

MULTIPLY-ACCUMULATE (MAC) UNIT FOR SINGLE-INSTRUCTION/MULTIPLE-DATA (SIMD) INSTRUCTIONS

5

BACKGROUND

[0001] Digital signal processors (DSPs) may operate as SIMD (Single-Instruction/Multiple-Data), or data parallel, processors. In SIMD operations, a single instruction is sent to a number of processing elements, which perform the same operation on different data. SIMD instructions provide for several types of standard operations including addition, subtraction, multiplication, multiply-accumulate (MAC), and a number of special instructions for performing, for example, clipping and bilinear interpolation operations.

15 [0002] Many DSP applications, including many speech codecs, require high performance 16-bit multiply-accumulate (MAC) operations. To achieve high performance for these 16-bit DSP applications, 64-bit SIMD instructions may be introduced. The 64-bit SIMD instructions may be used to handle media streams more efficiently and reduce register pressure and memory traffic since four 16-bit data items may be loaded into a 64-bit register at one time.

[0003] While high throughput is an important factor for achieving high performance, power consumption may also be an important consideration in designing DSPs for wireless/handheld products. Accordingly, MAC architectures

which are capable of high performance with low power demands may be desirable for use in DSPs.

BRIEF DESCRIPTION OF THE DRAWINGS

- 5 [0004] Figure 1 is block diagram of a dual multiply-accumulate (MAC) unit according to an embodiment.
- [0005] Figure 2 is a block diagram illustrating a MAC SIMD (Single-Instruction/Multiple-Data) operation according to an embodiment.
- 10 [0006] Figures 3A to 3C are flowcharts describing a MAC SIMD operation according to an embodiment.
- [0007] Figures 4A to 4C are block diagrams illustrating pipelined instruction sequences utilizing data forwarding according to an embodiment.
- 15 [0008] Figures 5A to 5C are block diagrams illustrating pipelined instruction sequences utilizing intermediate data forwarding according to an embodiment.
- [0009] Figures 6A and 6B are flowcharts describing a 32-bit X 32-bit MAC operation performed on a tightly coupled
- 20 dual 16-bit MAC unit according to an embodiment.
- [0010] Figure 7 is a block diagram of a mobile video unit including a MAC unit according to an embodiment.

DETAILED DESCRIPTION

[0011] Figure 1 illustrates a Multiply-Accumulate (MAC) unit 100 according to an embodiment. The MAC unit 100 may be used to perform a number of different SIMD (Single-Instruction/Multiple-Data) operations.

5 [0012] The MAC unit 100 may have a tightly coupled dual 16-bit MAC architecture. A 16-bit MAC SIMD operation 200 which may be performed by such a MAC unit is shown conceptually in Figure 2. The contents of two 64-bit registers, 202 (wRn) and 204 (wRm), may be treated as four
10 pairs of 16-bit values, A_0 - A_3 (wRn) and B_0 - B_3 (wRm). The first 16 bits to fourth 16 bits of wRn are multiplied by the first 16 bits to fourth 16 bits of wRm, respectively. The four multiplied results P_0 - P_3 are then added to the value in 64-bit register 206 (wRd), and the result is sent to a
15 register 206.

[0013] The MAC operation 200 may be implemented in four execution stages: (1) Booth encoding and Wallace Tree compression of B_1 and B_0 ; (2) Booth encoding and Wallace Tree compression of B_3 and B_2 ; (3) 4-to-2 compression, and
20 addition of the low 32-bits of the result; and (4) addition of the upper 32-bits of the result. These four stages may be referred to as the CSA0, CSA1, CLA0, and CLA1 stages, respectively.

[0014] Figures 3A to 3C illustrate a flow chart
25 describing an implementation 300 of the MAC operation 200

according to an embodiment. In the CSA0 stage, a MUX & Booth encoder unit 102 selects B_0 (16 bits) and encodes those bits (block 302). Control signals are generated, each of which select a partial product vector from the set $\{0, -A_0, -2A_0, A_0, 2A_0\}$. Nine partial product vectors, Pa0 to Pa8, are generated and passed to a MUX array 104 (block 304). All nine partial product vectors and the low 32 bits of the value in register 206 (wRd) are compressed into two vectors by a Wallace Tree unit 106 (block 306). The two vectors include a sum vector and a carry vector, which are stored in a sum vector flip-flop (FF) 108 and a carry vector FF 110, respectively.

[0015] A MUX & Booth encoder unit 112 selects B_1 (16 bits) and encodes those bits (block 308). Control signals are generated, each of which select a partial product vector from the set $\{0, -A_1, -2A_1, A_1, 2A_1\}$. Nine partial product vectors, Pb0 to Pb8, are generated and passed to a MUX array 114 (block 310). All nine partial product vectors and a zero vector are compressed into two vectors by a Wallace Tree unit 116 (block 312). The two vectors include a sum vector and a carry vector, which are stored in a sum vector FF 118 and a carry vector FF 120, respectively.

[0016] In the CSA1 stage, four vectors from the sum and carry vectors FFs 108, 110, 118, and 120 from the CSA0 stage are compressed into vectors Vs_0 and Vc_0 by a MUX & 4-to-2

compressor unit 122 (block 314). The MUX & Booth encoder unit 102 selects B_2 (16 bits) and encodes those bits (block 316). Control signals are generated, each of which select a partial product vector from the set $\{0, -A_2, -2A_2, A_2, 2A_2\}$.

5 Nine partial product vectors are generated (block 318). All nine partial product vectors and vector Vs_0 are then compressed into two vectors by the Wallace Tree unit 106 (block 320). The two vectors include a sum vector and a carry vector, which are stored in a sum vector FF 108 and a
10 carry vector FF 110, respectively.

[0017] The MUX & Booth encoder 112 selects B_3 (16 bits) and then encodes those bits (block 322). Control signals are generated, each of which select a partial product vector from the set $\{0, -A_3, -2A_3, A_3, 2A_3\}$. Nine partial product
15 vectors are generated (block 324). All nine partial product vectors and vector Vc_0 are then compressed into two vectors by the Wallace Tree unit 116 (block 326). The two vectors include a sum vector and a carry vector, which are stored in a sum vector FF 118 and a carry vector FF 120, respectively.

20 [0018] In the CLA0 stage, four vectors from FFs 108, 110, 118, and 120 from the CSA1 stage are sent to the 4-to-2 compressor unit 122 to generate vector Vs_1 and vector Vc_1 (block 327). The lower 32 bits of Vs_1 and Vc_1 are added by the carry look-ahead (CLA) unit 124 to generate the low 32
25 bits of the final result (block 328).

[0019] In the CLA1 stage, the upper bits of Vs_1 and Vc_1 are sign extended to two 32-bit vectors (block 330). The extended vectors and the upper 32-bits of wRd are then compressed into two vectors by a 3-to-2 compressor unit 126 (block 332). Two compressed vectors and carry-in bit from the CLA0 unit 124 are added together by CLA unit 128 to generate the upper 32-bits of the final result (block 334).

[0020] As described above, the Booth encoding and vectors compressing take two cycles to finish. In the first cycle, the results from both Wallace Tree units are sent back for further processing in the second cycle. Conventionally, all four vectors from FFs 108, 110, 118, and 120 would be sent back to the Wallace trees for further processing in the second cycle. However, it has been observed that the MUX & 4-to-2 compressor unit 122 may perform the 4-to-2 compression of the vectors faster than the MUX & Booth encoder units and the MUX arrays. Thus, only two vectors (Vs_0 and Vc_0) from the MUX & 4-to-2 compressor unit 122 are sent back to the Wallace Tree units 106 and 116. With this architecture, the feedback routings may be reduced and the Wallace Tree units 106, 116 made relatively smaller. Less feedback routings make the layout easier, which is desirable since routing limitations are an issue in MAC design.

[0021] Some conventional MAC implementations perform the 64-bit addition in one cycle. However, such MACs may not be

suitable for a very high frequency 64-bit datapath, and their results may not have enough time to return through the bypass logic, which is commonly used for solving data dependency in pipelining. Compared with conventional architectures, the dual MAC architecture shown in Figure 1 may be more readily implemented in very high frequency and low power application. The CLA1 stage may have less logic gates than that of CLA0 stage, which enables the final results to have enough time to return through the bypass logic, making this dual MAC architecture suitable for a high speed and low power 64-bit datapath.

[0022] The MAC unit may be used in a pipelined DSP.

Pipelining, which changes the relative timing of instructions by overlapping their execution, may increase the throughput of a DSP compared to a non-pipelined DSP. However, pipelining may introduce data dependencies, or hazards, which may occur whenever the result of a previous instruction is not available and is needed by the current instruction. The current operation may be stalled in the pipeline until the data dependency is solved.

[0023] Typically, data forwarding is based on a final result of an operation. For many DSP algorithms, the result of the previous MAC operation needs to be added to the current MAC operation. However, a MAC operation may take four cycles to complete, and the result of the previous MAC

operation may not be available for the current MAC operation. In this case, a data dependency called an accumulating dependency is introduced.

[0024] Figures 4A-4C show possible accumulating

5 dependency penalties for a standard data forwarding scheme.

The standard forwarding scheme is used to reduce the accumulating dependency penalty, where EX 402 is the execution stage for other non-MAC instructions. Even if the standard data forwarding is employed, an accumulating

10 dependency penalty is still two cycles in the worst case, which is shown in Figure 4A (note that, although there are three stalls 404 before the final result is available after the CLA1 stage, the first stall 404 in Figure 4A is due to a resource conflict in the Wallace Tree unit, which is not
15 counted as data dependency penalty). Two cycle penalties may be too severe for some DSP applications, and hence it is desirable to eliminate the accumulating dependency penalty.

[0025] The MAC unit 100 may be used to implement a new data forwarding scheme, referred to as intermediate data

20 forwarding, which may eliminate the accumulating dependency penalty. Instead of waiting for a final result from a previous operation, the intermediate data forwarding scheme forwards an intermediate result to solve data dependencies. Figures 5A-5C illustrate the sequences shown in Figures 4A-

4C, but implemented using an intermediate data forwarding technique.

[0026] As shown in Figures 5A-5C, the CSA0 stage 500 is segmented into two sub-stages 502 (BE0) and 504 (WT0) for Booth encoding and Wallace tree compressing, respectively, operands B_0 and B_1 . The CSA1 stage 506 is segmented into two sub-stages 508 (BE1) and 510 (WT1) for Booth encoding and Wallace tree compressing, respectively, operands B_2 and B_3 . The CLA0 stage 512 is segmented into two sub-stages 514 (4T2) and 516 (ADD0) for 4-to-2 compressing of vectors and low 32-bit addition of the final result. The CLA1 stage 518 includes the upper 32-bit addition of the final result 520 (ADD1).

[0027] In the cases shown in Figures 5A and 5B, the low 32-bits of intermediate vectors V_s , V_c of the first MAC instruction may be forwarded to the Wallace Tree units 106 and 116 for the second MAC instruction to solve the accumulating dependency. The upper 32-bit result of the first MAC instruction from the CLA1 unit 128 is forwarded to the MUX & 3-to-2 compressor unit 126. The stall 404 in Figure 5A is due to the Wallace Tree resource conflict, which is not counted as data dependency penalty.

[0028] In the case shown in Figure 5C, the final result of the first MAC instruction is not available when it is needed by the second MAC instruction, but the low 32-bit

result of the first MAC instruction is available. Instead of waiting for the final result, the low 32-bit result of the first MAC instruction is forwarded to the Wallace Tree unit 106 to solve the accumulating dependency. The upper 32-bit result of the first MAC instruction from the CLA1 unit 126 is forwarded to the MUC & 3-to-2 compressor unit 128.

[0029] The accumulating data dependency penalty comparisons between the standard data forwarding technique shown in Figures 4A to 4C and the intermediate data forwarding technique shown in Figures 5A to 5C are given in Table 1. As shown in Table 1, intermediate data forwarding may eliminate accumulating dependencies, which may enable relatively high throughput for many DSP applications.

	Penalty for case (A)	Penalty for case (B)	Penalty for case (C)
Standard data forwarding	2 cycles	2 cycles	1 cycle
Intermediate data forwarding	0 cycles	0 cycles	0 cycles

TABLE 1

[0030] A tightly coupled dual 16-bit MAC unit, such as that shown in Figure 1, may be used for 32-bit X 32-bit instructions as well as 16-bit SIMD instructions according to an embodiment. A 32-bit X 32-bit operation may be

divided into four 16-bit X 16-bit operations, as shown in the following equation:

$$\begin{aligned}
 &A[31:0] \times B[31:0] = (A[31:16] \times B[15:0] \times 2^{16} + \\
 &A[15:0] \times B[15:0]) + (A[31:16] \times B[31:16] \times 2^{16} + \\
 5 \quad &A[15:0] \times B[31:16]) \times 2^{16}.
 \end{aligned}$$

[0031] Figure 6 is a flow chart describing a 32-bit X 32-bit MAC operation 600 according to an embodiment. In the CSA0 stage, the partial product vectors of $A[15:0] \times B[15:0]$ are generated by the MUX & Booth encoder unit 102 (block 602). The Wallace Tree unit 106 compresses the partial product vectors into two vectors (block 604). The two vectors include a sum vector and a carry vector, which are stored in the sum vector FF 108 and the carry vector FF 110, respectively. The partial product vectors of $A[31:16] \times B[15:0]$ are generated by the MUX & Booth encoder unit 112 (block 606). The Wallace Tree unit 116 compresses the partial product vectors into two vectors (block 608). The two vectors include a sum vector and a carry vector, which are stored in the sum vector FF 108 and the carry vector FF 110, respectively.

[0032] In the CSA1 stage, two vectors from the sum vector FF 118 and carry vector FF 120 are shifted left 16 bits (block 610). The MUX & 4-to-2 compressor unit 122 compresses the shifted vectors and the other two vectors from the sum vector FF 108 and carry vector FF 110 into

vector Vs_0 and vector Vc_0 (block 612). The low 16 bit of Vs_0 and Vc_0 are sent to the CLA0 unit 124. The remaining bits are sent back to the Wallace Tree units 106 and 116. The final results from bit 0 to bit 15 are then generated by the

5 CLA0 unit 124 (block 614). The partial product vectors of $A[15:0] \times B[31:16]$ and the feedback vector from Vs_0 are then compressed into two vectors by the Wallace Tree unit 106 (block 616). The two vectors include a sum vector and a carry vector, which are stored in the sum vector FF 108 and

10 the carry vector FF 120, respectively. The partial product vector of $A[31:16] \times B[31:16]$ and the feedback vector from Vs_0 are then compressed into two vectors by the Wallace Tree unit 116 (block 618). The two vectors include a sum vector and a carry vector, which are stored in the sum vector FF

15 118 and the carry vector FF 120, respectively.

[0033] In the CLA0 stage, two vectors from the sum vector FF 118 and the carry vector FF 120 are shifted left 16 bits (block 620). The MUX & 4-to-2 compressor unit 122 compresses the shifted vectors and the other two vectors

20 from the sum vector FF 108 and the carry vector FF 110 into vector Vs_1 and vector Vc_1 (block 622). The low 16 bits of vectors Vs_1 and Vc_1 are added by the CLA0 unit 124. The final results from bit 16 to bit 31 are then generated (block 624).

[0034] In the CLA1 stage, the upper bits (from bit 16 to bit 47) of vectors V_{s1} and V_{c1} are added by the CLA1 unit 128 to generate the upper 32-bit final results (from bit 32 to bit 63) (block 626).

5 [0035] The MAC unit 100 may be implemented in a variety of systems including general purpose computing systems, digital processing systems, laptop computers, personal digital assistants (PDAs) and cellular phones. In such a system, the MAC unit may be included in a processor coupled
10 to a memory device, such as a Flash memory device or a static random access memory (SRAM), which stores an operating system or other software applications.

[0036] Such a processor may be used in video camcorders, teleconferencing, PC video cards, and High-Definition
15 Television (HDTV). In addition, the processor may be used in connection with other technologies utilizing digital signal processing such as voice processing used in mobile telephony, speech recognition, and other applications.

[0037] For example, Figure'7 illustrates a mobile video
20 device 700 including a processor 701 including a MAC unit 100 according to an embodiment. The mobile video device 700 may be a hand-held device which displays video images produced from an encoded video signal received from an antenna 702 or a digital video storage medium 704, e.g., a
25 digital video disc (DVD) or a memory card. The processor

100 may communicate with a cache memory 706, which may store instructions and data for the processor operations, and other devices, for example, an SRAM 708.

[0038] A number of embodiments have been described.

5 Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. For example, blocks in the flowchart may be skipped or performed out of order and still produce desirable results. Furthermore, the size of the
10 operands and number of operands operated on per SIMD instruction may vary. Accordingly, other embodiments are within the scope of the following claims.

15

CLAIMS

1. A method comprising:
performing a first compression operation in a first
multiply-accumulate operation in a pipeline;
5 generating two or more intermediate vectors in a first
compression operation in the first multiply-accumulate
operation; and
forwarding at least a portion of each of the two or
more intermediate vectors to a second multiply-accumulate
10 operation in the pipeline.

2. The method of claim 1, wherein said forwarding at
least a portion of each of the two or more intermediate
vectors comprises forwarding a lower portions of each of the
15 two or more intermediate vectors.

3. The method of claim 1, wherein said performing the
first compression operation comprises compressing a first
plurality of partial products into a first sum vector and a
20 first carry vector and compressing a second plurality of
partial products into a second sum vector and a second carry
vector.

4. The method of claim 1, wherein said generating two
25 or more intermediate vectors comprises compressing the first

and second sum vectors and the first and second carry vectors into an intermediate sum vector and an intermediate carry vector.

5 5. The method of claim 1, wherein said forwarding comprises forwarding at least a portion of each of the two or more intermediate vectors to a Wallace tree compression unit.

10 6. An article comprising a machine-readable medium which stores machine-executable instructions, the instructions causing a machine to:

perform a first compression operation in a first multiply-accumulate operation in a pipeline;

15 generate two or more intermediate vectors in a first compression operation in the first multiply-accumulate operation; and

forward at least a portion of each of the two or more intermediate vectors to a second multiply-accumulate
20 operation in the pipeline.

7. The article of claim 6, wherein the instructions causing the machine to forward at least a portion of each of the two or more intermediate vectors include instructions

causing the machine to forward a lower number of bits of each of the two or more intermediate vectors.

8. The article of claim 6, wherein the instructions
5 causing the machine to perform the first compression operation include instructions causing the machine to compress a first plurality of partial products into a first sum vector and a first carry vector and compress a second plurality of partial products into a second sum vector and a
10 second carry vector.

9. The article of claim 6, wherein the instructions causing the machine to generate two or more intermediate vectors include instructions causing the machine to compress
15 the first and second sum vectors and the first and second carry vectors into an intermediate sum vector and an intermediate carry vector.

10. The article of claim 6, wherein the instructions
20 causing the machine to forward include instructions causing the machine to forward at least a portion of each of the two or more intermediate vectors to a Wallace tree compression unit.

25 11. A method comprising:

compressing a first plurality of partial products into
a first sum vector and a first carry vector and compressing
a second plurality of partial products into a second sum
vector and a second carry vector in a first Wallace tree

5 compression stage of a multiply-accumulate operation;

compressing the first and second sum vectors and the
first and second carry vectors into a first intermediate sum
vector and a first intermediate carry vector; and

10 compressing the intermediate sum vector and a third
plurality of partial products and compressing the
intermediate carry vector and a fourth plurality of partial
products in a second stage of the multiply-accumulate
operation.

15 12. The method of claim 11, wherein the multiply-
accumulate operation comprises a single instruction/multiple
data (SIMD) operation.

13. The method of claim 11, further comprising:
20 generating the first plurality of partial products from
a first pair of operands;
generating the second plurality of partial products
from a second pair of operands;
generating the third plurality of partial products from
25 a third pair of operands; and

generating the fourth plurality of partial products
from a fourth pair of operands.

14. The method of claim 11, further comprising
5 forwarding the intermediate sum and carry vectors to a
second multiply-accumulate operation in a pipeline.

15. The method of claim 14, wherein said forwarding
comprises eliminating an accumulate data dependency in the
10 second multiply-accumulate operation.

16. An article comprising a machine-readable medium
which stores machine-executable instructions, the
instructions causing a machine to:

15 compress a first plurality of partial products into a
first sum vector and a first carry vector and compressing a
second plurality of partial products into a second sum
vector and a second carry vector in a first Wallace tree
compression stage of a multiply-accumulate operation;
20 compress the first and second sum vectors and the first
and second carry vectors into a first intermediate sum
vector and a first intermediate carry vector; and
compress the intermediate sum vector and a third
plurality of partial products and compressing the
25 intermediate carry vector and a fourth plurality of partial

products in a second stage of the multiply-accumulate operation.

17. The article of claim 16, wherein the multiply-
5 accumulate operation comprises a single instruction/multiple data (SIMD) operation.

18. The article of claim 16, further comprising instructions causing the machine to:

10 generate the first plurality of partial products from a first pair of operands;

generate the second plurality of partial products from a second pair of operands;

15 generate the third plurality of partial products from a third pair of operands; and

generate the fourth plurality of partial products from a fourth pair of operands.

19. The article of claim 16, further comprising
20 instructions causing the machine to forward the intermediate sum and carry vectors to a second multiply-accumulate operation in a pipeline.

20. The article of claim 16, wherein the instructions
25 causing the machine to forward include instructions causing

the machine to eliminate an accumulate data dependency in the second multiply-accumulate operation.

21. An apparatus comprising:

- 5 first and second Wallace tree compression units operative to compress vectors in first and second stages of a multiply-accumulate operation;
- a compressor operative to compress a plurality of vectors output from the first and second Wallace tree units
- 10 in the first stage of the multiply-accumulate operation into two intermediate vectors; and
- a data path from an output of the compressor to an input of a multiplexer, said multiplexer operative to selectively input one of said intermediate vectors to one of
- 15 said first and second Wallace tree compression units in the second stage of the multiply-accumulate operation.

22. The apparatus of claim 21, further comprising a dual multiply-accumulate unit.

20

23. The apparatus of claim 21, wherein the plurality of vectors comprise first and second sum vectors and first and second carry vectors.

24. The apparatus of claim 21, wherein the compressor comprises a four-to-two vector compressor.

25. The apparatus of claim 21, wherein the multiplexer
5 comprises a first multiplexer having an output coupled to the first Wallace tree compression unit and a second multiplexer having an output coupled to the second Wallace tree compression unit.

10 26. A system comprising:
a static random address memory; and
a processor coupled to the static random access memory,
said processor comprising a dual multiply-accumulate unit,
said unit including
15 first and second Wallace tree compression units
operative to compress vectors in first and second
stages of a multiply-accumulate operation,
a compressor operative to compress a plurality of
vectors output from the first and second Wallace tree
20 units in the first stage of the multiply-accumulate
operation into two intermediate vectors, and
a data path from an output of the compressor to an
input of a multiplexer, said multiplexer operative to
selectively input one of said intermediate vectors to
25 one of said first and second Wallace tree compression

units in the second stage of the multiply-accumulate operation.

27. The system of claim 21, wherein the multiplexer
5 comprises a first multiplexer having an output coupled to the first Wallace tree compression unit and a second multiplexer having an output coupled to the second Wallace tree compression unit.

10 28. A method comprising:
performing a multiply-accumulate operation on first and second 2n-bit operands as four n-bit operations.

29. The method of claim 28, wherein said performing
15 comprises:

generating partial product vectors from the lower n bits of the first operand and the lower n bits of the second operand;

generating partial product vectors from the upper n
20 bits of the first operand and the lower n bits of the second operand;

generating partial product vectors from the upper n bits of the first operand and the upper n bits of the second operand; and

generating partial product vectors from the lower n bits of the first operand and the upper n bits of the second operand.

5 30. The method of claim 28, further comprising:
compressing the partial products generated from the upper n bits of the first operand and the lower n bits of the second operand into two intermediate vectors; and
shifting the intermediate vectors left by n bits.

10

31. The method of claim 28, wherein said performing comprises performing the multiply-accumulate operation on a tightly coupled dual n-bit multiply-accumulate unit.

15 32. The method of claim 28, wherein n equals sixteen.

33. An article comprising a machine-readable medium which stores machine-executable instructions, the instructions causing a machine to:

20 perform a multiply-accumulate operation on first and second 2n-bit operands as four n-bit operations.

34. The article of claim 33, wherein the instructions causing the machine to perform includes instructions causing
25 the machine to:

generate partial product vectors from the lower n bits of the first operand and the lower n bits of the second operand;

5 generate partial product vectors from the upper n bits of the first operand and the lower n bits of the second operand;

generate partial product vectors from the upper n bits of the first operand and the upper n bits of the second operand; and

10 generate partial product vectors from the lower n bits of the first operand and the upper n bits of the second operand.

35. The article of claim 33, further comprising
15 instructions causing the machine to:

compress the partial products generated from the upper n bits of the first operand and the lower n bits of the second operand into two intermediate vectors; and

20 shift the intermediate vectors left by n bits.

36. The article of claim 33, wherein the instructions causing the machine to perform includes instructions causing the machine to perform the multiply-accumulate operation on a tightly coupled dual n-bit multiply-accumulate unit.

25

37. The article of claim 33, wherein n equals sixteen.