8x8 High Speed Schottky Multipliers

SN74S557 SN54/74S558

Features/Benefits

- **Industry-standard 8x8 multiplier**
- **Multiplies two 8-bit numbers; gives 16-bit result**
- **Cascadable; 56x56 fully-parallel multiplication uses only 34 multipliers for the most-significant half of the product**
- **Full 8x8 multiply in 60ns worst case**
- **Three-state outputs for bus operation**
- **Transparent 16-bit latch in 'S557**
- **Plug-in compatible with original Monolithic Memories' 67558**

Description

The 'S557/'S558 is a high-speed 8x8 combinatorial multiplier which can multiply two eight-bit unsigned or signed twos-complement numbers and generate the sixteen-bit unsigned or signed product. Each input operand X and Y has an associated Mode control line, X_M and Y_M respectively. When a Mode control line is at a Low logic level, the operand is treated as an unsigned eight-bit number; whereas, if the Mode control is at a High logic level, the operand is treated as an eight-bit signed twos-complement number. Additional inputs, R_S and R_U, (R, in the 'S557) allow the addition of a bit into the multiplier array at the appropriate bit positions for rounding signed or unsigned fractional numbers.

The 'S557 internally develops proper rounding for either signed or unsigned numbers by combining the rounding input R with X_M, Y_M, $\overline{X_M}$, and $\overline{Y_M}$ as follows:

$R_U = \overline{X_M} \cdot \overline{Y_M} \cdot R$ = Unsigned rounding input to 2^7 adder.

$R_S = (X_M + Y_M)\, R$ = Signed rounding input to 2^6 adder.

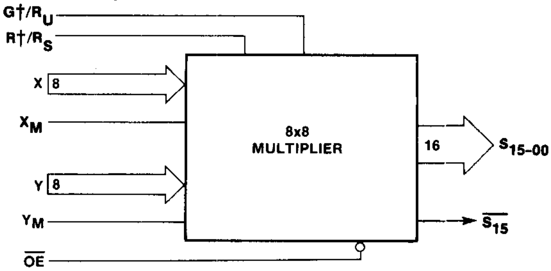
Since the 'S558 has no latches, it does not require the use of pin 11 for the latch enable input G, so R_S and R_U are brought out separately.

The most-significant product bit is available in both true and complemented form to assist in expansion to larger signed multipliers. The product outputs are three-state, controlled by an assertive-low Output Enable which allows several multipliers to be connected to a parallel bus or be used in a pipelined system. The device uses a single +5V power supply and is packaged in a standard 40-pin DIP.

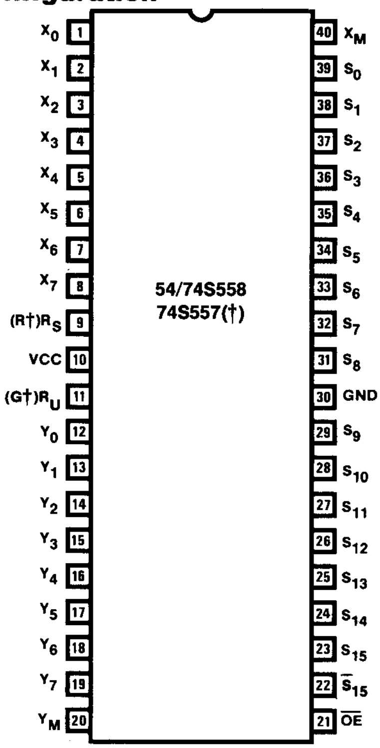
Ordering Information

PART NUMBER	PACKAGE	TEMPERATURE
54S558	J, (44), (L)	Military
74S557, 74S558	N,J,	Commercial

Logic Symbol



Pin Configuration

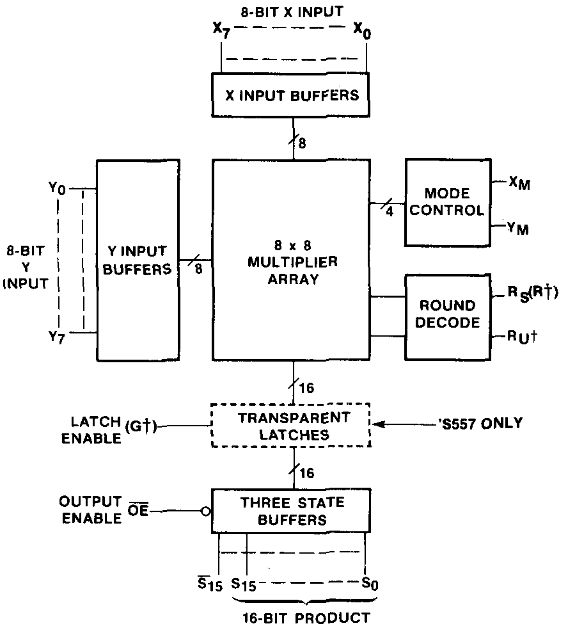


†For 74S557 Pin 9 is R and Pin 11 is G.

2175 Mission College Blvd. Santa Clara, CA 95054-1592 Tel: (408) 970-9700 TWX: 910-338-2376 TWX: 910-338-2374

Monolithic Memories

Logic Diagram



†For 74S557 Pin 9 is R and Pin 11 is G.

Absolute Maximum Ratings

Supply voltage V_{CC} .... 7.0 V
Input voltage .... 7.0 V
Off-state output voltage .... 5.5 V
Storage temperature .... −65° to +150°C

Operating Conditions

SYMBOL	PARAMETER	DEVICE	MILITARY MIN	MILITARY TYP	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP	COMMERCIAL MAX	UNITS
V_{CC}	Supply voltage	all	4.5	5	5.5	4.75	5	5.25	V
T_A	Operating free-air temperature	all	-55		125*	0		75	°C
t_{su}	X_i, Y_i to G set	'S557				40			ns
t_h	X_i, Y_i to G hold time	'S557				0			ns
t_w	Latch enable pulse width	'S557				15			ns

* Case temperature

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER	TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{IL}	Low-level input voltage					0.8	V
V_{IH}	High-level input voltage			2			V
V_{IC}	Input clamp voltage	V_{CC} = MIN	I_I = −18mA			−1.5	V
I_{IL}	Low-level input current	V_{CC} = MAX	V_I = 0.5V			−1	mA
I_{IH}	High-level input current	V_{CC} = MAX	V_I = 2.4V			100	μA
I_I	Maximum input current	V_{CC} = MAX	V_I = 5.5V			1	mA
V_{OL}	Low-level output voltage	V_{CC} = MIN	I_{OL} = 8mA			0.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN	I_{OH} = −2mA	2.4			V
I_{OZL}	Off-state output current	V_{CC} = MAX	V_O = 0.5V			−100	μA
I_{OZH}	Off-state output current	V_{CC} = MAX	V_O = 2.4V			100	μA
I_{OS}	Output short-circuit current*	V_{CC} = MAX	V_O = 0V	−20		−90	mA
I_{CC}	Supply current	V_{CC} = MAX			200	280	mA

* Not more than one output should be shorted at a time and duration of the short-circuit should not exceed one second.

† Typicals at 5.0V V_{CC} and 25°C T_A.

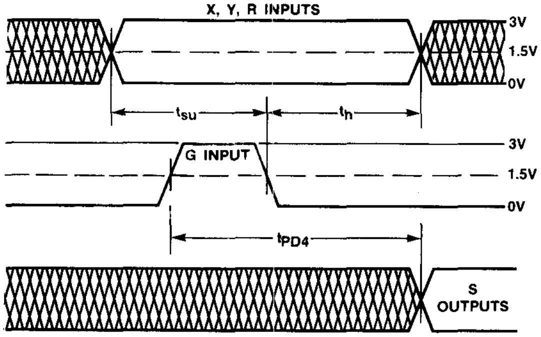
Switching Characteristics Over Operating Conditions

SYMBOL	PARAMETER	DEVICE	TEST CONDITIONS	MILITARY MIN	MILITARY TYP†	MILITARY MAX	COMMERCIAL MIN	COMMERCIAL TYP†	COMMERCIAL MAX	UNIT
t_{PD1}	X_i, Y_i to S_{7-0}	All	C_L = 30pF, R_L = 560Ω, see test figures		40	60		40	50	ns
t_{PD2}	X_i, Y_i to S_{15-8}	All			45	70		45	60	ns
t_{PD3}	X_i, Y_i to $\overline{S}_{15}$	All			50	75		50	65	ns
t_{PD4}	G to S_i	'S557			20	40		20	35	ns
t_{PXZ}	$\overline{OE}$ to S_i	All			20	40		20	30	ns
t_{PZX}	$\overline{OE}$ to S_i	All			15	40		15	30	ns

11

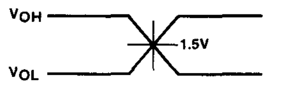
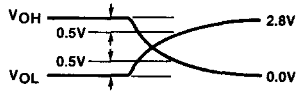
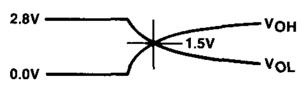
Timing Waveforms

Setup and Hold Times ('S557)

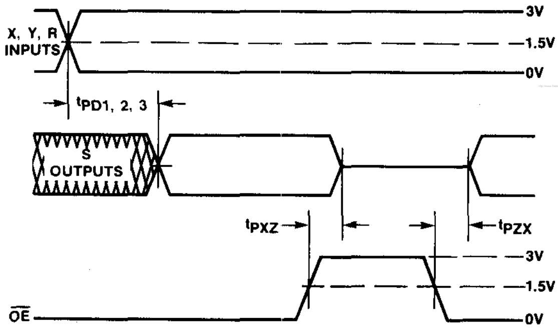


NOTE: If the rising edge of G occurs *before* ($t_{SU_{MIN}}$-$t_{W_{MIN}}$) from the inputs changing, then the applicable propagation delays are t_{PD}, t_{PD2} and t_{PD3}, (and not t_{PD4}). In this case the time at which the results arrive at the outputs depends on when the inputs change instead of when the rising edge of G occurs.

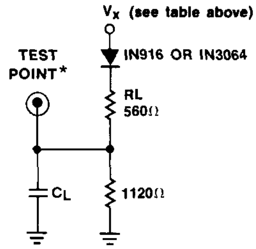
Test Waveforms

TEST	V_X		OUTPUT WAVEFORM — MEAS. LEVEL
All t_{PD}	5.0V		
t_{PXZ}	for t_{PHZ}	for t_{PLZ}	
	0.0V	5.0V	
t_{PZX}	for t_{PZH}	for t_{PZL}	
	0.0V	5.0V	

Propagation Delay

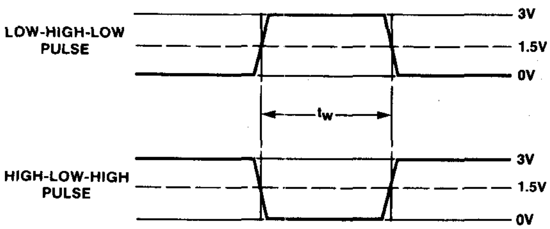


Test Load

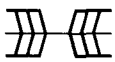


* The "TEST POINT" is driven by the output under test, and observed by instrumentation.

Latch-Enable Pulse Width ('S557)



Definition of Timing Diagram

WAVEFORM	INPUTS	OUTPUTS
	DON'T CARE; CHANGE PERMITTED	CHANGING; STATE UNKNOWN
	NOT APPLICABLE	CENTER LINE IS HIGH IMPEDANCE STATE
	MUST BE STEADY	WILL BE STEADY

SUMMARY OF SIGNALS/PINS	
X_7-X_0	Multiplicand 8-bit data inputs
Y_7-Y_0	Multiplier 8-bit data inputs
X_M, Y_M	Mode control inputs for each data word; LOW for unsigned data and HIGH for twos-complement data
S_{15}-S_0	Product 16-bit output
$\overline{S}_{15}$	Inverted MSB for expansion
R_S, R_U	Rounding inputs for signed and unsigned data, respectively ('S558 only)
G	Transparent latch enable ('S557 only)
$\overline{OE}$	Three-state enable for S_{15}-S_0 and $\overline{S_{15}}$ outputs
R	Rounding input for signed or unsigned data; combined internally with X_M, Y_M ('S557 only)

ROUNDING INPUTS
'S557

INPUTS			ADDS	
X_M	Y_M	R	2^7	2^6
L	L	H	YES	NO
L	H	H	NO	YES
H	L	H	NO	YES
H	H	H	NO	YES
X	X	L	NO	NO

'S558

INPUTS		ADDS		USUALLY USED WITH	
R_U	R_S	2^7	2^6	X_M	Y_M
L	L	NO	NO	X	X
L	H	NO	YES	H†	H†
H	L	YES	NO	L	L
H	H	YES	YES	*	*

†In mixed mode, one of these could be Low but not both.
*Usually a nonsense operation. See applications section of data sheet.

74S557 FUNCTION TABLE

INPUTS		PRODUCT RESULT FROM ARRAY	LATCH CONTENTS (INTERNAL TO PART)	OUTPUTS	FUNCTION
$\overline{OE}$	G	T_i	Q_i	S_i	
L	L	X	L	L	Latched
L	L	X	H	H	
L	H	L	(L)*	L	Transparent
L	H	H	(H)*	H	
H	L	X	(L)	Z	Hi-Z; Latched Data not Changed
H	L	X	(H)	Z	
H	H	X	(X)*	Z	Hi-Z

*Identical with product result passing through latch

MODE CONTROL INPUTS

OPERATING MODE	INPUT DATA		MODE CONTROL INPUTS	
	X_7-X_0	Y_7-Y_0	X_M	Y_M
Unsigned	Unsigned	Unsigned	L	L
Mixed	Unsigned	Twos-Comp.	L	H
	Twos-Comp.	Unsigned	H	L
Signed	Twos-Comp.	Twos-Comp.	H	H

11

Functional Description

The 'S557 and 'S558 multipliers are 8x8 full-adder Cray arrays capable of multiplying numbers in unsigned, signed, twos-complement, or mixed notation. Each 8-bit input operand X and Y has associated with it a mode control which determines whether the array treats this number as signed or unsigned. If the mode control is at High logic level, then the operand is treated as a twos-complement number with the most-significant bit having a negative weight; whereas, if the mode control is at a Low logic level, then the operand is treated as an unsigned number.

The multiplier provides all 16 product bits generated by the multiplication. For expansion during signed or mixed multiplication the most-significant product bit is available in both true and complemented form. This allows an adder to be used as a subtractor in many applications and eliminates the need for certain SSI circuits.

Two additional inputs to the array, R_S and R_U, allow the addition of a bit at the appropriate bit position so as to provide rounding to the best signed or unsigned fractional eight-bit result. These inputs can also be used for rounding in larger multipliers. In the 'S557, these two inputs are generated internally from the mode controls and a single R input.

The product outputs of the multiplier are controlled by an assertive-low $\overline{\text{Output Enable}}$ control. When this control is at a Low logic level the multiplier outputs are active, while if the control is at a High logic level then the outputs are placed in a high-impedance state. This three-state capability allows several multipliers to drive a common bus, and also allows pipelining of multiplication for higher-speed systems.

Rounding

Multiplication of two n-bit operands results in a 2n-bit product†. Therefore, in an n-bit system it is necessary to convert the double-length product into a single-length product. This can be accomplished by truncating or rounding. The following examples illustrate the difference between the two conversion techniques in decimal arithmetic:

$$\left.\begin{array}{l}39.2 \rightarrow 39\\ 39.6 \rightarrow 39\end{array}\right\}\ \text{Truncating}$$

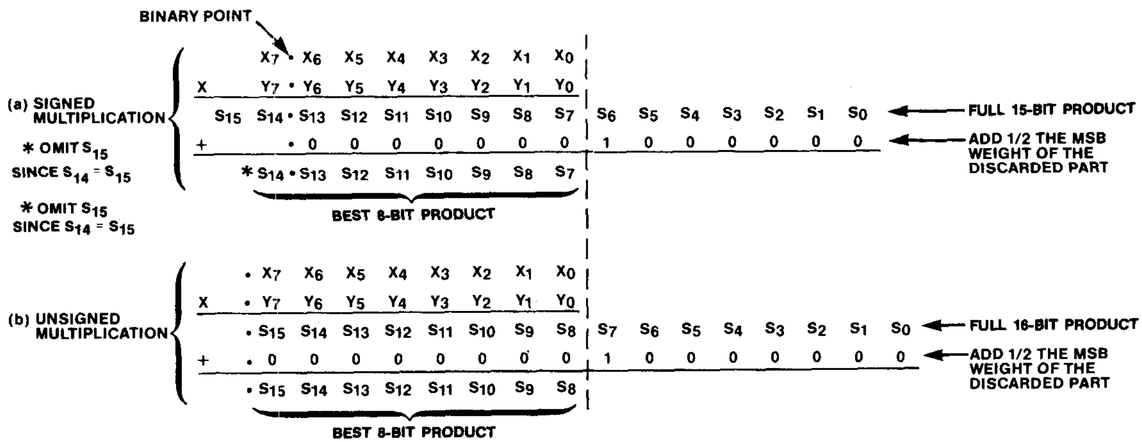
$$\left.\begin{array}{l}39.2 + 0.5 = 39.7 \rightarrow 39\\ 39.6 + 0.5 = 40.1 \rightarrow 40\end{array}\right\}\ \text{Rounding}$$

Obviously, rounding maintains more precision than truncating, but it may take one more step to implement. The additional step involves adding one-half of the weight of the single-length LSB to the MSB of the discarded part; e.g., in decimal arithmetic rounding 39.28 to one decimal point is accomplished by adding 0.05 to the number and truncating the LSB:

$$39.28 + 0.05 = 39.33 \rightarrow 39.3$$

The situation in binary arithmetic is quite similar, but two cases need to be considered: signed and unsigned data representation. In signed multiplication, the two MSBs of the result are identical, except when both operands are -1; therefore, the best single-length product is shifted one position to the right with respect to the unsigned multiplications. Figure 1 illustrates these two cases for the 8x8 multiplier. In the signed case, adding one-half of the S_7 weight is accomplished by adding 1 in bit position 6, and in the unsigned case 1 is added to bit position 7. Therefore, the 'S558 multiplier has two rounding inputs, R_S and R_U. Thus, to get a rounded single-length result, the appropriate R input is tied to V_{CC} (logic High) and the other R input is grounded. If a double-length result is desired, both R inputs are grounded for the 'S558, and the single R input is grounded for the 'S557.

†In general: multiplication of an M-bit operand by an N-bit operand results in an (M + N)-bit product.



NOTES:

(a) In signed (twos-complement) notation, the MSB of each operand is the sign bit, and the binary point is to the right of the MSB. The resulting product has a redundant sign bit and the binary point is to the right of the second MSB of the product. The best eight-bit product is from S_{14} through S_7, and rounding is performed by adding "1" to bit position S_6.

(b) In unsigned notation the best 8-bit product is the most significant half of the product and is corrected by adding "1" to bit position S_7.

Figure 1. Rounding the Result of Binary Fractional Multiplication

Signed Expansion

The most-significant product bit has both true and complement outputs available. When building larger signed multipliers, the partial products (except at the lower stages) are signed numbers. These unsigned and signed partial products must be added together to give the correct signed product. Having both the true and complemented form of the most-significant product bit available assists in this addition. For example, say that two signed partial products must be added and MSI adders are used; we then have the situation of adding together the carry from the previous adder stage plus the addition of the two negative most-significant partial-product bits. The result of adding these variables must be a positive sum and a negative carry (borrow). The equations for this are:

$$S = A \oplus B \oplus C$$

$$C_{OUT} = AB + BC + CA$$

where C is the carry-in and A and B are the sign bits of the two partial products.

Now an adder produces the equations:

$$S = A \oplus B \oplus C$$

$$C_{OUT} = AB + BC + CA$$

Examining these equations, it can be seen that, if the inversions of A and B are used, then the most significant sum bit of the adder is the sign extension bit.

$$\text{Sign ext} = AB + B\overline{C} + \overline{C}A = \overline{\overline{A}\overline{B} + \overline{B}C + C\overline{A}},$$

and the sum remains the same.

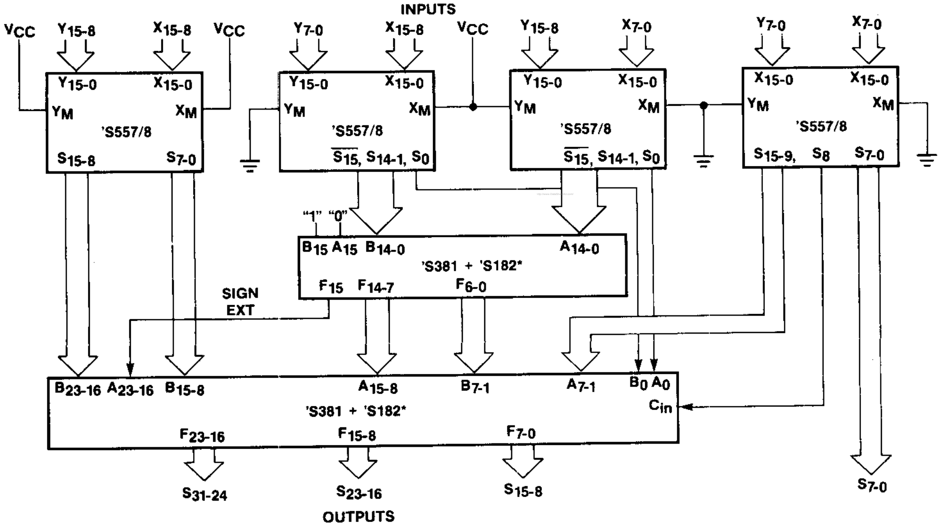
16x16 Twos-Complement Multiplication

The 16-bit X operand is broken into two 8-bit operands (X_7-X_0 and X_{15}-X_8), as is the Y operand. Since the situation is that of a cross-product, four partial products are generated as follows:

$$A = X_L * Y_L$$
$$B = X_L * Y_H$$
$$C = X_H * Y_L$$
$$D = X_H * Y_H$$

where the subscript L stands for bits 7-0, ("low or least-significant half), and the subscript H stands for bits 15-8.

Expanded twos-complement multiplication requires a sign extension of the B and C partial products. Thus, B_{15} and C_{15} need to be extended eight positions to the left (to align with D_{15}). In this approach two more adders are required. But the complement of the MSB ($\overline{S}_{15}$) on the 'S557/8 can be used to save these two adders. Figure 2 shows the implementation of 16x16 signed twos-complement multiplication in this manner.



* THESE ARE ADDER BLOCKS USING THE 'S381, A 4-BIT ALU FUNCTION GENERATOR, TO PERFORM A HIGH-SPEED ADD OPERATION. THE 'S182 IS A LOOKAHEAD CARRY GENERATOR AND REDUCES THE PROPAGATION DELAY. ALL OF THE ABOVE PARTS ARE AVAILABLE FROM MONOLITHIC MEMORIES INCORPORATED.

TOTAL MULTIPLY TIME = MULTIPLIER DELAY + ADDER LEVEL 1 DELAY + ADDER LEVEL 2 DELAY = 60+44+64 = 168 nsec

Figure 2. 16x16 Twos-Complement Signed Multiplication

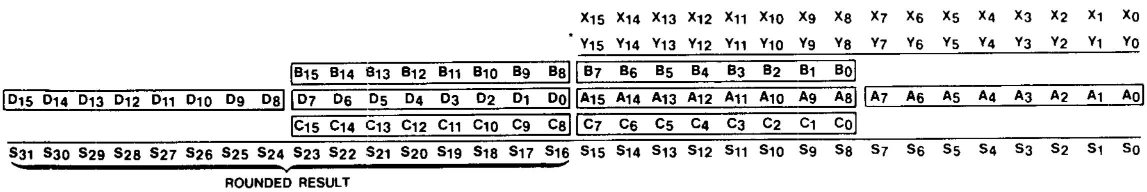


Figure 3. Unsigned Expansions of the 8x8 Multiplier to 16x16 Multiplication

Applications: How to Design Superspeed Cray Multipliers with '558s

by Chuck Hastings

Multiplication, as most of us think of it, is performed by repeated addition and shifting. When we multiply using pencil and paper, according to the familiar elementary-school method, we first write down the multiplicand, and then write down the multiplier immediately under it and underline the multiplier. Then we take the least-significant digit of the multiplier, multiply that digit by the entire multiplicand, and record the answer in the top row of our workspace, underneath the line. Then we repeat, using now the second-least-significant multiplier digit, and record that answer below the first one, pushed one digit position (that is, "shifted") to the left. This process continues until we run out of multiplier digits (or out of patience), at which point we add up the constants of the whole diamond-shaped workspace and record at the bottom an answer which consists of either m + n - 1 digits or m + n digits, where there are m digits in the multiplier and n digits in the multiplicand. An example, voila':

```
  125  (multiplicand)
 x107  (multiplier)
 ------------------------------------------
  875  (7 x 125)
 000   (0 x 125, shifted left one digit position)
125    (1 x 125, shifted left two digit positions)
 ------------------------------------------
13375  (sum of the above)
```

Figure 4. Decimal Multiplication

The decimal number system has no monopoly on truth — our ancestors simply happened to have ten fingers at the time when someone came up with the idea of counting. Binary numbers, as you know, are more copacetic than are decimal numbers with digital-logic elements, which like to settle comfortably into one voltage state ("High) or another ("Low"), rather than into one of ten different states. So we can repeat the above example using binary numbers, right? First, we convert our multiplicand and multiplier to binary:

$$125_{10} = 01111101_2$$
$$107_{10} = 01101011_2$$

The subscripts 10 and 2 refer to the "base" or "radix" of the number system, 10 for decimal and 2 for binary. (Remember your New Math?) For sneaky reasons to be revealed soon, I've used 8-bit binary numbers, which is one bit more than necessary for my example, and added a leading zero. So, we multiply:

```
               01111101₂ = 125₁₀
              x01101011₂ = 107₁₀
               --------
               01111101
              01111101
             00000000
            01111101
           00000000
          01111101
         01111101
        00000000
        ---------------
        0011010000111111 = 13375₁₀
```

Figure 5. Binary Multiplication

I've left off the remarks this time, but they're just like the remarks in the decimal example, at least in principle. Just in case you doubt this answer, I'll convert it back:

1	1	
1	2	
1	4	
1	8	
1	16	
1	32	
0	0	(64)
0	0	(128)
0	0	(256)
0	0	(512)
1	1024	
0	0	(2048)
1	4096	
1	8192	
0	0	(16384)
0	0	(32768)
	13375	

Figure 6. Binary-to-Decimal Conversion

Now look carefully at the diamond-shaped array of numbers in the workspace in Figure 5. Each row is either the multiplicand 01111101, or else all zeroes. The 01111101 rows correspond to "1" digits in the multiplier, and the all-zero rows to "0" digits in the multiplier. Life does get simpler in some ways when we switch to binary numbers: "multiplying a multiplier digit by the multiplicand" now means just gating a copy of the multiplicand into that position if the digit is "1," and not doing so if the digit is "0."

Seymour Cray, the master computer designer from Chippewa Falls, Wisconsin, whose career has spanned three companies (Univac, Control Data, and now Cray Research) and many inventions, first observed some time in the late 1950s that *computers* also could actually multiply this way, if one merely provided enough components. This last qualifying remark; in those days when even transistors, let alone integrated circuits, in computers were still a novelty was by no means a trivial one! To prove his point (and satisfy a government contract), Cray designed, and Control Data built, a 48x48 multiplier which operated in one microsecond, about 1960. This multiplier was part of a special-purpose array processor for a classified application, and was so big that a CDC 1604 (then considered a large-scale processor) served as its input/output controller. In principle, such a multiplier at that time would have had to consist of 48 48-bit full adders or "mills," each of which received one input 48-bit number from the outputs of the mill immediately above it in the array, and the other 48-bit number from a gate which either allowed the multiplicand to pass through, or else supplied an all-zero 48-bit number. Actually, these mills have to be somewhat *longer* than 48 bits. Anyway, that is at least 2304 full adders, and in 1960 a full-adder circuit normally occupied one small plug-in circuit card.

A later version of this multiplier, in the CDC 7600 supercomputer, could produce one 48x48 product out every 275 nanoseconds on a pipelined basis. The pipelining was asynchronous, and the entire humungous array of adders and gating logic could have up to three different products rippling down it at a given instant!

Back to the 1980s. Monolithic Memories has for several years produced an 8x8 Cray multiplier, the 67588, as a single 600-mil 40-pin DIP. After we invented this part, AMD second-sourced it, and by now it has become an industry standard. We now also have faster pin-compatible parts, the 54/74S558 and 74S557. Like other West Coast companies 2,000 miles from Wisconsin and Minnesota where Seymour Cray does his inventing, Monolithic Memories previously used the term "combinatorial multiplier" instead of "Cray multiplier" for this type of part. However, "combinatorial multiplier" has nine extra letters and five extra syllables, and also inadvertently implies that the technique involves combinatorial logic rather than arithmetic circuits. Some West Coast designs, including our 67588, use a modified internal array with only half as many full-adder circuits and slightly different interconnections, based on the two-bit "Booth-multiplication" algorithm (see reference 1), plus the two-bit "Wallace-tree" or "carry-save adder" technique (see references 2 and 3). Conceptually, however, the entire chip or system continues to operate as a Cray multiplier.

The '558, in particular can be thought of as a static logic network which fits exactly the binary multiplication example of Figure 5. (See now why I insisted on using 8-bit binary numbers?) There are no flipflops or latches whatever in the '558 — it is a "flow-through" device. Its 40 pins are used up as follows:

Use of Pins	Input, Output, or Voltage	Number of Pins
Multiplier	I	8
Multiplicand	I	8
Double-Length Product	O	16
Complement of Most-Significant Bit of Double-Length Product	O	1
3-State Output Enable	I	1
Number-Interpretation-Mode Control	I	2
Rounding Control for Product	I	2
Power and Ground	V	2
		40

Table 1. Use of Pins in the '558

The two number-interpretation-mode control pins, one for the multiplier and one for the multiplicand, allow the format for each of these two 8-bit input numbers to be chosen independently, as follows:

Control Input	Interpretation of 8-bit Input Number
L	8-bit unsigned
H	7-bit plus a sign bit

Table 2. Mode Control Input Encoding

The two rounding control pins allow either integer (right-justified) or fractional (left-justified) interpretation of the 14-bits-plus-sign double-length product of two 7-bits-plus-sign numbers for internal rounding of the double-length result to the most accurate 8-bit number. The control encoding is:

R_S Input	R_U Input	Effect
L	L	Disable Rounding
L	H	Round Unsigned
H	L	Round Signed
H	H	Nonsense (see below)

Table 3. Rounding Control Input Encoding

Rounding is normally disabled if the entire 16-bit double-length product output is to be used. If only an 8-bit subset of this product is to be used, this subset can be either bits 15-8 for unsigned rounding as shown in Figure 7, or bits 14-7 for signed rounding as shown in Figure 8. In either case, a "1" is forced into the '558's internal adder network at the bit position indicated by the arrow; adding a "1" into the bit position *below* the least-significant bit of the final answer has the effect of rounding, as you can see after a little thought. Obviously, forcing a "1" into *both* of these adder positions at the same time is a nonsense operation for most applications — it adds a "3" into the middle of the double-length result.

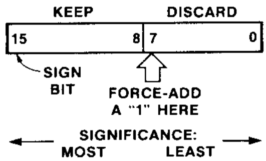


Figure 7. Unsigned Rounding

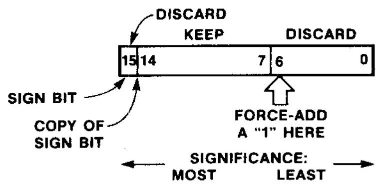


Figure 8. Signed Rounding

By now you probably have a fairly good idea of what a '558 is, and would like a few hints as to how to use it, right? First of all, there is an occasional application in things like video games for *very* fast multiplication, either 8x8 or 16x16, controlled by an 8-bit microprocessor, where there would be one '558 per system (see reference 4). More typically, however, the '558 is a building block, and several of them are used within one system; in fact, maybe more than several — "many." In the usual Silicon-Valley jargon, we can *cascade* a number of '558 (8x8) Cray-multiplier chips to create larger Cray multipliers at the systems level.

For the sake of concreteness, I'll discuss the case of 56x56 multipliers, which are appropriate in floating-point units which deal with "IBM-long-format" numbers which have a 56-bit mantissa. Any computer which emulates, or uses the same floating-point format as, any of the following computers can use such a multiplier:

11

IBM 360/370
Amdahl 470
Data General Eclipse
Gould/System Engineering SEL 32
Norsk Data 500 (different format)

There are two basic approaches: serial-parallel, and fully parallel. The serial-parallel approach uses seven '558s, and requires seven full multiply-and-add cycles. On the first cycle, the least-significant eight bits of the multiplier are multiplied by the entire multiplicand, and this partial product is saved. On the second cycle, the next-least significant eight bits of the multiplier are multiplied by the multiplicand, and that product (shifted eight bit positions to the left) is added into the first partial product to form the new partial product. And so forth, for five more cycles. It's almost like our decimal-multiplication example of Figure 1, except that instead of base-10 decimal digits we now have base-256 superdigits.

The fully-parallel approach totally applies Cray's usual design philosophy (sometimes characterized as "big, fast, and simple") at the systems level. It uses 49 '558s, in seven ranks; the 'i'th rank performs an operation corresponding to that done during the 'i'th cycle in the serial-parallel implementation. In principle, a complete mill is used to add the outputs of one rank of '558s to those of the rank above it. Or, alternatively, these mills can be laid out in a "tree" arrangement, such as:

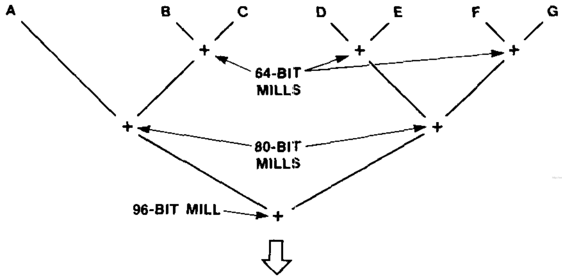


Figure 9. "Tree" Summing Arrangement of Mills for a 56x56 Cray Multiplier

Each letter stands for one rank of '558s, and each "+" stands for a mill of the indicated length. More involved "Wallace-tree" techniques are usually preferable. (See reference 3). If the least-significant half of the double-length product is *never* needed, only 34 'S558s are required. There is one subtlety which needs to be mentioned. If, conceptually, a '558 looks like a diamond —

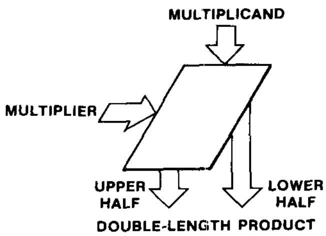


Figure 10. A Single '558 in "Diamond" Notation

then, the 8x56 multiplier for the serial-parallel configuration (which is also one rank of the fully-parallel configuration, which has seven such ranks) looks like this:

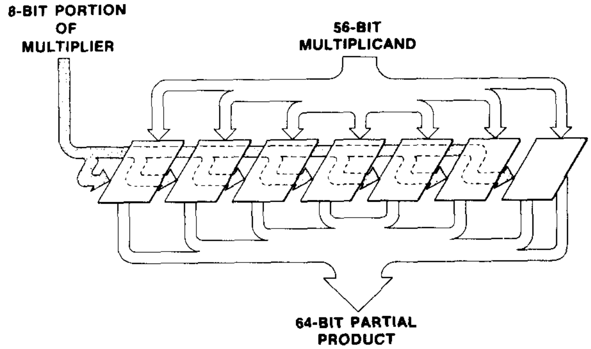


Figure 11. 8x56 Cray Multiplier in "Diamond" Notation

As you may discover after a moment's thought, each slanted double line in Figure 8 calls for addition of the outputs of *two* '558s — the eight most significant bits of one, and the eight least-significant bits of the next one to the left. There must also be an extra adder (or at least a "half adder") to propagate the carries from this addition all the way over to the left end of the result. The upshot is that an extra 56-bit mill is needed, in addition to the '558s. The eight least-significant bits of the least-significant '558 do not have to go through this mill, since they do not get added to anything else.

One final note: building up a large Cray-multiplier configuration out of '558s requires a *lot* of full adders, or else a lot of something else equivalent to them. Monolithic Memories also makes 74S381 (a 4-bit "ALU" or "Arithmetic Logic Unit") and the 74S182 (a carry-bypass circuit which works well with the '381); and two faster ALUs, the 54/74F381 and the 54/74F382 are in design. These ALUs and bypasses are excellent building blocks from which to assemble the mills used for summation within a rank of '558s, and also the mills used for tree-summation of the outputs of all ranks. For how to put together one of these mills using '381s, '382s, and '182s, see reference 1. For how to use PROMs as Wallace trees, see reference 3.

Now you can go ahead, design your Cray multiplier out of '558s, and start multiplying full-length numbers together in a fraction of a microsecond. Sound like fun?

References

1. "Doing Your Own Thing in High-Speed Digital Arithmetic," Chuck Hastings, Monolithic Memories Conference Proceedings Reprint CP-102

2. "Real-Time Processing Gains Ground with Fast Digital Multiplier," Shlomo Waser and Allen Peterson, *Electronics,* September 29, 1977.

3. "Big, Fast and Simple — Algorithms, Architecture, and Components for High-End Superminis," Ehud "Udi" Gordon and Chuck Hastings, 1982 *Southcon Professional Program,* Orlando, Florida, March 23-25, 1982, paper no. 21/3.

4. "An 8x8 Multiplier and 8-bit μP Perform 16x16-bit Multiplication," Shai Mor, EDN, November 5, 1979, Monolithic Memories Article Reprint AR-109.

NOTE: All of these references are available as application notes from Monolithic Memories Inc.