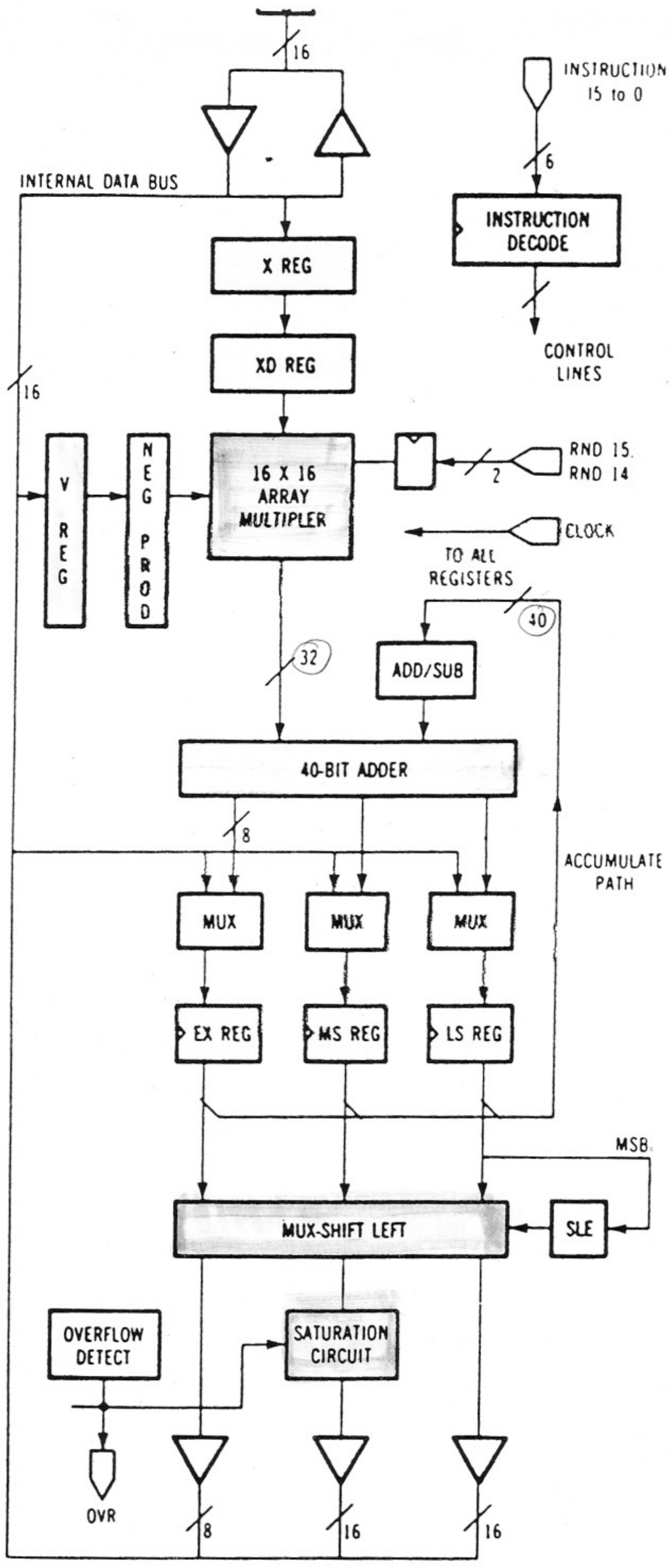


Single-port multipliers, such as Analog Devices' ADSP-1010, greatly reduce pin requirements, power consumption, and interface logic. The 16- x 16-bit CMOS 1010 features a 100-ns I/O cycle, and draws only 150 mW, all in a 28-pin DIP.

This design technique can be expanded to access several functions stored in ROM by using a preloadable counter. The counter is preloaded with the starting address of the desired program in ROM. A dedicated control bit from the ROM is used to flag the counter, denoting the end of the program segment. Note that a latch is placed in front of the preloadable counter so that a host microprocessor can feed the required starting addresses if desired. This design is illustrated below.

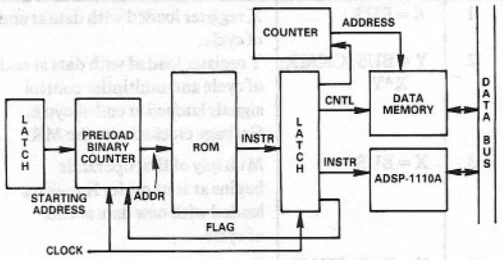


Figure 12.

PLA-Based State Machine

Instead of using the purely sequential approach of the ROM/counter solution, a latched PLA can generate the microcode. This state machine reduces real estate by entirely eliminating the latch (which is internal to the PLA) and the counter. No counter is needed since, in a state machine, the next output state is determined by the current output state. Use of state diagrams and CAD techniques provide the PLA truth table. However, in designs that require many states and complex state diagrams, a ROM-based sequential machine may be easier to implement.

Interfacing the ADSP-1110A to a Microprocessor

Because of its high speed, the ADSP-1110A can be used in an accelerator for microprocessors such as the Intel 286 or the Motorola 68000, performing dedicated macro-routines. The host microprocessor communicates with the ADSP-1110A via memory-mapping or I/O mapping (depending on the microprocessor). Also, the microprocessor and the ADSP-1110A must both have access to data memory. The microprocessor merely triggers the ADSP-1110A circuit and proceeds to perform some other task while the ADSP-1110A executes its macro-routine such as a matrix multiply, inverse function, square-root function, or digital filter. The following diagram illustrates a typical interface to a microprocessor.

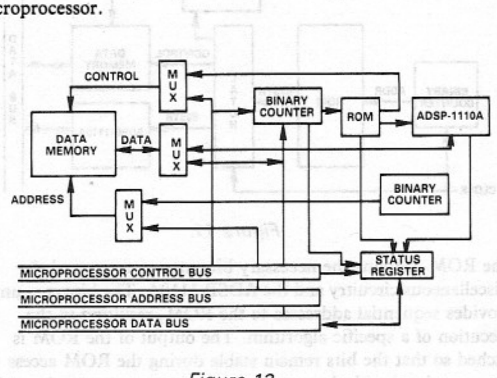


Figure 13.

Any signal that must communicate with the microprocessor is connected to the external status register. The status register is either I/O mapped or memory mapped. The overflow line from the ADSP-1110A, along with the end of program control flag, is also connected to the status register. Also, flags may be used to interrupt the microprocessor. Note that the high-speed data memory is available to both the microprocessor and the ADSP-1110A circuit. The multiplexers performing this selection are controlled by the microprocessor, with multiplexer select lines coming from the external status register.

APPLICATIONS

The ADSP-1110A is a high-performance component for a host of digital signal processing applications including FFT's, digital filters, and double-precision multiplication.

FFT Applications

The fast Fourier transform (FFT) is the principal algorithm used to analyze the frequency content of a signal. The FFT significantly reduces the time required to compute a Fourier transform by taking advantage of patterns in the computations to economize on multiplications. The ADSP-1110A performs the "butterfly," the key arithmetic operation in an FFT, entirely on-chip.

Figure 14 illustrates a decimation-in-time butterfly. As outlined by equations (1)

$$A_0' = A_0 + A_1 e^{j\theta} \tag{1}$$

$$A_1' = A_0 - A_1 e^{j\theta}$$

the complex number A_1 is multiplied by a rotation term, $R = e^{j\theta}$, and added to the complex number A_0, producing A_0'. A_1' is obtained by subtracting the complex product $A_1{*}R$ from A_0. The rotation R can be written:

$$R = e^{j\theta} = \cos\theta + j\sin\theta \equiv C + jS \tag{2}$$

In an FFT A_0 and A_1 are complex numbers. Let

$$A_0 = X_0 + jY_0 \tag{3}$$
$$A_1 = X_1 + jY_1$$

Then,

$$\begin{aligned}(A_1)e^{j\theta} &= (X_1 + jY_1)(C + jS) \\ &= (X_1C - Y_1S) + j(X_1S + Y_1C)\end{aligned} \tag{4}$$

allowing A_0' and A_1' to be represented as:

$$\begin{aligned}A_0' &= X_0 + jY_0 + [(X_1C - Y_1S) + j(X_1S + Y_1C)] \\ &= X_0' + jY_0'\end{aligned} \tag{5}$$

$$\begin{aligned}A_1' &= X_0 + jY_0 - [(X_1C - Y_1S) + j(X_1S + Y_1C)] \\ &= X_1' + jY_1'\end{aligned}$$

Expanding and equating real and imaginary terms yields:

$$\begin{aligned}X_0' &= X_0 + (X_1C - Y_1S) \\ Y_0' &= Y_0 + (X_1S + Y_1C) \\ X_1' &= X_0 - (X_1C - Y_1S) \\ Y_1' &= Y_0 - (X_1S + Y_1C)\end{aligned} \tag{6}$$

Equations 6 can be used by the ADSP-1110A to efficiently implement the butterfly computation. First, X_0 is loaded into

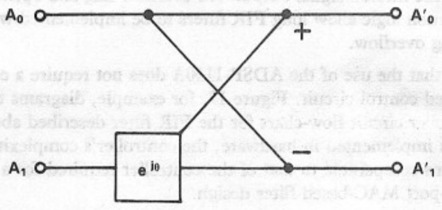


Figure 14. FFT "Butterfly" Diagram

MR by multiplying it by positive full scale (k=0111...1). ("+1" cannot be represented in fractional two's complement, though "–1" can be. Scaling all terms by the factor "0.111111111111111" [binary] fits all values of sine and cosine into fractional two's complement and introduces less error than consistently failing to represent positive unity.) X_0' is obtained as follows:

$$X_0' = kX_0 + X_1C - Y_1S \qquad (7)$$

Note that the factor k is not equal to unity. To ensure consistency in results, all stored cosine and sine factors (C and S) should similarly be scaled by k. Then, X_1' can be simply computed:

$$\begin{aligned} X_1' &= kX_0 - (X_1C - Y_1S) \qquad (8)\\ &= 2kX_0 - X_0' \end{aligned}$$

where X_0' is the accumulator's contents, and $2kX_0$ results from a mixed-mode multiplication of 2k and X_0. This operation represents a multiply/subtract, which illustrates an additional feature of the ADSP-1110A. Conventional MAC's cannot perform mixed-mode multiplies or multiply/subtracts.

Table VI provides the details for computing an FFT Butterfly with the ADSP-1110A. Each point requires ten cycles to compute the real component and ten cycles for the imaginary component. A 1024-point FFT requires 5120 butterflies.

A butterfly calculation contains a series of multiply/accumulates and multiply/subtracts. This presents a challenge in rounding the result, because rounded outputs from earlier cycles become inputs in later cycles. The rounding on cycles 4, 6, 14, and 16 ensures that the outputs (lines 7, 10, 17, and 20) are rounded correctly. Lines 4 and 14 round on bit 14, consistent with a left shift during output. However, lines 6 and 16 round on bit 15 to arrive at $X_1'$and Y_1'. In performing the multiply and subtract on these lines, the original round on lines 4 and 14 becomes inverted. To compensate, 2 must be added to the 14th bit, 1 to compensate for the previous round, and then 1 to round the current result. This can be easily accomplished in one step by adding a 1 to bit 15 rather than 2 to bit 14.

Cycle	Instruction	Comments
18'.	X = BUS	Load k
19'.	Y = BUS;CKMR;X_{TC}*Y_{TC}	Load X_0, MR = Previous
20'.	BUS = MS(sl)	Output Y_1' from previous butterfly
1.	X = BUS	Load C
2.	Y = BUS;CKMR;X_{TC}*Y_{TC} + MR	Load X_1, MR = kX_0
3.	X = BUS	Load S
4.	Y = BUS;CKMR; – X_{TC}*Y_{TC} + MR/RND14	Load Y_1, MR = $kX_0 + X_1C$
5.	X = BUS	Load 2k
6.	Y = BUS;CKMR;X_{US}*Y_{TC} – MR/RND15	Load X_0, MR = $kX_0 + (X_1C - Y_1S)$ + RND14
7.	BUS = MS(sl)	Output X_0'
8.	X = BUS	Load k
9.	Y = BUS;CKMR;X_{TC}*Y_{TC}	Load Y_0, MR = $kX_0 - (X_1C - Y_1S)$ + RND14
10.	BUS = MS(sl)	Output X_1'
11.	X = BUS	Load S
12.	Y = BUS;CKMR;X_{TC}*Y_{TC} + MR	Load X_1, MR = kY_0
13.	X = BUS	Load C
14.	Y = BUS;CKMR;X_{TC}*Y_{TC} + MR/RND14	Load Y_1, MR = $kY_0 + X_1S$
15.	X = BUS	Load 2k
16.	Y = BUS;CKMR;X_{US}*Y_{TC} – MR/RND15	Load Y_0, MR = $kY_0 + (X_1S + Y_1C)$ + RND14
17.	BUS = MS(sl)	Output Y_0'
18.	X = BUS	Load k
19.	Y = BUS;CKMR;X_{TC}*Y_{TC}	Load new X_0, MR = $kY_0 - (X_1S + Y_1C)$ + RND14
20.	BUS = MS(sl)	Output Y_1'

Table VI. Sample FFT "Butterfly" Sequence

FIR Filters

The ADSP-1110A is readily included in an FIR filter configuration. Figure 15 diagrams an N-tap finite impulse response (FIR) filter. FIR filters perform convolution in the time domain, corresponding to multiplication in the frequency domain. The coefficients h_i represent the filter's impulse response—the time domain equivalent of the filter's desired frequency response.

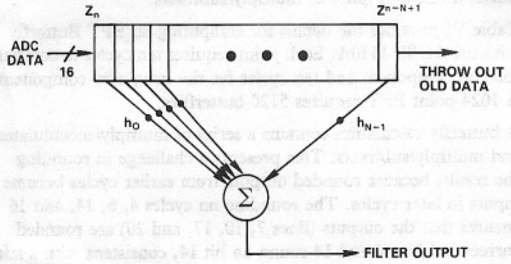
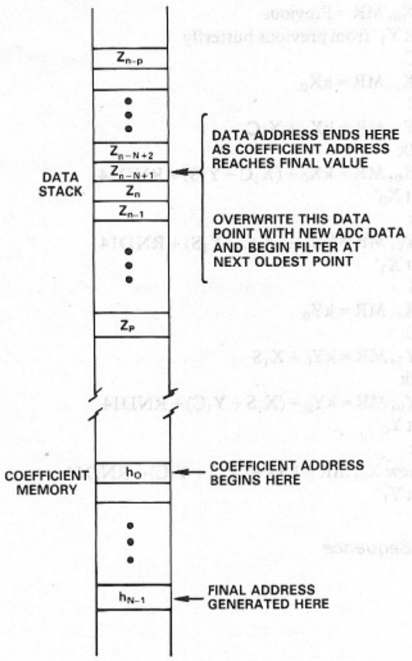


Figure 15. FIR Filter Design

When implemented with the ADSP-1110A, FIR filters employ a single RAM with a memory map as diagrammed in Figure 16. This contrasts with the multiple RAMs usually required in three-port MAC designs. Except in adaptive filters, where the filter response changes to meet changing system requirements, the filter coefficients remain constant.

Input data, on the other hand, is continuously updated. Each new data sample overwrites the oldest data point in RAM, an action that is tracked by an address counter. The Z_{n-N} sample, for instance, is overwritten with the new Z_n sample, and data points are addressed as a circular buffer.

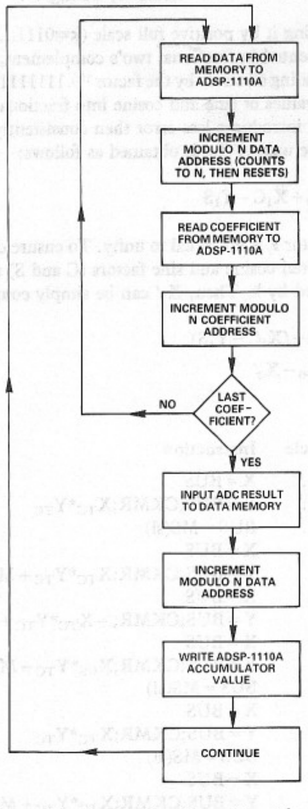


Z_{n-p}
Z_{n-N+2}
Z_{n-N+1}
Z_n
Z_{n-1}
Z_P
DATA STACK
DATA ADDRESS ENDS HERE AS COEFFICIENT ADDRESS REACHES FINAL VALUE
OVERWRITE THIS DATA POINT WITH NEW ADC DATA AND BEGIN FILTER AT NEXT OLDEST POINT
COEFFICIENT MEMORY
h_O
COEFFICIENT ADDRESS BEGINS HERE
h_{N-1}
FINAL ADDRESS GENERATED HERE

Figure 16. Memory Map

In the time interval between new data samples, the filter multiplies each of the N previously stored data samples, Z_i, by the respective filter coefficients, h_i. The resulting sum of the products represents the filtered signal output. An overflow flag and optional saturation logic allow long FIR filters to be implemented without risking overflow.

Note that the use of the ADSP-1110A does not require a complicated control circuit. Figure 17, for example, diagrams the controller circuit flow-chart for the FIR filter described above. When implemented in hardware, the controller's complexity remains comparable to that of the controller required for a three-port MAC-based filter design.



READ DATA FROM MEMORY TO ADSP-1110A
INCREMENT MODULO N DATA ADDRESS (COUNTS TO N, THEN RESETS)
READ COEFFICIENT FROM MEMORY TO ADSP-1110A
INCREMENT MODULO N COEFFICIENT ADDRESS
LAST COEF-FICIENT?
NO
YES
INPUT ADC RESULT TO DATA MEMORY
INCREMENT MODULO N DATA ADDRESS
WRITE ADSP-1110A ACCUMULATOR VALUE
CONTINUE

Figure 17. Controller Flow Chart for FIR Filter

IIR Filters

Infinite impulse reponse (IIR) filters use feedback to improve filter performance at the cost of a more complicated design. The principal advantage of an IIR filter is the relatively small number of multiplies needed to achieve a high-performance filter. The ADSP-1110A, unlike conventional MAC's, has architectual features that eliminate some of the disadvantages associated with implementing IIR's.

The time required for the ADSP-1110A to calculate a biquad section of an IIR filter is (5 MAC operations) × (two cycles/operation) + (output cycle), or eleven cycles. In the equation for a biquad,

$$Y_0 = a_0X_0 + a_1X_{-1} + a_2X_{-2} - b_1Y_{-1} - b_2Y_{-2} \qquad (9)$$

the coefficient b_1 generally lies between 1 to 2. The most conventional way to represent the coefficients and data is in fractional two's complement notation. Since this numbering system only ranges from -1 to 0.999. . . , all coefficients and data for the IIR filter have to be divided by 2 to handle b_1 when using a conventional MAC. To compensate, external shifters are needed on output to shift the result up by one bit (multiply by 2).

A coefficient in the $+2$ to -2 range can be handled with the ADSP-1110A by using a mixed-mode multiply. Since the coefficient's sign is known in advance, multiply/add and multiply/subtract operations can supply the sign to an unsigned magnitude number. The MS register is left-shifted as usual on output to obtain the correct result.

Stability is an important issue for IIR filters. The ADSP-1110A's wide accumulator, together with the hard-limiting provided by its saturation circuit, prevent the overflow problems and large-scale oscillations that often plague IIR filters.

Double-Precision Multiplies

In order to handle double-precision multiplication (multiplying two 32-bit two's complement numbers), conventional MACs require additional external logic. The ADSP-1110A, in contrast, performs these operations without external support. Moreover, the device performs a double-precision multiplication in fifteen cycles, seven of which represent overhead.

Equation 10 represents a double-precision multiply:

$$\begin{aligned} P &= (X)(Y) \\ &= (MSW_x + LSW_x \cdot 2^{-16})(MSW_y + LSW_y \cdot 2^{-16}) \qquad (10) \\ &= MSW_x \cdot MSW_y + (MSW_x \cdot LSW_y + MSW_y \cdot LSW_x) \cdot 2^{-16} \\ &\quad + LSW_x \cdot LSW_y \cdot 2^{-32} \end{aligned}$$

where P is the 64-bit product of two 32-bit two's complement numbers, X and Y. MSW_x represents the 16 most significant bits of word X, and LSW_x represents the 16 least significant bits. The product P equals the sum of partial products; each partial product's sign and significance must be taken into account in order to obtain the proper result.

A double-precision multiply requires no external logic. Furthermore, as illustrated in the following sequence, the ADSP-1110A performs the operation in 15 cycles.

In this double-precision multiply sequence, shown in Table VII, the four basic multiplications require only eight cycles. Cycles 5, 6, 11, 12, 13, 14, and 15 represent overhead.

Double-Precision MACs

The previous discussion concerned double-precision multiplies. The ADSP-1110A also readily handles double-precision multiply/accumulate operations. For example:

$$AP = \sum_{i=1}^{N} X_i Y_i \qquad (11)$$

$$= \sum_{i=1}^{N} (MSW_{xi} + LSW_{xi} \cdot 2^{-16})(MSW_{yi} + LSW_{yi} \cdot 2^{-16})$$

Cycle	Operation	Comments
1.	X = BUS	Load LSW_x.
2.	Y = BUS;CKMR; X_{US}*Y_{US}/RND15	Load LSW_y and multiply (unsigned).
3.	NOP	No op.
4.	Y = BUS;CKMR; X_{US}*Y_{TC} + MR	Load MSW_y and perform MAC (mixed-mode).
5.	LS = MS	MS shifts into LS. The LS of the $(LSW_x)(LSW_y)$ product is discarded.
6.	MS = EX	Shift EX into MS.
7.	X = BUS	Load MSW_x.
8.	Y = BUS;CKMR; X_{TC}*Y_{US} + MR/ RND14	Load LSW_y and perform MAC (mixed-mode) with round in bit 14.
9.	NOP	No op.
10.	Y = BUS;CKMR X_{TC}*Y_{TC} + MR	Load MSW_y and perform MAC (two's complement).
11.	LS = MS	Shift MS into LS.
12.	MS = EX	Shift EX into MS.
13.	CKMR	Clock the output registers. This loads the MAC from cycle 10 into the accumulator.
14.	BUS = MS(sl)	Output MS with left shift.
15.	BUS = LS(sl)/ RND14	Output LS with left shift. The SLE register provides an extra bit of precision.

Table VII.

where each X and Y is a 32-bit number, and AP is a 72-bit accumulated product. Note that AP can be expressed as the sum of accumulated partial products as follows:

$$AP = \left[\left[\sum_{i=1}^{N} (MSW_{xi})(MSW_{yi})\right]\right] + \qquad (12)$$

$$+ \left[\left[\sum_{i=1}^{N} ((MSW_{xi})(MSW_{yi}) + (LSW_{xi})(MSW_{yi}))\right]\right] \cdot 2^{-16}$$

$$+ \left\{\sum_{i=1}^{N} (LSW_{xi})(LSW_{yi})\right\} \cdot 2^{-32}$$

Computing the accumulated double-precision product AP requires the same basic sequence as in computing a single-precision MAC. Simply compute a summation of partial products, rather than the summation of products themselves.

The summation of partial products often leads to sums greater than 32-bits. The 8-bit extension register stores any overflow, letting the summation proceed without error. The output and shift cycles occur once each at the end of the appropriate partial product calculation. A 32-point double-precision FIR filter, requires (32-points)(4-multiplies)(2 cycles/multiply) $+9$ overhead cycles = 273 total cycles.

An optional procedure, which cuts the multiply/accumulate time by roughly 25%, entails omitting the $LSW_x \times LSW_y$ accumulation and instead adding ¼ of the number of accumulations to the final result. This removes the bias because the LSW's of both words have a mean value of ½ and when multiplied together have a mean product of ¼. Thus any bias in the answer is removed. A simple way to add 1's to the LSB is to assert the round control on the appropriate number of MAC operations.