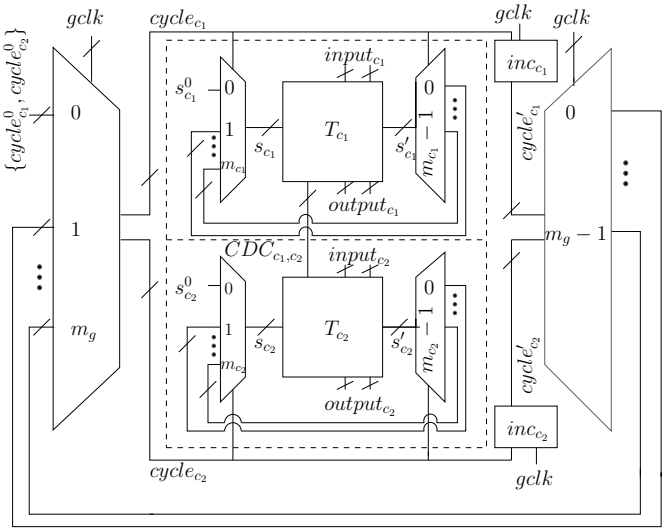


Fig. 2. Circuit diagram of two domain clock synchronization

(next) state assignment, CDC_{c_i,c_j} is a set of clock domain crossing signals between c_i and c_j, and $input_{c_i}$ ($output_{c_i}$) is the set of primary input (output). The two multiplexers around T_{c_i} are used to connect the next state assignment of the current cycle to the current state assignment of the next cycle through the use of the select line $cycle_{c_i}$. In this sense, the value of the select line determines the state of a clock domain at a particular cycle and allows one to model the sequential behavior of the design [5].

In order for the design in the dashed boxes to exhibit correct functionality, signals $cycle_{c_1}$ and $cycle_{c_2}$ should be assigned to cycles that synchronize the two clock domains. Our synchronization logic is denoted by the inc_{c_i} hardware module. The purpose of this module is to transition a clock domain to the next cycle when the clock is on a positive edge. If c_i is not on a positive edge then it remains in its current cycle. Its architecture is found in Fig. 3. The signal $edge^j_{c_i}$, for all $0 \leq j \leq m_g$, in Fig. 3 defines how the clock domain changes cycles, and is the main factor in achieving synchronization. It is a binary signal that must be constrained to `one` on a positive edge of the clock domain, otherwise it is `zero`. The purpose of $gclk$ is to order the positive edges of each clock domain temporally, according to the specification.

Returning to the description of Fig. 2, the two outer multiplexers essentially encode the current to next cycle assignment for inc_{c_i}. In this case $cycle_{c_i} \in \{cycle^0_{c_i}, \ldots, cycle^{m_g-1}_{c_i}\}$ is the current cycle assignment whereas $cycle'_{c_i} \in \{cycle^1_{c_i}, \ldots, cycle^{m_g}_{c_i}\}$ represents the next cycle assignment. These multiplexers are used to model the sequential

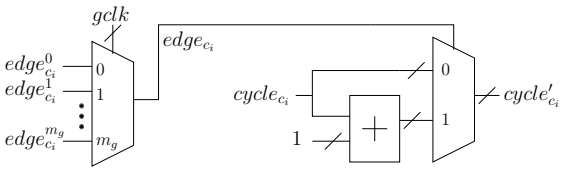


Fig. 3. Circuit diagram of the inc_{c_i} module

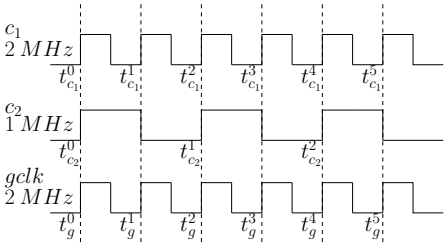


Fig. 4. Timing diagram of a multiple clock domain system with the lowest common multiple global clock `gclk`

behavior of the inc_{c_i} module. Notice that when $cycle'_{c_i} = cycle_{c_i}$, this indicates a cycle of the global clock where clock domain c_i is not transitioning states. The combination of the outer multiplexers and the inc_{c_i} module is hereafter called the *synchronizer*.

Clock synchronization ensures that state changes in each clock domain are ordered temporally according to the design specification. In more detail, it ensures that each clock domain is synchronized in terms of valid cycles with respect to the global clock, where a valid cycle is defined as follows:

Definition 1 *Cycle i for clock domain c_l is called valid w.r.t. cycle j of the global clock gclk iff $t^i_{c_l} \leq t^j_g < t^{i+1}_{c_l}$, where $0 \leq i \leq m_{c_l}$, $0 \leq j \leq m_g$, and $1 \leq l \leq n$ where n is the number of clock domains and m_{c_l} (m_g) is the maximum cycle c_l (gclk) can reach.*

Essentially, to achieve synchronization we have to prevent instances where one clock domain is processing a previous cycle and another is processing a future cycle w.r.t. the current cycle of the global clock. We also want to prevent clocks from exercising a different frequency to the one defined by the specification.

Example 1 *Suppose that in Fig. 4, the global clock is on t^0_g. This means that $t^0_{c_1}$ and $t^0_{c_2}$ are valid cycles because their edge occurs with t^0_g. Likewise, if the global clock is on t^1_g then $t^1_{c_1}$ and $t^0_{c_2}$ are valid, but $t^0_{c_1}$ is not, as c_1 has temporally completed that cycle.*

In the following, we show that the synchronizer hardware does indeed synchronize the clock domains of the design.

Theorem 1 *The synchronizer generates only valid cycles.*

Proof: To prove the theorem we apply induction on the number of cycles for the global clock. For the basis, let the global clock be on t^0_g. Then $t^0_{c_l} \leq t^0_g < t^1_{c_l}$, for any $1 \leq l \leq n$, holds since $t^0_g = t^0_{c_l}$ and the global clock has the smallest period so $t^0_g < t^1_{c_l}$. For the hypothesis, assume that given some j, cycle i of clock domain l is valid *i.e.,* $t^i_{c_l} \leq t^j_g < t^{i+1}_{c_l}$. In the inductive step, when the global clock increments to $j+1$, if c_l is not on a positive edge then i is not incremented by the adder, and therefore we have to show that the following inequality holds:

$$t^i_{c_l} \leq t^{j+1}_g < t^{i+1}_{c_l} \tag{2}$$

This corresponds to the case where c_l has not completed its cycle, thus $t^i_{c_l} \leq t^{j+1}_g$ holds since $t^i_{c_l} \leq t^j_g < t^{j+1}_g$. The inequality $t^{j+1}_g < t^{i+1}_{c_l}$ holds due to the following property of the global clock's frequency. The period of any clock can be represented by an integer multiple of the global clock's period. In the hypothesis, the smallest difference between t^j_g and $t^{i+1}_{c_l}$ is one period of the global clock. Since a positive edge did not occur, the difference must have been greater than one period.

On the other hand, if c_l is on a positive edge then i is incremented by the adder. Thus, the following inequality must hold:

$$t^{i+1}_{c_l} \leq t^{j+1}_g < t^{i+2}_{c_l} \tag{3}$$

This corresponds to the case where c_l has completed its cycle, and increments with the global clock. The inequality $t^{i+1}_{c_l} \leq t^{j+1}_g$ holds because $t^{i+1}_{c_l} = t^{j+1}_g$, and $t^{j+1}_g < t^{i+2}_{c_l}$ holds since $t^{i+2}_{c_l}$ is one period greater than $t^{i+1}_{c_l}$ and therefore greater than t^{j+1}_g. Since both inequalities hold it means that the synchronizer produces valid cycles. ■

Assume there exists a function $\lambda(x)$ that takes a bit vector as input and returns its integer representation (e.g. $\lambda(\langle 1,0,0\rangle) = 4$). Let $s_{c_i} = s^{\lambda(x)}_{c_i}$ represent a multiplexer encoding [5] where the output s_{c_i} is the $\lambda(x)$-th input of the multiplexer. This can be formally written as:

$$(\bigwedge_{j=0}^{m_{c_i}-1} ((\lambda(x) = j) \implies (s_{c_i} = s^j_{c_i}))) \equiv (s_{c_i} = s^{\lambda(x)}_{c_i}) \tag{4}$$

As such, the diagram in Fig. 2 can be represented by a three-scope QBF-based BMC instance formally described as follows:

$$\begin{aligned}
&\exists cycle^S_{c_1}, \ldots, cycle^S_{c_n}, s^S_{c_1}, \ldots, s^S_{c_n}, edge^S_{c_1}, \ldots, edge^S_{c_n}\ \forall gclk \\
&\exists s_C, s'_C, cycle_C, cycle'_C, edge_C, input_C, output_C, CDC_C \mid \\
&\bigwedge_{i=1}^{n} (I(s^0_{c_i}) \wedge T_{c_i}(s_{c_i}, s'_{c_i}, input_{c_i}, output_{c_i}, CDC_{c_i}) \\
&\wedge (s_{c_i} = s^{\lambda(cycle_{c_i})}_{c_i}) \wedge (s'_{c_i} = s^{\lambda(cycle_{c_i})+1}_{c_i}) \wedge I(cycle^0_{c_i}) \\
&\wedge (cycle_{c_i} = cycle^{\lambda(gclk)}_{c_i}) \wedge (cycle'_{c_i} = cycle^{\lambda(gclk)+1}_{c_i}) \\
&\wedge (\bigwedge_{j=0}^{m_g} I(edge^j_{c_i})) \wedge (edge_{c_i} = edge^{\lambda(gclk)}_{c_i}) \\
&\wedge (edge_{c_i} \implies (\lambda(cycle'_{c_i}) = \lambda(cycle_{c_i}) + 1)) \\
&\wedge (\neg edge_{c_i} \implies (\lambda(cycle'_{c_i}) = \lambda(cycle_{c_i}))) \\
&\wedge B(s^k_{c_l}))
\end{aligned} \tag{5}$$

where $CDC_{c_i} = \bigcup_{j=1}^{n} CDC_{c_i,c_j}$, for all $0 \leq i \leq n$. In both existential scopes notation s_C (s'_C) is used to denote variables $s_{c_1} \ldots s_{c_n}$ ($s'_{c_1} \ldots s'_{c_n}$), and $cycle_C$ ($cycle'_C$) denotes variables $cycle_{c_1} \ldots cycle_{c_n}$ ($cycle'_{c_1} \ldots cycle'_{c_n}$). This definition similarly applies to $edge_C$, $input_C$, $output_C$, and CDC_C. Furthermore, $gclk$ is the global clock, s_{c_i} is the current state, s'_{c_i} is the next state, $cycle_{c_i}$ is the current cycle, $cycle'_{c_i}$ is the next cycle, $I(cycle^0_{c_i})$, $I(edge^j_{c_i})$ and $I(s^0_{c_i})$ are the predicates recognizing valid initial constraints, and $edge_{c_i} = edge^{\lambda(gclk)}_{c_i}$ represents a multiplexer encoding similar to Equation 4. Also, $s^S_{c_i} = \{s^0_{c_i}, \ldots, s^{m_{c_i}}_{c_i}\}$. Furthermore, $cycle^S_{c_i} = \{cycle^0_{c_i}, \ldots, cycle^{m_g}_{c_i}\}$, and a similar argument applies for $edge^S_{c_i}$. Finally, $B(s^k_{c_l})$ is a bounded state at cycle k for some $1 \leq l \leq n$, which is determined to be reachable or not in the BMC process.

When compared to the approach in [3], the proposed QBF-based formulation has several benefits. First, it is able to remove redundant time-frames by ensuring that the next state assignment for every cycle is only computed once. Furthermore, as experiments later in the paper demonstrate, it is memory-efficient by orders of magnitude when compared to the one using an ILA approach. Due to the replacement of the ILA with universal quantification, designs with exceptionally long traces can be solved using our formulation whereas it may not be possible to generate an ILA due to the excessive memory required [5].

A. Optimizing Performance

In some instances, a positive edge of the global clock will not imply a state transition for some clock because its positive edge has yet to occur.

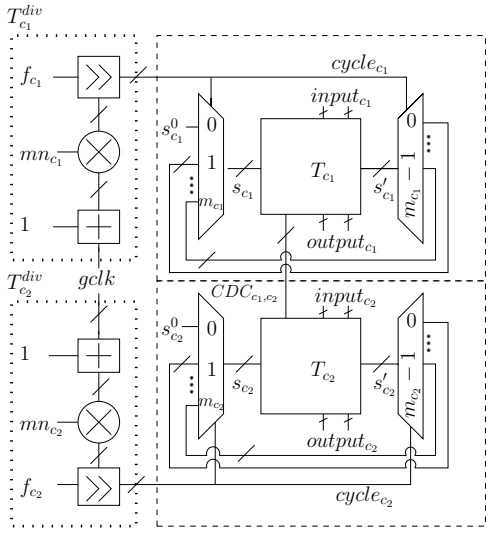


Fig. 5. Circuit diagram of two domain divider-based clock synchronization

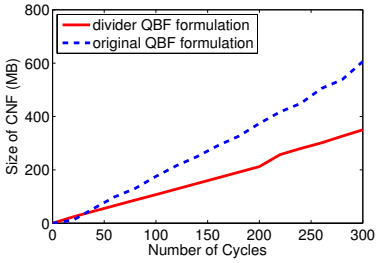


Fig. 6. Ratio of CNF size to number of cycles

In this case we have that $cycle'_{c_i} = cycle_{c_i}$ *i.e.,* a state change doesn't occur. In this context, T_{c_i} does not have to be computed because we are only interested in computing the transition relation when the positive edge of the clock occurs.

We can optimize performance in a QBF framework by adding a literal, $disable_{c_i}$, to every clause in T_{c_i} which represents a path from a driver flip-flop to an input flip-flop clocked by the same clock. Then for every cycle of the global clock, we add an implication that sets the literal to `true` (*i.e.,* turns off the transition relation), or sets the literal to `false` (*i.e.,* allows the transition relation to be solved) depending on whether $edge^j_{c_i}$ is encoding a positive edge of c_i. This results in the following equation:

$$\bigwedge_{i=1}^{n}\bigwedge_{j=0}^{m_g}((\lambda(gclk)=j) \implies (edge^j_{c_i} = \neg disable_{c_i})) \tag{6}$$

For clock domain crossing (CDC) paths we do not have to add a literal as we assume that multiple clock domain designs are implemented using asynchronous FIFOs or multi-flop synchronizers [8]. These synchronizers contain minimal or no combinational logic, which would not substantially impact the solving time. If we were to include these paths, the variables corresponding to them would have to be moved to the outer existential quantifier which has been shown to increase solving time [9].

B. Handling Latches

The formulation presented in the previous section can handle latches provided that inputs to latches are driven by flip-flops and/or clocks. This assumption is necessary because when a latch is enabled the data signal will behave as a clock *i.e.,* the latch will take the value of the data signal instantly. In the context of BMC, this would result in a clock signal (the data input to the latch) which can have an unbounded switching frequency. Between the period of one clock, the latch could have flipped values any number of times, which can theoretically require an infinite number of cycles to solve.

By applying the assumption, latched logic can be represented as a QBF-based ILA encoding that is clocked using a least common multiple frequency of the latch clock and the clock of the driver flip-flop. This will ensure that all state changes are encoded. If the data input is treated as a clock, then we would need a predefined input trace which specifies a periodic or non-periodic clock signal.

IV. Divider-based Clock Synchronization

A potential overhead in the formulation presented in Section III is that when encoding a large number of cycles, the multiplexers may become large which can potentially present a challenge to a QBF solver. Here we extend the proposed formulation by replacing the multiplexer-based clock synchronizer with a divider-based synchronizer. The net benefit is a smaller and easier to solve CNF formula.

In the previous formulation, two outer multiplexers are used in the synchronization of the clock domains. Intuitively, this causes an overhead because as the cycle bound for BMC increases, the number of paths in a multiplexer doubles with each new $gclk$ bit. Furthermore, once a path in a multiplexer is chosen, all other paths become redundant. This may add unnecessary work as a QBF solver must assign the variables on the unused paths. This observation motivates for a modification using a divider-based synchronizer, as follows.

The basic idea of the divider-based synchronizer is to compute $cycle_{c_i}$ by dividing the cycle of the global clock by some integer factor. In more detail, on every cycle of the global clock, one or more clock domains are changing states. Since all the clocks are periodic, a clock domain c_i changes states every time the global clock increments by some number of cycles, where this number is the factor mentioned above. For example in Fig. 4, every time the global clock increments by two cycles, c_2 increments by one cycle. Thus if we divide the current cycle of the global clock by this factor (here, the factor is two) the result is the current cycle of c_2. Note that each clock domain may have a different factor which will be a fixed value during a BMC run.

In the dashed boxes in Fig. 5, the hardware behind a QBF-based encoding for an ILA is illustrated. The divider portion is presented in the dotted box. For each clock domain, a divider output is attached to the select lines of the multiplexers, and the global clock is set as the numerator. The factor for each clock domain is used as the denominator. In the next subsection the divider circuitry will be described in detail.

By applying Theorem 1, the presented divider approach generates valid cycles. It also eliminates redundant time-frames because it only needs to compute the next state assignment for each clock cycle once.

Formally, the divider-based synchronizer in Fig. 5, can be formulated as a QBF instance by the following formula:

$$\begin{aligned}
&\exists s^S_{c_1},\ldots,s^S_{c_n}, mn_C, f_C\ \forall gclk\\
&\exists s_C, s'_C, cycle_C, input_C, output_C, CDC_C\ |\\
&\bigwedge_{i=1}^{n}(I(s^0_{c_i}) \wedge T_{c_i}(s_{c_i}, s'_{c_i}, input_{c_i}, output_{c_i}, CDC_{c_i})\\
&\wedge (s_{c_i} = s^{\lambda(cycle_{c_i})}_{c_i}) \wedge (s'_{c_i} = s^{\lambda(cycle_{c_i})+1}_{c_i})\\
&\wedge T^{div}_{c_i}(gclk, 1, mn_{c_i}, f_{c_i}, cycle_{c_i}) \wedge I(mn_{c_i}) \wedge I(f_{c_i})\\
&\wedge B(s^k_{c_l}))
\end{aligned} \tag{7}$$

where $T^{div}_{c_i}(gclk, 1, mn_{c_i}, f_{c_i}, cycle_{c_i})$ is the combinational circuitry for the dividers in the dotted boxes, $I(mn_{c_i})$ and $I(f_{c_i})$ are the predicates recognizing valid initial constraints. Also, mn_C (f_C) is used to denote variables $mn_{c_1}\ldots mn_{c_n}$ ($f_{c_1}\ldots f_{c_n}$). All other notation remains as described in Section III.

Fig. 6 shows a plot of memory vs. the number of cycles of the global clock. From this figure we see that the size of the CNF of the formulation in Section III increases sharply, while the divider-based formulation increases at a much slower rate. The reason for this is for every new bit of `gclk` the size of the outer multiplexers in the original formulation doubles. This is in contrast to the divider-based

TABLE I
MULTIPLE CLOCK DOMAIN BMC RESULTS

Design Info						uni-clk (SAT)		multi-clk (QBF)			divider-based (QBF)			optimization (QBF)		
Design	clks	Comb. Logic	State Elem.	# Latch	k	CNF (MB)	time (s)	CNF (MB)	time (s)	imprv (x)	CNF (MB)	time (s)	imprv (x)	CNF (MB)	time (s)	imprv (x)
rsdecoder1	3	13543	526	5	32	74.0	70.1	20.2	85.3	3.66x	13.4	80.4	5.52x	15.2	76.1	4.86x
rsdecoder2	3	13543	526	5	42	81.1	80.4	22.2	141.4	3.68x	16.4	121	4.94x	20.2	107.4	4.01x
ethernet1	5	70139	10558	12	26	486	250	127	490	3.83x	150	461	3.24x	162	430	3.00x
ethernet2	5	70139	10558	12	25	468	231	123	467	3.80x	157	420	2.98x	169	401	2.77x
ac97_ctrl1	16	17591	2482	137	68	279	368	80.2	1256	3.48x	71.2	1058	3.92x	83.5	1027	3.34x
ac97_ctrl2	16	17591	2482	137	129	539	∞	256	2900	2.10x	135	2821	3.99x	142	2412	3.79x
ac97_ctrl3	16	17591	2482	137	9	32.0	8.32	10.2	16.8	3.13x	14.4	16.5	2.22x	27.2	15.1	1.17x
vga1	3	89402	17110	8	9	237	57.0	60.1	158	3.94x	69.0	142	3.43x	71.6	132	3.31x
vga2	3	89402	17110	8	4	92.1	30.0	38.8	48.1	2.37x	46.0	45.6	2.00x	50.1	35.6	1.83x
vga3	3	89402	17110	8	100	3657	∞	1010	1320	3.62x	840	1212	4.35x	850	986	4.30x
mem_ctrl1	10	48006	1239	94	4	29.0	11.5	8.10	14.6	3.58x	11.3	12.5	2.57x	20.8	11.5	1.39x
mem_ctrl2	10	48006	1239	94	150	1226	∞	120	1012	10.2x	99.6	959	12.3x	138	845	8.88x
hpdmc1	34	9858	438	4	8	13.2	4.83	2.80	17.3	4.71x	3.40	29.4	3.88x	3.74	26.1	3.53x
inhouse-uart1	2	3912	340	0	10	5.90	1.18	1.50	1.84	3.93x	2.00	1.43	2.95x	2.12	4.18	2.68x
inhouse-uart2	2	3912	340	0	6	2.10	0.84	0.555	2.02	3.78x	0.902	1.91	2.32x	0.91	3.10	2.29x
inhouse-uart3	2	3912	340	0	20	12.3	2.47	2.67	20.2	4.61x	3.10	14.8	3.96x	3.45	16.6	3.56x
AVERAGE						452	92.5	118	496	4.03x	102	462	4.04x	109.9	408	3.45x

formulation, where increasing the divider input width by one does not double the size of the divider. Furthermore, if the number of cycles of the global clock is not a power of 2, some paths will be left dangling. Satisfying these dangling paths requires extra computational effort. For example, if the number of cycles to be computed is 100, the select line will be 7 bits long giving a total of 128 cycles, 28 of which are redundant and will be left dangling. This CNF size increase may cause a slow down in a solver when tackling a design with large multiplexers.

A. An Efficient Divider Implementation

We now describe an efficient implementation of the divider circuitry shown in Fig. 5. General purpose dividers can be large when translated into CNF. As such, the divider is implemented using unsigned integer division with a constant number similar to the approach presented in [10]. By using unsigned integer division we can implement the divider using a multiplier, an incrementer, and a shifter, which have simpler underlying circuitry. The idea behind this type of division is the fact that any number n can be divided by 2^f, where f is some integer, by using a shifter. This allows us to express division as such:

$$\lfloor \frac{n}{d} \rfloor = \lfloor mn \ * \ \frac{n}{2^f} \rfloor \tag{8}$$

where d is the denominator, and $mn = \frac{2^f}{d}$ is a *magic number* [10]. The magic number is a number which when multiplied by n then shifted right by f bits, produces the result of $\frac{n}{d}$. Integer operations are easier to compute than floating point operations. As a result, when computing the magic number it is rounded down to an integer. This produces a dividing error which can be counteracted by incrementing n by 1. Finally, we must choose a value for f, such that the error in the flooring operation in $\lfloor \frac{2^f}{d} \rfloor$ is eliminated, and such that our magic number has the same width N, as the width of the global clock. For this we choose $f = N + \lfloor log_2 d \rfloor$. The magic number and f are precomputed as the denominator does not change. When the synchronizer is generated the implemented divider is:

$$cycle_{c_i} \ = \ \lfloor (mn_{c_i} \ * \ (gclk + 1)) \ >> \ f_{c_i} \rfloor \tag{9}$$

where the constant mn_{c_i} (f_{c_i}) is the mn (f) for c_i, and $gclk$ is universally quantified.

V. EXPERIMENTAL RESULTS

This section presents experimental results for the proposed formulations. Experiments are run on a core i5-3570K workbench clocked at 3.40GHz, with a time limit of 7200 seconds. A limit of 4 GB is set. The backend QBF solving engine that we employ is RAREQS [9], and the SAT solving engine that we compare to is the latest version of MINISAT 2.2.0 [11].

We perform BMC on six designs from OpenCores [12], and one in-house real-life industrial design. Each design has a varying number of clocks that ranges from 2 to 34. Both reachable and unreachable properties are tested. Table I displays the results for these seven designs. Under Column 1, the alphabetic prefix represents the name of the design, and the numerical suffix represents the property tested. Columns 2-5 display the number of clocks, combinational logic gates, state elements, and latches, respectively. Finally, Column 6 denotes the number of cycles that are required to reach $B(s^k_{c_i})$.

Four BMC formulations are profiled and compared to each other. The original clock unification method [3] is encoded in SAT and its results are shown in the columns titled `uni-clk (SAT)` (Columns 7 and 8). The proposed QBF-based clock synchronization and the extended divider-based formulations are outlined in the remaining columns titled `multi-clk (QBF)` and `divider-based (QBF)`, respectively. In the final columns, the divider-based approach is augmented with the optimization from Section III-A and evaluated. For each experiment, the size of the CNF and the solving time are reported. The peak memory usage of each solver is analyzed later in this section. Finally, Columns 11, 14, and 17 display the memory savings for the proposed formulations when compared to the `uni-clk (SAT)` one.

From these numbers it can be seen that when compared to `uni-clk (SAT)`, `multi-clk (QBF)` manages to have a memory savings average of 74% while keeping a comparable run-time, where `uni-clk (SAT)` runs for 32% of the run-time for `multi-clk (QBF)`. The average slow down is computed individually, excluding the cases where the SAT-based approach runs out of memory (shown with ∞ in the table), and then averaging the results. Evidently, these memory savings are due to the elimination of the ILA and only using a single instance of the design (*i.e.,* transition relationship) in the QBF-based approach. At low bounds for k (range of $1 \ldots 50$, depending on the design), `multi-clk (QBF)` has the best memory savings, as the CNF of a $1 \ldots 6$ bit multiplexer is small. However, as the bound k increases to greater than 6 bits the memory savings drop due to the generation of larger multiplexers. For these values of k, the run-time slows down due to the time required to assign the variables for unused paths in the outer multiplexers.

The divider-based formulation improves upon the original Section III formulation at higher values of k. The more moderate memory savings at low bounds is due to the multiplexers in the original modeling being smaller than the dividers for low values of k. Both formulations perform similarly in run-time at low values of k. We also observe that `multi-clk (QBF)` saves about 50% more memory at low bounds when compared to [3], however at higher bounds the divider-based formulation saves 20%-90%. It is worth noting that in BMC, the trace length k is not known until the instance is solved. Thus the divider-based formulation is better due to its performance at higher bounds.

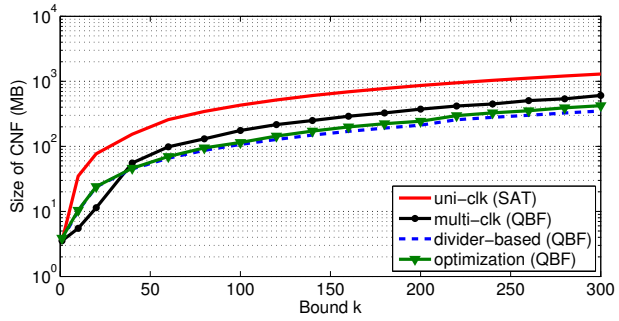


Fig. 7. Ratio of CNF size to bound k for design `ac97_ctrl` (in log scale)

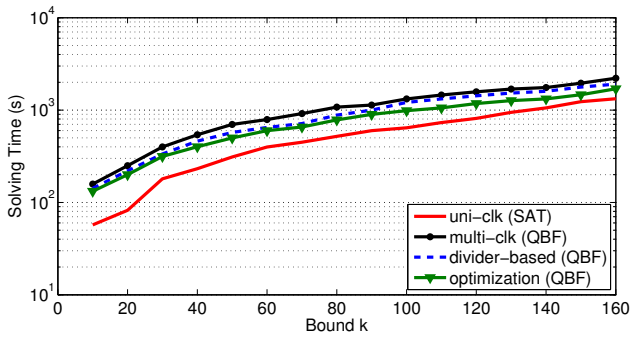


Fig. 8. Ratio of solving time to bound k for design `vga` (in log scale)

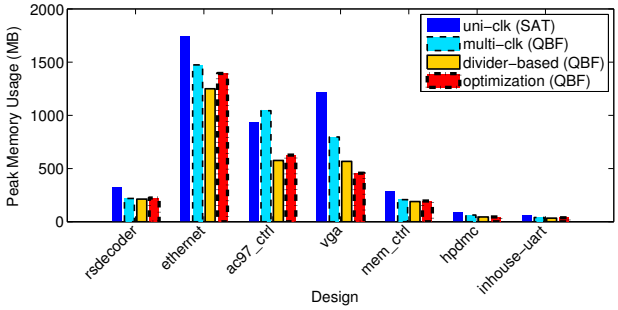


Fig. 9. Peak memory usage when running each solver

With the optimization presented in Section III-A, the divider-based formulation becomes the most efficient of all proposed methods. There is a slight increase in the CNF size due to the added clauses, but a run-time improvement is observed due to eliminating the need to recompute the transition relations for cycles that have previously been evaluated. The performance benefits are less pronounced for circuits with similar frequencies because there are less redundant time-frames. Thus, transition relations are computed on a more frequent basis, and the overhead introduced by the added clauses exceeds the performance benefits. This can be seen with the `inhouse-uart` design, which has a clock that is half the frequency of the other.

In terms of run-time, we observe that the SAT-based approach remains the fastest. Nevertheless, it is important to note that while the SAT-based approach takes 38% of the time that the divider-based formulation executes (if we exclude the mem-out cases), the divider-based formulation comes with a memory savings average of 76%, an attractive feat. Further, this QBF advantage is also demonstrated by entries in Table I which are infinity. We observe that in approximately 20% of the cases SAT runs out of memory when attempting to perform BMC, whereas the proposed QBF-based formulation does not. As work in QBF solvers remains a fertile research area, improvements in those engines will also benefit the proposed formulation too.

Fig. 7 illustrates the memory size of the CNF (in logarithmic scale) for SAT-based BMC versus the QBF-based one with respect to the number of clock cycles in the problem instance for circuit `ac97_ctrl`. As it can be seen from the plot, the ILA explodes in size when the number of cycles exceeds 100. On the other hand, the QBF-based instances do not increase as rapidly because the cycles are encoded by the universal time quantifier. We observe that the original formulation fairs comparably with the divider-based formulation, saving more memory at lower bounds due to the use of a small multiplexer. However, as the number of cycles increases, the size of the outer multiplexers also increases and its memory usage deteriorates.

Fig. 8 displays the solving time for the design `vga`, where the property was constrained at intervals of ten cycles. Results are again plotted in logarithmic scale while the graph is fitted to give the expected result at every cycle. The memory limit is relaxed in order to solve the instances. Due to the size doubling effect of the outer multiplexers when new clock cycle bits are added, the `multi-clk` formulation requires more computation time to process the multiplexers. The `divider-based` formulation does not grow as much because for every extra bit of the global clock, only a single bit adder and multiplier need to be added. This results in a smaller CNF when compared to the multi-clk approach, producing a run-time performance gain that is comparable to the one by the SAT-based formulation, as the number of cycles increases.

Finally, Fig. 9 presents the peak memory usage when solving the problem. The designs are constrained for the first property in Table I. Designs that are computed for a small number of cycles show little variation in memory usage, while designs with more cycles show a greater variation. This variation is primarily due to the increasing difference in CNF size between the QBF and SAT-based formulations as the bound k increases.

VI. Conclusion

Modern designs are architected to use multiple clock domains, a fact that introduces overhead during verification. This paper introduces a novel QBF-based synchronization scheme for the verification of designs with multiple clock domains, the first of its kind. The original scheme is later extended with the use of a divider-based synchronizer to further improve performance. Extensive experiments confirm the attractiveness of the approach as they demonstrate significant memory savings, with comparable run-times, when contrasted to the state-of-the-art.

References

[1] H. Foster, "Assertion-based verification: Industry myths to realities (invited tutorial)," in *Computer Aided Verification*, 2008, pp. 5–10.

[2] A. Biere, M. Heule, H. V. Maaren, and T. Walsh, *Handbook of Satisfiability*. IOS Press, Inc., 2009.

[3] M. Ganai and A. Gupta, "Efficient BMC for multi-clock systems with clocked specifications," in *IEEE ASP Design Automation Conf.*, 2007, pp. 310–315.

[4] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," ser. Advances In Computers, no. 60, 2003.

[5] H. Mangassarian, A. Veneris, and M. Benedetti, "Robust QBF encodings for sequential circuits with applications to verification, debug, and test," *IEEE Trans. on Computers*, vol. 59, no. 7, pp. 981–994, 2010.

[6] D. K. E. Clarke and K. Yorav, "Specifying and verifying systems with multiple clocks," in *IEEE Int'l Conf. on Comp. Design*, 2003, pp. 48–55.

[7] P. Bjesse and J. Kukula, "Automatic generalized phase abstraction for formal verification," in *IEEE/ACM Int'l Conf. on CAD*, 2005, pp. 1073–1079.

[8] N. Sharif, N. Ramzan, F. K. Lodhi, O. Hasan, and S. R. Hasan, "Quantitative analysis of state-of-the-art synchronizers: Clock domain crossing perspective," in *Emerging Technologies (ICET), 2011 7th IEEE International Conference on*, 2011, pp. 1–6.

[9] M. Janota, W. Klieber, J. Marques-Silva and E. Clarke, "Solving qbf with counterexample guided refinement," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2012, pp. 114–128.

[10] A. Robison, "N-bit unsigned division via n-bit multiply-add," in *IEEE Symp. on Computer Arithmetic*, 2005, pp. 131–139.

[11] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Int'l Conf. on Theory and Applications of Satisfiability Testing*, 2003, pp. 502–518.

[12] OpenCores.org, "http://www.opencores.org," 2007.