CHARGOGGAGGOGGMANCHAUGAGOGGCHAUBUNAGUNGAMAUG

A Novel Multiply-by-Three Circuit*

by
Caxton Foster, Edward Riseman, Fred Stockton and Conrad Wogrin
Computer Science Department, University of Massachusetts
Amherst, Massachusetts

Recently, while considering the connection of a 48 bit word computer to a 16 bit computer, we felt the need for a fast and inexpensive device that would multiply a binary address by a factor of three. Since 3N = N + 2N, there is an obvious solution of providing a normal adder circuit and presenting one set of inputs with N and the other with N-shifted left one place. But, there is a great deal of redundancy here since knowing one input we have complete knowledge of the other.

Seeking to take advantage of this redundancy, we have spent some twenty man hours and have saved one gate per stage over a conventional full adder. Since the ultimate circuit is to be a "one-of" this is scarcely economic, but it has been fun. Before going further, the interested reader may enjoy puzzling over the problem for himself. We make no pretense of minimality, but we have a circuit with 6 gates per stage and a propagation delay of one gate per stage.

Some More or Less Blind Alleys

Before stumbling on the "solution" presented in the next section we explored several ideas that did not seem to be too helpful. We mention them briefly below.

Any binary number can be thought of as sets of successive ones separated by one or more zeros. It is an interesting property of binary arithmetic that a string of S successive ones bounded on both ends by zeros will, when multiplied by 3, result (if considered by itself) in a pattern of "a one, a zero, S-2 ones, a zero, and a one" right justified under the original string. Thus 01111 produces 101101 etc. The only exception to this is the pattern 01 which produces 011. Two or more zeros between strings of ones isolate them from each other with no interference. Single zeros between two strings of ones cause the right-most two bits of the pattern generated by the left-hand string (normally equal to 01) to be complemented producing 10. This approach while interesting and probably extremely useful for a serial adder, did not help too much with our problem.

A second approach was to consider multiplets of input bits. For example, we examined the four possible values assumed by adjacent pairs (00, 01, 10, 11) together with the carry (if any) coming in from the right. This method consistently produced variations on a full adder per stage (or sometimes worse) and consequently was abandoned.

A third approach was to replace the original equation by: $3N = N + N + N$.

At the i^{th} stage the circuit must accept a value of 0 or 3 for N_i plus carry from the next lower stage and the one below that. This produces numbers as large as 5 (101) which represent (reading right to left) a sum value, a carry, and a double carry. This double carry output of a stage can be handled by sending it two stages to the left instead of the normal one stage shift for carry output, or by sending both the carry and the double carry to the stage on the left, one with weight of one, the other with a weight of two. Once again, this approach did not appear too fruitful.

The Present Method

The present method is based on the scheme of carry by pass as discussed by Gschwind (Gschwind, H., Design of Digital Computers. Springer-Verlag; New York, 1967). If both input digits are ONE at a given stage (in our case, if there are two adjacent ONEs) a carry will be guaranteed to be generated at this stage. If both inputs are ZERO (two adjacent ZEROs) then it is known that no carry can be generated at this stage. A ONE and a ZERO (alternating ONEs and ZEROs) will propagate a carry to the left if one arrives from the right, but will not generate a carry on its own.

Consider the i^{th} stage. To its right (toward less significant bit positions) any one of four different conditions may obtain. The immediately adjacent bit may have been ONE (conditions A and B) or ZERO (conditions C and D) and the nearest equal pair may have been a pair of adjacent ONEs (conditions B and D) or ZEROs (conditions A and C). Between this equal pair and the present stage there may have been 0, 1, 2, . . . pairs of alternating ONEs and ZEROs. Thus we can define the four conditions as:

$$A = (10)^n\ 100$$
$$B = (10)^n\ 11$$
$$C = (01)^n\ 00$$
$$D = (01)^n\ 011$$

where the power "n" represents the number of repetitions of the enclosed pattern (n = 0, 1, . . .).

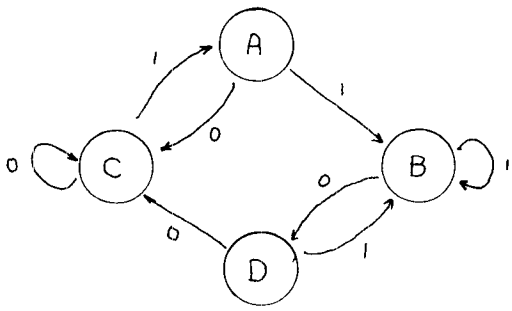

Figure 1. The state transition diagram

*The title of this paper refers to the Indian name of a lake in Massachusetts which reputedly means "You fish on your side, I fish on my side, nobody fish in the middle." The reason for choosing this name may be seen by examining Figure 5.

The conditions presented to the $i+1^{st}$ stage (the one to the left of the present stage) will be a function of the conditions at the present stage and the value of the i^{th} digit of N. Suppose that is a ONE. Then condition A will change into condition B, D will do the same, B remains B, and C converts into A. Figure 1 shows the complete transition diagram. The value to be output at a stage (that particular bit of the product) will also be a function of the input condition and the value of N_i. For conditions B and C the output is equal to N_i, while for states A and D the output is the inverse of N_i. Since both the behavior and the output of conditions A and D are identical, they can be "merged".

Look once again at Figure 1. If the input condition is "not C" (that is, it is A or B or D), and the N_i digit is a ONE, then the output condition is B. Similarly, if the input condition is "not B" and the N_i digit is ZERO then the output state is C. Figure 2 shows a circuit to realize this behavior constructed from two NOR gates. Condition A-D is represented by "neither B nor C." This represents a pseudo-carry propagation delay of only one gate which is twice as fast as a conventional full adder without carry look-ahead or carry bypass circuits.

The value of the product bit can be generated by the circuit of Figure 3. We thus have used a total of 6 gates per stage, which is one less than required by a conventional full adder.

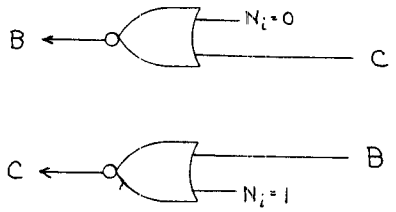

Figure 2. The "pseudo-carry" propagation circuit

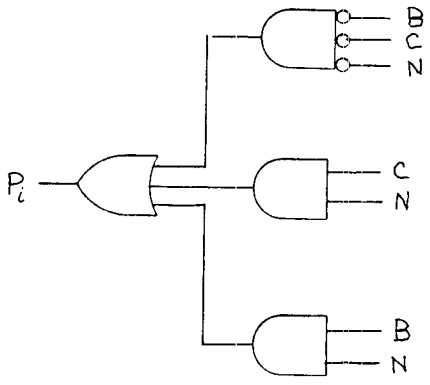

Figure 3. Circuit to generate the product bit

Other Uses

Now that we have this multiply-by-3 circuit we can find some other uses for it. For example, a conventional multiplier capable of multiplying two arbitrary numbers can be built by adding either ZERO or the value of the multiplicand into the shifted partial product depending on whether the bit of the multiplier is ZERO or ONE. Since a times-2 circuit is trivially simple, we can now have three possible outputs from the multiplicant register (N, 2N, and 3N) which, together with ZERO, can be selected for adding into the partial product depending on whether a pair of bits in the multiplier is 01, 10, 11, or 00 respectively. This would then form the final product with only half as many additions as would a conventional multiplier. Of course other methods of speeding up multiplication already exist.

Higher Products

Before going on to look at X7 and X15 multipliers, let us recast what we have learned about the X3 multiplier. At each stage the value of the input digit is either 0 or 3. Carry propagating in from the right can have a value of 0, 1, or 2. Thus the maximum input can be 5, which generates a sum of 2 and a carry for the next stage worth 4/2 or 2. Summarizing this in tabular form we have:

carry in	input = 0 carry out	input = 0 sum	input = 3 carry out	input = 3 sum
0	0	0	1	1
1	0	1	2	0
2	1	0	2	1

From this table we can construct "transition trees". These are shown in Figure 4.

Now suppose we write:

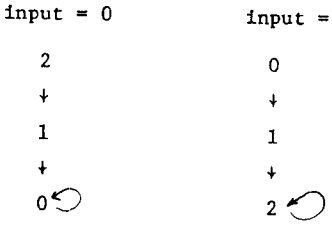

Figure 4. Transition trees for a X3 circuit

0	C	$C = \overline{X} \cdot \overline{B}$
1		
2	B	$B = X \cdot \overline{C}$

Figure 5. The Boolean equations for the pseudo carry lines of a times 3 circuit.

carry in	variables
0	C
1	
2	B

which means that if C=1, we know the carry in is 0, if B=1, the carry in is 2, if both are 0, the carry in is 2, and we "don't care" about the case where both B and C are 1. Note that this provides a form of "Gray Code". From this assignment and Figure 4,

we can generate Figure 5. Looking at the "falling" transition tree (X=0) we see that the new value of the carry will be 0 provided X=0 and the old carry was not 2. In equation form we write $C=\overline{X}\cdot\overline{B}$. Similarly, looking at the "rising" transition tree, we have a new carry of 2 provided X=1 and the old carry was not 0; or $B=X\overline{C}$. These two equations are, after substituting NOR circuits for ANDs, just what we derived by inspection of the patterns of bits above.

Thus encouraged, let us consider a times 7 circuit. The input variable X has values 0 and 1 corresponding to weights of 0, 7. Carry can range from 0 to 6. The table looks like:

	X = 0		X = 1[W(X)=7]	
carry in	carry out	sum	carry out	sum
0	0	0	3	1
1	0	1	4	0
2	1	0	4	1
3	1	1	5	0
4	2	0	5	1
5	2	1	6	0
6	3	0	6	1

The transition trees are shown in Figure 6 and a choice of pseudo-carry variable assignment in Figure 7. Note that we have not tried to minimize lines (this will catch up with us later on) but circuit delays. The equations for the product "S" of a given stage are

$$Z = B \vee C \vee E \vee F$$

$$S = \overline{X}\overline{Z} \vee X\overline{Z}$$

A similar treatment of a "times 15" circuit is shown in Figures 8 and 9.

Before the reader rushes off to design a new high speed multiplier based on these circuits, we should point out that a "times $2^n - 1$" circuit will require $2^n - 2$ pseudo carry lines per stage. Even for n = 10 this will be rather more than one would like to have, and for n = 32, will indeed be large.

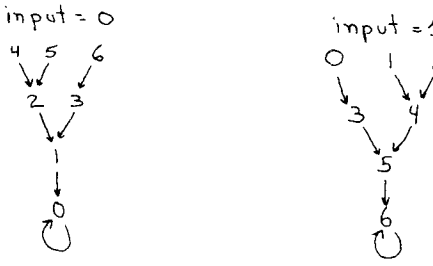

Figure 6. The transition trees for a times 7 circuit

carry	variables true	equations
0	AB	$B = \overline{X}\cdot A\cdot\overline{C}$
1	A	$A = \overline{X}\cdot\overline{F}$
2	AC	$C = \overline{X}DF$
3	none	
4	DE	$E = XA\overline{E}$
5	D	$D = X\overline{E}$
6	DF	$F = XD\overline{E}$

Figure 7. Variable assignment and Boolean equations for a times 7 circuit

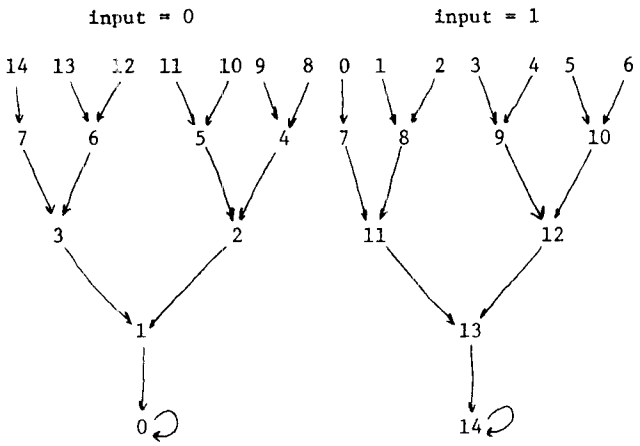

Figure 8. Transition trees for a times 15 circuit

carry	variables true	equations
0	ACG	$G = \overline{X}\overline{CH}$
1	AC	$C = \overline{X}A\overline{CJ}$
2	ACH	$H = \overline{X}AD\overline{J}$
3	A	$A = \overline{XN}$
4	ADI	$I = \overline{X}BE\overline{L}$
5	AD	$D = \overline{X}B\overline{N}$
6	ADJ	$J = \overline{X}BF\overline{N}$
7	none	
8	BEK	$K = XAC\overline{G}$
9	BE	$E = XA\overline{G}$
10	BEL	$L = XAD\overline{I}$
11	B	$B = X\overline{G}$
12	BFM	$M = XBE\overline{K}$
13	BF	$F = XB\overline{FK}$
14	BFN	$N = XF\overline{M}$

Figure 9. Variable assignment and Boolean equations for a times 15 circuit