DIGITAL CIRCUIT MACROMODELING WITH PSPICE

To Ching Nai Chou Sek Mak Peng Kin
(Faculty of Science & Technology, University of Macau, Macau)

Abstract Ordinary algebra is used to represent Boolean algebra on logic variables with states 0 and 1, so to achieve a unify approach to simulated both digital and analog circuit in PSPICE. Result on mixed A/D simulation shows a save in memory but generally longer run time.

Key words Mixed A/D simulation; Digital circuit macromodeling

I. Introduction

Mixed analog/digital (A/D) circuits are getting popular, giving rise to great demand on mixed A/D simulation. PSPICE, the most popular and general purpose analog circuit simulator, has been extended to simulate digital circuits since 1990. Here a separate digital simulator (which models digital circuits at the macromodel level) is employed. So A to D or D to A interface subcircuits are specially design and enable automatically inserted between analog and digital circuits to convert analog voltage to digital states (A to D) or digital state into analog voltage (D to A)[1]. So, in mixed analog/digital simulation, a large memory is needed. The author proposes a way to model digital circuits at macromodel level using PSPICE primitive so A to D and D to A interface subcircuits are no longer necessary and memory saving is achieved probably at some expense in running time. In some cases where analog and digital circuits are highly interlaced not only memory but also running time is saved.

There are three basic factors in digital macromodeling: (1) logic function, (2) time delay, (3) input, output interface. The first factor will be discussed in Section II and factors (2) and (3) will be covered in Section III.

II. Logic Function

1. It is well known that the basic building blocks of digital circuits are inverter, AND-gate, OR-gate with respective operations NOT, AND and OR form a set of functionally complete digital operations base on Boolean algebra[2,3]. Since analog circuits use ordinary algebra, the basic principle of this approach is to express Boolean algebra in terms of ordinary algebra.

First, the relation of Boolean and ordinary algebra in the three basic logical operations is depicted as follows:

Assume A and B are logical variables with states 0 and 1,
For NOT operation

\[ \overline{A} = 1 - A \]  

(1)

So, we can use “minus (-)” to express “NOT” in Boolean algebra as in Eq.(1).
For AND operation

\[ A \cdot B = A \times B \]  \hspace{1cm} (2)

So we can use "multiply (\times)" to express "AND" in Boolean algebra as in Eq.(2).

For OR operation \( A \oplus B = A + B \) when no more than one variable takes state 1 \((3a)\)

\[ \neq A + B \] when more than one variable takes state 1 \((3b)\)

In the first case, we can use "plus (+)" to express "OR" in Boolean algebra as in Eq.3(a).

But in general, by De Morgan’s Law:

\[ A \oplus B = \overline{A} \cdot \overline{B} \]  \hspace{1cm} (4)

From Eqs.(1), (2) and (4),

\[ A \oplus B = (1 - A)(1 - B) = 1 - A - B + AB = A + B - AB \]  \hspace{1cm} (5)

So, we can also express "OR" by ordinary algebra expression as in Eq.(5).

Eqs.(2), (3), (4) and (5) can be extended to multiple logic variables to model multiple input gates.

2. For more complicated digital systems, the logic relations are expressed in a truth table and logic expressions are generally expressed in “sum of products” of the input variables. It is easy to prove we can simply treat the logic expression as an ordinary algebra expression.

Without loss of generality, take XOR operation as example, it’s truth table is illustrated in Tab.1 and the logic expression is

\[ Q = \overline{A} \cdot B \oplus A \cdot \overline{B} \]  \hspace{1cm} (6)

It can be expressed in ordinary algebra as

\[ Q = (1 - A) \times B + A \times (1 - B) = A + B - 2A \times B \]  \hspace{1cm} (7)

It means “⊕” (OR) in Eq.(6) can be directly replaced by “+” (plus) in Eq.(7) and it can be verified to be correct (Tab.1).

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( Q )</th>
<th>( A + B - 2AB )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Actually, the truth table represents all the possible but not repeated states (4 states in Tab.1) of the digital system and each term in the logic expression (e.g. \( \overline{A} \cdot B \) or \( A \cdot \overline{B} \)) in Eq.(6) represents one state that gives \( Q=1 \). At any time the logic system can only exist in one state, so the two terms \( \overline{A} \cdot B \) or \( A \cdot \overline{B} \) cannot take 1 at the same time, so according to Eq.3(a), “⊕” (OR) in Eq.(6) can be replaced by “+” (plus) and Eq.(7) is correct.

3. Sometimes, some repeated terms are added to the “truth table” (also to the logic expression) to simplify hardware design.