A Dyadic Transformation Based Methodology to Achieve Coverage Driven Verification Goal

Swapnajit Mitra
Senior Manager, Hardware Engineering
Broadcom Limited
1320 Ridder Park Dr.
San Jose, CA 95131

***Abstract*- Cover property coverage is an essential measure for verification closure today. In this paper, a symbolic computation based methodology for analyzing why a cover property is not hitting in simulation is shown. The methodology is built on a dyadic transformation to reduce a sequence expression to a form that aids the analysis process.**

***Keywords*- Cover property, functional coverage, conjunctive normal form, SystemVerilog property, sequence.**

I. INTRODUCTION

Coverage driven verification has become the leading functional verification methodology to scale the enormity of the modern digital logic verification task. Cover property [1] construct provides a quantitative metric in a coverage driven verification methodology. Cover properties, often embedded within RTL code of a design, show if specific properties of that design have been exercised or 'hit' during functional simulation. A common observation is that majority of the cover properties in a constrained random verification environment are hit simply by running enough simulation seeds without needing any extra effort. However, a small number of the cover properties would not be hit easily and would need development of specific constraint (using the so-called 'directed random' approach) to hit these hard to achieve coverage targets. This is often a time consuming process involving multiple iterations between various RTL designers and verifiers. The multi-level process often involves steps, such as, how to constraint the stimuli, how to implement those constraints, running simulations to generate the waveform and coverage reports, checking if the coverage point under consideration has been hit or not, and if not, examining the waveform and re-evaluating the constraints, and possibly modifying them again. The overall process involves human effort and is prone to human errors.

This paper describes a methodology based on a dyadic transformation to reduce, if not eliminate, the need of human involvement in creating the specification of the constraint that would hit a currently un-hit cover property. It points out exactly which conditions have to be met simultaneously on a given clock cycle in order for the cover property to be hit.

Although this paper describes the proposed method for cover properties, it can be applied equally for any type of property (cover, assert or assume).

The organization of this paper is as follows. Section II describes prior work done in related fields. Section III introduces mathematical foundation for the work. Section IV shows a flowchart of the algorithm and Section V gives several examples in gradual difficulties. Finally, Section VI draws the conclusion.

II. PRIOR WORKS

To the best of the knowledge of the author, there is no previous work on conversion of literals spreading over two dimensional temporal and event space to one dimensional event space using Conjunctive Normal (i.e., AND of ORed expressions) and Disjunctive Normal (OR of ANDed expressions) Forms (CNF/DNF) based decision making in a property based environment that will be proposed in this paper. However, the foundation of this work is rooted to fundamental theories of symbolic computation.

While the generic Arithmetical Normal Form (ANF) of writing Boolean expression has been established since early 1900s, the implication of using CNF/DNF for symbolic computation were not brought to light before seminal work done by Zhegalkin [4]. This foundation was further utilized by other ground breaking works, such as, by Karnaugh [6] and Veitch [5] in what is known as Karnaugh map today and by Quine and McCluskey [7][8] for Quine–McCluskey algorithm.

III. MATHEMATICAL MODEL

Definition: A *Nominally Expressed Property* (NEP) in this paper is defined as a concurrent property $P(e(\{v\}), T)$ whose property expression is a generalized heterogeneous combination of conjunctive normal [2] and disjunctive normal forms [9] involving temporal or immediate events e on the set of variables $\{v\}$ with T as the time horizon for the life cycle of a single spawn of the property.

For this paper, any cover property under consideration is assumed to comply with the generic definition of NEP with no inherent constraint of any normality or simplification whatsoever. Assuming the form of the cover property to be generic also ascertains the generality of the methodology developed in this paper.

Most of the difficulty in determining the reason why a property is not hit comes from the complexity of the analysis of each sequence of the property expression and how those sequences interact with one another to eventually make the hit happen. If such a cover property expression can be represented in CNF, the analysis becomes much easier. In that case, it would imply that all components of the conjunctivity must be true for a cover property to be hit at a given clock tick. It follows similarly that at least one of the disjunctive components within each conjunctive component must be true to make the cover property hit. If a property expression can be represented in CNF, it will take a very simple analysis on the waveforms for each components of conjunctivity.

It should be noted that this method of analysis can be extended to assert properties (or assertions) as well. In such cases, the conjunctive components must *not* be all true at the same time per design intent. The same line of reasoning can be used for detecting why an assertion has been hit.

However, while any normalized boolean expression can be represented in a CNF, there exists no such method for an expression involving temporal transformation, such as, ##, |=>, #=#, and #-# in SystemVerilog [3] parlance, as it occurs in a property expression.

To achieve this, we introduce *Temporal Transformation Function* (TTF) $\Phi^{m,n}(c):c \leftarrow (c, m, n)$ as a function that returns true if any of the set of values a literal c had from m to n clock cycles earlier was true. Thus, $\Phi^{n,n}(c):c \leftarrow (c, n, n)$ represents if a literal c was true exactly n clock cycles earlier. For simplicity, shorthand of $\Phi^{n}(c)$ is used for the rest of the paper to represent $\Phi^{n,n}(c)$.

The TTF $\Phi^{m,n}(c)$ can be implemented very easily using common SystemVerilog constructs. An example implementation of this is shown in Fig. 2 with implicit clocking.

We define $\mathbb{T}$ as a dyadic transformation that maps an NEP compliant expression to a CNF involving a new set of events e' involving a transformed set of variables v' in the same time horizon T.

$$\mathbb{T}\ (e(\{v\}), T) \rightarrow \Lambda e'(\{v'\}, T) \tag{1}$$

The two parts that the dyadic transformation $\mathbb{T}$ consists of are

- Part A) the transformation on the variables that do not need any temporal transformation and
- Part B) on the ones that do.

Since any normalized boolean expression can be represented in a CNF without any change in the set of variables, v' can be expressed as a union of a proper subset of v and a converted form of rest of the variables of v through function Φ for the corresponding values of n (or m, n) associated with each variable. Or, in other words:

$$v' = s_1 \cup s_2 \mid s_1 \subset \{v\},\ s_2 = \{\ \Phi^{m,n}(\{v\}\text{-}s_1)\} \tag{2}$$

Substituting (2) in (1), we get,

$$\mathbb{T}\ (e(\{v\}), T) \rightarrow \Lambda e'(\{s_1 \cup s_2 \mid s_1 \subset \{v\},\ s_2 = \{\ \Phi^{m,n}(\{v\}\text{-}s_1)\}\}, T) \tag{3}$$

Using the formulation of T in (3), any property expression can be converted to its corresponding CNF.

Theorem 1*: A TTF $\Phi^{n}(c)$ is distributive over c if c is in CNF*.

Proof: Using $\vee$ and $\wedge$ to represent conjunction (i.e., logical AND) and disjunction (logical OR) operations, it is easy to see from Fig. 3 that

$$\Phi^{n}(\bigvee x_{ij}) = \Phi^{n}(x_1 \vee x_2 \vee \ldots) = \Phi^{n}(x_1) \vee \Phi^{n}(x_2) \vee \ldots \tag{4}$$

And similarly,

$$\Phi^{n}(\bigwedge x_{ij}) = \Phi^{n}(x_1 \wedge x_2 \wedge \ldots) = \Phi^{n}(x) \wedge \Phi^{n}(x_2) \wedge \ldots \tag{5}$$

Thus, applying (4) and (5) for a TTF on a CNF, we arrive at the rather simple result:

$$\Phi^n(\Lambda V x_{ij})=\Lambda(\Phi^n(V x_{ij}))=\Lambda V \Phi^n(x_{ij}) \quad (6)$$

Thus, a TTF is distributive over a CNF transformation [*QED*].

IV. Flowchart of the Algorithm

To summarize the mathematical model of the dyadic transformation developed in the previous section, the central theme of the algorithm is to convert any sequence expression into a CNF or DNF to determine on a specific clock cycle why a cover property has not been hit. Then the process can be repeated over the entire simulation time range. To avoid any repetition, for the rest of the paper, only CNF will be considered since the DNF will follow a similar but complementary path of analysis. While converting an immediate expression to CNF is trivial following classical logic analysis, a generic NEP may contain temporal expressions and the following flowchart in Fig. 1 summarizes how the TTF-based methodology developed in the previous section can be applied for these cases. Once a sequence expression is converted into a CNF, the sequence expression can only be true if each of the components of CNF is true.

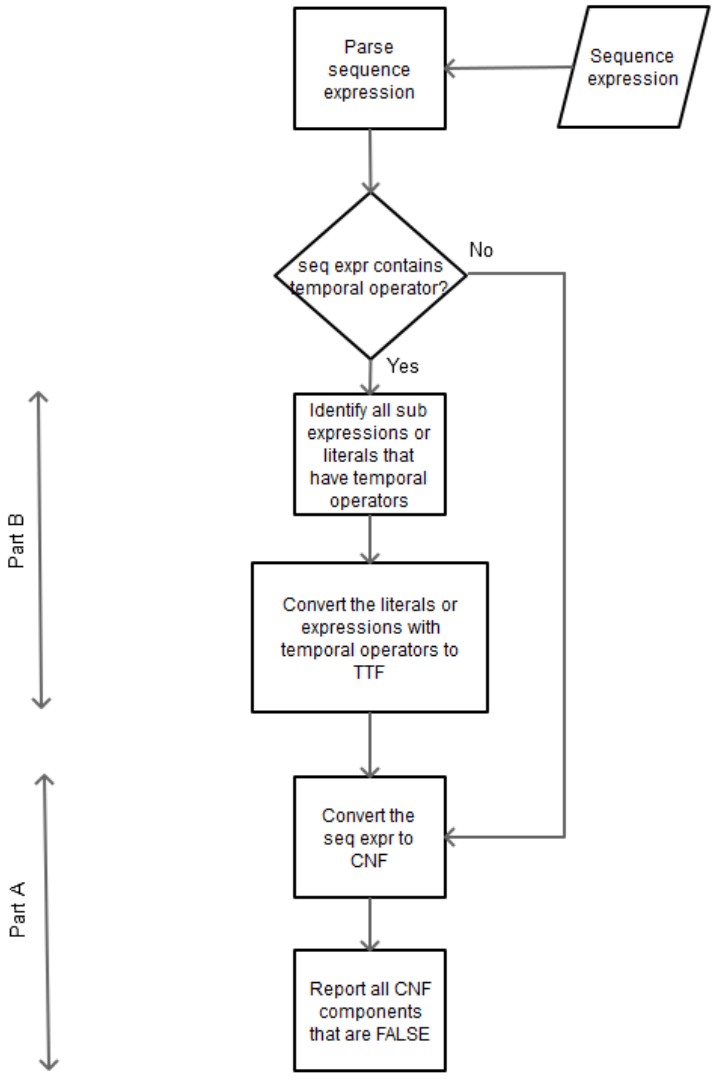

Figure 1: Flowchart of the methodology

As shown in the flowchart, a sequence expression parser receives the sequence expression. It first determines if the sequence expression has any sub-expression or literal that is operated by or preceded by a temporal expression. Without loss of generality, this paper primarily focuses on cycle delay (##`*N* or ##[`*M*:`*N*]) and implication (|-> and |=>) operations. However, the underlying theory remains the same with other temporal operators and expressions.

If the input expression has any temporal operator expression, the next steps are to identify each of them and then to create a corresponding TTF transformation for the literal or subexpression that is affected by the temporal expression. This constitutes Part B of the dyadic transformation.

After this part, or if the input sequence expression does not have any temporal expression, the next step is to create an equivalent CNF expression for the input sequence expression. Once the equivalent CNF expression is created, a logic parser analyzes each component of the CNF expression and finds out components that are false for a specific clock cycle. These components are responsible for the cover property to remain un-hit for that clock cycle. It then creates the report on the finding. These steps constitute Part A of the dyadic transformation.

V. Examples

Let *cp* be a cover property consisting of the sequence *seq* which has a sequence expression *s* (see Fig. 4). `*M* and `*N* are assumed to be macros for a positive integers such that `*M*>`*N*.

```
function bit ttf (bit c, int m, int n=-1);
bit equality;
int i;
if (n == -1) begin // if only m has been passed
   ttf = $past(c, m); // note that some simulators need constant m to be passed
end else begin // if both m and n have been passed
   equality = 1'b0;
   for (i=m; i<=n; i++) begin
      equality = equality | $past(c, i);
      if (equality == 1'b1) begin
         i = n+1;
      end
   end
   ttf = equality;
end
endfunction
```

Figure 2: SystemVerilog implementation of TTF equality

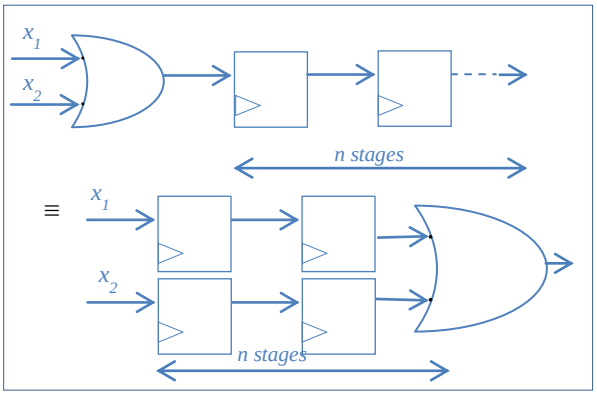

Figure 3: Φn(x1vx2v…) = Φn(x1)vΦn(x2) v…

```
sequence seq;
   s;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 4: Example cover property

Several examples are presented below from simple to more complex cases to illustrate the methodology. The examples have been deliberately chosen to show the cases in the minimalist way.

Case 1: s: (a∧c∨b∧c)

In this case, *s* has no temporal transformation (i.e., there is no Part A, and the transformation has Part B only) and corresponding CNF can be derived using only non-temporal transformation for *s* which is *c(a∨b)*. If *cp* is unhit, to hit *cp* both *c* and *(a∨b)* need to be hit. Since *(a∨b)* can be hit if either *a* or *b* is hit, *c* and one of *a* or *b* need to be hit.

```
sequence seq;
(write_en & data_valid) ||
(write_en && (retire_address[0:4]==addr));
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 5: Example of Case 1

An example of this is shown in Fig. 5. It is trivial to see in this example that `write_en` and either `data_valid` must be valid or `retire_address[0:4] == addr`) must be true.

Case 2: s: a∨b∧c

In this case also, *s* has no temporal transformation and corresponding CNF for *s* is *(a∨b)∧(a∨c)*. In order to hit *cp* both *(a∨c)* and *(a∨b)* need to be hit. Since *(a∨b)* can be hit if either *a* or *b* is hit, and *(a∨c)* can be hit if either *a* or *c* is hit, a or both *b* and *c* need to be hit.

Fig. 6 shows an example of this where the CNF is (`(irdy==0)|| $fell(trdy) && (irdy==0) || $fell(stop))`. So, either `(irdy==0)` or `$fell(trdy)` *and* either `(irdy==0)` or `$fell(stop)` must be true in order to hit the cover property.

```
sequence seq;
(irdy==0)|| $fell(trdy) && $fell(stop);
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 6: Example of Case 2

Case 3: s: a ##`N b

In this case, *s* does have a temporal transformation on *a*. Here, *s* can be expressed as a CNF as *ttf(a,`N)∧b*. To hit *cp* in this case, *ttf(a, `N)* and *b* need to be hit.

Two examples of this are shown in Fig. 7 through Fig. 13. In Fig. 7, a simple sequence expression involving one delay parameter (##5) is shown.

```
sequence seq;
a ##5 b;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 7: Sequence expression for Case 3 involving simple delay

This can be rewritten using the corresponding TTF as in Fig. 8. Thus, in order to hit *cp*, *ttf (a, 5)* and *b* must be true.

```
sequence seq;
ttf(a, 5) & b;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 8: Rewritten sequence expression using CNF for Case 3.

This is illustrated in the waveform diagram in Fig. 9.

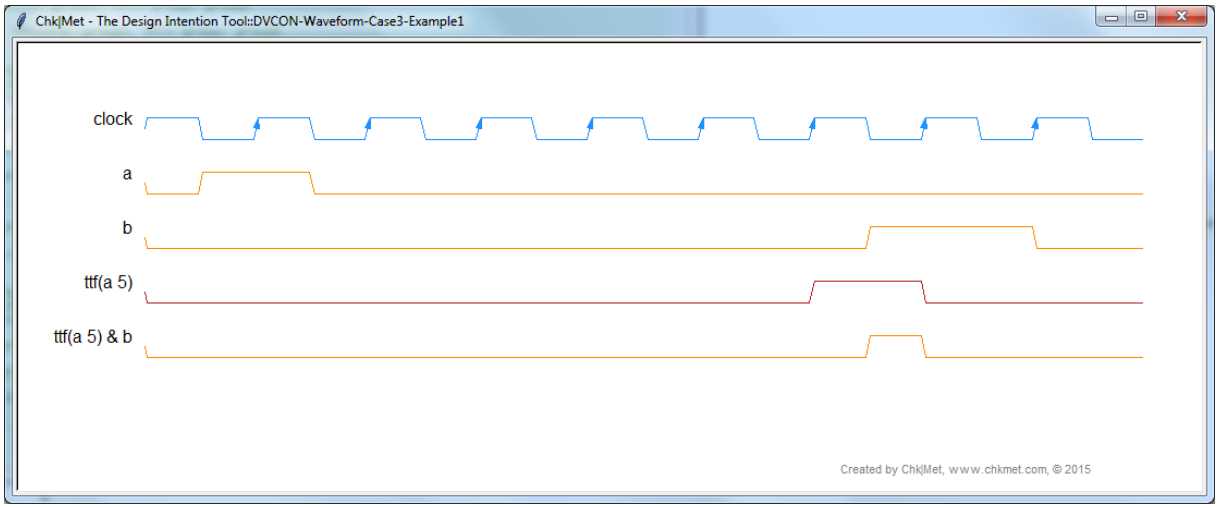

Figure 9: Timing Diagram for Case3, Example 1

Note that, *a* or *b* as literals can be sequence expressions themselves and thus, previous rules of redistribution as demonstrated in cases 1 and 2 will be still valid. One such example is given in Fig. 10.

```
sequence seq;
(a | b) ## 3 c;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 10: Example of Case 3 where a literal is a sequence expression itself

Here, the sequence expression *(a | b) ## 3 c* can be represented by equivalent CNF *ttf(a | b, 3) & c* (Fig. 11). To hit *cp* in this case, *ttf(a | b, 3)* and *c* need to be hit.

```
sequence seq;
ttf(a | b, 3) & c;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 11: CNF representation of the sequence expression in Fig. 10

This is illustrated in the waveform diagram in Fig. 12.

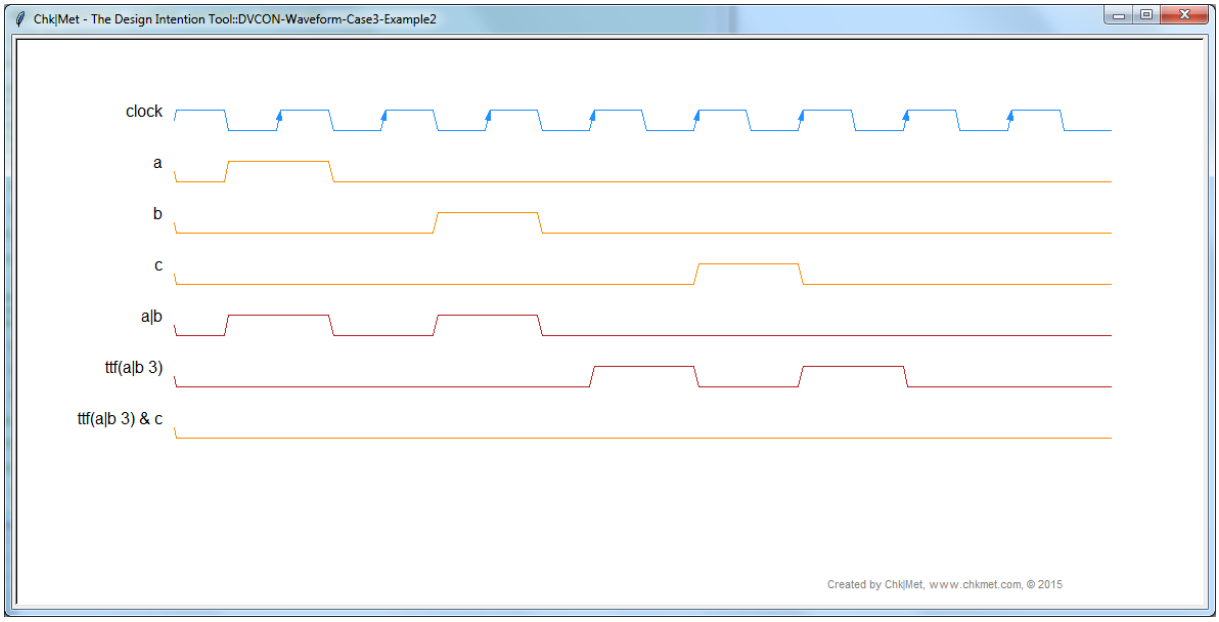

Figure 12: Timing Diagram for Case3, Example2

Alternately, using **Theorem 1**, the CNF can be further simplified to *ttf(a, 3) & c | ttf(b, 3) & c*. Thus, either *ttf(a, 3) & c* or *ttf(b, 3) & c* must be true (Fig. 13).

```
sequence seq;
ttf(a, 3) & c | ttf(b, 3) & c;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 13: Theorem 1 applied to the example of Fig. 10

Case 4: s: a ## [`M:`N]b

In this case, *s* has a temporal transformation on *a* over a spread of *`M* to *`N* clock cycles. Here, *s* can be expressed as a CNF as *ttf(a,`M,`N)* ∧ *b*. To hit *cp* in this case, *ttf(a, `M,`N)* and *b* must be hit.

This case is similar to the first example of Case 3 (Fig. 7) and the corresponding CNF form is shown in Fig. 14.

```
sequence seq;
ttf(a, m, n) & b;
endsequence
cp: cover property @(posedge clk) seq (m, n);
```

Figure 14: Example CNF for Case 4

Case 5a: s: a |-> ##[`M:`N]b

It should be noted that |-> is not a temporal transformation by itself. Thus, s in this case will be same as *ttf(a,`M,`N) |-> b*. But analyzing this further, it is easy to see that for cover property the non-vacuous success case will be *ttf(a,`M,`N)*∧*b*. Thus, to hit *cp*, both *ttf(a,`M,`N)* and *b* must be hit.

An example sequence expression after CNF transformation in this case will be the same as shown in Fig. 13.

Case 5b: s: a |=> ##[`M:`N]b

This is same as *a ##[`M+1:`N] `true |-> b*.

```
sequence seq;
a ##1 !b ##[2:4] b ##1 !b ##1 c;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 15: A combined example

The example in Fig. 15 shows a property definition that combines many of the above cases. The CNF for the sequence expression is shown in Fig. 16.

```
sequence seq;
ttf(ttf(ttf(ttf(a, 1) & !b, 2, 4) & b, 1) & !b, 1) & c;
endsequence
cp: cover property @(posedge clk) seq;
```

Figure 16: TTF based implementation of the combined example described in Fig. 13.

VI. Conclusion and Future Works

A methodology based on a dyadic transformation is proposed that will greatly aid, if not eliminate, human need for analysis of an un-hit cover property expression with both temporal and immediate components. A mathematical structure of the transformation is shown with associated examples.

Future work on this will include creating an automatic parser that will implement the mathematical model to report why a specific cover property is not hit in a specific clock cycle or during the entire simulation time range. Future work will also include expanding the mathematical model to deal with other temporal operators for a sequence, such as cycle range delay, consecutive repetition, etc.

Also, since the model is based on sequence and property in general, the same methodology can be used for analysis of assume or assert properties with minimal modification. This can be elaborated in future work as well.

Acknowledgment

The author likes to thank Shreyan Mayukh Mitra for help on some of the artworks. The timing diagrams used in this paper were created by *Chk|Met::The Design Intention Tool* (www.chkmet.com) and the flowchart was created by *Pencil* (pencil.evolus.vn).

References

[1] Zwolinski, Mark, Digital System Design with SystemVerilog (2009), pp 7-48 to 7-49.
[2] Hazewinkel , Michiel, ed. (2001), "Conjunctive normal form", Encyclopedia of Mathematics, Springer, ISBN 978-1-55608-010-4.
[3] Accellera, IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language (2012), pp 390.
[4] Zhegalkin, Ivan Ivanovich (1927). "On the Technique of Calculating Propositions in Symbolic Logic". Matematicheskii Sbornik. 43: 9–28.
[5] Veitch, Edward W., 1952, "A Chart Method for Simplifying Truth Functions", Transactions of the 1952 ACM Annual Meeting, ACM Annual Conference/Annual Meeting "Pittsburgh", ACM, NY, pp. 127–133.
[6] Karnaugh, Maurice, November 1953, The Map Method for Synthesis of Combinational Logic Circuits, AIEE Committee on Technical Operations for presentation at the AIEE summer General Meeting, Atlantic City, N. J., June 15–19, 1953, pp. 593–599.
[7] Quine, W. V. (Oct 1952). "The Problem of Simplifying Truth Functions". The American Mathematical Monthly. 59 (8): 521–531. doi:10.2307/2308219.
[8] McCluskey, E.J., Jr. (November 1956). "Minimization of Boolean Functions". Bell System Technical Journal. 35 (6): 1417–1444. doi:10.1002/j.15387305.1956.tb03835.x.
[9] Button, T., 2013, "Metatheory for truth functional logic". University of Cambridge, pp. 20 – 28. Also available online at the web address: http://people.ds.cam.ac.uk/tecb2/Metatheory.pdf.