LOGIC CIRCUIT VISUALIZATION

J. Markovski, A. Mišev, M. Gušev

Institute of Informatics, Faculty of Natural Sciences and Mathematics,
Sts. Cyril and Methodius University,
Arhimedova bb, PO BOX 162, Skopje, Macedonia
{jasen, infolab, marjan}@ii.edu.mk

Abstract: There are a lot of tools for logic circuit design and presentation. These applications have a lot of features, although they can be applied after the user draws the circuit. In this article we give an algorithm for logic circuit visualization for known connections between the elements. The logic circuit is defined by the connections between the elements regardless the circuit type. The algorithm is presented by flowchart and its application in real world application is shown.

Keywords: logic circuit, visualization technique, simulation, signal flow

1. Introduction

Logic circuits may be combinational or sequential. A combinational circuit consists of logic gates whose outputs at any time are determined directly by the present combination of inputs regardless to previous inputs. A sequential circuit employs memory elements in addition to the logic gates. [1]

The outputs of the combinational logic circuits are determined solely from the input variables. The outputs of the sequential logic circuits depend both on the input variables and the state of the memory elements. [1]

Visualization is interpretation of data into visual form. If the configuration of a logic circuit is given by the connections between the circuit elements, the visualization algorithm will present the corresponding logic circuit. The visual form of the circuit should be as readable as possible meaning that the signal flow should be clear after a short period of examination.

2. Analysis Of The Logic Circuit Composition

Any logic circuit is defined of at most four types of elements:

1. The input variables
2. The combinational logic elements

3. The memory elements
4. The output variables

Some of the element types can be omitted for various circuit types. For example, if the circuit is a combinational logic circuit, then there are no memory elements. If the logic circuit is a sequential logic circuit, then the input variables and the combinational logic can be omitted, because the memory elements can operate without any input variables or combinational logic. In all cases at least one output variable exists.

For definition of combinational logic elements we introduce classification into two categories:

Combinational logic elements whose inputs are determined only by the state of the memory elements. These elements are referred as CAM elements (Combinational logic elements After the Memory elements).

The rest of the combinational logic. These elements are referred as CBM elements (Combinational logic elements Before the Memory elements).

Five parts can define the logic circuit in our algorithm:

1. The input variables
2. The CBM elements
3. The memory elements
4. The CAM elements
5. The output variables

Our visualization algorithm orders the elements on the drawing panel according to this classification. The reason for this kind of arrangement is the signal flow. The input variables are presented as first elements in the circuit. Then this input is processed by CBM and passed to the memory elements (In some circuits, additional feedback from the memory elements or the CAM elements is required at this position.). The CAM elements process the output of the memory elements. The outputs from any circuit element are output variables.

When all circuit elements are arranged, they are linked by wires representing the connections between the elements.

At this point we introduce a coordinate system using one X – axis. The algorithm places the input variables sequentially one after another starting from point 0. The other elements are arranged sequentially in the direction of the X – axis in the following order: the CBM elements, the memory elements, the CAM elements and the output variables.

This is presented on Fig. 1.

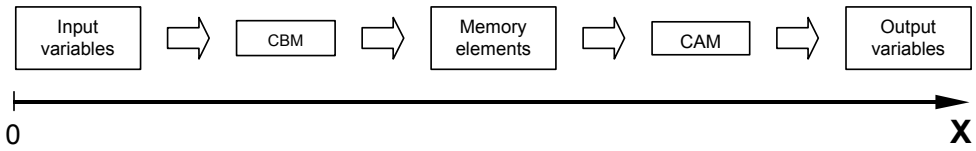

Figure 1: Scheme for visual presentation of the logic circuit

Every part of the circuit can contain an arbitrary number of corresponding circuit elements. The input and the output variables are easily arranged one after another. The combinational elements are arranged according to the inputs they require. If one combinational element requires the output from another combinational element then the first element will be placed after the second element according to the direction of the X – axis.

We define a notion of a column of circuit elements to be a set of circuit elements that have the same X coordinate. The combinational logic can be arranged in several columns. The memory elements are arranged in only one column.

RS flip-flop is a memory element with two inputs and one output. T flip-flop is a memory element with one input and one output. In order to implement RS flip-flop using T flip-flop we need two input variables and some combinational logic. The output variable is the output of the T flip-flop. We use XML as a programming tool to represent the logic circuit as presented on Fig. 2.

```
<Circuit>
   <Input ID="r">      <Sequence>0</Sequence>               </Input>
   <Input ID="s">      <Sequence>0</Sequence>               </Input>
   <CombLogic ID="NOT2" Input1="s" Type="NOT">                </CombLogic>
<CombLogic ID="AND1" Input1="NOT2" Input2="r" Type="AND">  </CombLogic>
   <CombLogic ID="NOT1" Input1="T1" Type="NOT">               </CombLogic>
   <CombLogic ID="AND2" Input1="AND1" Input2="T1" Type="AND"> </CombLogic>
   <CombLogic ID="AND3" Input1="s" Input2="NOT1" Type="AND">  </CombLogic>
   <CombLogic ID="OR1" Input1="AND2" Input2="AND3" Type="OR"> </CombLogic>
   <FlipFlop ID="T1" Input1="OR1" Type="T" Mem="0">         </FlipFlop>
   <OutPut>
      <CircuitElem ElementID="T1"> </CircuitElem>
      <CircuitElem ElementID="AND2">       </CircuitElem>
      <CircuitElem ElementID="AND3">       </CircuitElem>
      <CircuitElem ElementID="OR1"></CircuitElem>
   </OutPut>
</Circuit>
```

Figure 2: XML representation of the implementation of RS flip-flop with T flip-flop

The XML document shows the connections between the circuit elements. Element of type Input represent the input variables. Elements of type CombLogic represent the combinational logic elements and elements of type FlipFlop represent the memory elements. The output variables are references to some of the logic elements. The most important output variable is the output of the T flip-flop represented by the XML element T1.

The connections between the elements are made using the attributes `Input1` for the first input and `Input2` for the second input. For example the line:

```
<CombLogic ID="NOT2" Input1="s" Type="NOT">
</CombLogic>
```

represents that input of the combinational logic element `NOT2` is the circuit element `s`. `s` is one of the input variables in the logic circuit. By examining the XML document we notice that the output of `NOT2` is the first input of the element `AND1`, etc.

The visual form of the circuit after the application of our algorithm is presented on Fig. 3.

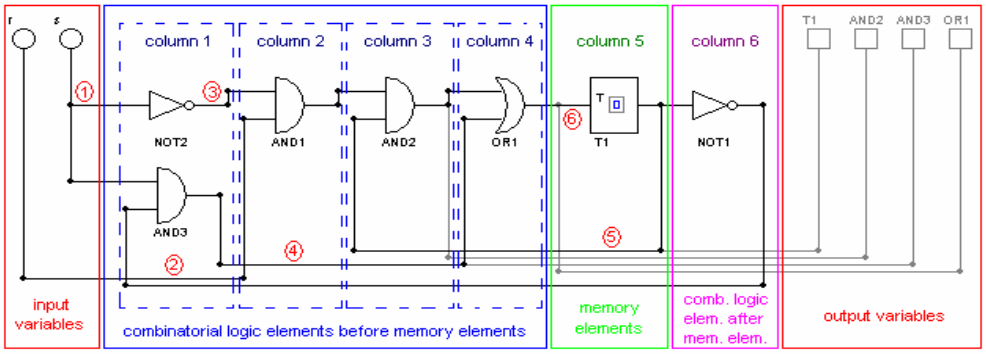

Figure 3: Implementation of the algorithm for logic circuit visualization

The input variables and the output variables are marked with red color. The CBM elements are marked with blue color. They are arranged in four columns. The memory elements are arranged in one column and they are marked with green color. The CAM elements are arranged in only one column and they are marked with pink color.

3. An Algorithm For Logic Circuit Visualization

Our logic circuit visualization algorithm is given on the Fig. 3. We use the following notations:

1. If *E* is a logic circuit element then the column in which *E* is arranged is *E.column*.
2. *Column* is an integer variable that represents the column in which the elements are being arranged.
3. *exist* is a logic variable that represents the condition if there are some un-arranged elements.

The algorithm places the elements according to the signal flow of the inputs. There are two reasons for this:

1. The natural form of visualization of a signal flow is from left to right.
2. Usually there are not too many elements whose inputs are determined by the memory elements.

When the elements are arranged into columns the wires that represent the connections are drawn. The wires are drawn using the following algorithm (see Fig. 2):

1. The inputs of the elements arranged in the first column that are determined by the input variables are represented by wires that come directly from the input variables. (This is marked on Fig. 2 with red label 1.)
2. The inputs of the elements arranged in the next columns that are determined by the input variables are represented by wires that go below the circuit elements. (This is marked on Fig. 2 with red label 2.)
3. If the input of some element is determined by the output of an element in the previous column then the wire is drawn directly between the two elements (this is marked on Fig. 2 with red label 3.), otherwise the wire is drawn below the circuit elements (this is marked on Fig. 2 with red label 4.).
4. The inputs of the CBM elements determined by the memory elements or the CAM elements are represented by wires drawn below the circuit elements. (this is marked on Fig. 2 with red label 5.).
5. The wires that represent the output variables are presented with wires with alternate color and are marked on Fig. 2 with red label 6 (they are drawn with gray color).

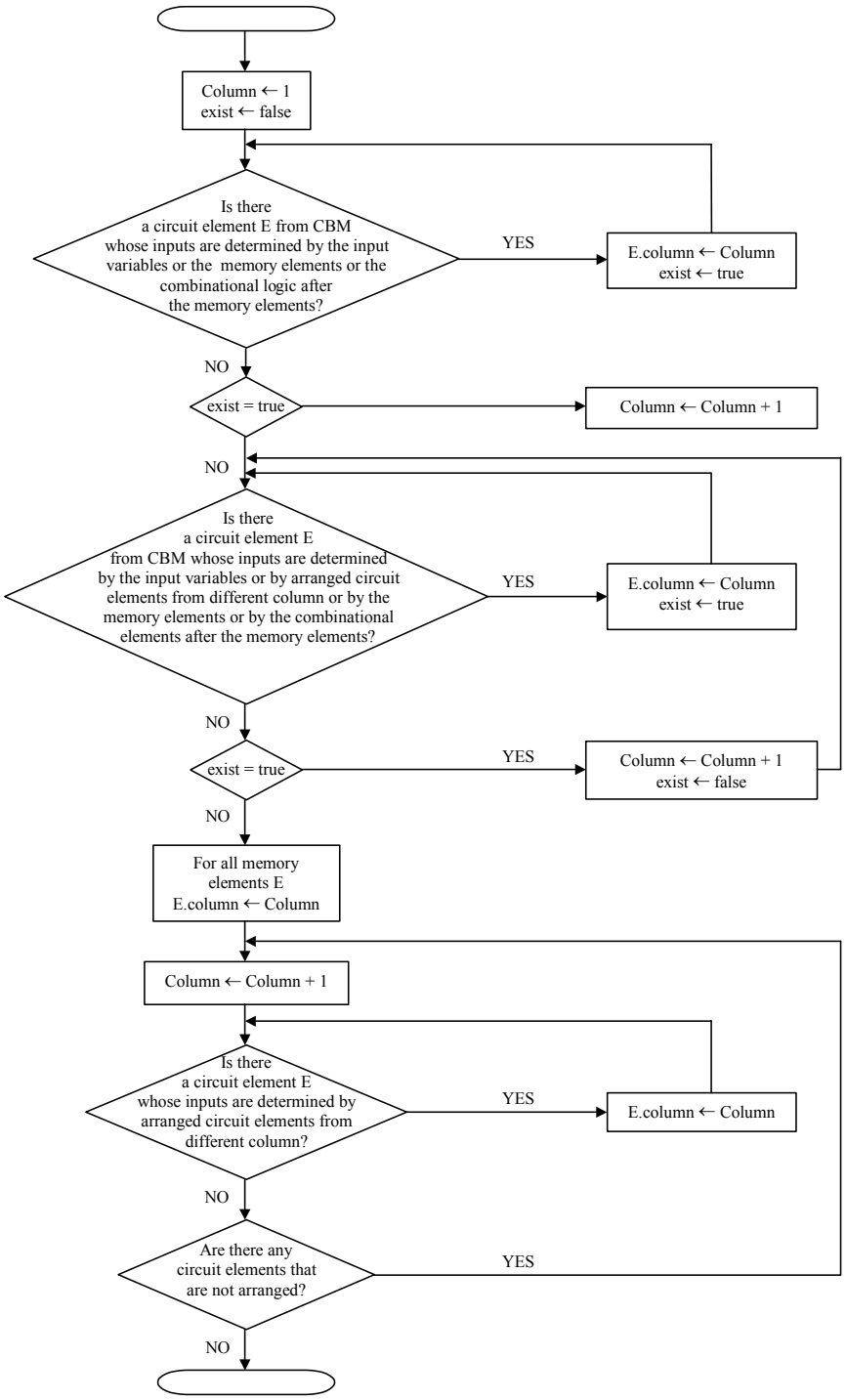

Figure 4: A flowchart of the algorithm used for logic circuit visualization

There are several commercial applications that use algorithms to rearrange logic circuits in order to minimize the intersections between the wires. However, this optimization is one of the preparations steps for printed circuits. There is no intention to present a better visual form of the logic circuit. We could not find an algorithm that considers the visualization of the logic circuit.

4. Conclusion

Logic circuit visualization algorithm is presented in this article. The logic circuit is defined by the connections between the elements.

An application for logic circuit simulation was realized to demonstrate the implementation of the algorithm. We used Java as programming language and XML as programming tool to represent the logic circuit.

We expect to improve the algorithm in several ways:

- The vertical distribution of elements in order to minimize the intersections between the wires.
- Straight lines in order to avoid rectangular folding of the wires.
- Implementation of planar graph theory in order to minimize the intersections of the wires.

5. References

1. Mano, M. M. (1991), *Digital Design*, Prentice Hall, Englewood Cliffs, New Jersey.
2. Mano, M. M. (1988), *Computer Engineering: Hardware Design,* Prentice Hall, Englewood Cliffs, New Jersey.
3. Mano, M. M. (1979), *Digital Logic and Computer Design,* Prentice Hall, Englewood Cliffs, New Jersey.
4. Flynn, M. J. (1995), *Computer Architecture: Pipelined and Parallel Processor Design,* Jones and Barlett Publishers, Sudburry, Massachutes.
5. Kuck, D. J. (1978), *The Structure Of Computers and Computations,* John Wiley & Sons, New York.
6. Hwang, K. (1979), Computer Arithmetic: Principles, Architecture and Design, John Wiley & Sons, New York.
7. Hennessy, J. L., Patterson, D. A. (1998) *Computer Architecture: A Quantitative Approach, second edition,* Morgan Kaufmann, San Francisco.
8. Patterson, D. A., Hennessy, J. L. (1998) *Computer Organization and Design, second edition,* Morgan Kaufmann, San Francisco.