Formal Modeling and Verification of PCHB Asynchronous Circuits

Ashiq A. Sakib, *Member, IEEE*, Scott C. Smith, *Senior Member, IEEE*, and Sudarshan K. Srinivasan, *Member, IEEE*

***Abstract*— Precharge half buffer (PCHB) is one of the major quasi-delay insensitive (QDI) asynchronous design paradigms, which has been utilized in several commercial applications due to its low power and inherent robustness. In industry, QDI circuits are often synthesized from a synchronous specification using custom synthesis tools. Design validation of the implemented QDI circuits mostly relies on extensive simulation, which may fail to detect corner-case bugs, especially in complex designs. Hence, a formal verification scheme for PCHB circuits is much needed. In this article, we present a formal verification methodology for PCHB circuits synthesized from a Boolean/synchronous specification, which is based on equivalence checking and can guarantee both safety (full functional correctness) and liveness (absence of deadlock). The approach is fast, scalable, and applicable to combinational as well as sequential PCHB circuits. We demonstrate the method using several multipliers, multiply and accumulate circuits (MACs), and IEEE International Symposium on Circuits and Systems (ISCAS) benchmarks.**

***Index Terms*— Asynchronous circuits, equivalence verification, formal methods, precharge half buffer (PCHB), quasi-delay insensitive (QDI).**

I. Introduction

THE synchronous design paradigm dominates today's semiconductor industry. However, this clocked approach is facing major challenges with today's high-speed, low-power design expectations, using processes with ever-increasing physical level variability. Operating frequencies in the GHz range complicate the existing clock management system, resulting in numerous timing-related issues, such as clock skew, clock jitter, etc. Furthermore, decreasing feature size results in higher power dissipation per unit area as well as more timing variability between dies due to increased process variations. Over the past few years, asynchronous, clockless quasi-delay insensitive (QDI) designs have been proven to be effective in circumventing the challenges faced by synchronous digital designs [1], due to their distributed switching (i.e., switching is not triggered simultaneously at the clock edge) and robustness against process, voltage, and timing (PVT) variations. The most recent 2013 international technology roadmap for semiconductors (ITRS) predicts asynchronous logic to account for more than 50% of IC global signaling in the multibillion-dollar semiconductor industry by 2027 [2], and the more recent 2018 IEEE International Roadmap for Devices and Systems (IRDS) lists asynchronous computing as a potential solution to reduce power consumption [3], but does not include a comparison of asynchronous versus synchronous signaling.

In industry, QDI circuits are often synthesized from their synchronous counterparts utilizing computer automated design (CAD) tools that cause the specification to undergo numerous transformations and optimizations, resulting in an implementation that is structurally very different from its specification. Any error in the synthesis tools will eventually result in an implementation error. Hence, validation is a critical step in the QDI design flow. Current QDI validation methods are mostly simulation-based, where the simulated behavior of the QDI implementation is compared to that of the specification. However, simulation alone cannot guarantee complete functional correctness (e.g., the FDIV bug in Intel's Pentium processor floating-point unit that went undetected during extensive simulation). Presently, formal verification methods are widely utilized in industry to complement traditional simulation methods in order to detect corner-case bugs.

Precharge half buffer (PCHB) [4] is one such commercially successful QDI design paradigm that has been utilized by major semiconductor companies, such as Intel and Achronix [1]. Although PCHB circuits are implemented commercially, there exist very few formal verification methods for such circuits, and the existing methods have several limiting factors. In [5], we discussed an equivalence verification methodology applicable only to combinational PCHB circuits. In this article, we extend our work to the verification of sequential PCHB circuits, based on equivalence checking. We further introduce additional changes to our previous combinational PCHB verification method [5], such that the verification method presented herein is applicable to both combinational and sequential PCHB circuits. Additionally, we present an analysis of all possible faults that may arise during synthesis and discuss how our approach detects all of those possible errors, ensuring complete correctness. Hence, the equivalence verification methodology illustrated in this article is a unified, fast, and highly scalable approach that can guarantee the safety and liveness of any combinational or sequential PCHB circuit.

Manuscript received April 2, 2019; revised July 5, 2019; accepted July 30, 2019. Date of publication September 13, 2019; date of current version November 22, 2019. This work was supported by the National Science Foundation under Grant CCF-1717420. *(Corresponding author: Scott C. Smith.)*

A. A. Sakib is with the Department of Electrical and Computer Engineering, Florida Polytechnic University, Lakeland, FL 33805 USA (e-mail: asakib@floridapoly.edu).

S. C. Smith is with the Department of Electrical Engineering and Computer Science, Texas A&M University Kingsville, Kingsville, TX 78363 USA (e-mail: scott.smith@tamuk.edu).

S. K. Srinivasan is with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58105 USA (e-mail: sudarshan.srinivasan@ndsu.edu).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2937087

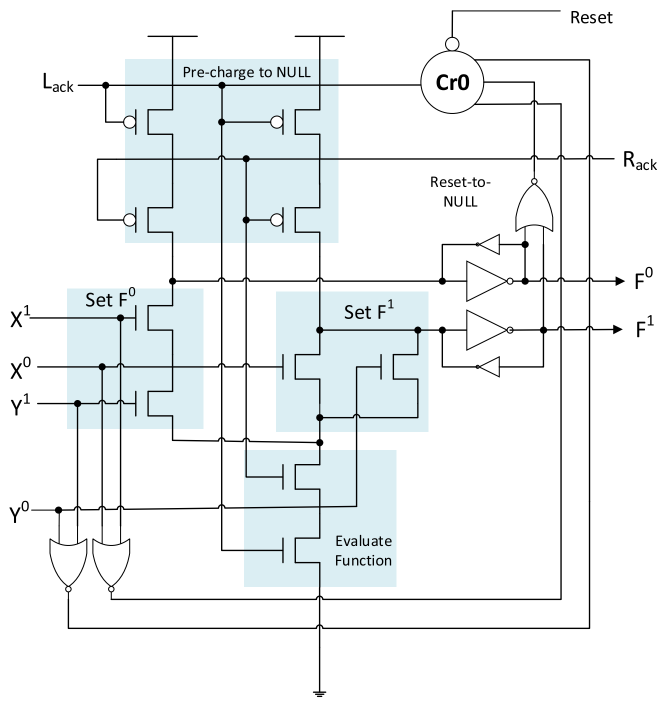


Fig. 1. PCHB NAND2 gate [7].

This article is divided into eight main sections. A brief overview of PCHB circuits and a review of related verification works are presented in Section II. All possible faults that could occur during PCHB synthesis are discussed in Section III. Section IV details our verification methodology for combinational PCHB circuits, which is then extended to sequential PCHB circuits in Section V. Section VI shows how our proposed verification method detects all possible faults, previously discussed in Section III. Verification results for several multipliers, multiply and accumulate circuits (MACs), and IEEE International Symposium on Circuits and Systems (ISCAS) benchmarks are presented in Sections IV-C and VII, followed by conclusions and directions for future work in Section VIII.

II. PCHB Background

A. PCHB Functionality

PCHB gates incorporate both registration and handshaking control [4]. In addition to performing specific logic functions, they also behave as memory elements. Therefore, an arrangement of multiple gates in a combinational PCHB circuit operates similarly to a synchronous pipeline (i.e., PCHB circuits themselves are not combinational; they include internal feedback, as shown in Fig. 1). The control consists of request and acknowledge signals from individual gates, *Rack* and *Lack*, respectively, and a combination of C-elements [6] to establish a well-defined handshaking scheme for synchronization. These unique features add to the complexity of the design, making formal verification of such circuits very challenging.

QDI circuits, such as PCHB, utilize multirail logic signals, such as dual-rail logic, where a signal consists of two wires, D^0 and D^1, which represent three states: DATA0 ($D^0 = 1$, $D^1 = 0$), DATA1 ($D^0 = 0$, $D^1 = 1$), and NULL ($D^0 = 0$, $D^1 = 0$). DATA0 and DATA1 are equivalent to Boolean logic 0 and 1, respectively. Unavailability of DATA is represented by NULL. ($D^0 = 1$, $D^1 = 1$) is an illegal state, as two rails cannot be asserted simultaneously. A PCHB gate has dual-rail inputs and outputs, X and Y, and F, respectively, for the NAND2 example in Fig. 1.

The set functions, F^0 and F^1, are implemented to achieve the particular gate functionality. The two-input NOR gates connected to both inputs' rails and the outputs' rails detect when a dual-rail signal is either DATA or NULL, and the C-element connects these completion detection signals to generate the gate's acknowledge signal, *Lack*. The weak inverter arrangement is used to hold the output DATA until precharged back to NULL to attain delay insensitivity. When *Lack* is logic 1, request-for-data (*rfd*), the inputs will eventually become DATA, and when *Lack* is logic 0, request-for-NULL (*rfn*), the inputs will eventually become NULL. The function evaluates and the output becomes DATA whenever both *Lack* and *Rack* are *rfd* and the X and Y inputs are DATA. If *Rack* is *rfd* and *Lack* is *rfn*, or vice versa, the state is held by the weak inverters. When *Lack* and *Rack* are both *rfn*, the output is precharged back to NULL. Whenever the inputs and outputs are all DATA, *Lack* changes to *rfn*; and when the inputs and output are all NULL, *Lack* changes to *rfd*. PCHB gates also include a *reset* input to initialize the gate's data output to NULL. This is done by initializing every gate's *Lack* output to logic 0, as shown in Fig. 1. Since a gate's *Rack* input comprises subsequent gates' *Lack* outputs that are combined through a C-element structure, all PCHB gates' *Lack* and *Rack* signals will both be logic 0, while reset is asserted, causing their data outputs to precharge to NULL.

Handshaking logic between PCHB gates can be implemented using either full-word or bit-wise completion [8], or some combination of the two. Full-word completion requires that the *Lack* signal of each PCHB gate in level$_i$ be conjoined by one or more C-elements to produce a single *Lack* signal, whose output is connected to the *Rack* signal of each PCHB gate in level$_{i-1}$, where a gate's level is the longest path (in terms of number of PCHB gates) from the circuit's primary inputs to that gate's output. On the other hand, bit-wise completion only sends the completion signal from PCHB gate b back to each PCHB gate whose output is an input to gate b.

Sequential PCHB circuits require at least $2N + 1$ latches in any feedback loop (FL) with N DATA tokens to avoid deadlock, and DATA tokens are inserted into a pipeline via a resettable latch (i.e., token buffer), whose data and *Lack* outputs are NULL and logic 0, respectively, while *reset* is asserted, to initialize the rest of the PCHB gates to NULL, as explained previously, and whose data output then changes to its initial DATA value when *reset* is deasserted [7]. However, PCHB gates themselves behave as latches, hence additional latches are not necessary if an FL already contains enough PCHB gates. For example, if there are 2 or more additional PCHB gates in an FL with 1 DATA token, then no additional latches are required since this loop would contain at least three latches. Note that nonresettable latches are designed and operate similar to that of regular PCHB gates, described previously, where the set function is output = input. Additionally, two resettable latches must be separated by a PCHB gate, which could be a nonresettable latch, in order to insert two DATA tokens; two adjacent resettable latches cause the system to deadlock immediately after reset. Also, an FL can never be all NULL (N) or all DATA (D). For example, in an FL consisting of three latches with one DATA token, (NNN) and (DDD) are illegal states, whereas any other of the six combinations are valid.

B. Related Verification Work

There have been several methodologies implemented to verify different types of asynchronous circuits. A trace-theory based method [9] was proposed to verify various asynchronous circuits at the gate level, such as Huffman circuits and Muller circuits, where the circuit behavior is represented as sets of traces, and the correctness properties are modeled as Petri nets. Methods based on refinement [10] and flow equivalence [11] have been used to verify bounded delay asynchronous paradigms, such as desynchronized circuits. A few formal verification methods have been developed for QDI NULL convention logic (NCL) circuits, including a method to verify the delay-insensitive property of combinational NCL circuits [12], and a method to check the functional equivalence of NCL circuits against their synchronous counterparts [13], both using the theory of well-founded equivalence bisimulation (WEB) refinement [14]. However, all of these approaches have been developed and tailored to a specific type of asynchronous paradigm, and none can be directly applied to PCHB circuits because of the major differences between paradigms.

There do exist some methods directly applicable to PCHB circuits, as described in the following, but these also have major limiting factors. In [15], a reverse synthesis-based approach that creates a high-level specification from a PCHB circuit is presented; however, in the case of a bug, the methodology does not address the issue of finding the error. For example, if the QDI circuit is buggy (e.g., a completion signal is missing in a completion network, such that under some extreme timing scenarios, the QDI circuit will malfunction), it is not clear if/how this will be preserved in the reverse synthesized output. Also, [15] is applicable only to control circuits, not to datapath circuits. Shih *et al.* [16] developed a deadlock verification scheme for sequential PCHB circuits that detect deadlocks by transforming the asynchronous pipeline into a Time Marked Graph, removing all edges containing initial tokens, and then detecting any remaining cycles (i.e., a deadlock-free circuit should be acyclic after removal of all initial token edges). The method effectively identifies deadlocks in any sequential PCHB circuit; however, it does not address verification of the combinational logic (C/L), neither functionality nor handshaking connections. Shih *et al.* [16] assumed that their optimized synthesis method for generating a combinational PCHB circuit from its Boolean specification, presented in [17], is correct. For example, inversion in a handshaking signal within the C/L would cause a deadlock, and swapped rails of a dual-rail signal would produce incorrect results, but not deadlock the system, neither of which would be detected by [16]. Saifhashemi *et al.* [18] presented an equivalence checking method to verify an asynchronous circuit implementation against its high-level asynchronous specification (e.g., using communicating sequential processes [19]), which is applicable to PCHB circuits. Our work herein differs from [18] in that we verify a PCHB circuit implementation against its synchronous specification.

In [20], we proposed a formal verification methodology for combinational PCHB circuits based on model checking, where the circuit to be verified is modeled as a transition system (TS), and correctness properties are specified using computational tree logic (CTL) [21]. We also developed a set of property templates for PCHB circuits, which can be used to verify any PCHB circuit that corresponds to a combinational Boolean circuit. Note that PCHB circuits themselves are not combinational, as each gate incorporates registration and control in addition to its logic function, as shown in Fig. 1. The templates can be classified as a set of local templates and one global template. The local templates are applicable locally to each PCHB gate and check for liveness of the circuit, which is the absence of deadlock. The global template checks for safety, i.e., under all circumstances, the circuit output is always correct. However, scalability is the major limiting factor of our previous model checking based verification method. Since each of the PCHB gates incorporates a hysteresis state holding capability with a complex handshaking scheme, the corresponding TS for a PCHB circuit is very complex, even for relatively simple circuits. This causes state space explosion, which in turn results in an infeasible verification time.

Therefore, in this article, we present an alternate approach to circumvent having to deal with the TS. Our proposed method is a unified verification approach based on equivalence verification that guarantees safety as well as liveness for any PCHB circuit, combinational or sequential; it is fast and highly scalable.

III. Enumeration of All Possible PCHB Faults

Our proposed verification method is applicable to a PCHB circuit synthesized from a Boolean/synchronous specification using the method presented in [17] and assumes that individual PCHB gates and C-elements are fault-free, which is consistent with standard gate-level verification methodologies. This type of PCHB circuit is depicted in Fig. 2, whose interface consists of a single *Rack* input, *reset* input, and *Lack* output, and one or more dual-rail inputs, *DI*, and dual-rail outputs, *DO*. The internal circuitry consists of a sea of PCHB gates, each designed as described in Section II-A, and a sea of C-elements [6] to implement the handshaking circuitry for communication between PCHB gates and with the external circuit interface. The developed method ensures that no interconnections between gates are erroneous and that the implemented PCHB function is equivalent to its Boolean/synchronous specification. Below is an enumeration of all possible faults that could occur in this type of PCHB circuit, referring to the signal names in Fig. 2, to aid the reader in visualizing the 18 resultant cases.

Case 1. Faulty Data Connection: Each PCHB gate receives its data inputs X from the circuit's primary data inputs *DI* and/or other PCHB gate data outputs F. A PCHB gate's data input X or primary circuit output *DO* could be the wrong dual-rail signal. For example, the F output of PCHB gate$_i$ should be connected to the X input of PCHB gate$_j$; however,

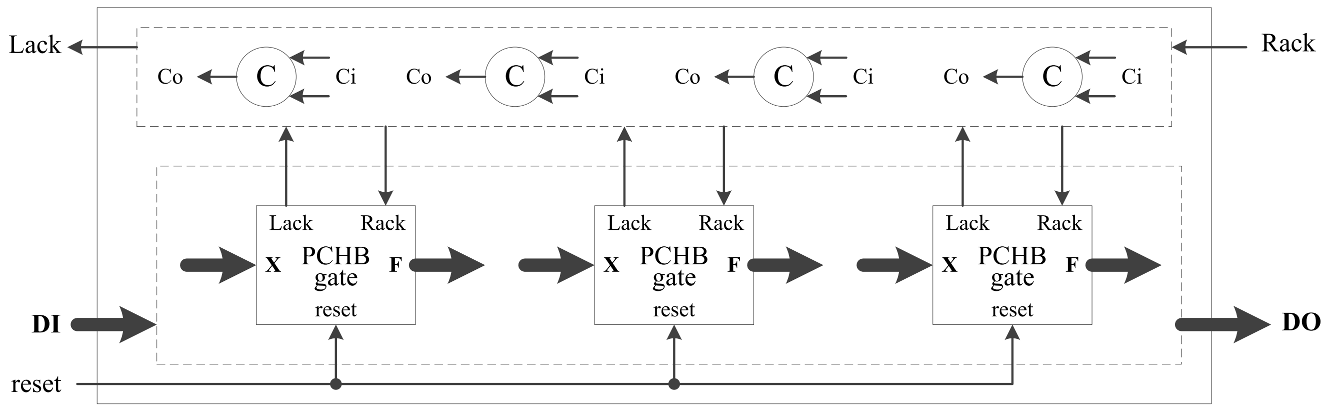


Fig. 2. Depiction of the PCHB circuit described in [17].

X is instead connected to the output of PCHB gate$_k$, which would result in a logical error, such that the specification and implementation circuits would not be functionally equivalent.

Case 2. Swapped Dual-Rail Connection: In PCHB circuits, all data signals are dual-rail logic, where two wires together represent one bit of data, as detailed in Section II-A. The rails of the dual-rail data input X or primary circuit output *DO* could be unintentionally swapped. For example, if the PCHB gate$_i$ output F is supposed to connect to the PCHB gate$_j$ input X, this implies that F_i^0 and F_i^1 should connect to X_j^0 and X_j^1, respectively. However, swapping the dual-rail connections would result in F_i^0 and F_i^1 connected to X_j^1 and X_j^0, respectively, which would correspond to the inversion of that signal, resulting in a logical error, such that the specification and implementation circuits would not be functionally equivalent.

Case 3. Rails From Different Signals: A PCHB gate's data input X or primary circuit output *DO* could be incorrectly composed of two different dual-rail signals' rails. For example, the X^1 input of PCHB gate$_i$ is connected to the F^1 output of PCHB gate$_j$ and the X^0 input of PCHB gate$_i$ is connected to the F^0 output of PCHB gate$_k$. This will result in the circuit deadlocking when F_j = DATA0 and F_k = DATA1, since X will never transition to DATA, and will result in X being an illegal value (i.e., $X^0 = 1$ and $X^1 = 1$) when F_j = DATA1 and F_k = DATA0.

Case 4. Rail Duplication: A PCHB gate's data input or primary circuit output could be incorrectly composed of the same rail of a dual-rail signal. For example, both rails of the X input of PCHB gate$_i$ are connected to the F^1 output of PCHB gate$_j$. This will result in the circuit deadlocking when F = DATA0, since X will never transition to DATA, and will result in X being an illegal value when F = DATA1.

Case 5. Handshaking Signal Connected to Data Signal: A PCHB gate's data input X or primary circuit output *DO* could be either partially or fully composed of one or two handshaking network signals (i.e., PCHB gate *Rack/Lack* signal or C-element output *Co*), which would result in the affected dual-rail signal being stuck at NULL for some cases, causing circuit deadlock, and being an illegal value for other cases.

Case 6. Incorrect Logic Implementation: The functionality of the PCHB circuit is not equivalent to its specification. For example, the specification F = AB + C is implemented as a PCHB circuit utilizing a two-input PCHB AND gate, followed by a two-input PCHB XOR gate, instead of a correct implementation that utilizes a two-input PCHB AND gate followed by a two-input PCHB OR gate.

Case 7. Non-PCHB Gate in Datapath: The datapath of PCHB circuits consists entirely of PCHB gates, all of which have one or more data inputs X, one or more data outputs F, a *Rack* input, a *Lack* output, and a *reset* input, as shown in Fig. 2. Any type of gate other than a PCHB gate in the datapath is an error, which may cause the circuit to deadlock or result in a logical error, such that the specification and implementation circuits would not be functionally equivalent.

Case 8. Incorrect Reset: Every PCHB gate includes a *reset* input for initialization, as described in Section II-A, all of which must be connected to the circuit's external *reset* input, which itself must not be connected to any other gate input. A PCHB latch with an incorrect reset value will either result in the circuit deadlocking or not being functionally equivalent to its specification. Take for example, a MAC, where all outputs should be reset to DATA0, according to its specification. If instead one or more of the PCHB implementation's outputs are reset to DATA1, the result of the first MAC operation will differ from its specification (i.e., $A_1 = A_0 + X_1 \times Y_1$, where $A_0 \neq 0$, versus the correct implementation: $A_1 = 0 + X_1 \times Y_1$). Instead, if an output is reset to NULL by using a nonresettable latch instead of the correct resettable one, this would result in a feedback path with no DATA tokens, which would cause the circuit to deadlock.

Case 9. Insufficient Latches in an FL: The four-phase QDI handshaking protocol utilized for PCHB circuits requires at least $2N + 1$ PCHB latches in an FL that contains N DATA tokens, in order to avoid deadlock [7]. For example, FLs with a single DATA token, such as a MAC (i.e., $A_i = A_{i-1} + X_i \times Y_i$), require at least three PCHB latches in every feedback path, otherwise, the circuit will deadlock. Since every PCHB gate includes an internal latch, at least two PCHB gates, in addition to the resettable latch to insert the initial DATA token, are sufficient for MAC FLs.

Case 10. Missing Handshaking Signal: Each PCHB gate$_j$ ($j \epsilon [1, N]$) whose data input X is a data output F of PCHB gate$_i$, must acknowledge PCHB gate$_i$, resulting in the *Lack* output of each PCHB gate$_j$ being conjoined via an N-input C-element structure, whose output *Co* is the *Rack* input of PCHB gate$_i$. For example, if a data output F of PCHB gate$_k$ is a data input X of PCHB gate$_y$ and the *Lack* output of PCHB gate$_y$ is not an input *Ci* to the C-element structure that generates the *Rack* input to PCHB gate$_k$, the circuit will deadlock under some timing scenarios.

Case 11. Additional Handshaking Signal: If the C-element structure that generates the *Rack* input for PCHB gate$_i$ contains a *Lack* input from PCHB gate$_j$ and a data output F of PCHB gate$_i$ is not a data input X of PCHB gate$_j$, the circuit may deadlock or slowdown, but could also operate correctly. This is not necessarily an error; however, the additional handshaking signal requires further inspection.

Case 12. External Lack Error: The external *Lack* output synchronizes all circuit primary data inputs *DI*. Hence, the *Lack* outputs of all PCHB gates that have a circuit primary data input *DI* as a data input X must be combined through a C-element structure to produce the external *Lack* output. Like Case 10, any missing *Lack* input to this C-element structure will cause the circuit to deadlock under some timing scenarios. Similar to, but different from Case 11, any additional *Lack* input to this C-element structure is an error, which may cause the circuit to slowdown or deadlock.

Case 13. External Rack Error: The external *Rack* input synchronizes all circuit primary data outputs *DO*. Hence, for each PCHB gate$_j$ whose data output F is a circuit primary data output *DO*, the external *Rack* input must either be the *Rack* input to gate$_j$ or an input *Ci* to the C-element structure that generates the *Rack* input for gate$_j$ (as would be the case when an external data output *DO* is fed back as an input X to another PCHB gate). Like Case 10, if the external *Rack* input is missing from the C-element structure that generates the *Rack* input for a PCHB gate whose data output F is a circuit primary data output DO, the circuit will deadlock under some timing scenarios.

Case 14. Non-C-Element in Handshaking Circuitry: PCHB handshaking circuitry is entirely composed of C-element structures, as depicted in Fig. 2, which consist of 0 or more C-elements that combine N *Lack* signals into a single *Rack* signal or the external *Lack*. Hence, any gate other than a C-element in the PCHB handshaking circuitry is an error, which will cause the circuit to deadlock under some timing scenarios.

Case 15. Data Signal Input to C-Element: As mentioned in Case 14, C-elements occur only in the handshaking circuitry to combine *Lack* signals; they are not utilized in the datapath. Hence, either rail of a data signal, X or F, being an input, *Ci*, to a C-element is an error, which will cause the circuit to deadlock under some timing scenarios.

Case 16. Data Signal Input to PCHB Gate Rack Input: As mentioned in Cases 10 and 13, a PCHB gate's *Rack* input may only be the output, *Co*, of a C-element, another PCHB gate's *Lack* output, or the external *Rack* input. Hence, either rail of a data signal, X or F, being a PCHB gate's *Rack* input is an error, which will cause the circuit to deadlock.

Case 17. C-Element Structure Feedback: As mentioned in Cases 10 through 13, a C-element structure combines multiple PCHB gate *Lack* outputs, and possibly the external *Rack* input, to generate PCHB gate *Rack* inputs or the external *Lack* output; hence, C-element structures are feedforward only, such that any FL within a C-element structure is an error, which will cause the circuit to deadlock.

Case 18. Shorted Output: An output, *Co*, of a C-element or any output of a PCHB gate, F or *Lack*, cannot be directly connected to any other PCHB gate output, F or *Lack*, or C-element output *Co* or any external input, *DI*, *Rack*, or *reset*. This would result in a wire short, causing the affected signal to be undefined when the logical values of the shorted wires differed.

These 18 cases comprise all possible faults that could occur in a PCHB circuit synthesized from a Boolean/synchronous specification using the method presented in [17]. In order to establish our claim that the abovementioned 18 cases comprise all possible faults, we analyze an exhaustive conjunction of PCHB gates, C-elements, and PCHB circuit external inputs and outputs, from which only a small set of connections is legal, and a large set of connections is illegal/faulty. We then illustrate how every faulty connection can be categorized by at least one of the above 18 fault case scenarios.

Let us consider a PCHB circuit with $N \geq 1$ PCHB gates, $M \geq 0$ C-elements, $Z \geq 1$ external data inputs, $Y \geq 1$ external data outputs, a single *Rack* input, a single *Lack* output, and a single *reset* input, as depicted in Fig. 2. Considering all possibilities, the dual-rail output F of PCHB gate$_k$ has the following 12 interconnection scenarios: it could be connected to: 1) dual-rail input(s) X of other PCHB gates$_j$, where $j \neq k$; 2) external data output *DO*; 3) dual-rail input(s) X of other PCHB gates, including gate$_k$; 4) *Rack* input of a PCHB gate; 5) C-element input *Ci*; 6) another PCHB gate data output F; 7) a PCHB gate *Lack* output; 8) a C-element output *Co*; 9) external data input *DI*; 10) external *Rack* input; 11) external *reset* input; or 12) external *Lack* output. Of these, only 1) and 2) are possibly correct, and will be expanded upon later; all other interconnection scenarios, 3) through 12), are faulty. Note that 3) corresponds to Case 9; 4) to Case 16; 5) to Case 7 or 15; 6) through 11) to Case 18; and 12) to Case 12 or 18. For 1), the dual-rail output F of PCHB gate$_k$ could be correctly connected to the data inputs X of PCHB gates$_j$, or could be incorrectly connected via a swapped rail connection (Case 2), being an input X to a wrong PCHB gate (Case 1 or 6), or only being a partial input X to a PCHB gate (Cases 3 or 4). For 2), the dual-rail output F of PCHB gate$_k$ could be correctly connected to an external data output *DO* or could be incorrectly connected via a swapped rail connection (Case 2), being connected to the wrong external data output *DO* (Case 1), or only being partially connected to the external data output *DO* (Case 3 or 4).

Considering all possibilities, the *Lack* output of PCHB gate$_k$ has the following 12 interconnection scenarios: it could be connected to: 1) *Rack* input of other PCHB gates$_j$, where $j \neq k$; 2) input *Ci* of one or more C-elements; 3) external *Lack* output; 4) *Rack* input of other PCHB gates, including gate$_k$; 5) dual-rail input X of a PCHB gate; 6) external data output *DO*; 7) PCHB gate data output F; 8) another PCHB gate *Lack* output; 9) a C-element output *Co*; 10) external data input *DI*; 11) external *Rack* input; or 12) external *reset* input. Of these, only 1), 2), and 3) are possibly correct, and will be expanded upon later; all other interconnection scenarios, 4) through 12), are faulty. Note that 4) corresponds to Case 9; 5) and 6) to Case 5, and 7) through 12) to Case 18. For 1), the *Lack* output of PCHB gate$_k$ could be correctly connected to the *Rack* input of other PCHB gates, or could be incorrectly connected by being the *Rack* input to a PCHB gate whose data output F was not an input X to gate$_k$ (Case 10). For 2), the *Lack* output of PCHB gate$_k$ could be correctly connected to C-element input(s) *Ci* or could be incorrectly connected by being an input *Ci* to a C-element structure that outputs the *Rack* input for a PCHB gate whose data output F was not an input X to gate$_k$ (Case 11). For 3), the *Lack* output of PCHB gate$_k$ could be correctly connected to the external *Lack* output or could be incorrectly connected if the external data inputs *DI* are connected to PCHB gates other than gate$_k$ (Case 12).

Considering all possibilities, the output of C-element$_b$ *Co* has the following 12 interconnection scenarios: it could be connected to: 1) input(s) *Ci* of other C-elements$_j$, where $j \neq b$; 2) external *Lack* output; 3) *Rack* input of a PCHB gate; 4) input(s) *Ci* of C-elements, including C-element$_b$; 5) dual-rail input X of a PCHB gate; 6) external data output *DO*; 7) PCHB gate data output F; 8) PCHB gate *Lack* output; 9) another C-element output *Co*; 10) external data input *DI*; 11) external *Rack* input; or 12) external *reset* input. Of these, only 1), 2), and 3) are possibly correct, and will be expanded upon later; all other interconnection scenarios, 4) through 12), are faulty. Note that 4) corresponds to Case 17; 5) to Case 5 or 14, 6) to Case 5, and 7) through 12) to Case 18. For 1), the output *Co* of C-element$_b$ could be correctly connected to other C-element inputs *Ci* or could be incorrectly connected by being part of the C-element structure that produces the *Rack* input for PCHB gate$_k$, where PCHB gate$_k$'s data output F was not an input X to all PCHB gates whose *Lack* outputs are inputs *Ci* to the C-element structure containing C-element$_b$ (Case 11), or by being connected to a C-element input *Ci* within the same C-element structure, forming an FL within the C-element structure (Case 17). For 2), the output *Co* of C-element$_b$ could be correctly connected to the external *Lack* output or could be incorrectly connected if the external data inputs *DI* are connected to PCHB gates other than those whose *Lack* outputs are the inputs *Ci* to the C-element structure containing C-element$_b$ (Case 12). For 3), the output *Co* of C-element$_b$ could be correctly connected to the *Rack* input of a PCHB gate, or could be incorrectly connected by being the *Rack* input to a PCHB gate whose data output F was not an input X to all PCHB gates whose *Lack* outputs are inputs *Ci* to the C-element structure containing C-element$_b$ (Case 11).

Each external data input *DI* is treated similarly to a PCHB gate data output F. Considering all possibilities, an external data input *DI* has the following 11 interconnection scenarios: it could be connected to: 1) the dual-rail input X of one or more PCHB gates; 2) external data output *DO*; 3) *Rack* input of a PCHB gate; 4) C-element input *Ci*; 5) another external data input *DI*; 6) PCHB gate data output F; 7) a PCHB gate *Lack* output; 8) a C-element output *Co*; 9) external *Rack* input; 10) external *reset* input; or 11) external *Lack* output. Of these, only 1) and 2) are possibly correct, and will be expanded upon later; all other interconnection scenarios, 3) through 11), are faulty. Note that 3) corresponds to Case 16, 4) to Case 7 or 15, 5) through 10) to Case 18, and 11) to Case 12 or 18. For 1), an external data input *DI* could be correctly connected to the data inputs X of PCHB gates, or could be incorrectly connected via a swapped rail connection (Case 2), being an input X to a wrong PCHB gate (Case 1 or 6), or only being a partial input X to a PCHB gate (Case 3 or 4). For 2), an external data input *DI* could be correctly connected to an external data output *DO* or could be incorrectly connected via a swapped rail connection (Case 2), being connected to the wrong external data output *DO* (Case 1), or only being partially connected to an external data output *DO* (Case 3 or 4).

The external *Rack* input is treated similarly to a PCHB gate *Lack* output. Considering all possibilities, the external *Rack* input has the following ten interconnection scenarios: it could be connected to: 1) *Rack* input of one or more PCHB gates; 2) input *Ci* of one or more C-elements; 3) external *Lack* output; 4) dual-rail input X of a PCHB gate; 5) external data output *DO*; 6) PCHB gate data output F; 7) PCHB gate *Lack* output; 8) a C-element output *Co*; 9) external data input *DI*; or 10) external *reset* input. Of these, only 1) and 2) are possibly correct, and they will be expanded upon later; all other interconnection scenarios, 3) through 10), are faulty. Note that 3) corresponds to Case 12 or 18; 4) and 5) to Case 5, and 6) through 10) to Case 18. For 1), the external *Rack* input could be correctly connected to the *Rack* input of PCHB gates or could be incorrectly connected by being the *Rack* input to a PCHB gate whose data output F was not an external data output *DO* (Case 11), or by not being connected to the *Rack* input of a PCHB gate whose output F is an external data output *DO* (Case 13). For 2), the external *Rack* input could be correctly connected to one or more C-element inputs *Ci*, or could be incorrectly connected by being an input *Ci* to a C-element structure that outputs the *Rack* input for a PCHB gate whose data output F is not an external output *DO* (Case 11), or by not being connected to a C-element input *Ci* that is part of a C-element structure that generates the *Rack* input of a PCHB gate whose output F is an external data output *DO* (Case 13).

Considering all possibilities, the external *reset* input has the following 11 interconnection scenarios: it could be connected to: 1) the reset input of one or more PCHB gates; 2) the data input X of a PCHB gate; 3) external data output *DO*; 4) *Rack* input of a PCHB gate; 5) C-element input *Ci*; 6) external data input *DO*; 7) PCHB gate data output F; 8) PCHB gate *Lack* output; 9) C-element output *Co*; 10) external *Rack* input; or 11) external *Lack* output. Of these, only 1) is possibly correct, and will be expanded upon later; all other interconnection scenarios, 2) through 11), are faulty. Note that 2) through 5) correspond to Case 8 and 6) through 11) to Case 18. For 1), the external *reset* input is correctly connected only if connected to every PCHB gate's *reset* input. Additionally, the external *reset* input is used to initialize the PCHB circuit, which must be reset to a live state and match the reset state of its corresponding Boolean/synchronous specification circuit, both covered by Case 8.

In summary, the above exhaustive enumeration of all possible interconnection combinations of PCHB gates, C-elements, and circuit primary inputs and outputs, proves that every

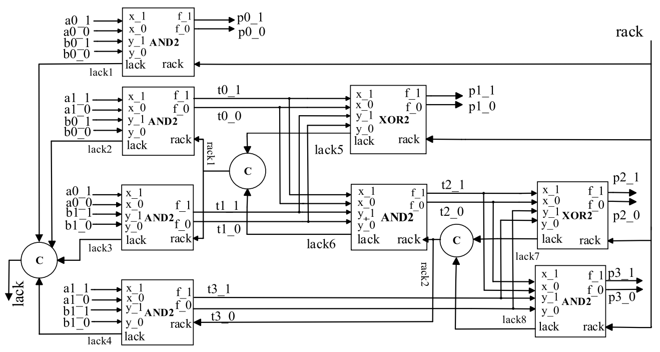


Fig. 3. PCHB 2 × 2 multiplier.

possible faulty connection maps to at least one of the 18 cases presented in this section, thereby proving that these 18 cases do indeed comprise all possible faults that could occur in a PCHB circuit.

IV. Equivalence Verification of Combinational PCHB Circuits

The developed methodology includes an equivalence verification scheme that verifies the functionality of a combinational PCHB circuit against its respective Boolean specification to ensure safety and a graph-based approach to ensure liveness and handshaking correctness, and both are described as follows. The ability of the proposed methodology to detect all possible faults is addressed in Section VI.

A. Safety Check of Combinational PCHB Circuits

The safety check requires two steps. First, a conversion algorithm takes the netlist of a combinational PCHB circuit as input and transforms that into a corresponding Boolean netlist. The generated Boolean circuit is then checked against the Boolean specification using an equivalence checker. To describe the methodology, the 2 × 2 PCHB multiplier, shown in Fig. 3, is used as an example. Note that although the PCHB multiplier is similar to a Boolean multiplier at the gate level, PCHB gate structures are far more complex. For example, a two-input Boolean NAND gate requires only four transistors, whereas the two-input PCHB NAND gate, shown in Fig. 1, requires 44 transistors to account for dual-rail signaling, registration, and handshaking control. In general, PCHB circuits require approximately 4–13 times more transistors than the corresponding Boolean circuits due to their complex features, as can be seen in Table I.

Fig. 4 shows the netlist format of the 2 × 2 PCHB multiplier. The first two lines correspond to all primary data inputs and outputs of the circuit, respectively. A dual-rail signal $a0$ is represented as "$a0\_1a0\_0$," where $a0\_1$ and $a0\_0$ are *rail*^{1} and *rail*^{0} of $a0$, respectively. Lines 3–10 represent the individual PCHB gates used in the circuit. The first column of each line represents the type of gate, where the right-hand number implies the number of gate inputs; e.g., *and*2 represents a two-input AND gate. The second column indicates the level of the gate, which is the longest path (in terms of number of PCHB gates) from the circuit's primary inputs to that gate's output. The remaining columns list the gate's data input(s), *reset* input, *Rack* input, *Lack* output, and data output(s), respectively. Type *Cn* in lines 11–13 represents an n-input C-element used to connect the PCHB handshake signals. Following *Cn* are its *n* inputs, and then its output.

```
1. a0_1a0_0, a1_1a1_0, b0_1b0_0, b1_1b1_0
2. p0_1p0_0, p1_1p1_0, p2_1p2_0, p3_1p3_0
3. and2  1  a0_1a0_0,b0_1b0_0  rst   rack  lack1   p0_1p0_0
4. and2  1  a1_1a1_0,b0_1b0_0   rst   rack1  lack2 t0_1t0_0
5. and2  1  a0_1a0_0,b1_1b1_0   rst   rack1   lack3 t1_1t1_0
6. and2  1  a1_1a1_0,b1_1b1_0   rst   rack2   lack4 t3_1t3_0
7. xor2  2   t0_1t0_0,t1_1t1_0   rst    rack   lack5   p1_1p1_0
8. and2  2   t0_1t0_0,t1_1t1_0   rst   rack2   lack6   t2_1t2_0
9. xor2   3   t2_1t2_0,t3_1t3_0   rst   rack   lack7   p2_1p2_0
10. and2 3   t2_1t2_0,t3_1t3_0  rst   rack   lack8   p3_1p3_0
11. C2     lack5, lack6    rack1
12. C2     lack7, lack8    rack2
13. C4     lack1, lack2, lack3, lack4   lack
```

Fig. 4. Netlist of the PCHB 2 × 2 multiplier.

(a)	(b)	
1. a0, a1, b0, b1	a0: fanout: [1 3]	comp_fanin: [1 2 3 4]
2. p0, p1, p2, p3	a1: fanout: [2 4]	comp_fanin: [1 2 3 4]
3. and2 1 a0,b0 p0	b0: fanout: [1 2]	comp_fanin: [1 2 3 4]
4. and2 1 a1,b0 t0	b1: fanout: [3 4]	comp_fanin: [1 2 3 4]
5. and2 1 a0,b1 t1	1: fanout: 0	comp_fanin: 0
6. and2 1 a1,b1 t3	2: fanout: [5 6]	comp_fanin: [5 6]
7. xor2 2 t0, t1 p1	3: fanout: [5 6]	comp_fanin: [5 6]
8. and2 2 t0, t1 t2	4: fanout: [7 8]	comp_fanin: [7 8]
9. xor2 3 t2, t3 p2	5: fanout: 0	comp_fanin: 0
10. and2 3 t2, t3 p3	6: fanout: [7 8]	comp_fanin: [7 8]
	7: fanout: 0	comp_fanin: 0
	8: fanout: 0	comp_fanin: 0

Fig. 5. (a) Converted Boolean netlist. (b) Fan_out and comp_fanin structure.

The PCHB netlist is automatically converted into its corresponding Boolean netlist, shown in Fig. 5(a), using a developed algorithm. Each dual-rail signal, including the primary inputs/outputs, are replaced with a corresponding Boolean signal. A PCHB gate structure containing all information related to individual gates is created by traversing the netlist. A Boolean gate structure is created by replacing each PCHB gate with its corresponding Boolean gate. Swapped rails of a dual-rail signal result in the introduction of an inverter. For example, if line 3 of Fig. 4 was instead "and2 1 a0_0a0_1 …," this would result in the following additional line in Fig. 5(a): "not 1 a0 a0_bar;" and line 3 of Fig. 5(a) would be changed to "and2 2 a0_bar,b0 p0." Therefore, any bug causing unintended rail swap in the implementation will be detected, as the added inverter will result in functional inequivalence between the specification and PCHB implementation. If a PCHB gate input's *rail*^{1} and *rail*^{0} are not part of the same dual-rail signal, an error message is generated noting the misconnection between rails, and where this occurs. Similarly, an error message is generated if any gate's data rails contain any handshaking signal(s). A further check flags any bug that causes only one rail of a dual-rail signal to be connected to both rails of another dual-rail signal. Another check ensures that the circuit's external *reset* input is connected to all PCHB gates' *reset* input, and not connected to anything else, flagging any gate or connection that violates this.

The converted Boolean netlist is then automatically encoded in the Satisfiability Modulo Theory Library (SMT_LIB)

TABLE I
VERIFICATION RESULTS FOR VARIOUS COMBINATIONAL PCHB CIRCUITS

PCHB Circuits	**# Transistors in Boolean Circuit**	**# Transistors in PCHB Implementation**	**Proposed Method Time (sec)**	**Model Checking [20] Time (sec)**
Full Adder	36	256 (i.e., 2×XOR2 and 3×NAND2)	<0.01	21.54
ISCAS C-17	24	308	0.01	138.62
4-to-16 decoder	136	1,188	0.01	222.6
32- bit MUX	386	4,436	0.23	413.20
4x4 Multiplier	456	3,196	0.06	Timed Out
8x8 Multiplier	2,256	15,980	12.18	Time Out Assumed
10x10 Multiplier	3,660	25,924	741.84	Time Out Assumed
11x11 Multiplier	4,488	31,790	7323.07	Time Out Assumed
12x12 Multiplier	5,400	38,256	62,823.67	Time Out Assumed
10x10Mul-B1	3,660	25,924	1.01	Time Out Assumed
10x10Mul-B2	3,660	25,924	0.06	Time Out Assumed
10x10Mul-B3	3,660	25,924	0.84	Time Out Assumed
10x10Mul-B4	3,660	25,924	0.79	Time Out Assumed
ISCAS c432	816	6,924	1.31	Time Out Assumed

language, using Python, and is then input to an SMT solver to check for functional equivalence between the transformed Boolean version of the original PCHB circuit and its corresponding Boolean specification. For the 2×2 multiplier example, the SMT solver checks for the following safety property: $F_{\text{PCHB\_Bool\_Equivalent}}$ $(a0, a1, b0, b1) = \text{MUL}\ (a, b)$, where $(a1, a0)$ and $(b1, b0)$ are the (most significant bit, least significant bit) of a and b, respectively. We use the Z3 SMT solver [22] to check for equivalence verification, but any combinational equivalence checker could be used.

B. Liveness and Handshaking Correctness Check

Liveness means the absence of deadlock in a circuit. For combinational PCHB circuits, proper connections between handshaking signals ensure liveness and proper synchronization. The same PCHB netlist shown in Fig. 4, used as input for the safety check method, is also utilized as input for the liveness check, to trace back the handshaking paths and C-element connections to verify proper handshaking, ensuring that every output generated by a particular input acknowledges that input. Procedure 1 illustrates the algorithm that checks the handshaking connections. A C-element structure and a PCHB gate structure containing all information related to C-elements (i.e., C-element type, inputs, and outputs) and individual gates (i.e., gate type, level, data inputs, rack, lack, and outputs), respectively, are created by traversing the netlist (lines 1 and 2 in Procedure 1). For each PCHB gate, i, its output is compared with every other PCHB gate j's inputs, $i \neq j$, to generate a fanout list, *fanout (i),* for PCHB gate i (line 3 in Procedure 1).

For example, referring to Fig. 4, *fanout* for the *and*2 gate on line 4 would contain the *xor*2 gate on line 7 and the *and*2 gate on line 8. For each PCHB gate, i, its *Rack* input is compared with every other PCHB gate j's *Lack* output, $i \neq j$, and every C-element's output, to generate a completion fanin list, *comp_fanin (i),* for PCHB gate i (line 4 in Procedure 1). For example, referring to Fig. 4, *comp_fanin* for the *and*2 gate on line 4 would contain the *xor*2 gate on line 7 and the *and*2 gate on line 8, since both of their *Lack* outputs are inputs to the C-element on line 11, whose output is the *Rack* input of the *and*2 gate on line 4. Similarly, a *fanout* and *comp_fanin* list is generated for each external data input.

After *fanout* and *comp_fanin* for each PCHB gate and external data input are calculated, as shown in Fig. 5(b) for the 2×2 multiplier example, *fanout*(k) is checked to ensure that it is a subset of *comp_fanin(k),* for all PCHB gates and external data inputs (lines 5–19 in Procedure 1). Bit-wise completion results in *fanout*(k) being equal to *comp_fanin*(k), while full-word completion results in *fanout*(k) being a proper subset of *comp_fanin*(k), with the restriction that each gate that is in *comp_fanin*(k) and not in *fanout*(k) must be from the immediate subsequent level of gate/input k. *fanout*(k) not being a subset of *comp_fanin*(k) could result in deadlock, while *fanout*(k) being a proper subset of *comp_fanin*(k), but violating the level restriction described, could either result in deadlock or may just decrease the circuit performance. Hence, if *fanout*(k) is a proper subset of *comp_fanin*(k), then each gate that is in *comp_fanin*(k) and not in *fanout*(k) is automatically inspected to ensure that it meets this level restriction. Even if the level restriction is met, a warning message is still generated to note the extra gate in the particular PCHB gate's *comp_fanin* list, to allow for easy manual inspection.

Additional checks ensure correct connection of the external *Rack* input and proper generation of the external *Lack* output. The external *Rack* signal should be connected to the *rack* inputs of all gates that produce the circuit's external data outputs. Similarly, the *lack* outputs of all gates that take primary data inputs as their inputs should be conjoined via a C-element structure to generate the external *Lack* output. The developed algorithm generates an appropriate descriptive error message, in case the PCHB circuit fails to satisfy any of these checks. Furthermore, it checks to ensure that no data signal is part of the handshaking connections and that no handshaking signal is part of a data signal. All these checks are performed during the process of creating the PCHB gate and C-element

Procedure 1 Procedure to Check Handshaking Connections

```
1: Create C_element_structure (PCHB_Netlist)
2: Create PCHB_gate_structure (PCHB_Netlist)
3: Create fanout (PCHB_gate_structure)
4: Create Comp_fanin (PCHB_gate_structure, C_element_structure)
5: for i←1 to num_pchb_gates do
6:     if fanout(i) = = Comp_fanin(i) then
7:         // no error
8:     else if fanout(i) ⊂ Comp_fanin(i) then
9:         for each variable v: v ∈ Comp_fanin(i) and v ∉ fanout(i) do
10:            if v.level ≤ gate(i).level then
11:                report level error message
12:            else
13:                report warning message
14:            end if
15:        end for
16:    else
17:        report error message
18:    end if
19: end for
```

structures (lines 1 and 2 in Procedure 1). Note that *fanout* 0 indicates an external output, while *comp_fanin* 0 denotes an external *Rack* input. The running time for this liveness check algorithm is $O(I + P) * (P + C)$, where I, P, and C are the number of external inputs, PCHB gates, and C-elements in the circuit, respectively.

C. Results

This equivalence verification methodology for combinational PCHB circuits has been demonstrated on several multipliers and ISCAS-85 [23] combinational circuit benchmarks, and the verification times are compared with the previous model checking based PCHB formal verification methodology [20]. As shown in Table I, the equivalence verification methodology presented herein is significantly faster than the model checking based approach for every circuit. Furthermore, this methodology was able to verify complex circuits with hundreds of gates, such as a 12 × 12 multiplier, whereas the model checking approach Timed Out for much smaller circuits, demonstrating the scalability of this approach. Since the model checking approach Timed Out for a 4 × 4 multiplier, we can safely assume that it would time out for more complex circuits, such as ISCAS c432 and other higher order multipliers. 10 × 10Mul-B1, 10 × 10Mul-B2, 10 × 10Mul-B3, and 10 × 10Mul-B4 are some of the tried buggy circuits. In 10 × 10Mul-B1, a bug was introduced in the data signals by incorrectly connecting one gate's dual-rail input to another dual-rail signal. 10×10Mul-B2 represents a logic element bug, where a PCHB AND gate was replaced with a PCHB NAND gate. 10×10Mul-B3 represents a handshaking connection bug, where the *Lack* output from PCHB gate i was not included in the C-element structure that generated the *Rack* input to PCHB gate j, even though PCHB gate j's data output was an input to PCHB gate i. 10 × 10Mul-B4 swaps the rails of one PCHB gate's input. In these, and every other buggy case tried, the proposed methodology detected and identified each bug very quickly. Verification was performed using the Z3 SMT solver [22] on an Intel Core i7-4790 CPU with 32 GB of RAM, running at 3.60 GHz. The verification times in Table I include only the Z3 runtime, as the netlist conversion times and time required to verify the handshaking signals were negligible in comparison.

V. Equivalence Verification of Sequential PCHB Circuits

As described in the previous section, our equivalence verification methodology proved to be a much faster and more scalable approach for combinational PCHB circuits, compared to the previous model checking method. Hence, in this section, we extend that approach to the verification of sequential PCHB circuits, which is far more complex due to datapath feedback.

The verification procedure requires three steps. In the first step, we take a sequential PCHB circuit and convert it to an equivalent synchronous circuit. We utilize the theory of WEB-refinement [14] to compare the synchronous netlist generated from the PCHB circuit with the original synchronous specification as the notion of correctness. The major advantage of applying WEB-refinement to the generated equivalent synchronous circuit instead of the actual PCHB circuit is that synchronous circuit signal transitions are much more deterministic compared to PCHB, where inputs can start propagating through the circuit any time and in any order, instead of at a predetermined clock edge, which makes the verification time much faster. The generated synchronous circuit, the specification synchronous circuit, and the WEB-refinement property are automatically encoded in the SMT-LIB language. The resulting equivalence property is then checked using an SMT solver. In the second step, we check the handshaking connections between components, which is similar to the combinational PCHB handshaking check discussed in Section IV-B. The third step consists of applying the method developed by Shih *et al.* [16] for deadlock verification of sequential PCHB circuits. Since this third step is fully detailed in [16], it is not discussed further herein, besides its brief overview in Section II-B.

To describe our methodology, we will use a MAC unit as an example circuit. Fig. 6(a) shows a synchronous MAC, where $A' = A + X \times Y$; Fig. 6(b) shows the equivalent PCHB version. Two latches are shown in the PCHB version such that all FLs contain at least three latches to avoid deadlock since PCHB gates themselves act as latches and the PCHB C/L contains at least one gate in every feedback path. However, some of the feedback paths only require the single resettable latch, as shown in Fig. 7(a), since they already contain at least two other PCHB gates. Although the synchronous and PCHB MACs seem similar, they are structurally very different. Synchronous registers are clocked, whereas alternating DATA/NULL transitions in PCHB are maintained via C-elements and a well-defined handshaking scheme.

A. Safety Check of Sequential PCHB Circuits

Fig. 7(a) shows the datapath connection diagram of a $4 + 2 \times 2$ PCHB MAC. (X_1, X_0) and (Y_1, Y_0) are the two bits of inputs X and Y, respectively. The product of X and Y is added with the 4-bit accumulator output A, where A_3 and A_0 are the MSB and the LSB, respectively. Each

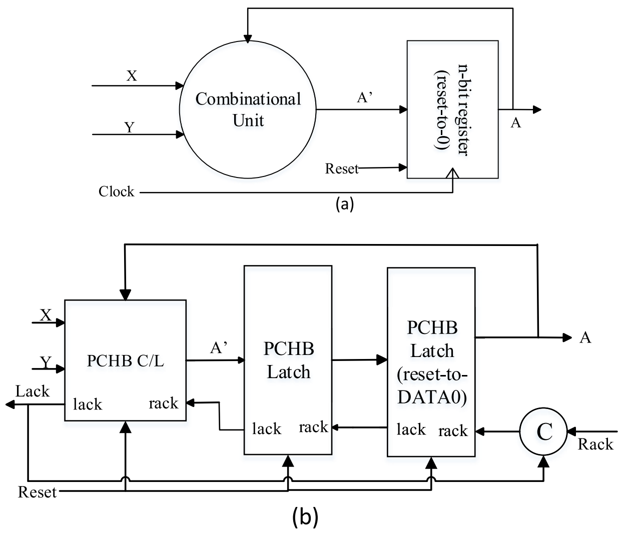
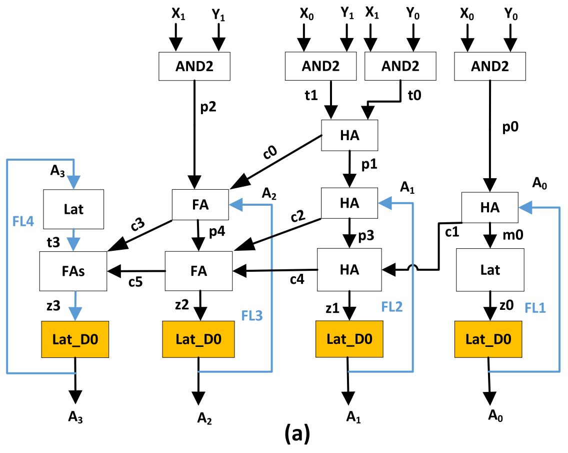
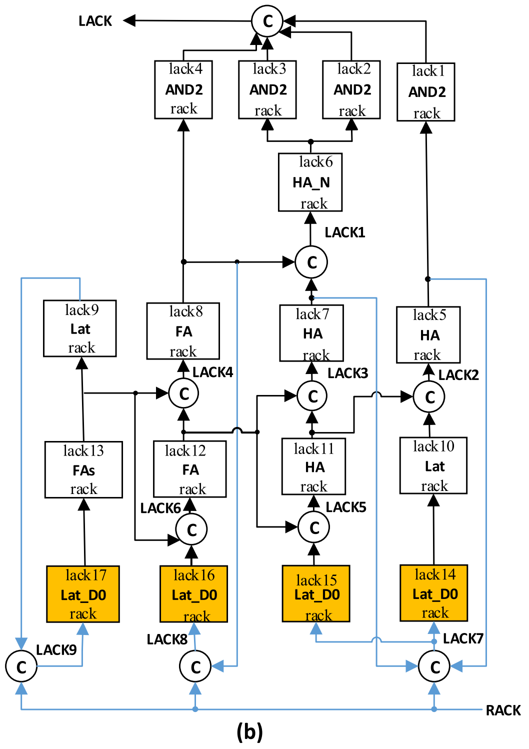


Fig. 6. (a) Synchronous MAC. (b) PCHB MAC.





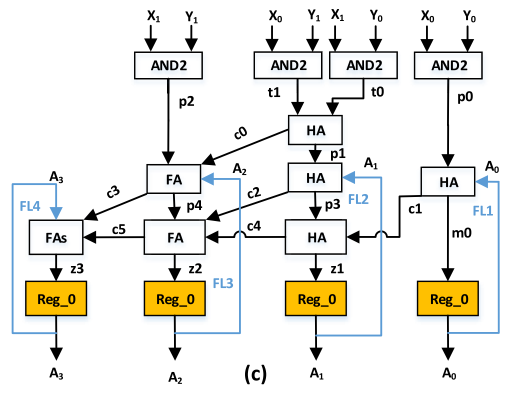


Fig. 7. 4 + 2 × 2 MAC. (a) PCHB datapath connections. (b) PCHB handshaking connections. (c) Equivalent synchronous circuit after conversion.

signal is dual-rail. *HA* and *FA* are optimized PCHB half-adder and full-adder components [24], respectively, whereas PCHB half-adders and full-adders were implemented using PCHB XOR2 and NAND2 gates for the previous multiplier circuits. The highlighted components, labeled as *Lat_D*0, are reset-to-DATA0 latches, while nonresettable latches are labeled as *Lat*. There are four FLs in Fig. 7(a), where FL1 and FL4 require an additional nonresettable latch to meet the required three latch minimum, while FL2 and FL3 do not, since these both already have two additional PCHB gates in their respective FL.

Fig. 8(a) shows the netlist of the PCHB 4 + 2 × 2 MAC, following the same structure described in Section IV-A. The first 2 lines are the circuit inputs and outputs, respectively; lines 3–13 are the PCHB C/L gates; lines 14–19 are the PCHB latches; and lines 20–29 are C-elements used in the handshaking network. Note that each of the *HA* and *FA* components contains two outputs, *sum* followed by *carry*. *FAs* is a modified *FA* component with only *sum* output.

This sequential PCHB netlist is then automatically converted into its equivalent synchronous netlist, depicted in Fig. 8(b), similar to the conversion process described in Section IV-A for combinational PCHB circuits. Each dual-rail signal is replaced with a corresponding Boolean signal and all handshaking signals and C-elements are eliminated. Procedure 2 illustrates the conversion algorithm for PCHB components (i.e., gates and latches). A PCHB component structure, *PCHB_comp*, containing all information related to individual components (i.e., gate type, data inputs, reset input, rack, lack, and data outputs) is created for each component by traversing the netlist (line 1 in Procedure 2). Each PCHB gate/component (e.g., AND gate or FA component), excluding latches, is replaced with its corresponding Boolean gate/component, which does not include a *reset* input. Every nonresettable latch, *Lat*, is eliminated by setting its output equal to its input, as those are added in FLs to avoid deadlock or to increase throughput via slack matching [25] and have no corresponding functionality in the equivalent synchronous circuit. Each resettable latch, *Lat_D*0 or *Lat_D*1, is replaced with its corresponding resettable synchronous register. Note that two adjacent resettable latches cause deadlock, as explained in Section II-A, hence Procedure 2 generates an error message if this occurs.

```
1. x0_1x0_0,x1_1x1_0,y0_1y0_0,y1_1y1_0
2. a0_1a0_0,a1_1a1_0,a2_1a2_0,a3_1a3_0
3. and2  1  x0_1x0_0,y0_1y0_0  reset  lack5  lack1  p0_1p0_0
4. and2  1  x1_1x1_0,y0_1y0_0  reset  lack6  lack2  t0_1t0_0
5. and2  1  x0_1x0_0,y1_1y1_0  reset  lack6  lack3  t1_1t1_0
6. and2  1  x1_1x1_0,y1_1y1_0  reset  lack8  lack4  p2_1p2_0
7. ha2  2  p0_1p0_0,a0_1a0_0  reset  LACK2  lack5  m0_1m0_0,c1_1c1_0
8. ha2  2  t0_1t0_0,t1_1t1_0    reset  LACK1  lack6  p1_1p1_0,c0_1c0_0
9. ha2  3  p1_1p1_0,a1_1a1_0  reset  LACK3  lack7  p3_1p3_0,c2_1c2_0
10. fa3  3  p2_1p2_0,a2_1a2_0,c0_1c0_0  reset  LACK4  lack8  p4_1p4_0,c3_1c3_0
11. ha2  4  p3_1p3_0,c1_1c1_0  reset  LACK5  lack11 z1_1z1_0,c4_1c4_0
12. fa3  5  p4_1p4_0,c2_1c2_0,c4_1c4_0  reset  LACK6  lack12  z2_1z2_0,c5_1c5_0
13. fas3  6  t3_1t3_0,c3_1c3_0,c5_1c5_0  reset  lack17  lack13  z3_1z3_0
14. lat   m0_1m0_0  reset   lack14    lack10    z0_1z0_0
15. lat_D0  z0_1z0_0    reset   LACK7    lack14    a0_1a0_0
16. lat_D0  z1_1z1_0    reset   LACK7    lack15    a1_1a1_0
17. lat_D0  z2_1z2_0    reset   LACK8    lack16    a2_1a2_0
18. lat   a3_1a3_0    reset   lack13    lack9     t3_1t3_0
19. lat_D0   z3_1z3_0    reset   LACK9    lack17    a3_1a3_0
20. C2  lack7,lack8  LACK1
21. C2  lack10,lack11  LACK2
22. C2  lack11,lack12  LACK3
23. C2  lack12,lack17  LACK4
24. C2  lack15,lack12  LACK5
25. C2  lack16,lack17  LACK6
26. C3  RACK,lack5,lack7  LACK7
27. C2  RACK,lack8  LACK8
28. C2  RACK,lack9  LACK9
29. C4  lack1,lack2,lack3,lack4  LACK
```

(a)

```
1.  x0,x1,y0,y1
2.  a0,a1,a2,a3
3.  and2  1  x0,y0  p0
4.  and2  1  x1,y0  t0
5.  and2  1  x0,y1  t1
6.  and2  1  x1,y1  p2
7.  ha2  2  p0,a0  m0,c1
8.  ha2  2  t0,t1  p1,c0
9.  ha2  3  p1,a1  p3,c2
10. fa3  3  p2,a2,c0  p4,c3
11. ha2  4  p3,c1  z1,c4
12. fa3  5  p4,c2,c4  z2,c5
13. fas3  6  a3,c3,c5  z3
14. Reg_0  m0  a0
15. Reg_0  z1  a1
16. Reg_0  z2  a2
17. Reg_0  z3  a3
```

(b)

Fig. 8. (a) PCHB $4 + 2 \times 2$ MAC netlist. (b) Equivalent converted synchronous netlist.

Procedure 2 Procedure to Generate Synchronous Circuit From PCHB Circuit

```
1:  Create PCHB_comp (PCHB_Netlist)
2:  for i←1 to num_pchb_components do
3:      if PCHB_comp(i).gate_type == Lat_D0 or Lat_D1 then
4:          if PCHB_comp(i).input connected to Lat_D0 or Lat_D1 then
5:              report error message
6:          end if
7:      else if PCHB_comp(i).gate_type == Lat then
8:          merge PCHB gates separated by PCHB_comp(i)
9:          delete PCHB_comp(i)
10:     end if
11: end for
12: for i←1 to num_pchb_components do
13:     convert_to_synchronous (PCHB_comp(i))
14: end for
```

Referring to Procedure 2, lines 2–11 delete nonresettable PCHB latches, which include lines 4–6 to check for adjacent resettable latches, such that after exiting that for loop, any remaining PCHB latch component directly corresponds to a synchronous register. Lines 12–14 then replace each PCHB component with its corresponding synchronous/Boolean component. For example, referring to Fig. 7(a), one can find that the two *Lat* components, one each in FL1 and FL4, are deleted, while all other components are replaced with their corresponding Boolean/synchronous component, resulting in the equivalent converted synchronous circuit depicted in Fig. 7(c), whose netlist is given in Fig. 8(b).

Functional equivalence checking of sequential PCHB circuits is more complicated than for combinational PCHB circuits described in Section IV-A. Sequential circuits require states and transitions between the states, such that both specification and implementation can be modeled as a TS. These TSs are then checked for equivalence using the theory of WEB refinement [14], whose detailed description can be found in [26]. Here, we provide a brief overview of the three key concepts of WEB refinement. The first is the refinement map, which is a function that maps implementation states to specification states and is used to bridge the abstraction gap between implementation and specification. The second concept is stuttering, which is a phenomenon where multiple but finite steps of the implementation can match a single transition of the specification. The third concept is that of a rank function, which is a function from implementation states to natural numbers, whose values decrease every time the implementation stutters. Rank functions are used to distinguish stutter from deadlock (which essentially amounts to infinite stutter). However, since the registers of the specification sequential circuit, *SPEC_SEQ*, and those of the equivalent synchronous circuit, *PCHB_SEQ*, automatically generated from the PCHB circuit as described, have a one-to-one mapping, there is no stuttering. Also, because of this one-to-one register mapping, the refinement map function is just a projection of each implementation register onto its corresponding specification register. Hence, the correctness proof obligation can be reduced to Proof Obligation 1.

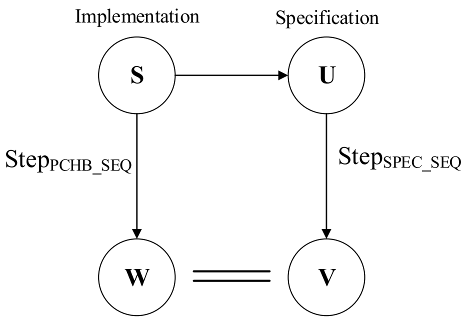


Fig. 9. Formulation of proof obligation to check the equivalence of PCHB_SEQ and SPEC_SEQ.

Fig. 9 is used to explain the proof obligation. S is a state of PCHB_SEQ and U is a SPEC_SEQ state, which includes a

$$\text{Proof Obligation}\,\mathbf{1} : \{\forall s : s \in S_{\text{PCHB\_SEQ}} :: [u = \text{Reg\_Proj}(s) \wedge w = \text{Step}_{\text{PCHB\_SEQ}}(s) \wedge \ v = \text{Step}_{\text{SPEC\_SEQ}}(u)] \Rightarrow w = v\}$$

projection of register values of state S. $Step_{PCHB\_SEQ}$ and $Step_{SPEC\_SEQ}$ are two functions that step *PCHB_SEQ* and *SPEC_SEQ* once, respectively. W and V are the next states of S and U, respectively. The proof obligation states that *PCHB_SEQ* and *SPEC_SEQ* are equivalent if the projection values of all registers in state W are equal to the values of their corresponding registers in state V. The proof obligation is encoded in SMT-LIB, and checked using an SMT solver. Note that such equivalence verification problems could be encoded as model checking problems; however, the model checking algorithms do not scale well due to state-space explosion, which is a well-known issue with model checking when used in the context of equivalence verification.

Additionally, since *PCHB_SEQ* and *SPEC_SEQ* have a one-to-one register mapping, the register reset values are compared to ensure that the two circuits have the same initial reset state. If a one-to-one register mapping does not exist, this is an error, since the method presented in [17] for PCHB circuit synthesis from a synchronous specification guarantees a one-to-one register mapping with the synchronous specification. For example, accidentally using a nonresettable PCHB latch instead of a resettable latch, as was supposed to be used, would result in one fewer register in *PCHB_SEQ*. Note that this error would not necessarily cause the PCHB circuit to deadlock (e.g., a loop that was supposed to contain more than 1 DATA token).

B. Liveness and Handshaking Correctness Check

Fig. 7(b) shows the handshaking connections between components for the $4+2\times 2$ PCHB MAC, where *RACK* and *LACK* are the external *request* and *acknowledge* signals, respectively. The same handshaking check algorithm detailed in Section IV-B is utilized to check the handshaking connections, with the only difference being the level restriction check for full-word completion (i.e., each gate that is in *comp_fanin*(k) and not in *fanout*(k) must be from the immediate subsequent level of gate/input k), since this level restriction no longer holds due to datapath feedback. Therefore, we do not require *level* for PCHB latch components and *level* for other PCHB gates is ignored for sequential circuits. To demonstrate this, we have utilized partial full-word completion in Fig. 7(b), by combining the request generation of the two LSB latches. The *comp_fanin* list of the 2 LSB Reg_D0 components will contain one additional signal each, *lack*7 for the LSB latch in FL1 and *lack*5 for the next LSB latch in FL2. This is not an error but may slow down the circuit. The handshaking algorithm generates a warning under such a scenario, highlighting the additional signal for further inspection. Following the handshaking check, the method developed by Shih *et al.* [16] for deadlock verification [16] is applied, as mentioned at the beginning of Section V. This method, along with our handshaking correctness check, guarantees liveness for sequential PCHB circuits.

VI. Detection of All Possible Faults

Section III enumerates the faults that could occur in a PCHB circuit synthesized using the method presented in [17] and proves that the 18 faults listed comprise all possible faults. In the following, we show how the proposed methodology detects all 18 of these faults. Cases 1–8 correspond to datapath faults, which are detected in our safety check; Cases 9–17 correspond to handshaking faults, which are detected in our liveness check; and Case 18 corresponds to electrical faults, which can occur either in the datapath or in handshaking circuitry, and are detected in our safety or liveness check, respectively.

Case 1: Faulty data connection, Case 2: Swapped dual-rail connection, and Case 6: Incorrect logic implementation would all result in functional inequivalence between implementation and specification and would be detected by the SMT solver. Case 3: Rails from different signals, Case 4: Rail Duplication, Case 5: Handshaking signal connected to a data signal, and Case 7: Non-PCHB gate in datapath would all be detected in the PCHB-to-Boolean netlist conversion algorithm, described in Sections IV-A and V-A. For Case 8: Incorrect reset, the external *reset* input not being connected to all PCHB gates' *reset* input would be detected in the PCHB-to-Boolean netlist conversion algorithm, while an incorrect reset value would be detected in the register reset value comparison check described at the end of Section V-A.

Case 9: Insufficient latches in an FL would be detected by the verification procedure proposed by Shih *et al.* [16], which follows our handshaking check, as described in Section V-B. Cases 10–13: Missing handshaking signal, Additional handshaking signal, External Lack error, and External Rack error would all be detected in the handshaking correctness check, described in Sections IV-B and V-B. Cases 14–17: Non-C-element in handshaking circuitry, Data signal input to C-element, Data signal input to PCHB gate Rack input, and C-element structure feedback would all be detected by the algorithm that generates the *fanout* and *comp_fanin* for each PCHB gate and external data input, described in Section IV-B.

Case 18: Shorted output would be detected by the PCHB-to-Boolean netlist conversion algorithm for shorted PCHB gate data outputs and by the *fanout*/*comp_fanin* generation algorithm for shorted PCHB *Lack* or C-element outputs. Hence, our proposed methodology will detect all 18 fault cases, which in Section III were proved to comprise all possible PCHB circuit faults; therefore, our proposed methodology guarantees full functional correctness for any PCHB circuit synthesized using the method presented in [17].

VII. Results

We have demonstrated the proposed methodology by verifying several different sized MACs and ISCAS sequential circuit benchmarks [27], as shown in Table II, which lists the verification time for each circuit. MAC circuits could be considered a special case since they only contain noninteracting FLs; however, the ISCAS benchmarks are more general and contain various interacting feedback paths. PCHB MAC circuits are complex sequential pipeline structures, with each PCHB component acting as a state holding element. For example, an optimized $20+10\times 10$ PCHB MAC requires 18 162 transistors, whereas its synchronous counterpart only requires 4710 transistors, demonstrating the substantially increased complexity of PCHB circuits versus their synchronous equivalent. The PCHB-to-Boolean netlist conversion time and the time to generate *fanout* and *comp_fanin* for each PCHB gate were negligible compared to the time required to perform

TABLE II
VERIFICATION RESULTS FOR VARIOUS SEQUENTIAL PCHB CIRCUITS

PCHB CIRCUITS	VERIFICATION TIME (IN SEC)			PCHB CIRCUITS	VERIFICATION TIME (IN SEC)		
	SAFETY CHECK	HANDSHAKING CHECK	TOTAL TIME		SAFETY CHECK	HANDSHAKING CHECK	TOTAL TIME
4+2x2 MAC	0.01	0.0053	0.0153	20+10x10 MAC- B6	0.23 (B)	1.7862	2.0162
8+4x4 MAC	0.07	0.0278	0.0978	20+10x10 MAC- B7	0.0109 (B)	1.7862	1.7971
16+8x8 MAC	12.06	0.7893	12.8493	20+10x10 MAC- B8i	0.17 (B)	1.7862	1.9562
20+10x10 MAC	813.03	1.7862	814.8162	20+10x10 MAC- B8ii	0.0206(B)	1.7862	1.8068
ISCAS s27	0.01	0.0010	0.0110	20+10x10 MAC- B10	813.03	3.0563 (B)	816.0863
ISCAS s208	0.17	0.0378	0.2078	20+10x10 MAC- B11	813.03	2.5998 (B)	815.6298
ISCAS s298	0.23	0.0459	0.2759	20+10x10 MAC- B12	813.03	1.8368 (B)	814.8668
ISCAS s444	0.76	0.1008	0.8608	20+10x10 MAC- B13	813.03	1.8989 (B)	814.9289
20+10x10 MAC- B1	0.11 (B)	1.7862	1.8962	20+10x10 MAC- B14	813.03	0.9063 (B)	813.9363
20+10x10 MAC- B2	0.14 (B)	1.7862	1.9262	20+10x10 MAC- B15	813.03	1.1856 (B)	814.2156
20+10x10 MAC- B3	0.0324 (B)	1.7862	1.8186	20+10x10 MAC- B16	813.03	1.1121 (B)	814.1421
20+10x10 MAC- B4	0.0175 (B)	1.7862	1.8037	20+10x10 MAC- B17	813.03	0.8555(B)	813.8855
20+10x10 MAC- B5	0.0154 (B)	1.7862	1.8016	20+10x10 MAC- B18	0.1333 (B)	1.7862	1.9195

the safety check by the Z3 SMT solver [22]; additionally, the handshaking check was also negligible, as seen in Table II. Note that Table II does not include the deadlock verification times, as this algorithm was developed by Shih *et al.* [16].

To check our methodology, we injected bugs into the $20 + 10 \times 10$ MAC, corresponding to all 18 fault cases, except for Case 9: Insufficient latches in an FL, as the method to detect this fault was already developed by Shih *et al.* [16]. The –Bn multipliers in Table II are the buggy circuits, where *n* corresponds to a *Case n* bug and (B) in either the safety check or handshaking check column denotes the check that detected the bug. -B8i corresponds to a latch reset to an incorrect value, while B8ii corresponds to a signal other than the external *reset* that is connected to a PCHB gate's *reset* input.

In every case, the proposed methodology detected the bug and produced a counterexample and/or descriptive error message, in order to assist in identifying where the error occurred. Verification was performed using the same computer described in Section IV-C.

VIII. CONCLUSION

Formal verification methodologies for various QDI paradigms have long been desired in the industry. Equivalence checking is one of the popular formal verification approaches that is highly scalable and efficient. This article proposes the first-ever equivalence checking based methodology to formally verify a QDI PCHB circuit against its synchronous specification. Our approach is scalable, fast, and applicable to any combinational or sequential PCHB circuit synthesized using the method presented in [17]. Additionally, our proposed method is also applicable to the optimized hybrid PCHB-WCHB circuits proposed in [17], since PCHB and weak-conditioned half buffer (WCHB) gates have the same interface and compatible handshaking mechanisms, such that it makes no difference to our verification approach if the transistor level gate structure is implemented in PCHB or WCHB fashion. Note that WCHB gates need to be designed properly to ensure weak conditioning [7] of their inputs (i.e., all gate outputs cannot become DATA/NULL until all gate inputs are DATA/NULL), which is required for delay-insensitivity. However, our verification methodology does not check this; as noted at the beginning of Section III, we assume that the transistor-level implementation of every PCHB/WCHB gate is correct, which is a typical assumption for gate-level verification methods. Verifying the transistor-level implementation of each gate, including correct weak-conditioning of WCHB gates, is a different problem. Since each PCHB/WCHB gate is small enough, this can be easily done via exhaustive simulation, which is a common method for verifying circuit primitives.

Also note that the proposed methodology can be used to check the equivalence of two PCHB circuits by applying the conversion technique to both PCHB circuits to obtain two corresponding synchronous circuits, verifying these two synchronous circuits against each other and performing the liveness and handshaking correctness checks on both PCHB circuits. Additionally, the proposed method can be utilized for PCHB circuits with more than one external *Lack* and/or *Rack*; but for this, the handshaking check tool requires additional information to specify for each external *Lack* output, which external data input(s) it acknowledges, and for each external *Rack* input, which external data output(s) it requests. Furthermore, our proposed approach is also applicable to cases where there is no one-to-one register mapping between the synchronous specification and the PCHB implementation. In this case, equivalence between the synchronous specification, SPEC_SEQ, and the synchronous circuit automatically generated from the PCHB circuit using our method, PCHB_SEQ, must be verified using a sequential equivalence checker, such as Cadence's JasperGold [28].

We have demonstrated that the proposed approach detects all possible faults, thereby guaranteeing correctness, and have proved that the 18 fault cases presented herein comprise all faults that could possibly occur in a QDI circuit comprised entirely of PCHB/WCHB gates and C-elements. Future work includes extending the proposed methodology to PCHB/WCHB circuits that include conditional communication, where DATA does not necessarily flow through every path and every loop every cycle [7] (e.g., flowing DATA only through a single ALU path to perform the selected operation, rather than through all ALU paths, as in a Boolean circuit, which can yield substantial power savings and improve the average performance).

References

[1] S. C. Smith and J. Di, "The future of asynchronous logic," in *VLSI Circuits for Emerging Applications*. Boca Raton, FL, USA: CRC Press, 2014.

[2] (2013). *International Technology Roadmap For Semiconductors*. Accessed: Jun. 2019. [Online]. Available: https://www.dropbox.com/sh/6xq737bg6pww9gq/AACQWcdHLffUeVloszVY6Bkla?dl=0&preview=2013_Design-v3.pdf

[3] *IEEE International Roadmap for Devices and Systems, Medical Devices Market Driver*. Accessed: Jun. 2019. [Online]. Available: https://irds.ieee.org/editions/2018/medical-devices-market-driver

[4] A. J. Martin and M. Nystrom, "Asynchronous techniques for system-on-chip design," *Proc. IEEE*, vol. 94, no. 6, pp. 1089–1120, Jun. 2006.

[5] A. A. Sakib, S. C. Smith, and S. K. Srinivasan, "An equivalence verification methodology for combinational asynchronous PCHB circuits," in *Proc. IEEE 61st Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2018, pp. 767–770.

[6] D. E. Muller, "Asynchronous logics and application to information processing," in *Switching Theory in Space Technology*. Stanford, CA, USA: Stanford Univ. Press, 1963, pp. 289–297.

[7] P. Beerel, R. Ozdag, and M. Ferretti, *A Designer's Guide to Asynchronous VLSI*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[8] S. C. Smith and J. Di, *Designing Asynchronous Circuits Using Null Convention Logic (NCL)* (Synthesis Lectures on Digital Circuits and Systems). San Rafael, CA, USA: Morgan Claypool, 2009.

[9] C. J. Myers, *Asynchronous Circuit Design*. New York, NY, USA: Wiley, 2001.

[10] S. K. Srinivasan and R. S. Katti, "Desynchronization: Design for verification," in *Proc. FMCAD*, P. Bjesse and A. Slobodová, Eds. Mountain View, CA, USA: FMCAD Inc., 2011, pp. 215–222.

[11] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.

[12] L. Neukom, O. Roig, K. Fant, A. Kondratyev, and A. Taubin, "Checking delay-insensitivity: 104 gates and beyond," in *Proc. 8th Int. Symp. Asynchron. Circuits Syst.*, 2002, p. 149.

[13] V. M. Wijayasekara, S. K. Srinivasan, and S. C. Smith, "Equivalence verification for NULL convention Logic (NCL) circuits," in *Proc. ICCD*, Oct. 2014, pp. 195–201.

[14] P. Manolios, "Correctness of pipelined machines," in *Formal Methods in Computer-Aided Design* (Lecture Notes in Computer Science), vol. 1954, W. A. Hunt, Jr., and S. D. Johnson, Eds., New York, NY, USA: Springer-Verlag, 2000, pp. 161–178.

[15] S. J. Longfield and R. Manohar, "Inverting Martin synthesis for verification," in *Proc. IEEE ASYNC*, May 2013, pp. 150–157.

[16] C. Shih, Y. Lai, and J. R. Jiang, "SPOCK: Static performance analysis and deadlock verification for efficient asynchronous circuit synthesis," in *Proc. ICCAD*, Nov. 2015, pp. 442–449.

[17] C. Chuang, Y. Lai, and J. R. Jiang, "Synthesis of PCHB-WCHB hybrid quasi-delay insensitive circuits," in *Proc. 51st ACM/EDAC/IEEE Design Automat. Conf. (DAC)*, Jun. 2014, pp. 1–6.

[18] A. Saifhashemi, H.-H. Huang, P. Bhalerao, and P. A. Beerel, "Logical equivalence checking of asynchronous circuits using commercial tools," in *Proc. DATE*, 2015, pp. 1563–1566.

[19] C. A. R. Hoare, *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, 1985.

[20] A. A. Sakib, S. C. Smith, and S. K. Srinivasan, "Formal modeling and verification for pre-charge half buffer gates and circuits," in *Proc. IEEE MWSCAS*, Aug. 2017, pp. 519–522.

[21] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[22] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science), vol. 4963, C. R. Ramakrishnan and J. Rehof, Eds., Springer, 2008, pp. 337–340.

[23] D. Bryan. *ISCAS Benchmark Circuits and Netlist Format*. Accessed: Dec. 2018. [Online]. Available: https://ddd.fit.cvut.cz/prj/Benchmarks/iscas85.pdf

[24] A. J. Martin, A. Lines, R. Manohar, U. Cummings, and M. Nystrom, "Pipelined asynchronous processing," U.S. Patent 6 658 550 B2, Dec. 2, 2003.

[25] P. Prakash and A. J. Martin, "Slack matching quasi delay-insensitive circuits," in *Proc. 12th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, Mar. 2006, p. 204.

[26] P. Manolios, "Mechanical verification of reactive systems," Ph.D. dissertation, Dept. Comput. Sci., Univ. of Texas at Austin, Austin, TX, USA, Aug. 2001.

[27] *ISCAS Benchmark Circuits*. Accessed: Dec. 2018. [Online]. Available: http://www.pld.ttu.ee/ maksim/benchmarks/iscas89/bench/

[28] *JasperGold Sequential Equivalence Checking App*. Accessed: Jun. 2019. [Online]. Available: https://www.cadence.com/content/cadence-www/global/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/jaspergold-sequential-equivalence-checking-app.html

Ashiq A. Sakib (M'14) received the B.Tech. degree from the Department of Electronics and Communication Engineering, West Bengal University of Technology, Kolkata, India, in 2013, and the Ph.D. degree from the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA, in 2019.

He is currently an Assistant Professor at the Department of Electrical and Computer Engineering, Florida Polytechnic University, Lakeland, FL, USA. His current research interests include formal verification of quasi-delay insensitive (QDI) asynchronous design paradigms, and design of advanced and ultralow-power digital circuits.

Dr. Sakib is a member of IEEE-HKN and Phi Kappa Phi.

Scott C. Smith (M'95–SM'06) received the B.S. degree in electrical engineering and the B.S. degree in computer engineering and the M.S. degree in electrical engineering from the University of Missouri at Columbia, Columbia, MO, USA, in 1996 and 1998, respectively, and the Ph.D. degree in computer engineering from the University of Central Florida, Orlando, FL, USA, in 2001.

He is currently a Professor and the Chair of the Department of Electrical Engineering and Computer Science, Texas A&M University Kingsville, Kingsville, TX, USA. He has authored 26 refereed journal articles, 71 refereed conference papers, one coauthored book, one co-edited book, and four additional book chapters. He holds eight U.S. patents. His current research interests include asynchronous logic, NULL convention logic, computer architecture, embedded systems, digital logic, field-programmable gate arrays, CAD tools for digital design, computer arithmetic, VHDL, VLSI, secure/trustable hardware, wireless sensor networks, robotics, and cyber physical systems.

Dr. Smith is a member of the National Academy of Inventors, Sigma Xi, IEEE-HKN, and Tau Beta Pi.

Sudarshan K. Srinivasan (M'07) received the B.E. degree in electrical and electronics engineering from the University of Madras, Chennai, India, in 2001, and the M.S. and Ph.D. degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2003 and 2007, respectively.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND, USA. His current research interests include the development and application of formal verification techniques to synchronous and asynchronous digital systems, real-time systems, embedded software, cyber-physical systems, computer architecture, low power design, secure/trustworthy computing, and pedagogy.

Dr. Srinivasan is an Associate Editor for *IET Computer and Digital Techniques*.