On a mathematical model of magnetic bubble logic

by EIJI YODOKAWA
Musashino Electrical Communication Laboratory, NTT
Tokyo, Japan

INTRODUCTION

In 1967, A. H. Bobeck showed that, under suitable magnetic conditions, small discrete cylindrical magnetic domains can be stably supported in thin platelets of certain orthoferrite materials, and these domains can be moved within the material by the application of suitable external magnetic fields. These cylindrical magnetic domains, commonly referred to as bubbles, have an application in performing memory function, since the presence or absence of a bubble at a particular location can be treated as binary information. Furthermore, these bubbles can be utilized to perform logic functions, since they can be manipulated by the application of external magnetic fields as well as by the magnetostatic repulsion between adjacent bubbles.

Some remarkable features of bubble devices are the realization of both memory and logic functions in the same platelet of bubble materials, the very low power dissipation, the high storage density and the low cost. For these reasons, bubbles have been investigated extensively in recent years. Especially, the design of a bubble computer is very interesting.

This paper is concerned with a mathematical model of bubble logic. A simple model using only the bubble transfer operation was studied earlier by R. L. Graham, who showed that there exist combinational functions of 11 or more variables that cannot be computed by his model. A. D. Friedman and P. R. Menon extended Graham's model by introducing different types of operation which enabled the computation of all combinational functions, and then discussed the problem of efficient computation from the viewpoints of time and space requirements and geometrical requirements imposed by the fact that bubble interactions can occur only between physically adjacent locations.

This paper first briefly reviews Graham's results. After that further properties of his model are investigated from the viewpoints of the realization of combinational functions and geometrical requirements mentioned above. Next, Graham's model is extended by introducing a different type of bubble interaction, the magnetostatic repulsion between adjacent bubbles, which seems to be more practical. The computational capabilities are especially investigated in the present model.

The features of the model are the use of only the two types of operation, i.e., bubble transfer and conditional bubble transfer, and the restriction on space used for the computation, which meet both the engineering and economical requirements.

DESCRIPTION OF THE MODEL

Operations on bubbles

Let \( S \) be a finite set of \( n \) discrete locations at which bubbles may lie. Let \( X \) be any subset of \( S \) which is actually occupied by bubbles. It is assumed, without loss of generality, that all locations of \( S \) are adjacent to one another so that interaction between any pair of locations is possible. The following two types of operation are introduced on the set \( S \). The first type of operation is the transfer denoted by the form \( e = (a, b) \), and the second one is the conditional transfer denoted by the form \( e = (ab, c) \). These operations are called commands. The command \( e = (a, b) \) or \( (ab, c) \) transforms the locations of the bubbles from \( X \) to \( X' \) by the following definitions.

\[
\begin{align*}
\text{transfer:} & \quad X' = \begin{cases} 
(X - \{a\} \cup \{b\}) & \text{if } a \in X, \; b \in X, \\
X & \text{otherwise}
\end{cases} \\
\text{conditional transfer:} & \quad X' = \begin{cases} 
(X - \{b\} \cup \{c\}) & \text{if } a, \; b \in X, \; c \in X, \\
X & \text{otherwise}
\end{cases}
\end{align*}
\]

A program \( P \) (of length \( k \geq 0 \)) is defined to be a finite sequence \( P = e_1, e_2, \ldots, e_k \) of commands. The command \( e = (a, b) \) or \( (ab, c) \) transforms the locations of the bubbles from \( X \) to \( X' \) by the following definitions.

\[
\begin{align*}
\text{transfer:} & \quad X' = \begin{cases} 
(X - \{a\} \cup \{b\}) & \text{if } a \in X, \; b \in X, \\
X & \text{otherwise}
\end{cases} \\
\text{conditional transfer:} & \quad X' = \begin{cases} 
(X - \{b\} \cup \{c\}) & \text{if } a, \; b \in X, \; c \in X, \\
X & \text{otherwise}
\end{cases}
\end{align*}
\]

A program \( P \) (of length \( k \geq 0 \)) is defined to be a finite sequence \( P = e_1, e_2, \ldots, e_k \) of commands. The length of any program \( P \) is denoted by \( |P| \). A program of length zero, called the empty program, is denoted by \( \lambda \). For any bubble location \( X \) and any program \( P = e_1, e_2, \ldots, e_k \), \( X^P \) is defined by

\[
X^P = (\ldots (X^{e_1})^{e_2} \ldots)^{e_k}
\]

Therefore, in general, a program \( P \) maps the set of all \( 2^n \) subsets of \( S \) into itself.

Two dimensional bubble logic

Let us now consider the problem of computing the Boolean functions of \( m \) variables with appropriate programs. Following Graham, two locations, \( a_1 \) and \( a_2 \), are used to represent...
the value of a binary variable \( x \), as shown in Figure 1. For \( m \) variables, a set of \( 2m \) locations is used which is imagined to be arranged in pairs, as illustrated in Figure 2.

A restriction is imposed on the locations, which can be used for the computation of the Boolean functions of \( m \) variables, namely \( S = \{ a_1, a_2, \ldots, a_m, a_{m+1}, \ldots, a_{2m} \} \).

Note that the space which can be used for the computation of the Boolean functions is not unbounded, but is defined by the input space.

For \( m \) variables, \( x_1, x_2, \ldots, x_m \), let \( W_1, W_2, \ldots, W_{2^m} \) be all \( 2^m \) inputs, and let \( X_1, X_2, \ldots, X_m \) be the corresponding bubble locations. Let \( f(x_1, x_2, \ldots, x_m) \) be a Boolean function of \( m \) variables. A Boolean function \( f(x_1, x_2, \ldots, x_m) \) is said to be realized by a program \( P \) if the following condition is satisfied:

There exist some location \( \rho \in S \) and some program \( P \) such that for \( i = 1, 2, \ldots, 2^m \),

\[
\rho \in X_\rho P \text{ if } f(W_i) = 1,
\]

\[
\rho \notin X_\rho P \text{ if } f(W_i) = 0.
\]

Note that: the value of a Boolean variable is represented by a program which can be used for the computation of the Boolean function represented by the presence or absence of a bubble in a particular location.

Let \( f_a(i) \) be the value of the function which is represented in a location \( a_i \) after application of the \( i \)th command \( c_i \), such that \( f_a(i) = f_a(i-1) \cdot f_a(i) \) or \( f_a(i) = f_a(i-1) + f_a(i) \).

Where \( \cdot \) + are logical AND and logical OR, respectively. We usually write \( f_a(i) f_b(i) \) instead of \( f_a(i) f_b(i) \). If \( c_{i+1} = (a, b) \), then

\[
f_a(i+1) = f_a(i) \cdot f_a(i)
\]

\[
f_a(i+1) = f_a(i) + f_a(i) \cdot f_a(i)
\]

Where \( \cdot \) + are logical AND and logical OR, respectively. We usually write \( f_a(i) f_b(i) \) instead of \( f_a(i) f_b(i) \).

Note that: the value of a Boolean variable is represented by a program which can be used for the computation of the Boolean function represented by the presence or absence of a bubble in a particular location.

Let \( f_a(i) \) be the value of the function which is represented in a location \( a_i \) after application of the \( i \)th command \( c_i \), such that \( f_a(i) = f_a(i-1) \cdot f_a(i) \) or \( f_a(i) = f_a(i-1) + f_a(i) \).

Where \( \cdot \) + are logical AND and logical OR, respectively. We usually write \( f_a(i) f_b(i) \) instead of \( f_a(i) f_b(i) \).

Note that: the value of a Boolean variable is represented by a program which can be used for the computation of the Boolean function represented by the presence or absence of a bubble in a particular location.

Let \( f_a(i) \) be the value of the function which is represented in a location \( a_i \) after application of the \( i \)th command \( c_i \), such that \( f_a(i) = f_a(i-1) \cdot f_a(i) \) or \( f_a(i) = f_a(i-1) + f_a(i) \).

Where \( \cdot \) + are logical AND and logical OR, respectively. We usually write \( f_a(i) f_b(i) \) instead of \( f_a(i) f_b(i) \).

Note that: the value of a Boolean variable is represented by a program which can be used for the computation of the Boolean function represented by the presence or absence of a bubble in a particular location.

In the present model of bubble logic, two types of commands can be used, i.e., transfer and conditional transfer, but in Graham's model, only transfer commands can be used. Thus, the present model is an extension of Graham's model. The main object of this paper is to examine the computational capabilities of the proposed model.

**SOME BASIC PROPERTIES OF GRAHAM'S MODEL**

In this section, Graham's results are briefly reviewed in order to compare them with the present results. After that, further properties of his model will be investigated from the viewpoints of the realization of the Boolean functions and geometrical requirements imposed by the fact that bubble interactions can occur only between physically adjacent locations.

The following non-decreasing overlap (NDO) theorem accredited to W. Shockley plays a fundamental role in Graham's model.

Theorem 1 (NDO Theorem): Let \( X_1 \) and \( X_2 \) be two arbitrary initial sets of locations of bubbles and let \( P \) be any program using only bubble transfer commands. Then

\[
| X_1 \cap X_2 | \geq | X_1 \cup X_2 |
\]

where \( | X | \) denotes the cardinality of set \( X \).

The first important consequence of the NDO theorem is the following.

Proposition 2: There is no replicating program \( P^* \) using only the bubble transfer commands.

A program \( P \) is said to be a replicating program if the following conditions are satisfied:

1. \( X \cap S_1 = X \cap S_2 \)
2. \( \theta(X^\cap S_2) = X \cap S_3 \)

In other words, the replicating program \( P^* \) creates a "copy" of \( X \cap S_1 \) in \( S_2 \) without disturbing \( X \cap S_3 \).

Another consequence of the NDO theorem is the following result.

Proposition 3: There is no binary addition program \( P^* \) using only the bubble transfer commands.

A binary addition program \( P^* \) performs binary addition in the following way.

The definition of a binary addition program in Graham's paper is repeated here.

Assume \( S_1 \) denotes a set of \( m \geq 1 \) pairs of locations of \( S \), \( S_2 \) denotes another set of \( m \) pairs of locations disjointed from \( S_1 \), and \( S_3 \) denotes a set of \( m \) pairs of locations disjointed from \( S_1 \) and \( S_2 \). These sets can be imagined as arranged as shown in Figure 3.
An integer \( M, 0 \leq M \leq 2^m \), can be represented in the \( m \) pairs of \( S_i \) by letting the \( j \)th pair of \( S_i \) denote the \( j \)th binary digit in the binary expansion of \( M \). This can be done using the configurations shown in Figure 1. Thus, for \( m = 5 \), the configuration shown in Figure 4 would denote the integer \( \text{10010}_2 = 18 \).

The binary addition program \( P^+ \) would operate by starting with \( S_3 \) in some fixed configuration and with arbitrary integers \( A \) and \( B \) loaded into \( S_1 \) and \( S_2 \), respectively, to form the initial state \( X \); after applying \( P^+ \) to \( X \) we should get the sum \( A+B \) in \( S_3 \).

The following theorem is the main result obtained by Graham, and indicates one of the limitations of his model for realizing Boolean functions.

**Theorem 4:** There exists a Boolean functions of \( n \) variables which cannot be realized by a program of bubble transfer commands.

At present, it is known that, for \( m = 1, 2, 3 \), and 4 all Boolean functions of \( m \) variables can be realized by his model, but it is not known whether all Boolean functions of 5 variables can be realized.

Some properties of programs using only transfer commands are now investigated.

Let \( P(f) \) denote a program which realizes a Boolean function \( f \), and let \( \rho(P, f) \) denote the location at which the value of \( f \) is represented by the presence or absence of a bubble, after applying a program \( P \). Hereafter, the same character shall be used to represent both a variable (or its complement) and its corresponding input location, since these are not confused.

For example, first consider a Boolean function \( f(x, y) \) defined by Table 1. Thus, in this case, \( \{ x, \bar{x}, y, \bar{y} \} \) will be taken to be the set of \( S \) as shown in Figure 5.

### Table 1. Truth table for \( f(x, y) \).

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
<th>( f(x, y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

It is easy in this case to find an appropriate \( P(f) \). For example, we can take

\[
P(f) = (\bar{y}, \bar{x}) (y, x) (\bar{y}, x)
\]

\[
\rho(P, f) = \bar{x}
\]

For any Boolean function \( f(x_1, x_2, \ldots, x_m) \) of \( m \) variables, we define \( f_{\bar{x}_1}, f_{(x_1, x_2)}, \text{and} \bar{f} \) by

\[
f_{\bar{x}_1} = f(x_1, x_2, \ldots, \bar{x}_1, \ldots, x_m)
\]

\[
f_{(x_1, x_2)} = f(x_1, x_2, \ldots, x_1, x_2, \ldots, x_m)
\]

\[
\bar{f} = 1 - f(x_1, x_2, \ldots, x_m)
\]

**Proposition 5:** If a program \( P(f) \) exists, then \( P(f_{\bar{x}_1}) \) is obtained from \( P(f) \) by complementing \( x_1 \) and letting

\[
\rho(P, f_{\bar{x}_1}) = \begin{cases} 
\bar{x}_1 & \text{if } \rho(P, f) = x_1 \\
x_1 & \text{if } \rho(P, f) = \bar{x}_1 \\
\rho(P, f) & \text{otherwise}
\end{cases}
\]

**Proposition 6:** If a program \( P(f) \) exists, then \( P(f_{(x_1, x_2)}) \) is obtained from \( P(f) \) by permuting \( x_1 \) and \( x_2 \) and letting

\[
\rho(P, f_{(x_1, x_2)}) = \begin{cases} 
x_1 & \text{if } \rho(P, f) = x_2 \\
x_2 & \text{if } \rho(P, f) = x_1 \\
\rho(P, f) & \text{otherwise}
\end{cases}
\]

These results follow directly from the definition of the program.

**Proposition 7:** If \( P(f) = (a_1, a_2, a_3, a_4, \ldots, a_n) \) and \( \rho(P, f) = a_{2r-1} \text{ or } a_{2r}, \) then

\[
P(f) = (a_2, a_1, a_4, a_3, \ldots, a_n, a_{n-1})
\]

\[
\rho(P, f) = \begin{cases} 
a_{2r-1} & \text{if } \rho(P, f) = a_{2r-1} \\
a_{2r} & \text{if } \rho(P, f) = a_{2r} \end{cases}
\]

It is easy to verify that these results are correct for \( m = 1 \).
Computational Capabilities of the Proposed Model

This section deals with some of the computational capabilities of the proposed model. First, it is shown that a binary addition program exists, while there is no binary addition program in Graham's model. Next, by an argument similar to that of Graham in deriving Theorem 4, it is shown that Boolean functions of 19 or more variables exist that cannot be realized by the proposed model.

Proposition 9: A binary addition program $P^+$ exists in the model.

Proof: Used here is the symbolic arrangement of locations for binary addition, shown in Fig. 3. A program $P^+$ can be constructed which performs one-bit addition, as follows.

$$P^+(a_i, b_i, c_i) = (\delta_i \delta_i, c_i) (\delta_i \delta_i, b_i) (\delta_i \delta_i, c_i)$$

In the preceding section, it was assumed that interactions are possible between any pair of bubble locations. Since interactions can occur only between bubbles in physically adjacent locations, this implies that bubbles which are required to interact must be brought into adjacent locations without affecting the bubbles in other locations prior to the application of the command. Therefore, the time taken for computing a function depends not only on the number of commands in the program but also on the layout of the bubble locations.

There are several open problems associated with the design of efficient programs for computing Boolean functions in Graham's model.

1. For a given Boolean function $f_j$, find a simple algorithm for determining whether a program $P(f_j)$ exists.

2. If a program $P(f_j)$ exists, find an algorithm for constructing it effectively.

3. If a program $P(f)$ exists, find a program with the smallest length for computing $f$.

4. For a given program, find the assignment of locations so as to minimize the total number of locations required for its implementation.

The programs realizing all Boolean functions of 3 variables and the corresponding input memory layouts with the minimum total locations are now presented.

Note that by Propositions 5, 6 and 7, it suffices to give the programs for one representative for each of 14 equivalence classes of functions of 3 variables under the operations of complementing and permuting the variables, and of complementing the functions. Table II consists of these representatives. Figure 6 shows the corresponding programs and the input memory layouts. Also note that, in each case, no locations used only as transit points are required.

For all Boolean functions of one variable and of 2 variables, we can easily obtain such programs and input memory layouts as shown above, but for 4 variables, it is not known whether such programs and input memory layouts exist.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
<th>$x_{10}$</th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table II. The representatives for Boolean functions of 3 variables.
where \( m \)-bits addition, is given by

Consequently, in general, a program \( P^+ (m) \) which performs \( m \)-bits addition, is given by

\[ P^+ (m) = P_1 P_2^+ \ldots P_{m+1}^+ (\sigma_{m+1}, \sigma_{m+2}) \]

where \( (\sigma_{m+1}, \sigma_{m+2}) \) comes from the carry.

The capability of Boolean function realization by the proposed model is now considered. Let \( f(x_0, x_1, \ldots, x_n) \) and \( g(x_0, x_1, \ldots, x_n) \) be two Boolean functions. If \( f(x_0, x_1, \ldots, x_n) \leq g(x_0, x_1, \ldots, x_n) \) for all \( 2^n \) inputs, then it is said that \( f \) implies \( g \), and written as \( f \subseteq g \).

Let \( f_x (t) \), \( f_y (t) \) and \( f_z (t) \) denote the Boolean functions realized at locations \( a, b \) and \( c \), respectively, after applying the \( t \)-th command \( c_t \) of program \( P \). It is immediately apparent that, if \( f_x (t) \subseteq f_y (t) \), then the application of the command \( (a, b) \) as the \((t+1)\)-th command of the program \( P \) changes nothing. Hence, it can be assumed that only the command \((a, b)\) is used for which, at the time of their application, \( f_x (t) \nmid f_y (t) \) (it is said that the pair \((f_x (t), f_y (t))\) is command-applicable).

Therefore, \( f_x (t) = f_y (t) \) for all \( t \) such that \((a, b)\) is the \((t+1)\)-th command of \( P \).

Consequently, at least \( r+1 \) triplets of the functions of \( D(t+1) \) are not command-applicable. It is known that, if exactly \( r \) of \( \binom{2^{m}}{2} \) pairs of the functions of \( D(t) \) are not command-applicable, after the application of the command \((a, b)\) as the \((t+1)\)-th command of the program, at least \( r+1 \) pairs of the functions of \( D(t+1) = \{ f_x (t), f_y (t), f_z (t) \} \) are not command-applicable, where it is assumed that the pair \((f_x (t), f_z (t))\) is command-applicable. Therefore, we have the following result.

Lemma 10: For any Boolean function \( f \) of \( m \) variables, it is possible to put

\[ \lg (P(f)) = \left( \binom{2^m}{2} - m \right) (2m-2) + \binom{2^m}{2} \]

Proof: In the initial state, the number of triplets which are command-applicable is \( \binom{2^m}{2} - m \) and that of the pairs is \( \binom{2^m}{2} \).

Now we show the main result:

Theorem 11: Boolean functions of 19 or more variables exist which cannot be realized by the model.
Proof: Let
\[ N = \left( \frac{2m}{2} \right) - m \left( \frac{2m-2}{2} \right) + \frac{2m}{2}. \]
Consider any program \( P = e_0 e_1 \ldots e_r \). In choosing the \( i \)-th command \( e_i \) of \( P \), there are, at most, \( N-i+1 \) possibilities for \( e_i \), since, after \( e_0 e_1 \ldots e_{i-1} \) has been applied, at least \( i-1 \) of the pairs and the triplets are not command-applicable. Therefore there are, at most,
\[ \prod_{i=1}^{N} (N-i+1) = N! \]
choices for the sequence of \( e_i \), since \( t \leq N \) by Lemma 10. Furthermore, by the application of one command, at most two new Boolean functions are generated. Hence there are, at most,
\[ 2N(N!) + 2m \]
Boolean functions which can be generated. On the other hand, the total number of Boolean functions of \( m \) variables is \( 2^m \). These expressions are listed for \( m = 18 \) and \( m = 19 \) in Table 3. Hence the theorem follows.

Table III. Bounds on the number of Boolean functions which can be generated.

<table>
<thead>
<tr>
<th>( m )</th>
<th>( 2N(N!) + 2m )</th>
<th>( 2^m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>( &gt; 10 , 83,520 )</td>
<td>( &lt; 10 , 78,914 )</td>
</tr>
<tr>
<td>19</td>
<td>( &lt; 101 , 001,536 )</td>
<td>( &gt; 101 , 576,826 )</td>
</tr>
</tbody>
</table>

**SUMMARY**

Some properties of a simple bubble model proposed by Graham have been investigated and the programs realizing all Boolean functions of 3 variables and the corresponding input memory layouts with the minimum total locations have been presented. This paper has extended Graham’s model by introducing a different type of command, i.e., the conditional bubble transfer. It has been shown that, in the proposed model, a binary addition program exists, and also Boolean functions of 19 or more variables exist which cannot be realized.

**ACKNOWLEDGMENTS**

The author would like to thank Dr. K. Noda for his support of the writing of this paper, and Dr. N. Ikeno and Dr. G. Nakamura for their valuable suggestions and many discussions.

**REFERENCES**