Mathematical Problems of Computer Science 22, 2001, 86–90.

The Presentation of Logic Circuit's Verilog Description through Linear Description

Pavel Yu. Vasilyan

Institute for Informatics and Automation Problems of NAS RA and YSU
E-mail: pavel_v@bi-line.am

Abstract

Linear Description of circuits is the generalization of Polish notation of arithmetic expressions. The importance of the use of Linear Description of circuits is justified by the necessity of studying the functional behavior and reliability of circuit. In the article presented an algorithm is described, which acquires Linear Description from an arbitrary logic circuit's Verilog Gate-Level model.

It is known that contemporary Very Large Scale Integrated Circuits cannot be described without using computer-aided techniques. Various hardware description languages are used to define a hardware model in terms of switches, gates, RTL (Register Transfer Level) or behavioral code, as well as to synthesize and simulate circuits. Verilog HDL is among such languages, and has evolved as a standard hardware description language in hardware design. This language is famous by the simplicity of description and its universality.

In order to study the functional behavior and reliability of logic circuits, as well as for solving other problems it is convenient to use the Linear Description of circuits [1].

Thus there is a necessity for algorithms of known languages for Linear Description and opposite translation. These algorithms will enable to acquire the linear description of a given circuit, as well as conduct opposite translation after solving several problems.

Hereby, an algorithm is presented, which acquires Linear Description from a Verilog model of arbitrary gate-level circuit.

The Linear Description of circuits is the generalization of Polish notation of arithmetic expressions [2].

Polish notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators.

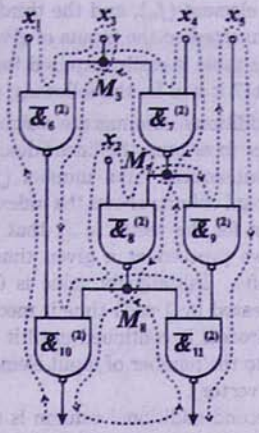
When this way is applied to gate-level circuits, the logic elements (logic gates) are recorded instead of operators, and inputs of the circuit instead of operands.

While recording arbitrary logic element, its name (logic function) is written, the number of the element as an index, and the number of inputs of the element in the parenthesis. For example, if the i-th element has 3 inputs and implements the function AND (&), then its record will be $\&_i^{(3)}$.

If the i-th element is a fanout element, then the branch point will be noted by M_i. When the i-th index occurs the first time while moving left and up, then we will write the mark $M_i^{(1)}$ and move forward. When it occurs not the first time, we will write the mark $M_i^{(0)}$ and return.

The picture shown represents the ISCAS-85 c17 Benchmark, which consists of 5 inputs, 2 outputs and 6 logic gates, each of which implements the logic function "Negation of AND" ($\overline{\&}$).

In order to acquire the linear description of the circuit, it is necessary to move up and left (towards the direction of dashed lines shown in the picture) starting from the first output, and recording each occurred element. We will not record anything while returning down and right. As a result, the linear description of the circuit will be the following:



$$\overline{\&}_{10}^{(2)}\overline{\&}_6^{(2)}x_1M_3^{(1)}x_3M_8^{(1)}\overline{\&}_8^{(2)}x_2M_7^{(1)}\overline{\&}_7^{(2)}M_3^{(0)}x_4\overline{\&}_{11}^{(2)}M_8^{(0)}\overline{\&}_9^{(2)}M_7^{(0)}x_5.$$

Let's denote the corresponding record of any circuit S by $h(S)$. The record of the circuit S is a finite word composed of the elements of alphabet $D = \{x_i, f_j^{(n)}, M_k^{(1)}, M_k^{(0)}\}$ $i, j, n, k \in \{1, 2, 3, ...\}$. Clearly, not all the words are record of any circuit. For example, there is no circuit, which has a description $M_1^{(0)}M_2^{(1)}f^{(3)}M_1^{(0)}$.

The conditions noted in [1] are necessary and sufficient for any word defined in D to be a record of some circuit.

The main advantage of the linear description of logic circuits is that there is a one-to-one correspondence between each continuous line segment weighted -1 and some sub-circuit [1].

The Verilog model of ISCAS-85 c17 Benchmark, shown in the picture above, is the following:

```
module Circuit17 (out10, out11, in1, in2, in3, in4, in5);
// I/O Port declarations
output out10, out11;
input in1, in2, in3, in4, in5;
// Internal wire declarations
wire w6, w7, w8, w9;
// Gate instructions
nand (w6, in1, in3);
nand (w7, in3, in4);
nand (w8, in2, w7);
nand (w9, w7, in5);
nand (out10, w6, w8);
nand (out11, w8, w9);
edmodule
```

A table is used to describe the process of obtaining linear description algorithmically from the Verilog model, using the above-mentioned method. The first column of the table includes the index of the element (n), the second one includes the implemented logic function

of the element (f_n), and the third column is a vector ($in_n[]$), where the indices of elements are connected to the inputs of given element.

The table is easily acquired from the description of Verilog model of the circuit. For the Circuit17 it will be the following:

Index	Logic function	Input Vector	Index of current input	Number of output branches
n	f_n	in_n	j_n	m_n
1	INPUT	$\emptyset$	0	0
2	INPUT	$\emptyset$	0	0
3	INPUT	$\emptyset$	0	1
4	INPUT	$\emptyset$	0	0
5	INPUT	$\emptyset$	0	0
6	NAND	1 (1), 3 (2)	0	0
7	NAND	3 (1), 4 (2)	0	1
8	NAND	2 (1), 7 (2)	0	1
9	NAND	7 (1), 5 (2)	0	0
10	NAND	6 (1), 8 (2)	0	0
11	NAND	8 (1), 9 (2)	0	0

Two additional columns are assigned to the table in order to acquire the linear description.

Each subsequent input number (j_n) of the first additional column shows the index of the current element for the vector $in_n[]$, that is which input shall we proceed at a given time (towards up and left). Initially its value is 0, and then it is increased by 1 each time it meets an element. This process is continued until it is less than or equal to the number of input elements $|in_n|$ (size of the vector).

The second additional column is a non-negative number and shows the number of output branches for the given element (m_n). If it is zero, then the output of the element is not branched. This number can be easily acquired by fixing the repetitions of the given index in all in_n vectors of the table.

The algorithm uses stack S (LIFO) to ensure the return. The queue Q (FIFO) is also used in the algorithm, which is given beforehand and contains all the indices of output elements of the circuit.

When the algorithm meets the subsequent element for the first time (when $j_n = 0$), it records the element index in the stack if the number of inputs is greater than 1. This is done to ensure the return process by reading the index of that particular element from the stack.

Initially, the algorithm acquires the first index of the output element from the queue Q, deletes it from the queue, and observes the whole sub-circuit based on that output element. The algorithm concludes when all the outputs are observed and the queue Q is empty.

Hereby, the block-scheme of the algorithm is presented, where the following procedures have been used:

$n \leftarrow top(Q), n \leftarrow top(S)$	Assign to n the vertex value of the queue or stack,
$push(S, n)$	Add the element n on the top of the stack S,
$pop(S)$	Delete the top value of the stack S,
$Dequeue(Q)$	Delete the top element of the queue Q,
$record(A)$	Record A at the end of the Linear Description.

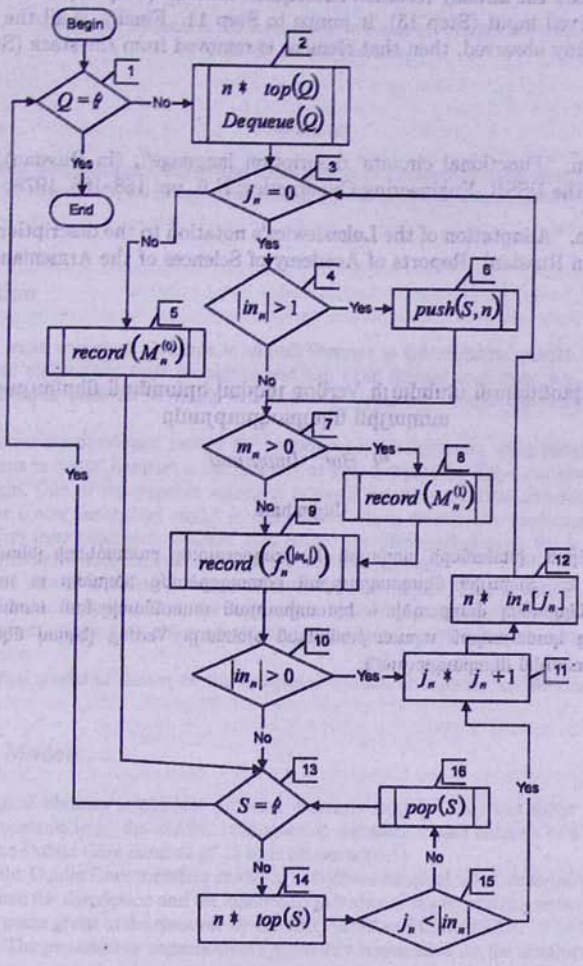
Let's describe the algorithm through a block-scheme.

In Step 1 the algorithm checks the set Q of indices of output elements. If it is empty, then the algorithm concludes.

In Step 2 the algorithm acquires the index of subsequent output element from the set Q

and removes it from the set.

In Step 3 the algorithm checks whether the given element occurs the first time or not. If not, it records the mark $M^{(0)}$ of that element in the Linear Description (Step 5), moving to Step 13. If it occurs the first time, then in Step 4 the algorithm checks the number of inputs to that element, and if that number is greater than 1, then the element is recorded in stack (Step 6), to ensure that the other inputs will be observed while returning.



In Step 7 the algorithm checks whether the output of a given element is branched or not. If not, then only that element is recorded in the Linear Description (Step 9). Otherwise, it records also the mark $M^{(1)}$ of that element (Step 8).

In Step 10 the algorithm checks the size of inputs of the current element. If it is zero,

then the given element is an input of the circuit and it is necessary to return. Otherwise, in Step 11, j_n is increased by 1 to observe the subsequent input of the element. Then the algorithm assigns to n the index of element connected to j_n-th input (Step 12), and returns to Step 3 (that is, moves up and left through the subsequent input).

In Step 13, the algorithm checks whether the stack is empty or not. If so, then the sub-circuit based on that output is already recorded, and it returns to Step 1. Otherwise, the algorithm takes the already recorded subsequent element (Step 14), and if that element contains unobserved input (Step 15), it jumps to Step 11. Finally, if all the inputs of the element are already observed, then that element is removed from the stack (Step 16).

References

[1] Sh. Bozoyan, "Functional circuits' description language", (in Russian), Academy of Sciences of the USSR, Engineering Cybernetics, N 6, pp. 158–166, 1978.

[2] Sh. Bozoyan, "Adaptation of the Lukasiewicz's notation to the description of functional circuits", (in Russian), Reports of Academy of Sciences of the Armenian SSR, vol. 63, N 4, 1979.

Տրամաբանական սխեմայի Verilog լեզվով գրառման ներկայացումը տողային նկարագրությամբ

Պ. Յու. Վասիլյան

Ամփոփում

Տրամաբանական սխեմաների տողային նկարագրությունը բանաձևերի լեհական գրառման ընդհանրացումն է: Տողային նկարագրության օգտագործումը հարմար ու արդյունավետ է սխեմաների ֆունկցիոնալ վարքագծի և հուսալիության ուսումնասիրման համար: Աշակված է ալգորիթմ, որը կամայական տրամաբանական սխեմայի Verilog լեզվով նկարագրությունը թարգմանում է տողային նկարագրության: