The Error Characteristics of the Binary Rate Multiplier

A. DUNWORTH AND J. I. ROCHE

Abstract—The binary rate multiplier is a device which has been used for many years in hybrid computing (operational digital techniques) and control systems as a means for generating a pulse train of average frequency proportional to the value of a binary number stored in a register. In general, the pulse spacing is irregular and the number of pulses generated in a given time fluctuates above and below the number which would be produced by a perfectly regular pulse train at the same average frequency. These fluctuations constitute a short-term frequency error, the value of which is an important parameter in the design of pulse rate digital systems incorporating binary rate multipliers. This article analyzes the conditions under which maximum positive and negative errors occur, and expressions are derived from which the magnitude of such errors may be calculated.

Index Terms—Binary rate multiplier, digital control systems, error analysis, hybrid computation, operational digital techniques, pulse rate systems, synthesized pulse trains.

INTRODUCTION

The binary rate multiplier is a device for producing, by digital means, an output train of pulses whose average frequency is proportional to the value of a binary integer stored in a register. It has been used as a building block in a number of special-purpose computing and control systems [1]–[3]. The simplest form is shown in Fig. 1.

An input pulse train of frequency \( f \) is applied to a series of flip-flop dividers. The noncarry sides of these divider stages supply pulse trains at frequencies of \( f/2, f/4, f/8, \) etc. and no pulses in any of these pulse trains are coincident. These pulse trains can be selected by a set of \( \text{AND} \) gates controlled by the stages of a flip-flop register \( R \) and recombined in an \( \text{OR} \) gate to give a single pulse train. If the input \( f \) consists of equally spaced pulses then the output has an average frequency of

\[
f_o = \sum_{r=0}^{k} b_{k-r} \left( \frac{f}{2^{k+1}} \right) = \frac{f}{2^{k+1}} \sum_{r=0}^{k} b_{k-r} 2^{k-r},
\]

(1)

i.e.,

\[
f_o = \frac{fB}{2^{k+1}} = \frac{fB}{(B_{\text{max}} + 1)}
\]

(2)

where \( B \) is the number stored in the register (in pure binary coded form) and \( B_{\text{max}} \) is the maximum capacity of the register, that is, \( 2^{k+1} - 1 \) if the register has \( k+1 \) flip-flop stages.

The output pulse train is, in general, irregular even if the input is uniform since the output is derived from the input by selectively omitting some pulses without altering the basic spacing. The above expression for \( f_o \) implies measurement over a sufficiently long time to make the irregularity insignificant. Alternatively there is no error if \( f_o \) is measured over an integral number of complete cycles of the flip-flop divider, that is, over \( N2^{k+1} \) input pulses. For example, let the number of input pulses in a given time be \( A \) and assume that initially all the flip-flops in the divider chain are in their \( \text{RESET} \) state. If \( A \) happens to be \( N2^{k+1} \), then the number of output pulses \( N_0 \) is \( AB/2^{k+1} \) and the circuit can be regarded as an exact multiplier. In general, for any \( A \) the quantity \( AB/2^{k+1} \) is not an integer and the actual number of output pulses \( N_0 \) corresponding to \( A \) is an integer close to \( AB/2^{k+1} \) but not necessarily the nearest integer. We will define the instantaneous error as

\[
\epsilon = N_0 - \frac{AB}{2^{k+1}},
\]

(3)

i.e., \( \epsilon \) is the difference between \( N_0 \) and a continuous variable increasing linearly at the same average rate.

A knowledge of the instantaneous error is useful when the multiplier forms part of a closed loop in which the multiplier output is compared to another pulse train to produce a difference error pulse rate. The small fluctuations in the multiplier output may then be reproduced as relatively much larger fluctuations in the difference pulse rate, and it is normally necessary to remove these fluctuations by some form of filtering process [1].

The error is thus a quantity which fluctuates about zero and becomes zero at least once in every cycle of \( 2^{k+1} \) input pulses. In the following sections the conditions are discussed under which maximum positive and negative errors occur, and expressions are presented from which the magnitude of such errors may be calculated.

DERIVATION OF THE GENERAL EXPRESSION FOR THE ERROR

Referring to Fig. 1, suppose that initially all the flip-flop divider stages are in the \( \text{RESET} \) condition and let a total of \( A \) regularly spaced pulses be applied to the input of stage \( k \). The number \( A \) may be expressed in binary form as
where \( L \) is an integer which has a lower limit given by \( 2^{L+1} - 1 \geq A \). In particular, the value of \( L \) may be chosen to be \( \geq k \) where \((k+1)\) is the number of stages in the flip-flop divider chain.

The number of noncarry pulses produced by stage \( k \) is equal to

\[
\frac{A}{2} \text{ if } A \text{ is even,}
\]
\[
\frac{1}{2} (A + 1) \text{ if } A \text{ is odd, or}
\]
\[
\frac{1}{2} (A + a_0) \text{ in general.}
\]

The number of carry pulses produced by stage \( k \) is

\[
\frac{1}{2} (A - a_0) = \sum_{i=1}^{L} a_i 2^{i-1},
\]
and this number of pulses provides the input to stage \((k-1)\).

Similarly, stage \((k-1)\) produces \( \frac{1}{2} \left[ \frac{1}{2} (A - a_0) + a_1 \right] \) noncarry pulses, i.e., \( \frac{1}{2} [A + 2a_1 - a_0] \), and in general, the stage \( s \) produces \( N_s \) noncarry pulses where

\[
N_s = \frac{1}{2^{k-s+1}} \left[ A + 2^{k-s+1} a_{k-s} - \sum_{i=0}^{k-s} a_i 2^i \right].
\]

If the register \( R \) (with \( k+1 \) stages) in Fig. 1 contains a number \( B \) in pure binary coded form where

\[
B = \sum_{i=0}^{k} b_i 2^i,
\]
then the number of output pulses

\[
N_0 = \sum_{i=0}^{k} b_i N_s.
\]

The error \( \epsilon \) may be expressed as \( \epsilon = N_0 - (AB/2^{k+1}) \) (3), and substituting for \( N_0 \) from (9) and \( B \) from (8) leads to

\[
\epsilon = \sum_{s=0}^{k} \left( \frac{b_s}{2^{k-s+1}} \left[ A + 2^{k-s+1} a_{k-s} - \sum_{i=0}^{k-s} a_i 2^i - A \right] \right),
\]
i.e.,

\[
\epsilon = \frac{1}{2} \sum_{s=0}^{k} b_s \left[ 2a_{k-s} - \sum_{i=0}^{k-s} a_i 2^{i-s} \right].
\]

Note that the error is independent of any coefficients in \( A \) more significant than \( a_s \).

Equation (10) may also be rearranged in the equivalent form

\[
\epsilon = \frac{1}{2} \sum_{s=0}^{k} a_s \left[ 2b_{k-s} - \sum_{i=0}^{k-s} \frac{b_i}{2^{k-s-i}} \right].
\]

**MAXIMUM ERRORS FOR A GIVEN \( B \) REGISTER COUNT**

If we consider an arbitrary fixed value of \( B \) represented by a binary bit pattern of \( b_k b_{k-1} \cdots b_0 \), then the error \( \epsilon \) given by (11) may be maximized by suitable choice of the coefficients of \( A \). The general results and some particular examples are given in Table I which distinguishes three error conditions. Equation (11) effectively gives the value of the error \( \epsilon \) occurring immediately after the \( A \) pulse which establishes the particular \( A \) count of \( a_k a_{k-1} \cdots a_0 a_0 \). The maximum positive error calculated from (11) is the largest positive error that can be obtained under any conditions for that particular \( B \) register count. However, if \( A_{\text{min}} \) represents the \( A \) count for the maximum negative error derived from (11), then a somewhat larger negative error
can be obtained by calculating the error immediately before the next occurring A pulse, i.e., immediately before the A pulse that establishes an $A$ count of $(A_{\text{max}}+1)$.

As shown by Rule I of Table I, the $A$ count for the maximum positive error condition may be obtained for a given binary bit pattern of $b_n b_{n-1} \cdots b_1 b_0$ by writing down the bits of $B$ in reverse order, replacing all leading 0's by $x$'s (optional 0 or 1). If $A_{\text{max}}$ is the resultant $A$ value, then the conditions for maximum negative error occurring before or after an A pulse may be obtained, respectively, by taking the 2's or 1's complement of $A_{\text{max}}$ (Rules II and III).

**Absolute Maximum Errors**

*(Evaluated Over all $A$ and $B$ Values)*

The expressions given in column 3 of Table I maximize the errors over all $A$ for any particular value of $B$. The overall error maxima with regard to all possible $A$ and $B$ values may therefore be obtained by maximizing the error equations of Table I by a suitable choice of the coefficients $b_n b_{n-1} \cdots b_1 b_0$. The analysis is straightforward but somewhat lengthy, and the results are summarized in Table II. For any given length of register and for each of the three types of error maxima there are two values of the $B$ counter content which give maximum errors. Corresponding to each critical $B$ counter content there is a single $A$ value within each complete cycle of $2^n$ pulses at which the specified error maximum occurs. It should be noted that all the $A$ and $B$ bit pattern entries in Table II contain a basic recurrent pattern of the form 0101 \ldots that is broken in some cases by a repeated binary 1 or 0 at the beginning and/or end of the sequence. Corresponding bit patterns for $A$ and $B$ are related by the rules given in Table I (Rules I, II, and III).

The absolute maximum positive and negative error conditions correspond to the same $B$ register counts. If we define $\epsilon_{\text{dev}}$ as the peak-to-peak positive-to-negative error deviation, then

$$\epsilon_{\text{dev}} = \epsilon_{\text{max}}(+) - \epsilon_{\text{max}}(-).$$

(12)

Substituting the expression for $\epsilon_{\text{max}}(+) \text{ and } \epsilon_{\text{max}}(-)$ given in column 4 of Table II we derive

$$\epsilon_{\text{dev}} = \frac{k}{3} \frac{10}{9} - \frac{(1)^k}{9 \times 2^{k}}.$$

(13)

This expression may be written in terms of $m = k + 1$ where $m$ is the number of stages in the $B$ register or the number of stages in the flip-flop divider of Fig. 1:

$$\epsilon_{\text{dev}} = \frac{m}{3} + \frac{7}{9} - \frac{(1)^m}{9 \times 2^{m-1}}.$$

(14)

The peak-to-peak error deviation occurs for either of the two $B$ counts given by

$$B_1 = \frac{2^{m+1} + (-1)^m}{3}$$

(15)

$$B_2 = \frac{5 \times 2^{m-1} - (-1)^m}{3}.$$

(16)

The numerical examples given in Table I include the maximum error conditions for an 8-bit register. The entries marked a) correspond to the maximum negative and positive errors obtained, respectively, before or

---

**TABLE I**

<table>
<thead>
<tr>
<th>Error condition (for a given fixed $B$ value evaluated over all possible $A$ values)</th>
<th>$A$ count of form $x = x_{n-1} x_{n-2} \cdots x_1 x_0$ corresponding to a $B$ count of the form $b_{n-1} \cdots b_1 b_0$ for the specified error condition.</th>
<th>Error equation for $\epsilon$</th>
<th>Examples ${\text{values marked a)}}$ are absolute maxima for all $A$ and any 8-bit value of $B$)</th>
</tr>
</thead>
</table>
| **Error $\epsilon$ (occurring immediately after an $A$ pulse) has the maximum possible positive value** | $\begin{align*}
&| \begin{align*}
&\text{Rule I:} \\
&b_{k-1} = 1 \\
&b_{k-2} = 1 \\
&b_{k-3} = 1 \\
&\vdots \\
&b_0 = 1 \\
&\text{All leading 0's by } x \text{'s (optional 0 or 1).}
\end{align*}
\end{align*}$ | $\begin{align*}
&\frac{1}{2} \sum_{i=0}^{n} b_i \left[ 2^{i-1} - \sum_{i=0}^{n} b_i \right]
\end{align*}$ | $\begin{align*}
&\begin{align*}
&\text{Examples:} \\
&\text{A pulse count and equivalent binary bit pattern} \\
&\text{B register count and equivalent binary bit pattern}
\end{align*}
\end{align*}$ | $\begin{align*}
&\text{Error value}
\end{align*}$ |
| **Error $\epsilon$ (occurring immediately after an $A$ pulse) has the maximum possible negative value** | $\begin{align*}
&| \begin{align*}
&\text{Rule II:} \\
&b_{k-1} = 0 \\
&b_{k-2} = 0 \\
&b_{k-3} = 0 \\
&\vdots \\
&b_0 = 0 \\
&\text{All leading 0's by } x \text{'s (optional 0 or 1).}
\end{align*}
\end{align*}$ | $\begin{align*}
&\frac{1}{2} \sum_{i=0}^{n} b_i \left[ 2^{i-1} - \sum_{i=0}^{n} b_i \right]
\end{align*}$ | $\begin{align*}
&\begin{align*}
&\text{Examples:} \\
&\text{A pulse count and equivalent binary bit pattern} \\
&\text{B register count and equivalent binary bit pattern}
\end{align*}
\end{align*}$ | $\begin{align*}
&\text{Error value}
\end{align*}$ |
### TABLE II
Maximum Errors Evaluated over all A and B Values

<table>
<thead>
<tr>
<th>Error Condition</th>
<th>B register count and the equivalent binary bit pattern ( b_kb_{k-1} \cdots b_2b_1 )</th>
<th>A pulse count for the specified error condition and the equivalent binary bit pattern ( a_{a-1} \cdots a_0a_1 )</th>
<th>Error Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum positive error occurring immediately after an A pulse input</td>
<td>( \frac{4 \times 2^k + 1}{3} ) [ \begin{array}{c} \text{1010} \cdots \text{1011 (k odd)} \end{array} ]</td>
<td>( \frac{5 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1101} \cdots \text{0101 (k odd)} \end{array} ]</td>
<td>( \varepsilon_{\text{max}} (+) ) = ( \frac{k}{6} + \frac{5}{9} - \frac{(-1)^k}{9 \times 2^{k+1}} )</td>
</tr>
<tr>
<td></td>
<td>( \frac{4 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1010} \cdots \text{10101 (k even)} \end{array} ]</td>
<td>( \frac{4 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1010} \cdots \text{10101 (k even)} \end{array} ]</td>
<td></td>
</tr>
<tr>
<td>Maximum negative error occurring immediately after an A pulse input</td>
<td>( \frac{2 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{0101} \cdots \text{0101 (k odd)} \end{array} ]</td>
<td>( \frac{2 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{0101} \cdots \text{0101 (k odd)} \end{array} ]</td>
<td>( \varepsilon'_{\text{max}} (-) ) = ( -\frac{k}{6} + \frac{1}{18} + \frac{(-1)^k}{9 \times 2^{k+1}} )</td>
</tr>
<tr>
<td></td>
<td>( \frac{2 \times 2^k + 1}{3} ) [ \begin{array}{c} \text{0101} \cdots \text{01001 (k even)} \end{array} ]</td>
<td>( \frac{2^k - 1}{3} ) [ \begin{array}{c} \text{0010} \cdots \text{10101 (k even)} \end{array} ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \frac{2^k + 1}{3} ) [ \begin{array}{c} \text{0010} \cdots \text{1011 (k odd)} \end{array} ]</td>
<td>( \frac{2^k + 1}{3} ) [ \begin{array}{c} \text{0010} \cdots \text{1011 (k odd)} \end{array} ]</td>
<td></td>
</tr>
<tr>
<td>Maximum negative error occurring immediately before an A pulse input</td>
<td>( \frac{4 \times 2^k + 1}{3} ) [ \begin{array}{c} \text{1010} \cdots \text{1011 (k odd)} \end{array} ]</td>
<td>( \frac{2^k + 1}{3} ) [ \begin{array}{c} \text{0010} \cdots \text{1011 (k odd)} \end{array} ]</td>
<td>( \varepsilon''_{\text{max}} (-) ) = ( -\frac{k}{6} + \frac{5}{9} + \frac{(-1)^k}{9 \times 2^{k+1}} )</td>
</tr>
<tr>
<td></td>
<td>( \frac{4 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1010} \cdots \text{10101 (k even)} \end{array} ]</td>
<td>( \frac{2 \times 2^k + 1}{3} ) [ \begin{array}{c} \text{0101} \cdots \text{01101 (k even)} \end{array} ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \frac{5 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1101} \cdots \text{0101 (k odd)} \end{array} ]</td>
<td>( \frac{2 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{0101} \cdots \text{01101 (k odd)} \end{array} ]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( \frac{5 \times 2^k - 1}{3} ) [ \begin{array}{c} \text{1101} \cdots \text{0101 (k even)} \end{array} ]</td>
<td>( \frac{2^k - 1}{3} ) [ \begin{array}{c} \text{0010} \cdots \text{10101 (k even)} \end{array} ]</td>
<td></td>
</tr>
</tbody>
</table>
after any $A$ pulse. These examples indicate that for an 8-bit binary rate multiplier, the maximum error difference between the number of output pulses actually produced and the number which would be produced by a perfect multiplier generating a uniform pulse rate at the same average frequency is less than plus or minus two pulses.

REFERENCES


Table-Lookup/Interpolation Function Generation for Fixed-Point Digital Computations
H. M. AUS and G. A. KORN

Abstract—For very fast function generation with small fixed-point digital computers, the n-bit argument word $x$ is split into an $N$-bit word representing the breakpoint abscissa $X_i$ for table-lookup by indirect addressing, and an $n-N$ bit word representing the interpolation difference $X-X_i$. The complete procedure takes only about 50 machine cycles (50 μs) for equal breakpoint intervals, and about 70 machine cycles for variable breakpoint density. A sine-cosine generator for digital or hybrid-computer simulation is exhibited as an example.

Index Terms—Fixed-point computation, function generator, simulation, sine-cosine generator, table-lookup.

INTRODUCTION

The approximation of piecewise-continuous functions

$$Y = F(X)$$

(1)

by piecewise-linear (polygonal) functions in the form

$$Y = A_i X + B_i \quad X_i \leq X < X_{i+1}$$

(2)

or

$$Y = A_i (X - X_i) + Y_i \quad (X_i \leq X < X_{i+1})$$

(3a)

$$A_i = \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i}$$

$$B_i = Y_i - A_i X_i$$

(3b)

(Fig. 1) is widely employed in analog computations. Methods for finding suitable breakpoint abscissas $X_i$ and slopes $A_i$ for good approximation of a given function $F(X)$ range all the way from saeman's-eye techniques to dynamic programming [2], [3], [6]. In general, a polygonal approximation will be formed by secants crossing and recrossing the given curve [1]. To minimize temperature drift effects, analog diode function generator outputs should serve, if at all possible, only as corrections to linear approximation functions $AX+B$, although carefully made diode function generators can have errors within 0.05 percent of half-scale.

It will be interesting and fruitful to consider application of table-lookup/interpolation techniques to function generation with small fixed-point digital computers, which do not permit sufficiently fast multiplication and division to make polynomial or rational-function approximations attractive. The 7- to 15-μs multiplication times quoted for the so-called "fast arithmetic" options of such small digital processors usually conveniently neglect the 15- to 30-μs setup times required for a complete multiplication routine. Table-lookup routines, on the other hand, require storage for at least a set of breakpoint coordinates $Y_i = F(X_i)$. In general, a simple step-function approximation $Y = Y_i (X_i \leq X < X_{i+1})$ would require an impossibly large number of breakpoints $X_i$ so that at least linear interpolation is indicated. The number of breakpoints (and thus the amount of storage) required for a given function-approximation accuracy will depend on the variations of the slope of $F(X)$ in each breakpoint interval. As an especially important example, sin $X (0 \leq X \leq \pi/2)$ can be represented within 0.001 percent of half-scale (5 decimal digits, or 17 bits) with 256 evenly spaced breakpoints. By contrast, the more frequently "oscillating" function sin $16X (0 \leq X \leq \pi/2)$ would require more breakpoints;