Equivalence Verification for NULL Convention Logic (NCL) Circuits

Vidura M. Wijayasekara, Sudarshan K. Srinivasan, and Scott C. Smith
Department of Electrical and Computer Engineering
North Dakota State University
Fargo ND 58104
Email: vidura.wijaysekara@my.ndsu.edu, sudarshan.srinivasan@ndsu.edu, and scott.smith.1@ndsu.edu

***Abstract*—NULL Convention Logic (NCL) circuits are asynchronous circuits and find application in SoC design due to their delay-insensitive nature, which allows ease in resolution of timing issues in IP component reuse for SoC. NCL components are typically synthesized from synchronous circuits. For any design paradigm to be feasible, verification is an important factor. We present a formal verification methodology for checking equivalence of NCL circuits against their synchronous parent circuits. The methodology includes a procedure that computes the reachable states of NCL sequential circuits and a refinement mapping function that can be used to map NCL circuit states onto synchronous circuit states. The methodology is demonstrated by verifying the correctness of several NCL circuits.**

Keywords*—asynchronous/NCL circuits, equivalence checking, refinement.

I. Introduction

The synchronous design paradigm is facing many challenges as fabrication technologies scale down to keep up with the demand for high-performance low-power circuits. Such challenges include dominant wire delays that cannot be accurately determined until the later stages of the design cycle, and managing clock skew. In addition to these design challenges, high power consumption and noise are also inherent challenges in the synchronous design domain that are becoming increasingly difficult to manage. Asynchronous circuits in general have properties that could potentially solve many of the issues that are becoming dominant in the synchronous domain [1]. Therefore, in recent years, a lot of research work has been done to develop design methodologies and processes to integrate asynchronous circuits in commercial systems [2][3].

NULL Convention Logic (NCL) circuits are *delay-insensitive* (DI) asynchronous circuits. In comparison to synchronous circuits, DI circuits have lower power consumption, less noise, and lower electro-magnetic interference. The delay-insensitive nature of DI circuits can also be exploited to ease component reuse in complex SoC designs, especially for SoCs that employ multiple clocks [1]. DI asynchronous circuits are correct-by-construction designs that do not require extensive timing analysis as needed by *bounded-delay* asynchronous circuits (micropipelines). Also, performance of delay-insensitive asynchronous circuits is close to average case performance. Bounded-delay circuits, on the other hand, yield worst case performance [1].

Functional testing and verification for asynchronous circuits is a challenging problem because the control components of asynchronous designs are highly non-deterministic and exhibit a prohibitively large state space. Therefore, exhaustive testing is very hard. Formal verification techniques have been successfully integrated in synchronous commercial design cycles and have shown to significantly improve functional coverage and find corner case bugs. In this paper, we propose a formal verification methodology for NCL circuits.

The proposed verification methodology is developed in the context of NCL synthesis. A recent trend to overcome the design challenges for asynchronous circuits (such as lack of CAD tools and design complexity) is to develop tools to synthesize asynchronous circuits from synchronous circuits. Tools have been developed to automate the synthesis of NCL circuits from synchronous circuits [2]. The goal of our verification methodology is to check the equivalence of the synthesized NCL circuit against the synchronous circuit that was input to the synthesis tool, which we call the parent synchronous circuit. Why would such an equivalence verification methodology be useful in practice? Consider the integration of IP blocks in a multi-rate SoC environment. Such an integration in the synchronous domain is a challenging task. The designer may opt to use a technology in the DI design paradigm such as NCL circuits to integrate the multi-rate blocks.

After generating the NCL design, the resulting NCL circuit will have undergone a considerable transformation w.r.t. the original synchronous circuit. The designer cannot assume that the resulting NCL circuit will not have bugs. The designer will instead have to expend time and resources to verify the NCL circuit. Even though the NCL circuits are correct-by-construction, the synthesis tools that generate NCL circuits from the synchronous circuit may have bugs. Also, there maybe other sources of bugs such as manual tinkering of the NCL circuit after synthesis. The commercial design process will definitely require functional verification of the NCL circuit used in the SOC. *The full verification of the NCL circuit cannot be bypassed because it is correct-by-construction or by performing some checks on the connections of the NCL circuit.* Our equivalence verification methodology comes to the aid here as it is targeted at checking the correctness of the NCL circuit against the parent synchronous circuit. Equivalence

This work was funded in part by NSF grants CCF-1117164 and CCF-1242043.

978-1-4799-6492-5/14/$31.00 ©2014 IEEE

checking technology has been very effective and useful in the synchronous domain.

The rest of the paper is organized as followed. First, we describe prior work related to verification of asynchronous circuits in Section II. An introduction to NCL circuits is given in Section III. Equivalence verification for combinational and sequential NCL circuits are described in Sections IV and V, respectively. Experimental results are presented in Section VI. Finally, we conclude in Section VII.

II. Related Work

Related work on verification technology for asynchronous circuits can be classified as property checking approaches [4][5] or methods based on trace theory [6]. The trace theory approaches target the verification of gate-level asynchronous circuits. In trace theory based approaches, the circuit is modeled as a petri-net. The correctness property is also modeled as a petri-net. In contrast, our equivalence verification approach is based on the theory of Well-Founded Equivalence Bisimulation (WEB) refinement. In WEB refinement, both the specification and implementation are modeled as transition systems. We are not aware of any trace theory based methods that have been developed for equivalence verification of synchronous and NCL circuits. Approaches based on property checking can be used for NCL circuits, but, are cumbersome because a large number of properties are required and also the properties themselves can be hard to write leading to erroneous specifications [6].

Loewenstein [7] verified some properties of a counter-flow pipeline using the HOL theorem prover. Counter-flow pipelines are asynchronous in nature with results flowing in the pipeline in a direction opposite to that of instruction flow. The NCL circuits we verify do not use the counter-flow mechanism. Also, our correctness proofs are based on the use of decision procedures and are highly automated.

Verbeek [8] verified deadlock freedom of DI circuits compiled from Click library (a DI primitives library) using SAT/SMT instances of the circuit. We verify NCL circuits, which is a different DI design paradigm from the technology implemented in the Click library. Also, we verify safety, i.e., we verify that the implementation (NCL circuit) behaves correctly as given by the specification (synchronous circuit).

A method to check the delay insensitive property of combinational NCL circuits is presented in [9]. Our method verifies the functional equivalence of both combinational and sequential NCL circuits against the parent synchronous circuit. A method to verify desynchronized pipelined circuits against ISA-type specifications using refinement is presented in [10]. Desynchronized circuits are bounded-delay circuits. In contrast, NCL circuits are delay-insensitive. Hence, their intrinsic properties are different and require domain-specific verification methods. Also, our specifications are the parent synchronous circuit used for synthesis of NCL circuits. We do not use an ISA-type specification. To our knowledge, there is no prior work that uses formal methods to verify the functional equivalence of NCL circuits against their synchronous parent circuits.

Cortadella et al. [11] have used flow equivalence (FE) to prove the correctness of their desynchronization method; and FE is well suited for this purpose. However, they have not demonstrated verification based on FE. Why do we use refinement instead of FE? Refinement is a more general notion. For example, one requirement of FE is that the specification and implementation should have the same set of registers. This requirement is not satisfied when comparing NCL circuits and their synchronous parents.

III. Background: NCL Circuits

NCL circuits are four phase (i.e., dual-rail) DI logic systems. In dual-rail encoding every Boolean value is represented using two rails/wires, D^0 and D^1. Boolean zero and one are represented by setting the value of D^0D^1 to 10 and 01, respectively. Unavailability of valid data (i.e., NULL) is represented by value 00. The two rails cannot be asserted at the same time, which is an illegal state.

NCL has 27 threshold gates that implement all functions of four or fewer Boolean variables. A THmn gate, where m $\leq$ n, has n inputs and a threshold of m. When m or more inputs are asserted, the output of the gate is asserted. The output is deasserted only when all inputs are deasserted. For all other input combinations, the output of the gate does not change. Hence, all 27 threshold gates are designed with hysteresis and hold a state. A THnn gate is equivalent to an n-input C-element. A TH1n gate acts as an n-input OR gate. Data propagates through an NCL circuit as wavefronts alternating between DATA and NULL. The NULL wavefront (spacer) brings the output of all the threshold gates to logic zero in preparation for the next set of DATA inputs.

Sequential circuits are implemented using NCL registers. A single-bit dual-rail NCL register has two data input rails (I^0, I^1), two data output rails (O^0, O^1), a request signal (K_i) as a control input, and an acknowledge signal (K_o) as a control output. The register receives requests from destination registers through K_i. When K_i is asserted (*rfd*), DATA is requested, and when K_i is deasserted (*rfn*), NULL is requested. When K_o (the acknowledge signal) is deasserted, the register acknowledges that DATA is received and requests for NULL from the source registers. When K_o is asserted, the register acknowledges that NULL is received and requests for DATA. When a source sends data to multiple destinations, the K_o signals output from the destination registers are combined in a completion component to form a single K_o signal, which is the K_i input to the source register. Completion component synchronizes its input K_o signals. The NCL register latches in the input and sets K_o to *rfd*/*rfn* when input is NULL/DATA and K_i is *rfn*/*rfd*, respectively. More details on NCL circuits can be found in [1].

IV. Equivalence Verification for Combinational NCL Circuits

We first describe the equivalence verification of combinational NCL circuits. Without loss of generality, consider an NCL circuit X with i dual-rail inputs, g NCL gates, and o dual-rail outputs. A subset of the gates form the dual-rail outputs of the circuit. Therefore, we have $2 * o \leq g$. x_1^{I0}, ..., x_i^{I0} denote the i D^0 inputs of X. x_1^{I1}, ..., x_i^{I1} denote the i D^1 inputs of X. x_1^{O0}, ..., x_o^{O0} denote the o D^0 outputs of X. x_1^{O1}, ..., x_o^{O1} denote the o D^1 outputs of X. x_1^G, ..., x_g^G denote the g gate values of

X. An NCL circuit is in the NULL state if all gate outputs (states) are deasserted as given by the following equation.

$$\bigvee_{j=1}^{g} x_j^G = 0$$

If the inputs or outputs of the circuit have valid DATA, then for each dual-rail wire, the D^0 wire should be the negation of the D^1 wire. D^1 and D^0 both cannot be either 0 or 1. The condition that the inputs have valid DATA is given below:

$$\bigwedge_{j=1}^{i} x_j^{I0} \oplus x_j^{I1} = 1$$

NCL circuits are operated as follows. First, a NULL wave is propagated through the circuit, which deasserts all the inputs. That the outputs of all gates are deasserted when a NULL wavefront is propagated, is a pre-condition for the DATA wave to work correctly. The proof obligation to be checked to verify that an NCL circuit satisfies the above requirement is given below.

Proof Obligation 1.

$$\Big\langle \forall x_1^{I0},...,x_i^{I0},x_1^{I1},...,x_i^{I1},x_1^G,...,x_g^G \in \{0,1\} :: \langle y_1^G,...,y_g^G\rangle = nclstep(x_1^{I0},...,x_i^{I0},x_1^{I1},...,x_i^{I1},x_1^G,...,x_g^G) \wedge \neg(\bigvee_{j=1}^{i} x_j^{I0}) \wedge \neg(\bigvee_{j=1}^{i} x_j^{I1}) \implies \neg(\bigvee_{j=1}^{g} y_j^G)\Big\rangle$$

In the above, *nclstep*() corresponds to a single step of the circuit and is modeled as a function that takes the circuit inputs and the current state of the gates in the circuit as input. The *nclstep*() function outputs the values of the next state of the gates in the circuit.

A combinational NCL circuit is correct w.r.t. its synchronous counterpart, if for all combinations of valid data inputs, the outputs of both circuits are the same. However, inputs to synchronous circuits are Boolean, and inputs to NCL circuits are dual-rail. We also encounter the same issue when checking the equality of outputs. To map valid dual-rail DATA to Boolean values, we exploit the fact that D^1 values represent the corresponding Boolean values, and D^0 values are always the negation of the corresponding D^1 values. Also, the NCL circuit operates under the assumption that the DATA wave is preceded by a NULL wave. Therefore, all the gate outputs in the circuit are deasserted. If *syncstep*() corresponds to a single step of the synchronous circuit, the next proof obligation gives the equivalence correctness condition of a combinational NCL circuit against its synchronous counterpart.

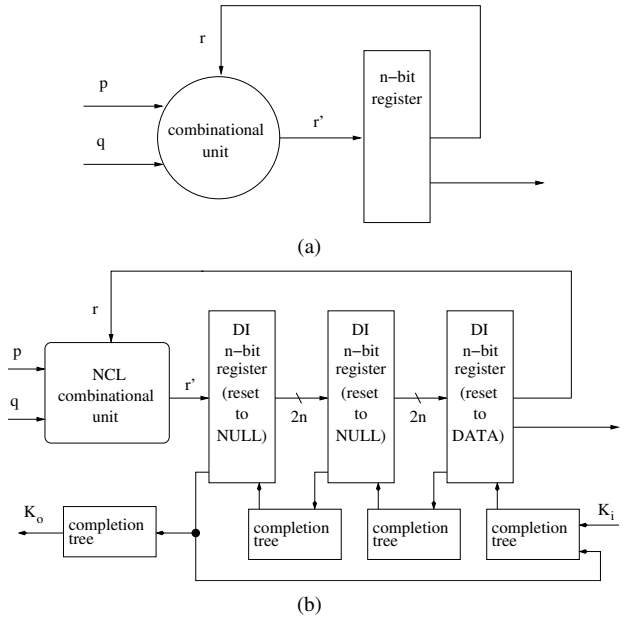

Fig. 1. Multiplier accumulator circuit (MAC): (a) synchronous, (b) NCL.

Proof Obligation 2.

$$\Big\langle \forall x_1^{I0},...,x_i^{I0},x_1^{I1},...,x_i^{I1},x_1^G,...,x_g^G \in \{0,1\} :: \langle y_1^G,...,y_g^G\rangle = nclstep(x_1^{I0},...,x_i^{I0},x_1^{I1},...,x_i^{I1},x_1^G,...,x_g^G) \wedge \neg(\bigvee_{j=1}^{g} x_j^G) \wedge \bigwedge_{j=1}^{i} x_j^{I0} \oplus x_j^{I1} \wedge \langle z_1,...,z_o\rangle = syncstep(x_1^{I1},...,x_i^{I1}) \implies \bigwedge_{j=1}^{o} z_j = y_j^{O1} \wedge \bigwedge_{j=1}^{o} y_j^{O0} \oplus y_j^{O1}\Big\rangle$$

Above, $z_1,...,z_o$ represent the outputs of the synchronous circuit. Verification of proof obligations 1 and 2 guarantee the correctness of an NCL combinational circuit w.r.t. its synchronous counterpart. Verification is performed by modeling the circuits in the SMT2-LIB modeling language. An SMT solver can be used to check the proof obligations. We use the Z3 SMT solver for verification.

V. Equivalence Verification for Sequential NCL Circuits

In this section, we present a methodology for verification of sequential NCL circuits. The methodology is illustrated using the example shown in Fig 1. Fig 1.(a) is the circuit for a synchronous multiplier and accumulate (MAC) unit. The circuit takes two 4-bit inputs p and q, and computes $r' \leftarrow r + pq$. Fig 1.(b) shows an NCL version of the MAC unit synthesized from the synchronous circuit in Fig 1.(a). Any feedback loop in an NCL circuit requires at least 3 NCL registers to ensure that the loop does not get deadlocked [1]. Other than the inclusion of two additional registers, the circuits look similar from the figure. However, there are many other differences. The data path (as mentioned earlier) is dual-rail in the NCL circuit, and the MAC combinational unit is implemented using threshold gates. Also, the registers in the

synchronous circuit are clocked, whereas progress in the NCL circuit is achieved using the movement of alternate DATA and NULL waves that is achieved using completion logic communication between adjacent NCL registers.

There are many states the NCL circuit can be in and there is not a straightforward relationship between the synchronous circuit and its NCL counterpart. Hence, we need a general theory of equivalence to relate and verify sequential NCL circuits against their synchronous counterparts.

A. WEB Refinement

We use the notion of Well-Founded Equivalence Bisimulation (WEB) refinement for the equivalence verification problem. A formal and detailed description of WEB refinement is provided in [12][13]. Here, we provide a brief overview of the key features of WEB refinement relevant to the problem at hand. WEB refinement is a notion of equivalence that can be used to check if an implementation system satisfies its specification system, even if the implementation and specification are defined at very disparate levels of abstraction. We consider the NCL circuit to be verified as the implementation, and the synchronous parent circuit as the specification.

In the context of refinement, digital systems are modeled as transition systems (TSs). A TS is a three tuple and includes the set of states of the system, a transition relation that defines the state transitions of the system, and a labeling function that defines what is observable in each state. The behaviors of a system modeled as a TS are defined using the notion of paths. A *path* in a TS is a sequence of states, say s^0,s^1,s^2, ..., where s^0 is the initial state and s^0 transitions to s^1, s^1 transitions to s^2, and so on. An infinite sequence of such states is a *full path*. The behaviors of a system is the set of all full paths in the TS of the system.

The implementation behaves correctly as given by the specification, if every behavior of the implementation is matched by a behavior of the specification and vice versa. However, the implementation and specification may not have the same timing behavior. For example, if the implementation is an NCL circuit and the specification is a synchronous circuit, the NCL circuit may take many steps to match a single step of the synchronous circuit. This phenomenon is known as stuttering. To account for such situations, multiple but finite transitions of the implementation are allowed to match a single transition of the specification.

Another issue is that to check equivalence, synchronous states and NCL circuit states need to be compared. However, these states can look very different. While synchronous states use D-FF based registers, NCL circuit states use NCL registers that encode data in dual-rail format. Also, there need not be a direct mapping between the synchronous registers and the NCL registers. For the MAC example, the synchronous circuit has one register, while the NCL circuit has three registers. WEB refinement employs refinement maps, functions that map implementation states to specification states to bridge this abstraction gap.

Manolios [13] has shown that it is enough to prove the following refinement-based correctness formula to establish the equivalence of an implementation and specification based on WEB refinement. *rank* is used to distinguish stutter from deadlock. *rank* is a witness function from implementation states to natural numbers whose value decreases when the implementation stutters.

Definition 1. *(Refinement-Based Correctness Formula)*

$$\begin{aligned}\big\langle \forall w \in I :: \;& s = \textit{refinement-map}(w) \wedge u = \textit{SStep}(s) \;\wedge \\ & v = \textit{IStep}(w) \wedge u \neq \textit{refinement-map}(v) \\ \Rightarrow \;& s = \textit{refinement-map}(v) \wedge \textit{rank}(v) < \textit{rank}(w)\big\rangle\end{aligned}$$

In the above, *SStep()* and *IStep()* are functions that define the transitions of the implementation and specification. I is the set of all the implementation states reachable from the initial states of the implementation. Note that in order to establish the equivalence of an implementation and specification system, all the reachable states of the implementation should be checked to see if they satisfy the above correctness formula. The correctness formula given above is expressible in a decidable fragment of first-order logic. Therefore, verifying equivalence of an implementation and a specification based on WEB refinement can be accomplished automatically using an SMT solver if a suitable refinement map and rank function are available.

B. Reachability for Sequential NCL Circuits

The first step in our verification methodology is to find the reachable states of the implementation NCL circuit. Registers in the NCL circuit are initialized to either hold DATA (D) or NULL (N) in a manner that ensures liveness of the circuit. For the MAC example, the registers $r_1r_2r_3$ are initialized to NND. Not every syntactically possible state of the three registers can be reached from the NND initial state. For example, states DDD and NNN will be never reached. We have developed a procedure to compute the reachable states of sequential NCL circuits. We require one assumption, that the NCL circuit has only one input source and one destination output. A large class of circuits fall under this category. We plan to tackle circuits with multiple input sources and destination outputs for future work.

To compute reachable states of an NCL circuit, we define the notion of an N/D state of the circuit.

Definition 2. *For an NCL circuit with n registers, an N/D state of the circuit is an n-tuple $\langle r_1,...,r_n\rangle$, such that for $1 \leq l \leq n$, $r_l = N$ if register l is holding NULL and $r_l = D$ if register l is holding valid DATA.*

Next, we develop a procedure that given an N/D state of a circuit, computes its next N/D state. This procedure is based on the notion of the connectivity matrix of an NCL circuit. In the definition below, R is the set of NCL registers in the circuit.

Definition 3. *The connectivity matrix of an NCL circuit $M_C = [a_{lm}]_{|R|\times|R|}$ such that a_{lm} = 1 if there is a data channel from register l to register m. Otherwise a_{lm} = 0.*

The following functions are used in Definition 4 that gives a function to compute the next N/D state of register l. For the functions, $r_{N/D} \in \{\text{D,N}\}$. If $r_{N/D}$ =D, then *src-st*(l,$r_{N/D}$) returns true only if all source registers to register l are in DATA state.

If $r_{N/D}$ =N, then *src-st*(l,$r_{N/D}$) returns true only if all source registers to register l are in NULL state.

$$\textit{src-st}(l, r_{N/D}) = \bigwedge_{0<m\leq n} \Big\{ \big(M_c[m][l] = 1\big) \rightarrow \big(r_m = r_{N/D}\big) \Big\}$$

Similarly, *dst-st*(l,$r_{N/D}$) evaluates the status of the destination registers of register l. If $r_{N/D}$ =D, then *dst-st*(l,$r_{N/D}$) returns true only if all destinations of register l are in DATA state. If $r_{N/D}$ =N, then *dst-st*(l,$r_{N/D}$) returns true only if all destinations of register l are in NULL state.

$$\textit{dst-st}(l, r_{N/D}) = \bigwedge_{0<m\leq n} \Big\{ \big(M_c[l][m] = 1\big) \rightarrow \big(r_m = r_{N/D}\big) \Big\}$$

Definition 4.

$$\textit{get-nxt-st}(l) = \begin{cases} D, & \big(r_l = N \wedge \textit{src-st}(l,D) \wedge \textit{dst-st}(l,N)\big) \bigvee \\ & \big(r_l = D \wedge \neg\big[\textit{src-st}(l,N) \wedge \textit{dst-st}(l,D)\big]\big) \\ N, & default \end{cases}$$

get-nxt-st(l) computes the next N/D state of register l. If the current N/D state of register l is NULL, all source registers to register l are in DATA state, and all destination registers of register l are in NULL state, then the next N/D state of register l is DATA. Otherwise, register l remains in NULL state in the next state. When the current N/D state of register l is DATA, the next N/D state of register l is NULL if source registers to register l are in NULL state and all the destination registers to register l are in DATA state. Else, register l remains DATA in the next state.

An NCL circuit has transient states and settling states. Settling states are stable. An NCL circuit transitions from a settling state only when there is a change in the inputs to the circuit. Note that on the input side, an NCL circuit has a data input. On the output side, an NCL circuit has a K_i input. Before the circuit transitions from one settling state to another, it transitions through a series of states called transient states. These states occur temporarily and are unstable. Given an input N/D settling state, Procedure 1 computes the next N/D settling state. Without loss of generality, for a circuit with n registers, we assume that r_1 is connected to the data input to the circuit, and r_n is connected to K_i on the output side.

The construction of refinement maps depends on the number and distribution of unique DATA wavefronts in the reachable settling states of an NCL circuit. In the reset state, the number of DATA wavefronts is equal to the number of pipeline stages in the corresponding synchronous circuit. This is a result of how the NCL circuits are initialized, which is discussed in the following subsection. However, this may not be true for all settling states. If the number of DATA wavefronts is greater or less than the number of stages in the synchronous circuit, it is harder to construct refinement mapping functions. To address this issue, we introduce a design-for-verification technique that ensures that the number of DATA wavefronts in all settling states is an invariant and is equal to the number of DATA wavefronts in the reset state, which is again equal to the number of stages in the synchronous circuit. *The design-for-verification technique makes two additional connections in the circuit. The first is to connect the K_o of register r_1 to the input of the completion tree of register r_n. The second is to introduce a data channel from r_n to r_1.* The data channel can simply be introduced by connecting a bit in r_n that is always initialized to the value 0, to a bit in r_1. These additional connections ensures that a DATA wavefront can enter the circuit iff a DATA wavefront exits the circuit. Thus, the number of DATA wavefronts in the circuit always remains a constant. This design-for-verification technique essentially causes the circuit to behave as if there is a feedback loop from the last NCL register to the first NCL register. Procedure 1 exploits the property that the number of DATA wavefronts in the reachable states are invariant.

Procedure 1 Procedure to compute the next N/D settling state

```
1:  procedure GETNEXTSETTINGSTATE(⟨r_1,...,r_n⟩)
2:      settled ← false
3:      1-flipped ← false
4:      n-flipped ← false
5:      while settled = false do
6:          if 1-flipped = true then
7:              r'_1 ← r_1
8:          else
9:              r'_1 ← get-nxt-st(1)
10:         if r'_1 ≠ r_1 then
11:             1-flipped ← true
12:         if n-flipped = true then
13:             r'_n ← r_n
14:         else
15:             r'_n ← get-nxt-st(n)
16:         if r'_n ≠ r_n then
17:             n-flipped ← true
18:         for l ← 2 to n−1 do
19:             r'_l ← get-nxt-st(l)
20:         if ⋀_{l=1}^{n} (r_l = r'_l) then
21:             settled ← true
22:         else
23:             ⟨r_1,...,r_n⟩ ← ⟨r'_1,...,r'_n⟩
```

Procedure 1 uses three flags. The *settled* flag indicates if a settling state has been reached. Since r_1 is connected to the input data, its current value will be flipped (D to N or N to D) exactly once during the computation of the transient states and next settling state. Similarly, since r_n is connected to K_i, r_n value will also be flipped exactly once. *That r_1 and r_n are flipped exactly once when transitioning from one settling state to another is also a result of the design-for-verification technique. 1-flipped* and *n-flipped* keep track of whether r_1 and r_n have been flipped yet or not, respectively. All the flags are initialized to *false*. In the procedure, r_l indicates the current state and r'_l indicates the next state. The while loop (lines 5 to 23) is repeated until the next settling state is reached (*settled* is true). In the while loop, next state of r_1, the register that the input is connected to, remains unchanged if it has already been flipped once (*1-flipped* is true). Otherwise, the next N/D state of r_1 is computed using *get-nxt-st*() function (lines 6 to 9). If the next N/D state of r_1 is different from the current state of the register, *1-flipped* flag is set (lines 10 to 11). The next state of r_n is computed similarly using the *n-flipped* flag.

The next N/D state of all other registers are computed using the *get-nxt-st*() function (lines 18 to 19). Once the next state of all the registers are computed, the next state of the NCL circuit is compared with the current state of the circuit. If the two states are equal, the *settled* flag is set causing the loop to terminate (lines 20 to 21). The next state at the loop termination is the next settling state. Else, the computed next state is set as the current state for the next loop run (lines 22 to 23). The time complexity of the procedure is $O(n^3)$. Using the procedure we get two settling states for the MAC example, NND and DDN. The transition from NND state to DDN state, transitions through the transient states DND and DNN. Similarly, transition from DDN to NND state transitions through NDN and NDD.

The reachable settling states and transitions between settling states form a transition system. For the MAC example, the transitions are NND to DDN and DDN to NND. We then check that the NCL circuit satisfies these transitions.

C. Refinement Maps

In this section, we derive a formula that can be used to construct refinement maps from NCL circuit states to parent synchronous circuit states. An NCL circuit is synthesized from an m-stage synchronous circuit by replacing the registers of the synchronous circuit with two or more NCL registers. Registers with self feedback loops should be replaced with a minimum of three NCL registers to avoid deadlocks. Hence, the n registers of the NCL circuit can be divided into m groups of two or more registers that correspond to the m stages of the synchronous pipeline. The last register in every group is initialized to DATA state at reset. Other NCL registers are initialized to NULL state. Therefore, there are m unique DATA wavefronts in the NCL circuit in the reset states. Also, our design-for-verification technique ensures then that every state of the NCL circuit reachable from reset state also has only m unique DATA wavefronts. Unique DATA wavefronts are separated from each other by NULL wavefronts (spacers).

We number registers such that r_2 is the register with input from r_1, r_3 is the register that has input from r_2 and so on. As such, we are assuming that the circuit has a linear pipeline structure with arbitrary feedback loops, but a large class of circuits can be cast into this structure. Register r_v holds a unique DATA wavefront if $r_v^D r_{v-1}^N$ is true, where for $1 < v \leq n$, r_v^D is a predicate that is true iff r_v is holding a DATA wavefront, and r_v^N is a predicate that is true iff r_v is holding a NULL wavefront. The DATA wavefront in r_1 is always unique.

Since the number of unique DATA fronts in the settling states of the NCL circuit is always equivalent to the number of registers in the parent synchronous circuit (m), the problem of finding a refinement map from the NCL circuit to the synchronous circuit reduces to finding the unique DATA wavefronts of the settling states of the NCL circuit and projecting them onto the stages of the synchronous circuit. Note that how the DATA wavefronts in NCL circuit states are projected depends on how the DATA wavefronts are distributed in that state, which is captured by the N/D settling states. As such each N/D settling state corresponds to a set of the NCL circuit states. The refinement map is constructed by having one projection function per N/D settling state. To construct such refinement maps, we introduce the notion of projection-predicates.

Definition 5. *A projection-predicate $p_{u\leftarrow v}$ is a predicate that is true only when register r_v of an NCL circuit state maps to stage u of the parent synchronous circuit.*

$$p_{u\leftarrow v} = \begin{cases} r_v^D . r_{v-1}^N . & ,(1 \leq u < v \leq n) \\ \left(\wedge_{k=1}^{u-1} \neg p_{k\leftarrow v}\right).\left(\wedge_{k=1}^{v-2} \neg p_{u\leftarrow k}\right) & \\ r_1^D & ,(v = 1 \wedge u = 1) \\ 0 & ,default \end{cases}$$

In the above definition, the first condition for $p_{u\leftarrow v}$ to be true is that r_v should hold a unique DATA wavefront ($r_v^D r_{v-1}^N$). Then, we should determine if the unique DATA wavefront in r_v is the u^{th} unique DATA wavefront. Since the distribution of the DATA wavefronts in an NCL state is highly variable, it is hard to formulate an expression that would capture all possible combinations of DATA wavefront distributions with r_v holding the u^{th} unique DATA wavefront. We tackle this problem using an inductive approach to compute the projection-predicates using $p_{1\leftarrow 1}$ as the base case. The rest of the projection-predicates are constructed inductively. $p_{1\leftarrow 1}$ is true iff r_1^D. We provide two conditions in terms of projection-predicates that if satisfied, the unique DATA wavefront in r_v is the u^{th} unique DATA wavefront. The first condition is that the DATA wavefront in r_v should not be mapping to any of the previous stages. The second condition is u^{th} stage of the synchronous circuit should not have a mapping from register in the NCL circuit before r_v.

We now provide a formula for the refinement map, which gives the value of each register s_u of synchronous pipeline for a given NCL state, using the projection-predicates and projection functions. For $1 \leq u \leq m$:

$$s_u = \begin{cases} pf_{u\leftarrow 1}(r_1), & p_{i\leftarrow 1} \\ pf_{u\leftarrow 2}(r_2), & p_{i\leftarrow 2} \\ \vdots & \\ pf_{u\leftarrow n}(r_n), & p_{i\leftarrow n} \end{cases}$$

Projection function $pf_{u\leftarrow v}(r_v)$ projects register r_v of the NCL circuit to register u of the synchronous circuit. The value of the synchronous register is generated by extracting the value of the D^1 wires of the register r_v. Using the above refinement map we can instantiate the general web refinement theorem (Subsection V-A) for equivalence verification problem of NCL circuits as follows.

$$\Big\langle \forall \langle r_1, ..., r_n \rangle \in \textit{reachable-states} :: \\ \bigwedge_{u=1}^{m} \left[s_u = pf(\langle r_1, ..., r_n \rangle, p_{u\leftarrow 1}, ..., p_{u\leftarrow n}) \right] \wedge \\ \langle s'_1, ..., s'_m \rangle = \textit{syncstep}(\langle s_1, ..., s_m \rangle) \wedge \\ \langle r'_1, ..., r'_n \rangle = \textit{nclstep}(\langle r_1, ..., r_n \rangle) \wedge \\ \neg \bigwedge_{u=1}^{m} \left[s'_u = pf(\langle r'_1, ..., r'_n \rangle, p'_{u\leftarrow 1}, ..., p'_{u\leftarrow n}) \right] \\ \Rightarrow \bigwedge_{u=1}^{m} \left[s_u = pf(\langle r'_1, ..., r'_n \rangle, p'_{u\leftarrow 1}, ..., p'_{u\leftarrow n}) \right] \Big\rangle$$

In the above proof obligation *reachable-states* is the set of reachable states of the NCL circuit found from the reachability

TABLE I. RESULTS

Model	No. of NCL gates	Time (s)	Memory (MB)
mul4X4	91	0.04	1.10
mul4X4-2s	279	0.32	2.94
mac4X4	202	2.77	6.68
mac5X5	281	10.63	8.12
mac6X6	370	96.17	16.59
mac7X7	472	788.93	24.83
mac8X8	587	13,527.12	47.97
mac8X8-B1	587	1.27	10.60
mac8X8-B2	587	4.00	13.63

procedure described in Subsection V-B. $pf()$ is the refinement map constructed from projection predicates and projection functions. *nclstep*() steps the NCL circuit to the next settling state. By proving the above proof obligation together with the reachability invariants mentioned in the Subsection V-B, equivalence of the NCL circuit can be verified against its synchronous parent circuit using theory of WEB refinement. In this work, we consider only the safety part of WEB refinement, i.e., we only check that if the NCL circuit makes progress, that progress is correct w.r.t. to the specification (synchronous circuit). We do not verify liveness, which we plan to address in future work.

VI. RESULTS

The presented verification method was used to verify 9 NCL circuits including two buggy circuits. Verification was performed on a 1.86GHz Intel® Celeron (R) CPU 540 with a 1 MB L2 cache. The SMT solver used was Z3 [14]. Results are summarized in Table I. mul4X4 is a 4-bit combinational multiplier circuit. mul4X4-2s is a pipelined 4-bit multiplier circuit that has two stages. mac$n$Xn circuits are n-bit wide MAC units. Finally, mac8X8-B1 and mac8X8-B2 are buggy 8-bit MAC units. In mac8X8-B1, a bug was injected to the data path by connecting a wrong wire to the input of an adder. In mac8X8-B2, a bug was injected in the completion tree path of the circuit by connecting K_i^0, which is generated by the completion tree for r_0, to r_1 instead. If multiple bugs are present, the solver presents counter examples until all bugs are resolved. The verification of NCL circuits is a complex problem. In fact, as the datapath width of the MAC circuit was increased from 4 to 8, the verification times increased exponentially as can be seen from Table I.

VII. CONCLUSIONS

Our contributions include a design-for-verification technique that reduces the state space of NCL circuits and ensures that the number of data tokens in reachable states of the circuit is an invariant. We exploit this property to develop a procedure to compute the reachable states of NCL circuits. We then use the reachable states to compute mapping functions or refinement maps for equivalence verification. The techniques were demonstrated using several NCL circuits. We found that for the MAC benchmarks, increase in the data path size causes verification times to increase exponentially. For future work, we plan to develop abstraction techniques to improve scalability of our equivalence verification approach.

REFERENCES

[1] S. C. Smith and J. Di, *Designing Asynchronous Circuits using NULL Convention Logic (NCL)*, ser. Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool Publishers, 2009.

[2] R. B. Reese, S. C. Smith, and M. A. Thornton, "Uncle - an rtl approach to asynchronous design," in *ASYNC*, 2012, pp. 65–72.

[3] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 25, no. 10, pp. 1904–1921, Oct 2006.

[4] J. R. Burch, "Combining ctl, trace theory and timing models," in *Automatic Verification Methods for Finite State Systems*, ser. Lecture Notes in Computer Science, J. Sifakis, Ed., vol. 407. Springer, 1989, pp. 334–348.

[5] T. Yoneda and B.-H. Schlingloff, "Efficient verification of parallel real-time systems," *Formal Methods in System Design*, vol. 11, no. 2, pp. 187–215, 1997.

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

[7] P. Loewenstein, "Formal verification of counterflow pipeline architecture," in *Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, 1995.

[8] F. Verbeek and J. Schmaltz, "Verification of building blocks for asynchronous circuits," in *ACL2*, ser. EPTCS, R. Gamboa and J. Davis, Eds., vol. 114, 2013, pp. 70–84.

[9] A. Kondratyev, L. Neukom, O. Roig, A. Taubin, and K. Fant, "Checking delay-insensitivity: 104 gates and beyond," in *Asynchronous Circuits and Systems, 2002. Proceedings. Eighth International Symposium on*, April 2002, pp. 149–157.

[10] S. K. Srinivasan and R. S. Katti, "Desynchronization: design for verification," in *FMCAD*, P. Bjesse and A. Slobodová, Eds. 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. on CAD of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 1904–1921, 2006.

[12] P. Manolios, "Mechanical verification of reactive systems," Ph.D. dissertation, University of Texas at Austin, August 2001, see URL `http://www.cc.gatech.edu/~manolios/-publications.html`.

[13] ——, "Correctness of pipelined machines," in *Formal Methods in Computer-Aided Design–FMCAD 2000*, ser. LNCS, W. A. Hunt, Jr. and S. D. Johnson, Eds., vol. 1954. Springer-Verlag, 2000, pp. 161–178.

[14] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *TACAS*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds., vol. 4963. Springer, 2008, pp. 337–340.