filters and linear transforms, having an effective 18 to 16 bit precision, can be realized in hardware. Data throughput rates are comparable to those obtained by fully pipelined conventional architectures. Finally, an error model for the multiplier was derived and experimentally tested.

The developed autoscale multiplier may prove to be an important innovation in the areas of shift invariant following and the DFT using high-speed RNS arithmetic.

REFERENCES


Comments on "An \(O(n)\) Parallel Multiplier with Bit-Sequential Input and Output"

H. J. SIPS

Abstract—For the realization of a bit-sequential multiplier with operands of length \(n\), Chen and Willoner\(^4\) suggest a circuitry consisting of \(2n\) identical modules. It is shown that if a slightly different arrangement of the modules is taken, the number of modules is reduced to \(n\). Furthermore, the implementation in circuit form can be made more simple.

Index Terms—Computer arithmetic, on-line algorithms, parallel multiplier, pipeline, real-time algorithms.

In their paper Chen and Willoner\(^4\) describe a parallel multiplier with bit-sequential input and output, least significant bits first. A possible implementation of the multiplier module without control logic is given in their paper.

It is not mentioned that the additional control logic is rather complex. In order to allow pipelining, each module needs a set of 5 storage cells to hold the \(A\) and \(B\) inputs and \(S\), \(C_1\), and \(C_2\) outputs.

A possible complete implementation is given in Fig. 1. The \(a\) and \(b\) inputs (common to all modules) are the numbers \([a_0,\ldots,a_i]\) and \([b_0,\ldots,b_i]\), which have to be multiplied. The \(B'(-1)\) and \(A'(-1)\) inputs are the shifted bit pair from module \(j - 1\) generated in the previous time step. \(C_1(-1)\) and \(C_2(-2)\) are the carry signals from modules \(j - 1\) and \(j - 2\).

A central circuitry (counter, demultiplexer) generates the selector signals \([\sigma_0,\ldots,\sigma_{2n+1}]\). Signal \(\sigma_{i-j/2}\) determines whether the current bit pair \((a, b)\) or the shifted bit pair \(A'(j - 1), B'(j - 1)\) is loaded into the register pair \((R_1, R_2)\). The signals \(a, b,\) and \(\alpha,\beta\) (common to all modules) and \(\sigma_{i+j/2}\) take care of the proper generation of the \(A(i, j)'s\) and \(B(i, j)'s\) according to the algorithm described in the above paper.

At time step \(i = 0,\ldots, n + 1\), the signals \(a, b,\alpha,\beta,\) and \(\sigma\) are defined as follows:

\[
a = \begin{cases} 
\sigma_{i+1} & \text{if } i = 0,\ldots, n - 1 \\
\text{DON'T CARE} \quad \text{otherwise}
\end{cases}
\]

\(0018-9340/82/0400-0325$00.75 © 1982 IEEE\)
It can be seen that $\alpha$ and $\beta$ can be made from $a$ and $b$ by delaying $a$, $b$ one time step (apart from the zero condition).

The storage cells $R_3$, $R_4$, $R_5$ hold $C_1$, $C_2$, and $S$, respectively. If the product bits $\{P_1, \ldots, P_{2n}\}$ are outputted in bit-sequential form, an additional 3-state gate is required.

The modules need not be identical since the gate circuits before $R_1$ and $R_2$ are only needed in the uneven modules. With additional clock and clear signals each module has 18 interconnections to the outside world.

However, it can be seen that once a bit of the product has been calculated the module remains stable. Instead of shifting the $|A'(j-1), B'(j-1)|$ bit pair to the next module, the same module can be reused, if the previously produced product bit is outputted to the next sequential device. This conforms to the bit-sequential nature of the multiplier.

In Fig. 2 the different arrangement of the modules is shown.

The five input bits generated for the $j$th module during the $i$th iteration of the multiplication algorithm are as follows:

\[
\begin{align*}
A(i,j) &= \begin{cases} 
\beta_j \cdot a_j & \text{if } j < i \\
0 & \text{if } j \geq i
\end{cases} \\
B(i,j) &= \begin{cases} 
\alpha_i \cdot b_j & \text{if } j \leq i \\
0 & \text{if } j > i
\end{cases} \\
C_i(i,j) &= S_{23}[A(i-1,j), B(i-1,j), C_i(i-1,j)] \\
\sigma_j &= \begin{cases} 
1 & \text{if } i = j \\
0 & \text{if } i \neq j.
\end{cases}
\end{align*}
\]

Instead of $2n$ modules only $n$ modules are required.

The full operation of the multiplier is illustrated in Fig. 3, which is the same example as in Chen and Willoner’s paper. If the product does not exceed the $n$ bits, only $n/2$ modules are required. In Fig. 4 a possible implementation is given. A module, including clock and clear signals, has only 12 connections to the outside world. Furthermore, the module implementation in Fig. 4 has a smaller number of gates than the module implementation in Fig. 1. Also, only $n + 1$ select-lines $\sigma_i$ are required.
Correction to "Error-Correcting Parsers for Formal Languages"
EIICHI TANAKA AND KING SUN FU

I. INTRODUCTION

In our earlier paper we proposed an algorithm S for error-correcting parsing of formal languages. It is found recently that two additional CS conditions and a routine for detecting infinite loop are required for the algorithm S.

II. THE TWO CS CONDITIONS

1) If \( i_r \in S(i_p), i_r \in S(i_c), Ch(i_p) = i_p, Ch(i_r) = i_r, i_r \neq i_p, \psi_3(\bar{F}') = i_{k_r}, \bar{F} \notin i_{t+1,k} \), then rules \( BC \rightarrow A \), \( BC \rightarrow BA \) and \( BC \rightarrow AC \) cannot be applied for \( i_r \) and \( i_c \), where \( \psi_3(\bar{F}') \) is the \( i \)th entry of \( \bar{F}' \).

2) If \( i_p \in S(i_b), i_c \in S(i_c), i_c \in R(i_r), i_r \in S(i_c), i_r \neq i_c, \bar{F} \notin S(i_r), \psi_3(\bar{F}') = i_{r_c}, \bar{F} \in t_{ij}, \psi_3(\bar{F}') = i_{c_c} \) and \( \bar{F} \notin i_{t+1,k} \), then rules \( BC \rightarrow A \), \( BC \rightarrow BA \) and \( BC \rightarrow AC \) cannot be applied for \( i_r \) and \( i_c \), where \( \psi_3(\bar{F}') = i_{r_c} \) and \( Cx(i_A) = i_{r_c} \) mean that \( i_r \) is a child of \( i_3 \) and that \( AB \rightarrow CB \) or \( BA \rightarrow BC \), respectively. \( i_A > i_{r_c} \) means that \( i_A \) is an ancestor of \( i_{r_c} \).

III. THE DETECTION OF AN INFINITE LOOP

Use the symbol "→c" instead of "→" to distinguish from an ordinary derivation, and call it C-derivation. C-derivation is defined by the following.

1) Let \( \vec{E} = A_i(i_1)A_i(i_2) \cdots A_n(i_n) \) and \( i_k > i_s (s = 1, 2, \cdots, k - 1, k + 1, \cdots, n) \). If \( A_k(i_k) \rightarrow_c \vec{E} \), then

\[
\vec{E} = \bar{E}A_k(i_k) c \rightarrow_c \bar{E}A_k(i_k) \vec{E}.
\]

2) Assume that \( i_1 \) (or \( i_n \)) is the largest number among \( i_1, i_2, \cdots, i_n \). If \( A_i (i_1) \) (or \( A_n (i_n) \)) has the left context \( \bar{E}c \) (or the right context \( \vec{E}c \))

\[
\vec{E} = A_i(i_1) \vec{E} \rightarrow_c \bar{E} A_i(i_1) \vec{E} \quad \text{or} \quad \vec{E} = \bar{E} A_n(i_n) \rightarrow_c \bar{E} A_n(i_n) \vec{E}.
\]

Assume that \( E_j \in T_{n,p}, \psi_3(E_j) = i_j \) (\( j = 1, 2, 3 \)) and \( i_3 > i_2 > i_1 \). Let \( A_j, i_j \) and \( r_j \) be a nonterminal, the number of \( A_j \) and the number of the rule used to create \( i_j \). Define a triple \( (E_1, E_2, E_3) = (A_j, i_j, r_j) \) (\( j = 1 \), \( 2, 3 \)). Define \( \psi = (B_1, r_1) \cdots (B_3, r_3) \). If \( \psi = E_1(E_2)E_3 \), write \( E(E_2) \), not \( E(E_1) \). Suppose that the following derivations are obtained:

\[
E(i_1) \rightarrow c E(i_1) \rightarrow c E(i_2)
\]

and

\[
E(i_2) \rightarrow c E(i_2) \rightarrow c E(i_1).
\]

To summarize, the advantages of this approach are as follows:
- only \( n \) modules are required to produce the \( 2n \)-bit product of two \( n \)-bit operands,
- there is only one product output,
- the interconnection scheme in the \( n \)-module case is more simple than in the \( 2n \)-module case. The \( n \)-modules have only nearest neighbor interconnections.

Manuscript received January 5, 1982.
E. Tanaka is with the Department of Information Science, Utsunomiya University, Utsunomiya, Japan.
K. S. Fu is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.