An Equivalence Verification Methodology for Asynchronous Sleep Convention Logic Circuits

Mousam Hossain, Ashiq A. Sakib, Sudarshan K. Srinivasan, and Scott C. Smith
Department of Electrical and Computer Engineering
North Dakota State University, Fargo, ND, USA

***Abstract*—Sleep Convention Logic (SCL) is an emerging ultra-low power Quasi-Delay Insensitive (QDI) asynchronous design paradigm with enormous potential for industrial applications. Design validation is a critical concern before commercialization. Unlike other QDI paradigms, such as NULL Convention Logic (NCL) and Pre-Charge Half Buffers (PCHB), there exists no formal verification methods for SCL. In this paper, we propose a unified formal verification scheme for combinational as well as sequential SCL circuits, based on equivalence checking, which verifies both safety and liveness. The method is demonstrated using several multipliers, MACs, and ISCAS benchmarks.**

Keywords— equivalence checking, formal verification, QDI

I. Introduction

SCL, also known as Multi-Threshold NULL Convention Logic (MTNCL) [4], integrates the MTCMOS technique for leakage power reduction with the popular NCL architecture [1] to provide an ultra-low power, high speed QDI asynchronous paradigm. Formal verification techniques are widely used in the semiconductor industry to validate commercial designs; however, there are currently no formal verification schemes for SCL circuits. A few formal verification techniques exist for other QDI circuits, such as NCL [5, 17] and Pre-Charge Half Buffers (PCHB) [7, 18]; but, those methods are not directly applicable to SCL circuits because of its unique structure. In this paper, we propose a formal verification method based on equivalence checking that verifies both functionality and handshaking control of combinational and sequential SCL circuits.

The rest of the paper is organized as follows: Section II provides an overview of the SCL framework, background, and related verification works. The proposed formal verification method is presented in Section III, followed by demonstration of the method verifying several benchmark circuits in Section IV. Finally, conclusions and future work are discussed in Section V.

II. Previous Work

A. Sleep Convention Logic (SCL)

As mentioned earlier, SCL combines MTCMOS technology with NCL. Like NCL, it utilizes dual-rail logic (i.e., two wires to represent 1 bit of data, where (D^1=0, D^0=1) = DATA0, (D^1=1, D^0=0) = DATA1, (D^1=0, D^0=0) = NULL or absence of DATA, and (D^1=1, D^0=1) is an invalid state). Also like NCL, SCL circuits are comprised of the same 27 threshold gates consisting of all functions of 4 or fewer variables; however, NCL gates include hysteresis, which requires all inputs to be deasserted before the output is deasserted, whereas SCL gates include an additional *sleep* input, which de-asserts the gate output when asserted. Because of this sleep mechanism, SCL circuits do not need to be input-complete or observable, as required for NCL circuits [4].

As shown in Fig. 1, an SCL pipeline is similar to NCL, where each stage consists of a QDI register (R_i), combinational logic (C/L_i), and completion detection (C_i). However, SCL instead utilizes early completion [3], and each stage component is slept to NULL by asserting the corresponding *sleep* signal rather than transitioning to NULL by propagating a NULL wavefront.

B. Related Work on Formal Verification

Wijayasekara et al. [5] have proposed an idea to verify NCL circuits based on the theory of WEB-Refinement [6], where the specification and implementation are both modeled as Transition Systems (TSs). Due to the multitude of feedback loops caused by handshaking control, the TSs become very complex with a huge state-space, resulting in state space explosion and infeasible verification times. Sakib et al. [7] describe a model checking based approach for PCHB circuits that also models the PCHB circuit as a TS, but this again suffers from state space explosion. A deadlock verification scheme for delay insensitive circuits based on the Click Library [8] is proposed in [9]; however, this method is not directly applicable, as SCL circuits are structurally very different. There also exists several Design-For-Test (DFT) techniques for NCL [10, 11] and SCL [12]; however, only testing is not sufficient to ensure complete functional correctness. Since scalability is the major limiting factor for the aforementioned similar methods [5, 7], this paper avoids modelling the SCL circuit as a TS, and instead develops an alternate approach that is scalable and addresses the limiting factors encountered in other methods.

III. SCL Equivalence Verification Methodology

In industry, QDI circuits are typically synthesized from corresponding synchronous specifications, which go through a series of transformations, resulting in the synthesized circuit being structurally very different from the original specification. For such a scenario, equivalence checking is a widely used formal technique that ensures functional and logical equivalence between two structurally different systems. As detailed below, the proposed method requires two steps, the first to ensure *safety* (i.e., functional correctness), and the second to check handshaking correctness to ensure *liveness* (absence of deadlock).

A. Safety Check for Combinational SCL Circuits

Fig. 2a shows an SCL 2×2 multiplier. The C/L is comprised of SCL 2-input AND functions (AND2) and Half Adders (HAs), shown in Figs 2c and 2d, respectively. The STAGE 1 and 3 registers are the input and output registers, respectively; and the STAGE 2 register is an intermediate pipelining register used to increase throughput. All registers are reset to NULL, since there is no datapath feedback. Comp1, Comp2, and Comp3 are the completion units that generate the *sleep* signals for their respective stages. K_i is the external *request* input, and $Ko = k_{o1}$ is the external *acknowledge* output. *SLP* is an external *sleep* input that sleeps the STAGE 1 completion unit.

The netlist of the SCL 2×2 multiplier is shown in Fig. 3a. The line numbers are used for ease of reference only and do

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

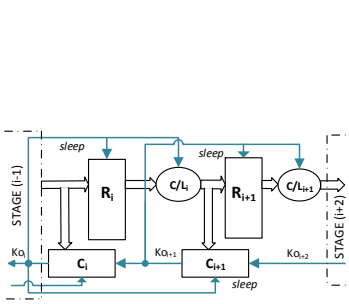
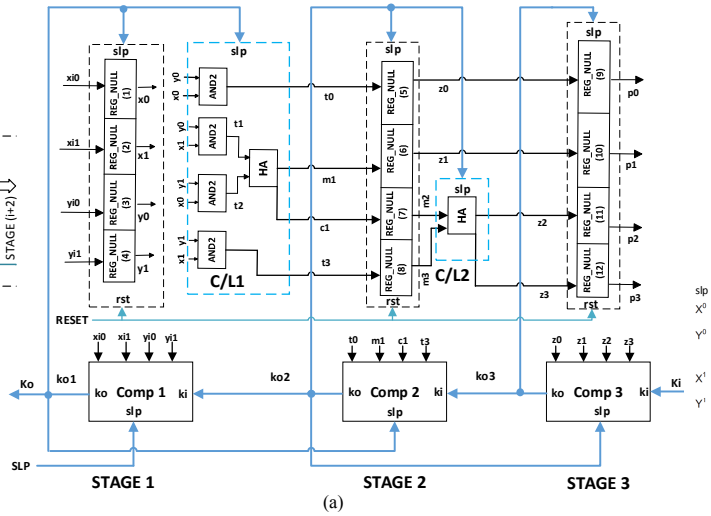
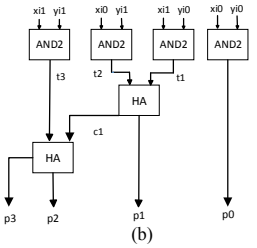


Fig. 1: SCL SECRII w/o nsleep architecture





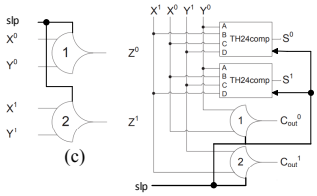


Fig. 2: (a) 2×2 SCL multiplier (b) Equivalent Boolean 2×2 multiplier (c) SCL AND2 function (d) SCL HA

not appear in the actual netlist. The first two lines denote the set of primary inputs and primary outputs, respectively. For any dual-rail signal, *a*, *a_0* and *a_1* denote a^0 and a^1, respectively. Lines 3-18 denote the gate structure of the C/L, where the first column indicates the *type* of SCL gate, the second column refers to the *level* of the gate (i.e., the longest path to the gate from any primary input, not counting registers), and the third, fourth, and fifth columns correspond to the gate's *data inputs*, *sleep input*, and *output*, respectively. Lines 19-30 are the 1-bit dual-rail SCL registers, where the first column corresponds to the *reset type*, columns 2 and 3 are the data $input^0$ and $input^1$ rails, respectively, column 4 is the *sleep* input, and columns 5 and 6 are the data $output^0$ and $output^1$ rails, respectively. Lines 31-33 are the three completion units, where *Comp* in the first column denotes an early completion component, the second column indicates the *Ki* input that comes from the following stage completion, the third column includes all of the completion unit's *data inputs*, the fourth column is the *sleep* input, and the last column is the *output*. Note that this SCL netlist is generated by processing the original SCL netlist to order the components as described above and to combine all of the completion unit gates into the single completion component described above. Each completion component, such as *Comp2* in Fig. 2a, is comprised of *TH12/TH24comp* gates and *THnn SCL* gates arranged in a tree structure, followed by a final inverted NCL *TH22* gate (without *sleep*), as shown in [4]. When processing the original SCL netlist to obtain the abstracted completion component shown in the Fig. 3a netlist, we ensure that all data inputs to a completion unit go to *TH12*/*TH24comp* gates, and their outputs form a tree of SCL *THnn* gates, whose output, along with the K_i input, is input to an inverted NCL *TH22* gate that produces the K_o output, and that all SCL gates have the same *slp* input.

The safety check first reduces the SCL netlist, as shown in Fig. 3a, into an equivalent Boolean netlist, as shown in Fig. 3b. Each SCL C/L gate is replaced with its corresponding Boolean function, omitting the *sleep* input. Each rail of a dual-rail signal is treated as a distinct Boolean signal, which requires the addition of an inverter for each primary circuit input, to generate its complement to replace each input's $rail^0$, used in the C/L, as shown in lines 3-6. Similar to the SCL netlist structure, the first two lines in the converted netlist correspond to the set of primary inputs and primary outputs, respectively. Each subsequent line corresponds to a C/L gate, where the first column denotes the *type* of gate, the second column denotes the gate's *level*, the third column denotes the gate's *inputs*, and the fourth column denotes the gate's *output*. Note that the C/L *sleep* input, SCL registers, and completion units are removed, since these are not utilized in the Boolean circuit; and their connections will be verified as part of the liveness check, explained next.

The Boolean netlist obtained from the SCL netlist is then checked against the specification Boolean function. We use the Z3 [13] SMT solver to perform this check. For the example 2×2 multiplier with two 2-bit dual-rail inputs, *x(1:0)* and *y(1:0)*, the SMT solver checks the following property: $F_{SCL\_Bool\_Equv}$ (*x0, x1, y0, y1*) → *MUL* (*x(1:0)*, *y(1:0)*), where $F_{SCL\_Bool\_Equv}$ is the function corresponding to the Boolean circuit obtained from converting the SCL circuit to be verified, (*x1, x0)* and (*y1, y0)* are the (MSB, LSB) of *x* and *y*, respectively, and *MUL* corresponds to the Boolean specification of the multiplier circuit. It also checks that the $rail^0$ and $rail^1$ outputs in the converted netlist are complements of each other: i.e., $F_{SCL\_Bool\_Equv}$ (*p0_1, p1_1, p2_1, p3_1*) → ¬ $F_{SCL\_Bool\_Equv}$ (*p0_0, p1_0, p2_0, p3_0*).

B. Handshaking Check for Combinational SCL Circuits

The 2×2 multiplier in Fig. 2a implements the SECRII w/o *nsleep* architecture, where the registers, completion units, and C/L are all slept during the NULL cycle. The output of the STAGE 1 completion unit, *ko1,* is the *slp* input of the STAGE 1 registers (1-4), the STAGE 1 C/L, C/L1, and the STAGE 2 completion unit, Comp 2. We developed an algorithm that takes an SCL netlist, as shown in Fig. 3a, and converts it into a graph structure, where each register, threshold gate, and completion unit are modeled as nodes. The directed edges going into and out from a node correspond to the inputs and outputs of that particular node, respectively. The algorithm traverses the graph to gather the needed information in order to verify that the handshaking exactly follows the SCL protocol. For registers, completion units, and threshold gates, the following information is stored: data inputs, *sleep* input, and data output(s). After gathering this information, the following handshaking checks are performed:

- Register *sleep* and Completion data inputs: Each stage register's data inputs must be exactly the same as for the stage's completion unit; and the completion unit output

```
1.  xi0_0,xi0_1,xi1_0,xi1_1,yi0_0,yi0_1,yi1_0,yi1_1
2.  p0_0,p0_1,p1_0,p1_1,p2_0,p2_1,p3_0,p3_1
3.  th12  1  x0_0,y0_0  ko1  t0_0
4.  th22  1  x0_1,y0_1  ko1  t0_1
5.  th12  1  x1_0,y0_0  ko1  t1_0
6.  th22  1  x1_1,y0_1  ko1  t1_1
……
11. th24comp  2  t2_0,t1_1,t1_0,t2_1  ko1  m1_0
12. th24comp  2  t2_0,t1_0,t2_1,t1_1  ko1  m1_1
13. th12  2  t2_0,t1_0  ko1  c1_0
……
15. th24comp  3  m2_0,m3_1,m3_0,m2_1  ko2  z2_0
16. th24comp  3  m2_0,m3_0,m2_1,m3_1  ko2  z2_1
17. th12  3  m2_0,m3_0  ko2  z3_0
18. th22  3  m3_1,m2_1  ko2  z3_1
19. Reg_NULL  xi0_0  xi0_1  ko1  x0_0  x0_1
20. Reg_NULL  xi1_0  xi1_1  ko1  x1_0  x1_1
……...
30. Reg_NULL  z3_0  z3_1  ko3  p3_0  p3_1
31. Comp  ko2  xi0_0,xi0_1,xi1_0,xi1_1,yi0_0,yi0_1,yi1_0,yi1_1  SLP  ko1
32. Comp  ko3  t0_0,t0_1,m1_0,m1_1,c1_0,c1_1,t3_0,t3_1  ko1  ko2
33. Comp  Ki  z0_0,z0_1,z1_0,z1_1,z2_0,z2_1,z3_0,z3_1  ko2  ko3
```

```
1.  Inputs: xi0_1,xi1_1,yi0_1,yi1_1
2.  Outputs:  p0_0,p0_1,p1_0,p1_1,p2_0,p2_1,p3_0,p3_1
3.  not  1  xi0_1  xi0_0
4.  not  1  yi0_1  yi0_0
5.  not  1  xi1_1  xi1_0
6.  not  1  yi1_1  yi1_0
7.  th12  2  xi0_0,yi0_0  p0_0
8.  th22  1  xi0_1,yi0_1  p0_1
9.  th12  2  xi1_0,yi0_0  t1_0
10. th22  1  xi1_1,yi0_1  t1_1
11. th12  2  xi0_0,yi1_0  t2_0
12. th22  1  xi0_1,yi1_1  t2_1
13. th12  2  xi1_0,yi1_0  t3_0
14 th22  1  xi1_1,yi1_1  t3_1
15. th24comp  3  t2_0,t1_1,t1_0,t2_1  p1_0
16. th24comp  3  t2_0,t1_0,t2_1,t1_1  p1_1
17. th12  3  t2_0,t1_0  c1_0
18. th22  2  t1_1,t2_1  c1_1
19. th24comp  4  c1_0,t3_1,t3_0,c1_1  p2_0
20. th24comp  4  c1_0,t3_0,c1_1,t3_1  p2_1
21. th12  4  c1_0,t3_0  p3_0
22. th22  3  t3_1,c1_1  p3_1
```

Fig. 3: (a) SCL 2×2 multiplier netlist (b) Converted equivalent Boolean netlist

must be the register's *sleep* input. As an example, the inputs of registers (1-4) in STAGE1 are also the data inputs to completion unit, Comp1. Hence, the output of Comp1, *ko1*, is the *slp* input for registers (1-4).

• Sleep for C/L: Each completion unit sleeps its stage's register and C/L, such that every SCL C/L gate's sleep input should be the same as its preceding register's sleep input. Hence, for each gate, *i,* we create a *gate_fanin(i)* list that traces back all inputs of $gate_i$ to their originating register. For example, the TH12 gate on line 13 of Fig. 3a, corresponds to the TH12 gate that generates the $c1^0$ output of the HA in C/L1 in Fig 2a. Tracing this gate's inputs back to their generating registers yields x1_0, y0_0, x0_0, y1_0, resulting in a *gate_fanin* list of Reg1, Reg2, Reg3, Reg4, which all have the same *slp* input as the TH12 gate. Once the *gate_fanin* list for all gates are computed, all registers in each *gate_fanin(i)* list are inspected to ensure that they all have the same *slp* input, and that this sleep input is also the *slp* input for $gate_i$. If the *gate_fanin* list contains registers from multiple stages (i.e., different *slp* inputs), or if the gate's *slp* input differs from its corresponding input register's *slp* input, then an error is generated.

• Completion output, and *slp* and *Ki* inputs: $Comp_i$'s *Ki* input must be the output of $Comp_{i+1}$, and its *slp* input must be the output of $Comp_{i-1}$. In Fig. 2a, $Comp_2$'s *Ki* input is the output of $Comp_3$, and its *slp* input is the output of $Comp_1$. The first and last stages are slightly different. $Comp_1$'s (i.e., the completion unit whose data inputs are the circuit's external data inputs) *slp* input must be the external *SLP* input, and its output must be the external *Ko* output; the last stage completion's (i.e., the completion unit associated with the register that produces the external data outputs) *Ki* input must be the external *Ki* input.

C. Safety Check for Sequential SCL Circuits

Verification of sequential SCL circuits are much more complex because of datapath feedback, which requires at least 2N+1 SCL registers in a feedback loop with *N* DATA tokens in order to avoid deadlock [14]. Hence, common practice when synthesizing an SCL circuit from its synchronous specification is to replace every synchronous register with three SCL registers, reset to NULL, DATA, NULL. For the 4+2×2 MAC example in Fig. 4b, the output register is replaced with registers 5-8 and 15-22, as shown in the Fig. 4a SCL implementation. In addition to the fed back accumulator output, the STAGE 1 registers (registers 1-8) also include the external data inputs, *Xi (1:0)* and *Yi (1:0)*, while the STAGE 2 register is an additional reset-to-NULL register included to increase performance. Note that STAGE 1 and 2 could be combined into a single stage or STAGE 3 could be removed without causing deadlock.

The netlist structure for a sequential SCL circuit is similar to that for a combinational circuit, shown in Fig. 3a. The only differences are the inclusion of C-elements in the feedback loop handshaking, and reset-to-DATA registers, REG_DATA0 in this case. The algorithm to convert the SCL netlist into its equivalent synchronous netlist is also similar to the combinational SCL conversion described in Section III.A, with the following additions: each reset-to-DATA register is converted into an equivalent 2-bit Boolean register, one bit for the SCL register's $rail^1$ output and the other for its $rail^0$ output. Same as for the combinational circuit conversion, the reset-to-NULL registers and all *sleep* signals are omitted from the converted synchronous netlist, as these will be verified in the subsequent handshaking check. For sequential circuits, the additional C-elements are also omitted from the converted synchronous netlist, and will also be verified in the handshaking check.

The theory of WEB-Refinement [6] is then utilized to check for equivalence between the converted synchronous netlist and the original synchronous specification. We assume that the I\O mapping and the register mapping between the specification and implementation circuits are provided. The converted netlist, synchronous specification, and equivalence check properties are modeled in SMT-LIB, and the properties checked using the Z3 SMT solver. For the 4+2×2 MAC, the following properties are checked: Considering symbolic state transitions for any current state values of 2-bit primary inputs, *x_cs(1:0)* and *y_cs(1:0)*, and 4-bit accumulator values *acc_cs(3:0)*, the next state of the converted synchronous netlist obtained from the SCL implementation should be equivalent to the next state of the synchronous specification; i.e., $F_{SCL\_Sync\_Eq}$ *(x0_cs, x1_cs,*

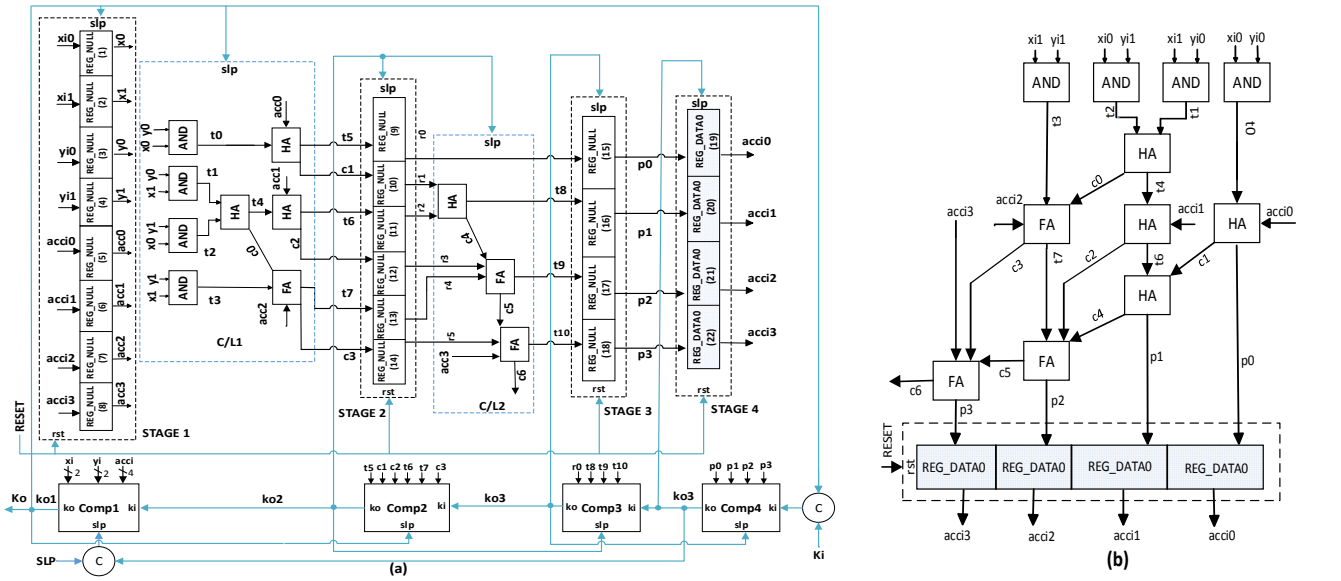


Fig. 4. (a) 4+2×2 SCL MAC (b) Converted equivalent Boolean circuit

y0_cs, y1_cs, acc0_cs, acc1_cs, acc2_cs, acc4_cs)→ F_{Sync_Spec} (x_cs(1:0), y_cs(1:0), acc_cs(3:0)). For each register in the converted synchronous netlist, we also check that its output rail^0 and rail^1 are complements of each other.

D. Handshaking Check for Sequential SCL Circuits

Handshaking verification is the same as for combinational SCL circuits, except that the first and last registers of a feedback loop include an extra C-element, as shown in Fig. 4a, which requires the following two additional checks: The *Ki* input for a feedback loop's output register's completion (e.g., Comp4 in Fig. 4a) must be the combination (via a C-element) of its downstream register's completion's *Ko* (external *Ki* input in Fig. 4a) and its feedback loop input register's completion's *Ko* (*ko1* in Fig. 4a), instead of only its downstream register's completion's *Ko,* as in combinational SCL circuits. The *slp* input for a feedback loop's input register's completion (e.g., Comp1 in Fig. 4a) must be the combination (via a C-element) of its upstream register's completion's *Ko* (external *SLP* input in Fig. 4a) and its feedback loop output register's completion's *Ko* (*ko3* in Fig. 4a), instead of only its upstream register's completion's *Ko,* as in combinational SCL circuits.

IV. Results

The algorithms described in Section III were implemented using Python. As tabulated below, several SCL sequential MACs and combinational multipliers and ISCAS benchmarks [15, 16] were successfully verified applying the proposed methodology, executed on an Intel® Core™ i7-4790 CPU with 32GB of RAM running at 3.60 GHz, using the Z3 SMT solver [13] to check for functional equivalence. Additionally, a number of buggy circuits were tested, including circuits with erroneous handshaking signals, such as an incorrect *sleep* signal connection to a C/L gate (i.e., 20+10×10 MAC B1) and a C/L gate with one data input being a signal's rail0 instead of the correct rail1 (i.e., 20+10×10 MAC B2). For all buggy cases, the proposed approach was able to flag the errors, providing a descriptive message indicating the erroneous connection for handshaking errors, and producing counter examples to trace back the error path (via the SMT solver) for cases of functional in-equivalence. Since B2 was caught during the safety check, its verification time was much less than for B1, which was detected in the handshaking check. Time to convert an SCL netlist to its equivalent Boolean/synchronous netlist was negligible compared to safety and handshaking check times, and therefore was not included in Table I.

V. Conclusions and Future Work

This paper proposes an equivalence verification methodology for both combinational and sequential SCL circuits. The method currently only works for sequential circuits without interacting feedback loops, such as a MAC, since the handshaking is otherwise much more complicated, and requires the development of an algorithm to map each register in the converted synchronous circuit to its corresponding register in the original synchronous specification. These problems will be tackled in future work, such that the developed approach will be applicable to any arbitrary sequential circuit. Scalability of the approach can be improved further using abstraction techniques, and a commercial equivalence checker instead of an SMT solver.

TABLE I. Verification Results for Various SCL Circuits

SCL Circuit	Verification Times (in sec.)		
	Safety Check	*Handshaking Check*	*Total time*
8x8 MUL	10.62	0.055	10.675
10x10 MUL	683.49	0.1536	683.64
12x12 MUL	49,963.05	0.316	49,963.36
ISCAS c17 [15]	0.01	0.002	0.012
ISCAS c432 [15]	1.03	0.0468	1.0768
ISCAS s27 [16]	0.09	0.002	0.092
12+6x6 MAC	1.69	0.031	1.721
16+8x8 MAC	12.03	0.1007	12.131
20+10x10 MAC	1,581.72	0.213	1581.93
24+12x12 MAC	1,40,780.13	0.4078	1,40,780.5
20+10x10 MAC B1	1483.22	0.3403	1483.5603
20+10x10 MAC B2	0.17	0.197	0.367

Acknowledgment

This work is supported by the National Science Foundation under Grant No. CCF-1717420.

REFERENCES

[1] K. M. Fant and S. A. Brandt, "NULL convention logic: a completeand consistent logic for asynchronous digital circuit synthesis," In Proc. IEEE International Conference on Application Specific Systems, Architectures and Processors, August 1996, pp. 261–273.

[2] 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.

[3] S. C. Smith, "Speedup of self-timed digital systems using early completion," In Proc. IEEE Computer Society Annual Symposium on VLSI, April 2002, pp. 98–104.

[4] L. Zhou, R. Parameswaran, F. Parsan, S. Smith, and J. Di, "Multi-Threshold NULL Convention Logic (MTNCL): An Ultra-Low Power Asynchronous Circuit Design Methodology," *Journal of Low Power Electronics and Applications*, vol. 5, no. 2, May, pp. 81–100, 2015.

[5] Vidura M. Wijayasekara, S.K.Srinivasan and S. C. Smith, "Equivalence verification for NULL Convention Logic (NCL) circuits," In Proc. IEEE 32nd International Conference on Computer Design (ICCD), Oct 2014, pp. 195-201.

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

[7] 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 International Midwest Symposium on Circuits and Systems, August 2017, pp. 519-522.

[8] A. Peeters, F. te Beest, M. de Wit & W. Mallon, "Click elements: An implementation style for data-driven compilation," In Proc. IEEE Symposium on Asynchronous Circuits and Systems (ASYNC'10), 2010, pp. 3-14.

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

[10] V. Satagopan, B. Bhaskaran, W. K. Al-Assadi, S. C. Smith, and S. Kakarla, "DFT techniques and automation for asynchronous NULLconventional logic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 10, pp. 1155–1159, October 2007.

[11] W. Al-Assadi and S. Kakarla, "Design for test of asynchronous NULLconvention logic (NCL) circuits", *Journal of Electronic Testing*, vol. 25, no. 1, 2009, pp. 117–126.

[12] F. Parsan, S. C. Smith, W. K. Al-Assadi, "Design for Testability of Sleep Convention Logic", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 743-753, 2015.

[13] 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

[14] T. E. Williams, "Self-Timed Rings and Their Application to Division", Ph.D. Thesis, CSL-TR-91-482, Department of Electrical Engineering and Computer Science, Stanford University, 1991.

[15] D. Bryan, The ISCAS '85 benchmark circuits and netlist format [online] Available: https://ddd.fit.cvut.cz/prj/Benchmarks/iscas85.pdf. [Accessed Feb. 1, 2019].

[16] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," In Proc. Int'l. Symp. Circuits and Systems, 1989, pp. 1929-1934.

[17] V. M. Wijayasekara, A. T. Rollie, R. G. Hodges, S. K. Srinivasan, S. C. Smith, "Abstraction techniques to improve scalability of equivalence verification for NCL circuits," Electron. Letters, 2016, 52, (19), pp. 1594–1596.

[18] A. A. Sakib, S. C. Smith, and S. K. Srinivasan, "An equivalence verification methodology for combinational asynchronous PCHB circuits," In Proc. IEEE International Midwest Symposium on Circuits and Systems, 2018, pp. 767 - 770.