A. NEW HIGH-SPEED MULTIPLIER DESIGN

With the advent of MSI and LSI integrated circuit technology, there is no doubt that digital multipliers of very high speed can be achieved, once it is agreed what should be incorporated in these chips.¹ In the meantime, we can achieve a very fast design by simultaneously exploiting the mathematical structure of binary two's-complement multiplication and existing MSI circuits which can be adapted in a natural way to the structure of this task. Accordingly, we shall show that the expression for binary multiplication can be rewritten to suggest use of the 74181 Arithmetic Logic Unit (ALU) in a straightforward way that achieves high speed, simple layout, and very little logic external to the ALU array.

In order to display the desired structure of multiplication, we shall consider the multiplication of two 4-bit two's-complement numbers. Let each such number be represented as

¹This work was supported by the National Institutes of Health (Grant 5 PO1 GM14940-05), and by the Joint Services Electronics Programs (U.S. Army, U.S. Navy, and U.S. Air Force) under Contract DAAB07-71-C-0300.)
(XI. COGNITIVE INFORMATION PROCESSING)

\[ Z = z_3 z_2 z_1 z_0 = -2^3 z_3 + 2^2 z_2 + 2^1 z_1 + 2^0 z_0, \]

where each \( z_i \) is either 1 or 0, so that the product of two such numbers is

\[ X Y = 2^6 x_3 y_3 - 2^5 (x_3 y_2 + x_2 y_3) + 2^4 (-x_3 y_1 + x_2 y_2 - x_1 y_3) \]
\[ + 2^3 (-x_3 y_0 + x_2 y_1 + x_1 y_2 - x_0 y_3) \]
\[ + 2^2 (x_2 y_0 + x_1 y_1 + x_0 y_2) + 2^1 (x_1 y_0 + x_0 y_1) + 2^0 x_0 y_0. \]

This sum is commonly arranged in an array in which each column contains factors of like powers of 2, as in Fig. XI-1. The factors can be further rearranged, as

\[
\begin{array}{c}
-x_3 y_0 + x_2 y_0 + x_1 y_0 + x_0 y_0 \\
-x_3 y_1 + x_2 y_1 + x_1 y_1 + x_0 y_1 \\
-x_3 y_2 + x_2 y_2 + x_1 y_2 + x_0 y_2 \\
+x_3 y_3 + x_2 y_3 - x_1 y_3 - x_0 y_3
\end{array}
\]

Fig. XI-1. Array representation of two's-complement multiplication.

\[
\begin{array}{cccc}
 x_2 y_0 & x_1 y_0 & x_0 y_0 \\
 x_2 y_1 & x_1 y_1 & x_0 y_1 \\
 x_2 y_2 & x_1 y_2 & x_0 y_2 \\
 x_3 y_3
\end{array}
\]

\[
\begin{array}{cccc}
 x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 \\
 x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3
\end{array}
\]

Fig. XI-2. Multiplication array, grouped in positive and negative terms.
shown in Fig. XI-2, in order to group positive and negative terms. At this stage, each row has a constant factor for each term such as \( y_0 \) for the top row, and these can be factored out as bits that control the conditional inclusion of a given row in the final sum. Thus, in Fig. XI-3, the top row \((x_2 \ x_1 \ x_0)\) will be added into the sum just in case \( y_0 = 1 \), and similarly for the other rows. Figure XI-3 also shows 6 conditional terms to be summed, but one of these, \( x_3y_3 \), affects only the most significant bit position. If this term is included in any other row (say, row 6), the only change

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 \ x_1 \ x_0 )</td>
<td>( y_0 )</td>
</tr>
<tr>
<td>( x_2 \ x_1 \ x_0 )</td>
<td>( y_1 )</td>
</tr>
<tr>
<td>( x_2 \ x_1 \ x_0 )</td>
<td>( y_2 )</td>
</tr>
<tr>
<td>( x_2 \ x_1 \ x_0 )</td>
<td>( y_3 )</td>
</tr>
</tbody>
</table>

\[ ((\text{A} + \text{B}) + \text{C}) - (\text{D} + \text{E}) \]

Fig. XI-3. Multiplication array, grouped by rows and their respective control bits.

Fig. XI-4. Illustrating the parallel nature of the multiplication task.
in the final result might be a carry out of the most significant bit. When sign exten-
sion is not desired, it is thus possible to incorporate the \(x_3^y_3\) term in row 6, and
we are left with just 5 rows to sum. It is this representation of the product as a
sum of conditional terms that can be exploited by the ALU design.

Having rewritten the product as a sum of terms, each conditioned on a control
bit, the next step is to minimize the sum-and-carry delays by exploiting the inher-
ent parallelism of the array. Referring to the five rows as A through E, Fig. XI-4
shows how a tree structure permits simultaneous sums to be computed, rather than
performing each indicated sum in serial, left-to-right order. In Fig. XI-4, each
box denotes an ALU adder, and we assume that each add is completed in \(\Delta\) seconds.
During the first \(\Delta\) seconds two adds are completed, followed by one add in each of
the two succeeding intervals. There are two advantages in this scheme. First,
the total delay would be \(4\Delta\) seconds if the adds were done serially, but when the
parallelism is utilized, only \(3\Delta\) seconds of delay result. More generally, for larger
sized numbers \(N\) bits long, a similar binary tree would lead to \((\log_2 N)\Delta\) seconds
delay, whereas a serial procedure would require \((N-1)\Delta\) seconds delay. For \(N = 16\),
the saving is \(\Delta(15-4) = 11\Delta\), a very substantial figure. When \(N\) is not a power
of 2, some branches of the full binary tree are pruned, but the saving in time
because of parallelism is still obtained. The second advantage is that the binary
tree arrangement can be implemented in a straightforward and natural way by using
the 74181 ALU, 24-pin MSI package.

The 74181 ALU, shown in Fig. XI-5, operates on two 4-bit inputs in a manner
prescribed by the four control bits, \(S_0\) through \(S_3\), and the Mode Control bit \(M\), to
produce a single 4-bit output. As shown in Fig. XI-6, all of the needed control
functions can be realized by appropriate use of \(S_0\) through \(S_3\), \(M\), and \(C_0\), the last
being the input carry to the least significant bit. Note that only the double condi-
tional sum, \((A \text{ if } z) + (B \text{ if } y)\), requires extra circuitry to translate the condition
bits \(z\) and \(y\) into ALU controls, but that this circuitry is very simple, containing
only an XOR gate and an inverter.

Figure XI-7 shows the complete design for a \(4 \times 4\) multiplier, in which the con-
trol circuitry is shown in detail. Depending on the size of the multiplier desired,
extra time savings may be realized by appropriate partitioning of the array and
insertion of carries, but the basic details remain the same. The authors have
designed \(16 \times 16\) and \(16 \times 24\) arrays, which illustrate further refinements. These
designs are available to the interested reader.

A further advantage of the ALU is its wide availability. Originally, it was
introduced in TTL, but Schottky TTL and MECL 10,000 versions are now avail-
able. Worst-case multiplication times will depend on which one of these pack-
ages is used, but a \(16 \times 16\) design should yield a completion time of 95-100 ns
LOGIC DIAGRAM

Fig. XI-5. Logic diagram for the 74181 Arithmetic Logic Unit.

<table>
<thead>
<tr>
<th>BASIC FUNCTION</th>
<th>CONTROL SIGNALS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>A+B</td>
<td>1</td>
</tr>
<tr>
<td>A-B</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONTROL FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+B</td>
</tr>
<tr>
<td>A-B</td>
</tr>
<tr>
<td>A+(B if y)</td>
</tr>
<tr>
<td>(A if z)+(B if y)</td>
</tr>
</tbody>
</table>

Fig. XI-6. Control functions for the 74181 Arithmetic Logic Unit.
in Schottky TTL. This is considerably faster than the performance obtainable from specialized multiplier packages, such as the Fairchild 9344 or the Advanced Micro Devices AM 2505. Since the ALU package has many uses, it is relatively inexpensive, particularly considering the resulting multiplier speed. The package count, and hence power, is high (approximately $N(N+1)/4$ for an $N \times N$ multiply; for $N=16$, 69 ALU's were required) but layout is simple, and no other design incorporating standard commercial MSI packages has been able to yield the speed of this ALU array.

Certainly faster or cheaper multipliers have been built. The ALU in a binary tree, however, appears to be an optimal choice when very high speed is desired from standard commercial packages.

J. Allen, E. R. Jensen

References