A Halting Algorithm to Determine the Existence of Decoder

ShengYu Shen, *Member, IEEE*, Ying Qin, LiQuan Xiao, KeFei Wang, JianMin Zhang, and SiKun Li

***Abstract*—Complementary synthesis automatically synthesizes the decoder circuit of an encoder. It determines the existence of decoder by checking whether the encoder's input can be uniquely determined by its output. However, this algorithm will not halt if the decoder does not exist.**

To solve this problem, we propose a novel halting algorithm to check the existence of decoder. For every unfolding of the encoder's transition function, we not only check whether its input can be uniquely determined by its output, we also check whether there exists a loop-like path that reaches a particular state, from which another loop-like path can produce an output sequence that corresponds to two different input letters. These two checks will confirm respectively the existence and non-existence of decoder, and thus confirm the halting character of our new algorithm.

To illustrate its usefulness and efficiency, we have run our algorithm on several complex encoders, including PCIE and Ethernet. Experimental results show that, our new algorithm always halts properly by distinguishing correct encoders from incorrect ones, and it can be more than three times faster than our previous work.

***Index Terms*—Halting Algorithm, Complementary Synthesis**

I. Introduction

Complementary synthesis [1], [2] is proposed by us to automatically synthesize the decoder circuit E^{-1} of an encoder E. It determines the existence of E^{-1} by checking the parameterized complementary condition(***PC***), i.e., whether E's input can be uniquely determined by its output on a bounded unfolding of E's transition function.

However, this algorithm will not halt if E^{-1} does not exist. We have proposed another algorithm in [3] to solve this problem by constructing an onion-ring between PC and its over-approximation, and checking whether E is in all these rings.

This new algorithm is very complicate and slow. So we propose in this paper yet another halting algorithm, which is both straightforward and faster.

For every unfolding of the encoder's transition function, we not only check whether its input can be uniquely determined by its output, we also check whether there exists a loop-like path that reaches a particular state, from which another loop-like path can produce an output sequence that corresponds to two different input letters. These two checks will confirm respectively the existence and non-existence of decoder, and thus confirm the halting character of our new algorithm.

The authors are with the School of Computer, National University of Defense Technology, ChangSha, HuNan, 410073 CHINA. e-mail: {syshen, qy123, lqxiao, kfwang, jmzhang, skli}@nudt.edu.cn, (see http://www.ssypub.org/).

We have implemented this algorithm with the OCaml language, and solved the generated SAT instances with Zchaff SAT solver [4]. The benchmark set includes several complex encoders from industrial projects (e.g., PCIE and Ethernet), and their slightly modified variants without corresponding decoders. Experimental results show that, our new algorithm always halts properly by distinguishing correct encoders from incorrect ones, and is much faster than [3]. All experimental results and programs can be downloaded from http://www.ssypub.org.

The remainder of this paper is organized as follows. Section II presents background materials. Section III presents our new algorithm. Section IV describes how to remove redundant output letters to minimize circuit area. Section V presents experimental results. Finally, Section VI concludes with a note on future work.

II. Preliminaries

A. Propositional Satisfiability

For a Boolean formula F over a variable set V, the **Propositional Satisfiability Problem**(abbreviated as **SAT**) is to find a satisfying assignment $A : V \rightarrow \{0, 1\}$, so that F can be evaluated to 1. If such a satisfying assignment exists, then F is **satisfiable**; otherwise, it is **unsatisfiable**.

A computer program that decides the existence of such a satisfying assignment is called a **SAT solver**, such as Zchaff [4], Grasp [5], Berkmin [6], and MiniSAT [7].

B. Recurrence Diameter

A circuit can be modeled by **Kripke structure** $M = (S, I, T, A, L)$, with a finite state set S, the initial state set $I \subseteq S$, the transition relation $T \subseteq S \times S$, and the labeling of the states $L : S \rightarrow 2^A$ with atomic proposition set A.

Kroening et al. [8] defined the **state variables recurrence diameter** with respect to M, denoted by $rrd(M)$, as the longest loop-free path in M starting from an initial state.

$$rrd(M) \stackrel{def}{=} \max\{i | \exists s_0 \dots s_i : I(s_0) \wedge \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k\} \quad (1)$$

In this paper, we define a similar concept: the **uninitialized state variables recurrence diameter** with respect to M, denoted by $uirrd(M)$, is the longest loop-free path in M.

$$uirrd(M) \stackrel{def}{=} \max\{i|\exists s_0 \dots s_i : \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \wedge \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k\} \quad (2)$$

The only difference between these two definitions is that our $uirrd$ does not consider the initial state.

These definitions are only used in proving our theorems below. Our algorithm does not need to compute these diameters.

C. The Original Algorithm to Determine the Existence of Decoder

The complementary synthesis algorithm [1] includes two steps: determining the existence of decoder and characterizing its Boolean function. We will only introduce the first step here.

The encoder E can be modeled by a Mealy finite state machine [9].

***Definition** 1:* **Mealy finite state machine** is a 5-tuple $M = (S, s_0, I, O, T)$, consisting of a finite state set S, an initial state $s_0 \in S$, a finite set of input letters I, a finite set of output letters O, a transition function $T : S \times I \rightarrow S \times O$ that computes the next state and output letter from the current state and input letter.

As shown in Figure 1, as well as in the remainder of this paper, the state is represented as a gray round corner box, and the transition function T is represented by a white rectangle. We denote the state, input letter and output letter at the n-th cycle respectively as s_n, i_n and o_n.

A sufficient condition for the existence of E^{-1} is, there exist three parameters p, d and l, so that i_n of E can be uniquely determined by the output sequence $< o_{n+d-l}, \dots, o_{n+d-1} >$. As shown in Figure 2, d is the relative delay between $< o_{n+d-l}, \dots, o_{n+d-1} >$ and the input letter i_n, while l is the length of $< o_{n+d-l}, \dots, o_{n+d-1} >$, and p is the length of prefix state transition sequence used to rule out some unreachable states.

Thus, as shown in Figure 2, the parameterized complementary condition(PC) [1] can be defined formally as:

***Definition** 2:* **Parameterized Complementary Condition (PC) :** For encoder E, $E \vDash PC(p, d, l)$ holds if i_n can be uniquely determined by $< o_{n+d-l}, \dots, o_{n+d-1} >$ on state transition sequence $< s_{n-p}, \dots, s_{n+d-1} >$. This equals the unsatisfiability of $F_{PC}(p, d, l)$ in Equation (3). We further define $E \vDash PC$ as $\exists p, d, l : E \vDash PC(p, d, l)$.

$$F_{PC}(p, d, l) \stackrel{def}{=} \left\{ \begin{array}{c} \bigwedge_{m=n-p}^{n+d-1}\{(s_{m+1}, o_m) \equiv T(s_m, i_m)\} \\ \wedge \quad \bigwedge_{m=n-p}^{n+d-1}\{(s'_{m+1}, o'_m) \equiv T(s'_m, i'_m)\} \\ \wedge \quad \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \\ \wedge \quad i_n \neq i'_n \end{array} \right\} \quad (3)$$

The 2nd and 3rd lines of Equation (3) correspond respectively to two unfolded instances of E's transition function. The only difference between them is that a prime is appended to every variable in the 3rd line. The 4th line forces the output sequences of these two unfolded instances to be the same, while the 5th line forces their input letters to be different.

We iterate through all valuations of d, l and p, from the smaller one to the larger one, until we find one valuation of d,l and p that makes Equation (3) unsatisfiable. Then the existence of decoder is proved.

However, if the decoder does not exist, this algorithm will never halt. This problem will be solved in next section.

III. A Halting Algorithm to Determine the Existence of Decoder

A. A Sufficient and Necessary Condition for the Non-existence of Decoder

PC in Definition 2 only define how to detect the existence of decoder E^{-1}, but not how to detect the non-existence of E^{-1}. So the key to a halting algorithm is to find out a sufficient and necessary condition for the non-existence of E^{-1}.

According to Definition 2 and Figure 2, E^{-1} exists if there is a parameters tuple $< p, d, l >$, such that every path longer that p always reaches a particular state s_n, in which the input letter i_n can be uniquely determined by the output sequence $< o_{n+d-l}, \dots, o_{n+d-1} >$.

So, Intuitively, E^{-1} does not exist if for every parameters tuple $< p, d, l >$, we can always find another tuple $< p', d', l' >$ with $p' > p$,$l' > l$ and $d' > d$, such that PC does not hold.

This case can be detected by the SAT instance in Figure 3, which is similar to Figure 2, except three new constraints used to detect loop-like paths on state sequences $< s_{n-p}, \dots, s_{n+d-l} >$,$< s_{n+d-l}, \dots, s_n >$ and $< s_{n+1}, \dots, s_{n+d} >$.

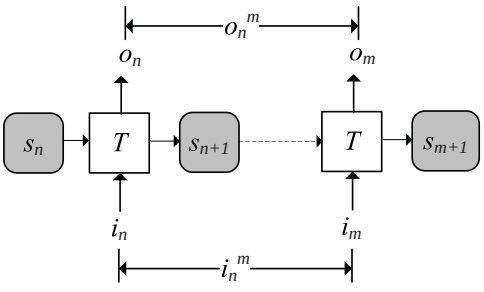

Fig. 1. Mealy finite state machine

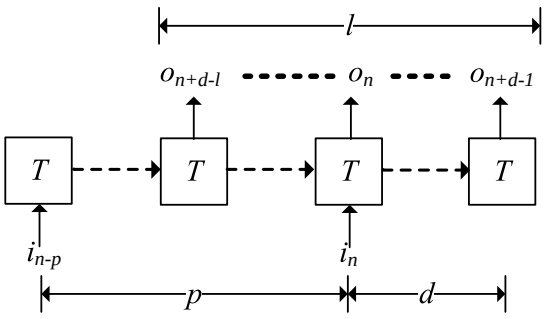

Fig. 2. The parameterized complementary condition

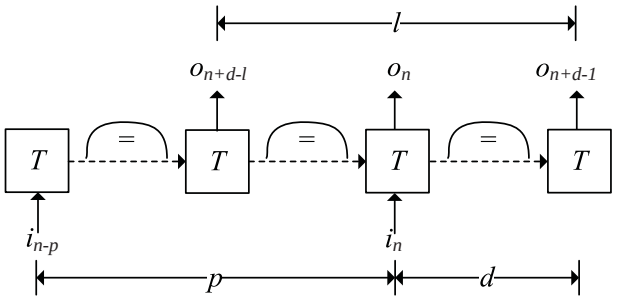

Fig. 3. The loop-like non-complementary condition

According to line 2 and 3 of Equation (3), there are actually two unfolding of transition function T, so we need to detect these loop-like paths on both of them, i.e., on the miter machine $M \times M$ defined below:

***Definition** 3:* **Miter machine :** For Mealy machine $M = (S, s_0, I, O, T)$, its Miter machine is $M^2 = (S^2, s_0^2, I^2, O^2, T^2)$, where

1) $S^2 = S \times S$
2) $s_0^2 = s_0 \times s_0$
3) $I^2 = I \times I$
4) $O^2 = O \times O$
5) T^2 is defined as $(< s_{m+1}, s'_{m+1} >, < o_m, o'_m >) = T^2(< s_m, s'_m >, < i_m, i'_m >)$ with $(s_{m+1}, o_m) = T(s_m, i_m)$ and $(s'_{m+1}, o'_m) = T(s'_m, i'_m)$.

Thus, we define the loop-like non-complementary condition below to determine the non-existence of E^{-1}:

***Definition** 4:* **Loop-like Non-complementary Condition (*LN*) :** For encoder E and its Mealy machine $M = (S, s_0, I, O, T)$, assume its miter machine is $M^2 = (S^2, s_0^2, I^2, O^2, T^2)$, then $E \vDash LN(p, d, l)$ holds if i_n can not be uniquely determined by $< o_{n+d-l}, \ldots, o_{n+d-1} >$ on state transition sequence $< s_{n-p}, \ldots, s_{n+d-1} >$, and there are loops on $< s^2_{n-p}, \ldots, s^2_{n+d-l} >$,$< s^2_{n+d-l}, \ldots, s^2_n >$ and $< s^2_{n+1}, \ldots, s^2_{n+d} >$. This equals the satisfiability of $F_{LN}(p, d, l)$ in Equation (4). We further define $E \vDash LN$ as $\exists p, d, l : E \vDash LN(p, d, l)$.

$$F_{LN}(p,d,l) \stackrel{def}{=} \left\{ \begin{array}{c} \bigwedge_{m=n-p}^{n+d-1}\{(s_{m+1}, o_m) \equiv T(s_m, i_m)\} \\ \wedge \quad \bigwedge_{m=n-p}^{n+d-1}\{(s'_{m+1}, o'_m) \equiv T(s'_m, i'_m)\} \\ \wedge \quad \bigwedge_{m=n+d-l}^{n+d-1} o_m \equiv o'_m \\ \wedge \quad i_n \neq i'_n \\ \wedge \bigvee_{x=n-p}^{n+d-l-1} \bigvee_{y=x+1}^{n+d-l}\{s_x = s_y \wedge s'_x = s'_y\} \\ \wedge \bigvee_{x=n+d-l}^{n-1} \bigvee_{y=x+1}^{n}\{s_x = s_y \wedge s'_x = s'_y\} \\ \wedge \bigvee_{x=n+1}^{n+d-1} \bigvee_{y=x+1}^{n+d}\{s_x = s_y \wedge s'_x = s'_y\} \end{array} \right\} \quad (4)$$

By comparing Equation (3) and (4), it is obvious that their only difference is the last three newly inserted lines in (4), which will be used to detect loops on the following three state sequences:

$$\begin{array}{c} Prefix_{p,d,l} =< s^2_{n-p}, \ldots, s^2_{n+d-l} > \\ Left_{p,d,l} =< s^2_{n+d-l}, \ldots, s^2_n > \\ Right_{p,d,l} =< s^2_{n+1}, \ldots, s^2_{n+d} > \end{array} \quad (5)$$

The correctness of this approach will be proved in next subsection.

B. Proving Correctness

Before proving correctness of our approach, we need some lemmas.

***Lemma** 1 ():* For two tuples $< p, d, l >$ and $< p', d', l' >$, if $Prefix_{p',d',l'}$,$Left_{p',d',l'}$ and $Right_{p',d',l'}$ are respectively longer than $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$, then $E \vDash PC(p, d, l) \rightarrow E \vDash PC(p', d', l')$.

Proof: It is obvious that $F_{PC}(p, d, l)$ is a sub-formula of $F_{PC}(p', d', l')$, so the unsatisfiability of former one implies the unsatisfiability of latter one. Thus, $E \vDash PC(p, d, l) \rightarrow E \vDash PC(p', d', l')$ holds. ■

The following two theorems will respectively prove that

1) For any E, PC and LN can not hold for the same time.
2) For every E, either PC or LN holds. That is, there does not exist a third possibility.

***Theorem** 1 ():* For any E, PC and LN can not hold for the same time. That is, $E \vDash LN \rightarrow \neg\{E \vDash PC\}$

Proof: Let's prove by contradiction. Assume that $E \vDash LN$ and $E \vDash PC$ both hold. Then there exist $< p, d, l >$ and $< p', d', l' >$, such that $E \vDash PC(p, d, l)$ and $E \vDash LN(p', d', l')$.

On one hand, $E \vDash LN(p', d', l')$ implies that there are loops in state sequences $Prefix_{p',d',l'}$,$Left_{p',d',l'}$ and $Right_{p',d',l'}$. By expanding these loops, we can get another tuple $< p", d", l" >$, such that $Prefix_{p",d",l"}$,$Left_{p",d",l"}$ and $Right_{p",d",l"}$ are respectively longer than $Prefix_{p',d',l'}$,$Left_{p',d',l'}$ and $Right_{p',d',l'}$. and $E \vDash LN(p", d", l")$, which means $F_{LN}(p", d", l")$ is satisfiable.

$F_{PC}(p", d", l")$ is a sub-formula of $F_{LN}(p", d", l")$, so $F_{PC}(p", d", l")$ is also satisfiable, which means that $E \vDash PC(p", d", l")$ does not hold.

On the other hand, according to Lemma 1, we know that $E \vDash PC(p", d", l")$ holds.

This contradiction concludes the proof. ■

***Theorem** 2 ():* For every E, either PC or LN holds. That is, $E \vDash LN \leftarrow \neg\{E \vDash PC\}$

Proof: Let's prove by contradiction. Assume that $E \vDash LN$ and $E \vDash PC$ both do not hold. Then for every $< p, d, l >$ and $< p', d', l' >$, $F_{PC}(p, d, l)$ is satisfiable, while $F_{LN}(p', d', l')$ is unsatisfiable.

Thus, assume the $uirrd(M^2)$ is the uninitialized state variables recurrence diameter of E's Miter machine. Let's define $< p, d, l >$ as:

$$\begin{array}{c} p = uirrd(M^2) * 2 + 2 \\ d = uirrd(M^2) + 1 \\ l = uirrd(M^2) + 1 \end{array} \quad (6)$$

With this definition, it is obvious that the length of $Prefix_{p,d,l}$,$Left_{p,d,l}$ and $Right_{p,d,l}$ are both longer than $uirrd(M^2)$. This means that there are loops in these three paths, which will make $F_{LN}(p, d, l)$ satisfiable. This will contradict with the fact that $F_{LN}(p', d', l')$ is unsatisfiable for every $< p', d', l' >$.

This contradiction concludes the proof. ■

Theorem 1 and 2 means that, we can enumerate all combination of $< p, d, l >$ from smaller one to larger one, and check $E \models PC(p,d,l)$ and $E \models LN(p,d,l)$ in every iteration. This process will eventually terminate with one and only one answer between PC and LN. This implementation will be presented in the next subsection.

C. Algorithm Implementation

Algorithm 1 check_PCLN

```
1: for bound = 1 → ∞ do
2:   for p = 0 → bound − 1 do
3:     for r = 0 → bound − 1 − p do
4:       l = bound − p
5:       if F_PC(p, d, l) is unsatisfiable then
6:         print "E^{-1} exists with < p, d, l >"
7:         halt;
8:       else if F_LN(p, d, l) is satisfiable then
9:         print "E^{-1} does not exist"
10:        halt;
11:      end if
12:    end for
13:  end for
14: end for
```

According to Theorem 1 and 2, Algorithm 1 will eventually terminate at line 6 or 9.

IV. Removing Redundant Output Letters

Although Algorithm 1 is sufficient to determine the existence of E^{-1}, the parameters found by line 6 of Algorithm 1 contain some redundancy, which will cause unnecessary large overhead on run time of characterizing and circuit area.

For example, as shown in Figure 4, assume that l is the smallest parameter value that leads to $E \models PC(p,d,l)$, and $l < d$, which means that i_n is uniquely determined by some output letters o_k with $k > n$.

We further assume that line 6 of Algorithm 1 find out $E \models PC(p,d,l')$. It is obvious that $l' > d$, which make i_n to depend on some redundant o_k with $k \leq n$.

So $o_{n+d-l'}^{n+d-l-1}$ is the sequence of redundant output letters, which should be removed to prevent them from being instantiated as latches in circuit E^{-1}.

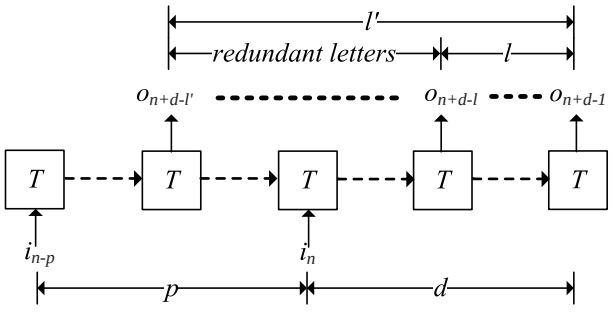

Fig. 4. Redundant Output Letters

At the same time, also according to Figure 4, larger p and d also generate larger SAT instance for the characterization algorithm.

So, we use Algorithm 2 to minimize $< p, d, l >$ before passing it to the characterization algorithm:

Algorithm 2 $RemoveRedundancy(p,d,l)$

```
1: for p' = p → 0 do
2:   if F_PC(p' − 1, d, l) is satisfiable then
3:     break
4:   end if
5: end for
6: for d' = d → 0 do
7:   if F_PC(p', d' − 1, l) is satisfiable then
8:     break
9:   end if
10: end for
11: for l' = l − (d − d') → 0 do
12:   if F_PC(p', d', l' − 1) is satisfiable then
13:     break
14:   end if
15: end for
16: print "final result is < p', d', l' >"
```

V. Experimental Results

We have implemented this algorithm with the OCaml language, and solved the generated SAT instances with Zchaff SAT solver [4]. All experiments are run on a PC with a 2.4GHz Intel Core 2 Q6600 processor, 8GB memory and CentOS 5.2 linux. All experimental results and programs can be downloaded from http://www.ssypub.org.

A. Benchmarks

Table I shows information of the following benchmarks.

1) A XGXS encoder compliant to clause 48 of IEEE-802.3ae 2002 standard [10].
2) A XFI encoder compliant to clause 49 of the same IEEE standard.
3) A 66-bit scrambler used to ensure that a data sequence has sufficiently many 0-1 transitions , so that it can run through high-speed noisy serial transmission channel.
4) A PCIE physical coding module [11].
5) The Ethernet module of Sun's OpenSparc T2 processor.

TABLE I
Information of Benchmarks

	XGXS	XFI	scrambler	PCIE	T2 ethernet
Line number of verilog source code	214	466	24	1139	1073
#regs	15	135	58	22	48
Data path width	8	64	66	10	10

TABLE II
EXPERIMENTAL RESULTS ON PROPERLY DESIGNED ENCODERS

		XGXS	XFI	scrambler	PCIE	T2 ethernet
[3]	time chk PC(sec)	1.06	70.52	5.74	2.40	66.37
	d, p, l	1,1,1	0,3,2	0,2,2	2,1,1	4,1,1
ours	time chk PC(sec)	0.39	21.31	3.27	0.61	65.34
	d, p, l	1,2,1	0,3,2	0,2,2	2,2,1	4,2,1

B. Determining the existence of decoder for Properly Designed Encoders

The 2nd and 4th rows of Table II compares the run time of checking $E \vDash PC$ between [3] and our approach. It is obvious that our approach is much faster than that of [3].

The 3rd and 5th rows compare the discovered parameter values, and some minor differences are found on parameter p. This is caused by the different orders in checking various parameter combinations.

C. Determining the existence of decoder for Improperly Designed Encoders

To further show the usefulness of our algorithm, we need some improperly designed encoders without corresponding decoders.

We obtained these improperly designed encoders by modifying each benchmark's output statements, such that they can explicitly output the same letter for two different input letters. In this way, input letter i_n can never be uniquely determined by E's output sequence.

The 2nd row of Table III shows the result of [3], while the 3rd row shows the result of our algorithm. This results indicate that our algorithm always terminated correctly, and is much faster than [3].

VI. CONCLUSIONS AND FUTURE WORKS

This paper proposes a faster halting algorithm that checks whether a particular encoder has corresponding decoder. Theoretical analysis shows that our approach always distinguishes correct encoders from their incorrect variants and halts properly. Experimental results shows that our approach is much faster than that of [3].

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for their hard work.

This work was fund by project 60603088 supported by National Natural Science Foundation of China, and the Program for Changjiang Scholars and Innovative Research Team in University No IRT0614.

TABLE III
COMPARING RUN TIME OF IMPROPERLY DESIGNED ENCODERS

	XGXS	XFI	scrambler	PCIE	T2 ethernet
[3](sec)	0.98	35.08	2.54	1.36	17.39
ours(sec)	0.33	14.44	2.43	0.56	10.14

REFERENCES

[1] ShengYu Shen, JianMin Zhang, Ying Qin, SiKun Li. Synthesizing Complementary Circuits Automatically. in ICCAD'09, pp 381-388, 2009.
[2] ShengYu Shen, Ying Qin, KeFei Wang, LiQuan Xiao, JianMin Zhang, SiKun Li. Synthesizing Complementary Circuits Automatically. IEEE transaction on CAD of Integrated Circuits and Systems, 29(8):1191-1202, 2010.
[3] ShengYu Shen, Ying Qin, JianMin Zhang, SiKun Li. A Halting Algorithm to Determine the Existence of Decoder. in FMCAD'10, pp xx-xx, 2010. http://www.ssypub.org/ssy_fmcad10.pdf
[4] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. Chaff: Engineering an Efficient SAT Solver. In DAC'01, pp 530-535, 2001.
[5] João P. Marques Silva, Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. in ICCAD'96, pp 220-227, 1996.
[6] E. Goldberg, Y Novikov. BerkMin: A Fast and Robust Sat-Solver. in DATE'02, pp 142-149, 2002.
[7] N. Eén, N. Sörensson. Extensible SAT-solver. in SAT'03, pp 502-518, 2003.
[8] D. Kroening, Ofer Strichman. Efficient Computation of Recurrence Diameters. in VMCAI'03, pp 298-309, 2003.
[9] Mealy, George H. A Method for Synthesizing Sequential Circuits. Bell Systems Technical Journal v 34, pp 1045-1079, 1955.
[10] *IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks Specific requirements Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/s Operation*, IEEE Std. 802.3, 2002.
[11] PCI Express Base Specification Revision 1.0. Download from http://www.pcisig.com