US 20140074901A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2014/0074901 A1**

(43) **Pub. Date: Mar. 13, 2014**

Olofsson et al.

(54) **BANDWIDTH EFFICIENT INSTRUCTION-DRIVEN MULTIPLICATION ENGINE**

(71) Applicant: **ANALOG DEVICES TECHNOLOGY**, Hamilton (BM)

(72) Inventors: **Andreas D. Olofsson**, Lexington, MA (US); **Baruch Yanovitch**, Ra'anana (IL)

(73) Assignee: **ANALOG DEVICES TECHNOLOGY**, Hamilton (BM)

(21) Appl. No.: **14/055,177**

(22) Filed: **Oct. 16, 2013**

Related U.S. Application Data

(63) Continuation of application No. 12/008,334, filed on Jan. 10, 2008, now Pat. No. 8,589,469.

(60) Provisional application No. 60/879,760, filed on Jan. 10, 2007.

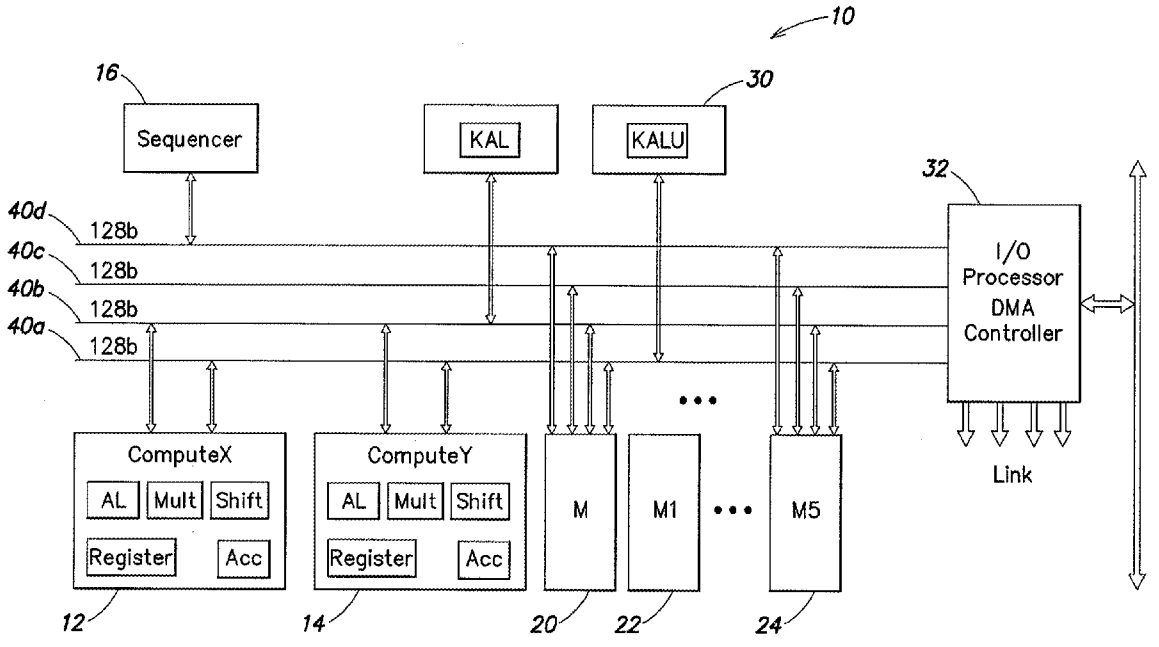
Publication Classification

(51) **Int. Cl.**
G06F 17/10 (2006.01)

(52) **U.S. Cl.**
CPC *G06F 17/10* (2013.01)
USPC **708/209**

(57) **ABSTRACT**

Multiplication engines and multiplication methods are provided for a digital processor. A multiplication engine includes multipliers, each receiving a first operand and a second operand; a local operand register having locations to hold the first operands for respective multipliers; a first operand bus coupled to the local operand register to supply the first operands from a compute register file to the local operand register; a second operand bus coupled to the plurality of multipliers to supply one or more of to the second operands from the compute register file to respective multipliers; and a control unit responsive to a digital processor instruction to supply the first operands from the local operand register to respective multipliers, to supply the second operands from the compute register file to respective multipliers on the second operand bus and to multiply the first operands by the respective second operands in the respective multipliers, wherein one or more of the first operands in the local operand register are reused by the multipliers in two or more multiplication operations.



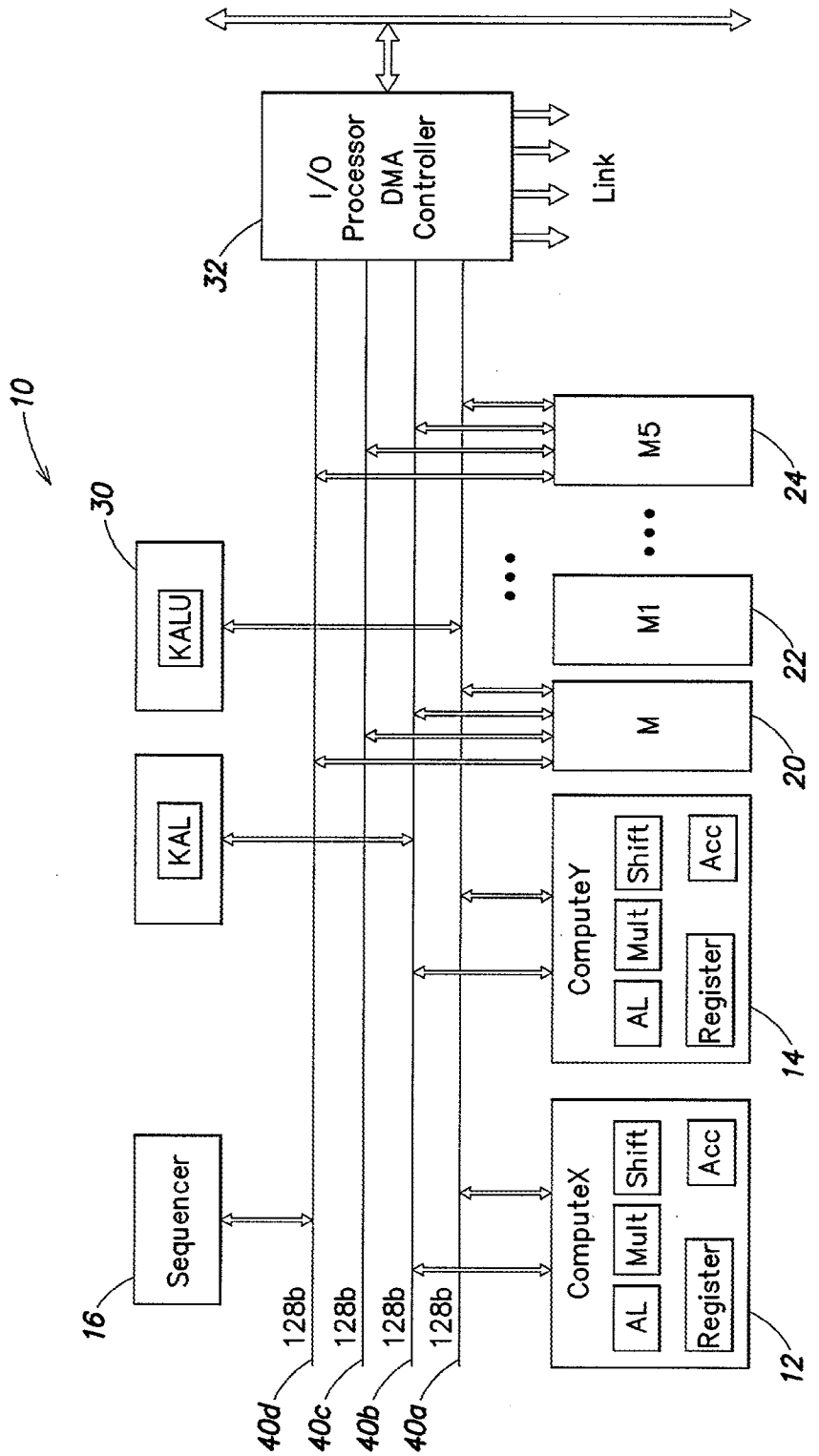

10
I/O Processor DMA Controller
Link
32
M5
24
M1
22
M
20
KALU
30
KAL
ComputeY
AL
Mult
Shift
Acc
Register
14
ComputeX
AL
Mult
Shift
Acc
Register
12
Sequencer
16
128b
128b
128b
128b
40d
40c
40b
40a

FIG. 1

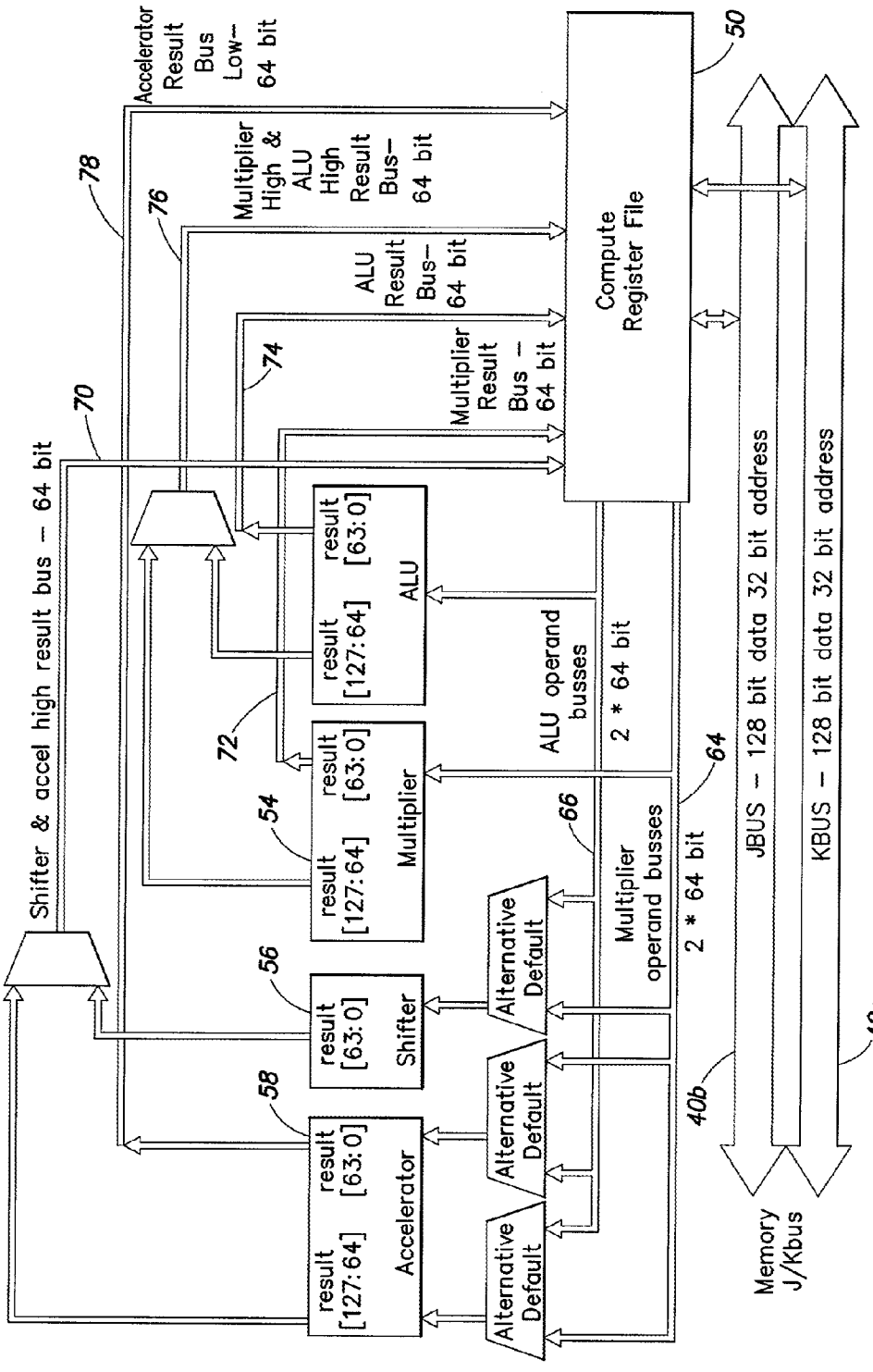

Accelerator Result Bus Low– 64 bit
Multiplier High & ALU High Result Bus– 64 bit
ALU Result Bus– 64 bit
Multiplier Result Bus – 64 bit
Compute Register File
50
78
76
74
70
Shifter & accel high result bus – 64 bit
result [63:0]
result [127:64]
ALU
72
result [63:0]
result [127:64]
Multiplier
54
ALU operand busses
2 * 64 bit
66
64
56
result [63:0]
Shifter
58
result [63:0]
result [127:64]
Accelerator
Alternative Default
Alternative Default
Alternative Default
Multiplier operand busses
2 * 64 bit
JBUS – 128 bit data 32 bit address
KBUS – 128 bit data 32 bit address
40b
40a
Memory J/Kbus

FIG. 2

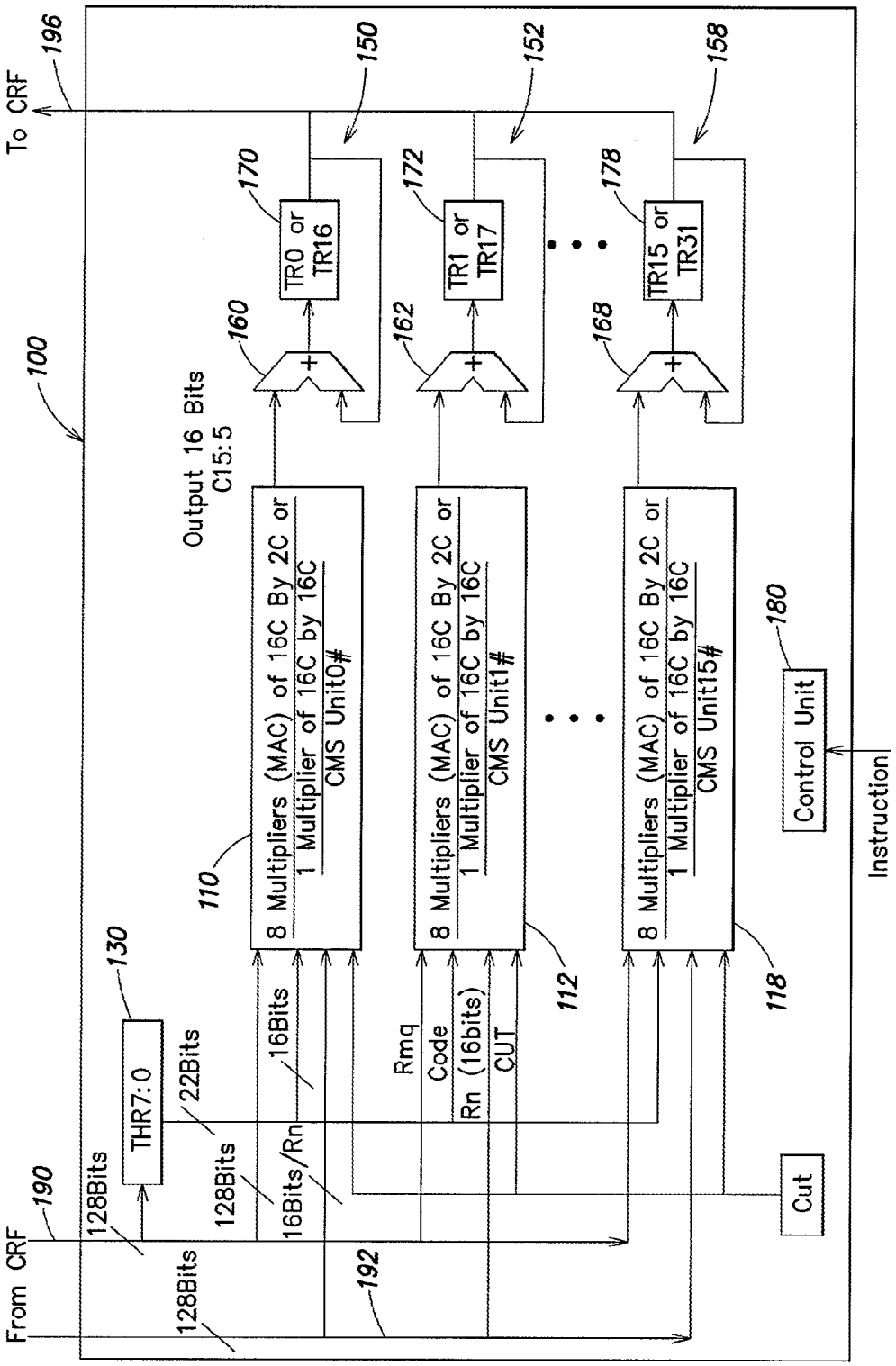

To CRF
196
150
152
158
170
172
178
TR0 or TR16
TR1 or TR17
TR15 or TR31
160
162
168
100
Output 16 Bits C15:5
8 Multipliers (MAC) of 16C By 2C or 1 Multiplier of 16C by 16C CMS Unit0#
8 Multipliers (MAC) of 16C By 2C or 1 Multiplier of 16C by 16C CMS Unit1#
8 Multipliers (MAC) of 16C By 2C or 1 Multiplier of 16C by 16C CMS Unit15#
110
112
118
180
Control Unit
Instruction
130
THR7:0
22Bits
16Bits
Rmq
Code
Rn (16bits)
CUT
Cut
128Bits
16Bits/Rn
190
From CRF
192

FIG. 3

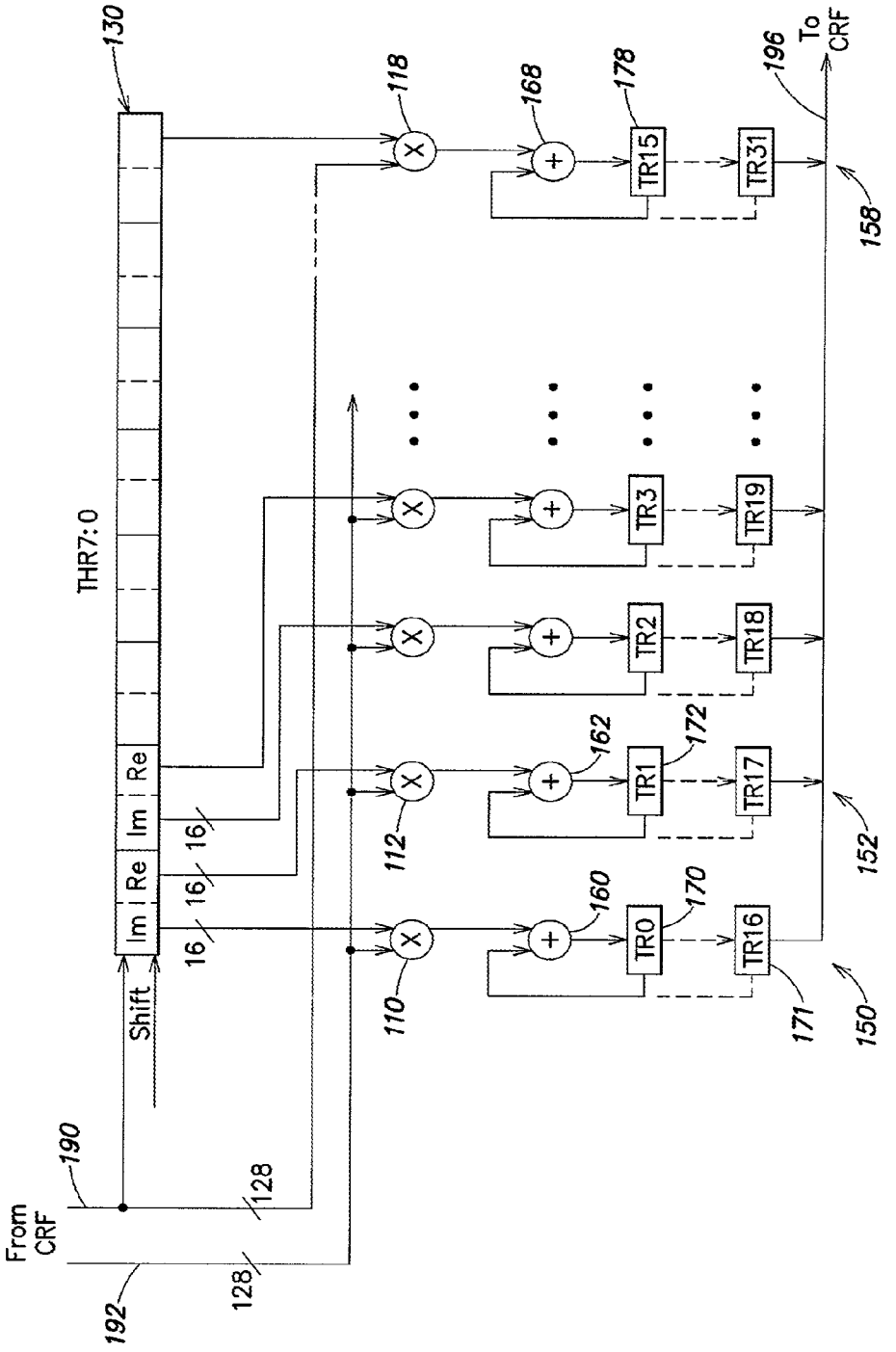

130
THR7:0
Im
Re
Im
Re
Shift
16
16
16
From CRF
190
192
128
128
110
112
118
X
+
160
162
168
TR0
TR1
TR15
170
172
178
TR16
TR17
TR31
171
150
152
158
196
To CRF
TR2
TR3
TR18
TR19

FIG. 4

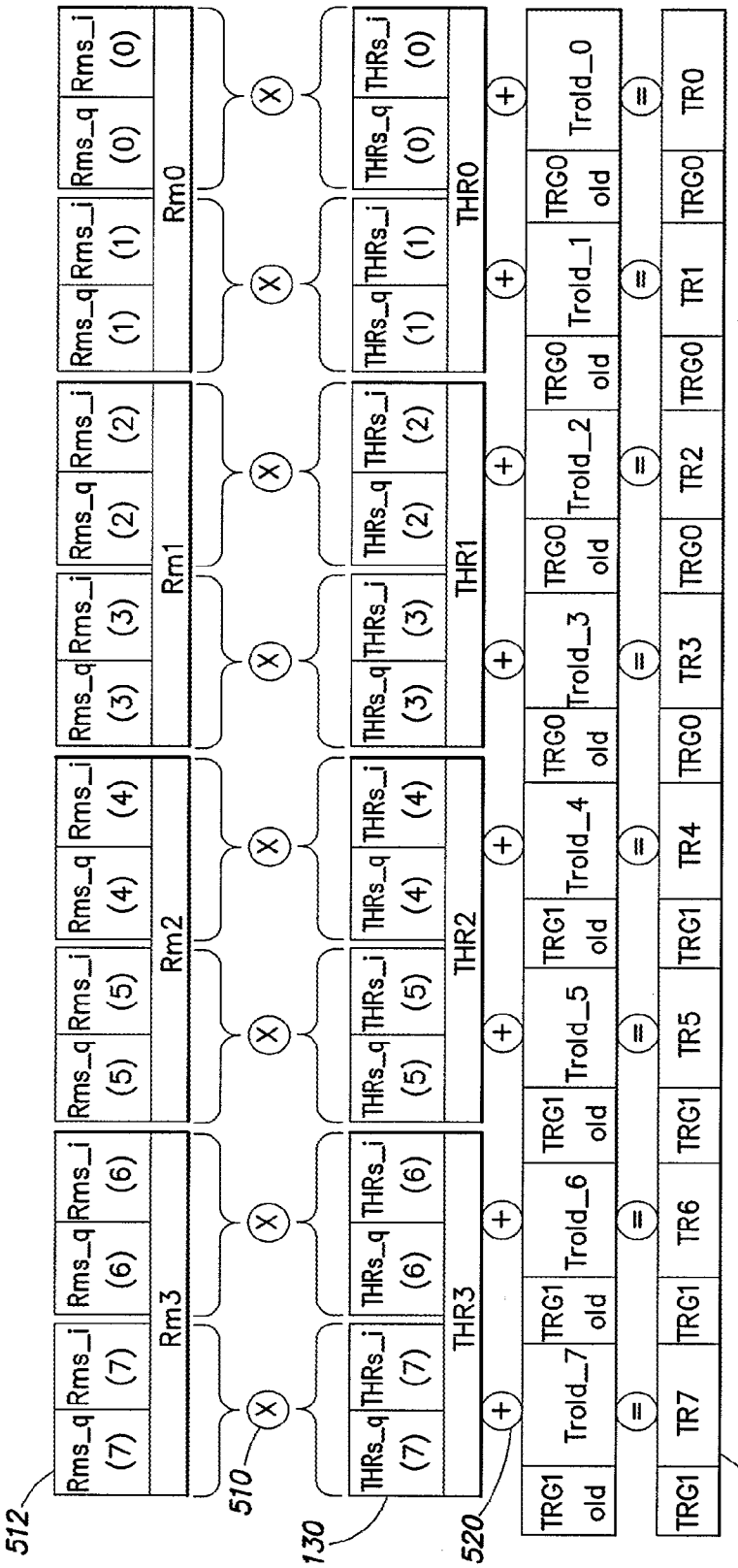

512
510
130
520
522
Rms_q (7)
Rms_i (7)
Rms_q (6)
Rms_i (6)
Rms_q (5)
Rms_i (5)
Rms_q (4)
Rms_i (4)
Rms_q (3)
Rms_i (3)
Rms_q (2)
Rms_i (2)
Rms_q (1)
Rms_i (1)
Rms_q (0)
Rms_i (0)
Rm3
Rm2
Rm1
Rm0
THRs_q (7)
THRs_i (7)
THRs_q (6)
THRs_i (6)
THRs_q (5)
THRs_i (5)
THRs_q (4)
THRs_i (4)
THRs_q (3)
THRs_i (3)
THRs_q (2)
THRs_i (2)
THRs_q (1)
THRs_i (1)
THRs_q (0)
THRs_i (0)
THR3
THR2
THR1
THR0
TRG1 old
Trold_7
Trold_6
Trold_5
Trold_4
Trold_3
Trold_2
Trold_1
Trold_0
TRG0 old
TR7
TR6
TR5
TR4
TR3
TR2
TR1
TR0
TRG1
TRG0
X
+
=

FIG. 5

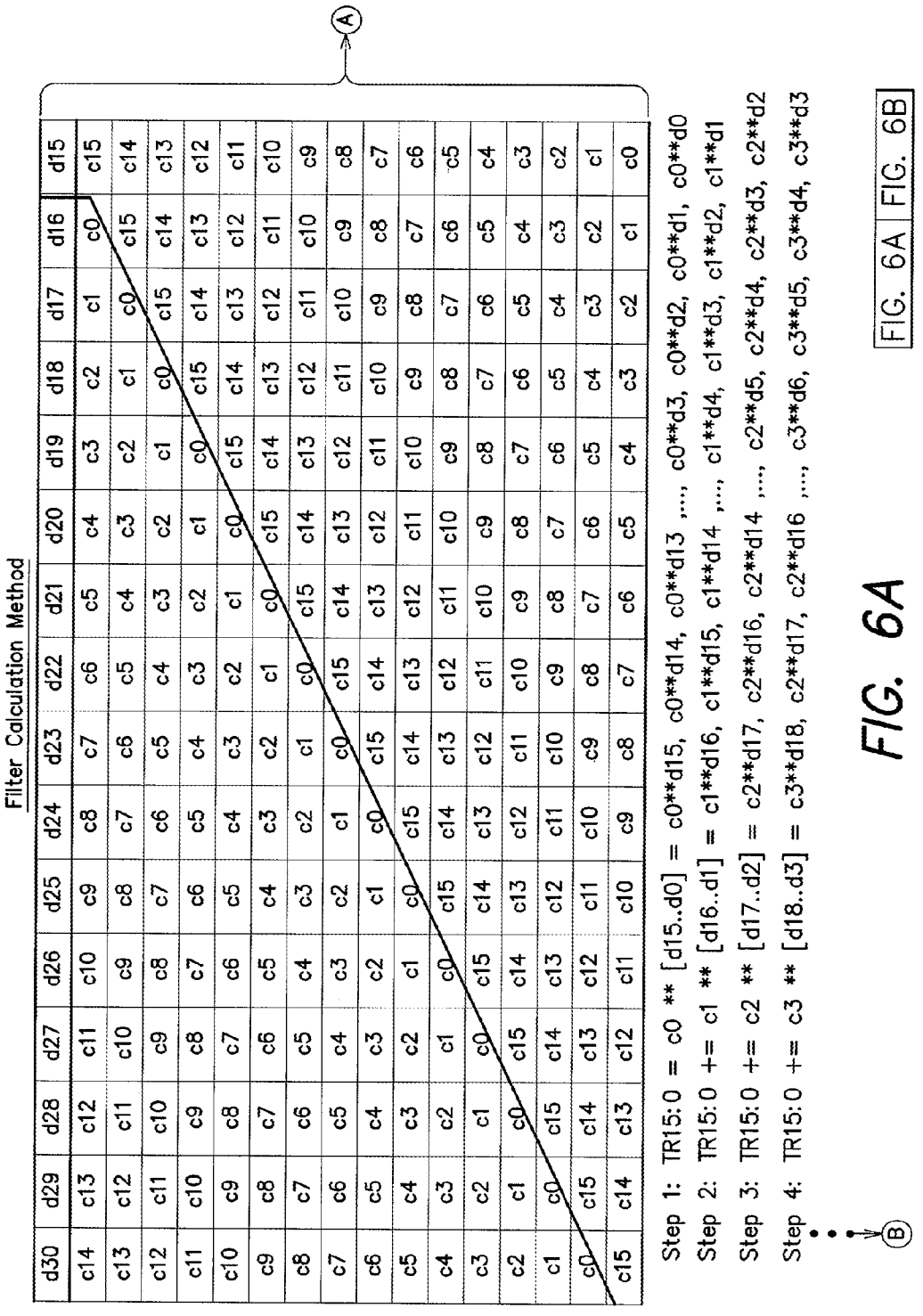


FIG. 6A

FIG. 6A | FIG. 6B

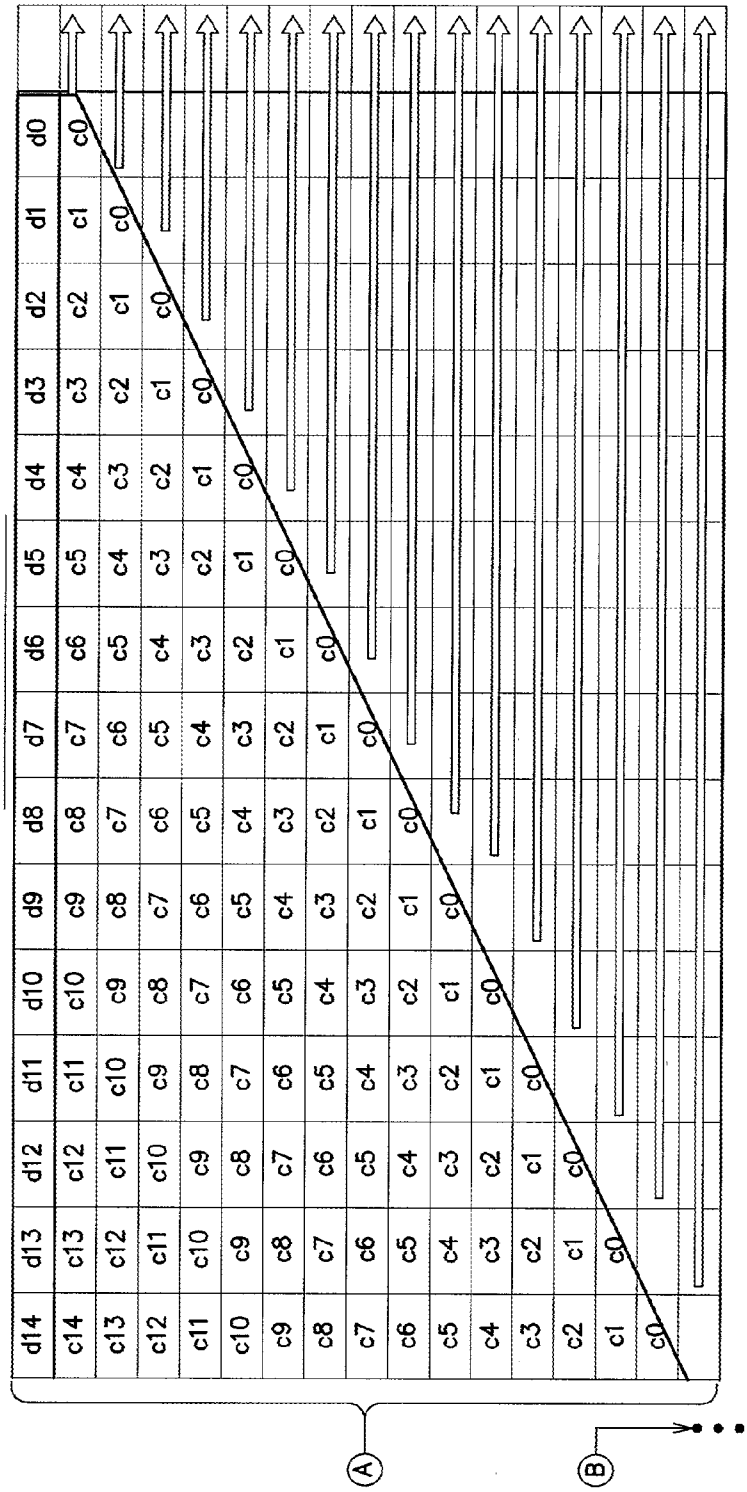

Filter Calculation Method
d14 d13 d12 d11 d10 d9 d8 d7 d6 d5 d4 d3 d2 d1 d0
A
B
Step 14: TR15:0 += c13 ** [d28..d13] = c13**d28, c13**d27, c13**d26 ,..., c13**d16, c13**d15, c13**d14, c13**d13
Step 15: TR15:0 += c14 ** [d29..d14] = c14**d29, c14**d28, c14**d27 ,..., c14**d17, c14**d16, c14**d15, c14**d14
Step 16: TR15:0 += c15 ** [d30..d15] = c15**d30, c15**d29, c15**d28 ,..., c15**d18, c15**d17, c15**d16, c15**d15
End of 16 (32 in the two Compute blocks) Filters Calculation
Step 17: TR31:16 += c0 ** [d31..d16] = c0**d31, c0**d30, c0**d29 ,..., c0**d19, c0**d18, c0**d17, c0**d16

FIG. 6B

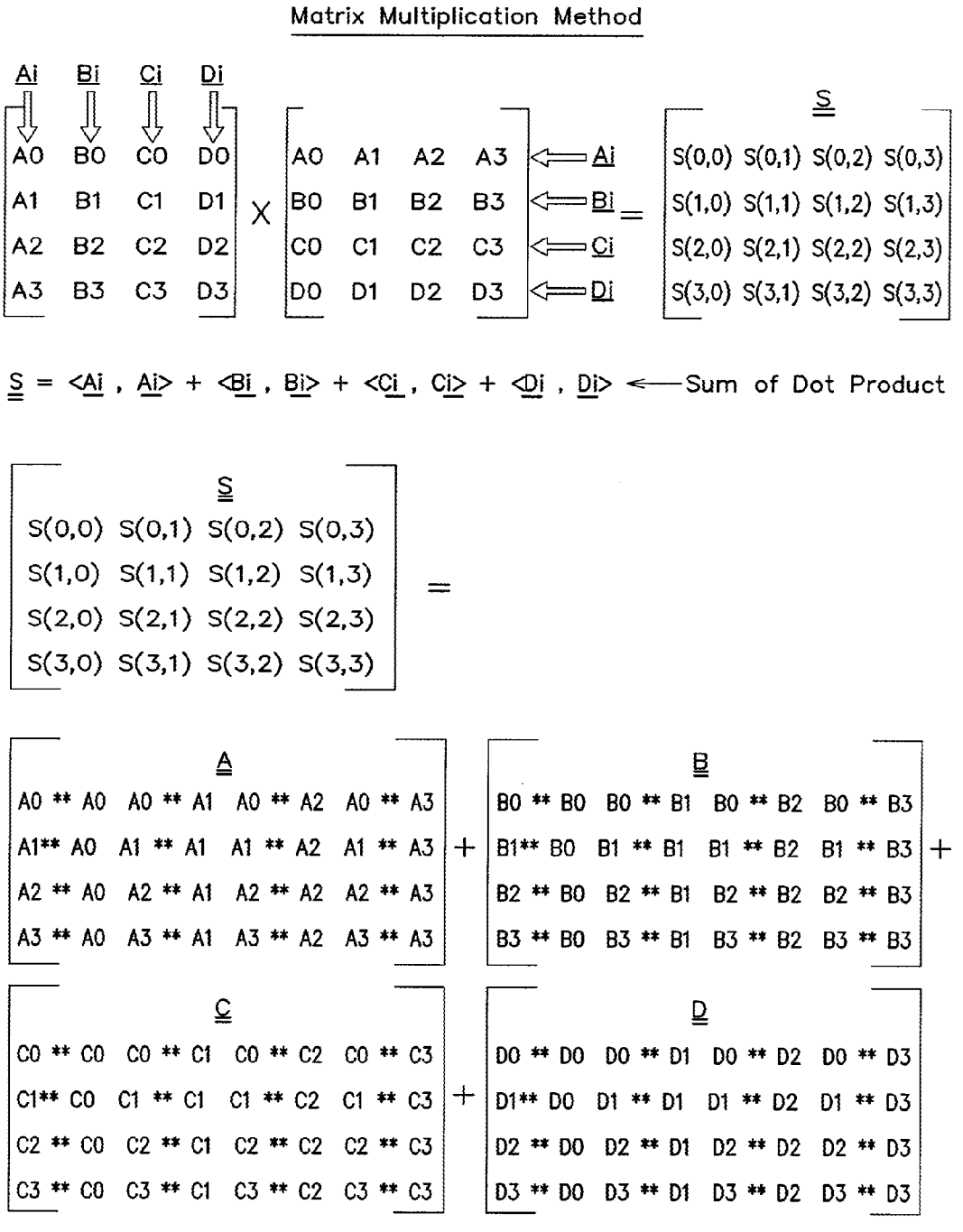

Matrix Multiplication Method
Ai Bi Ci Di
A0 B0 C0 D0
A1 B1 C1 D1
A2 B2 C2 D2
A3 B3 C3 D3
X
A0 A1 A2 A3 ⟸ Ai
B0 B1 B2 B3 ⟸ Bi
C0 C1 C2 C3 ⟸ Ci
D0 D1 D2 D3 ⟸ Di
=
S
S(0,0) S(0,1) S(0,2) S(0,3)
S(1,0) S(1,1) S(1,2) S(1,3)
S(2,0) S(2,1) S(2,2) S(2,3)
S(3,0) S(3,1) S(3,2) S(3,3)
S = <Ai , Ai> + <Bi , Bi> + <Ci , Ci> + <Di , Di> ⟵Sum of Dot Product
S
S(0,0) S(0,1) S(0,2) S(0,3)
S(1,0) S(1,1) S(1,2) S(1,3)
S(2,0) S(2,1) S(2,2) S(2,3)
S(3,0) S(3,1) S(3,2) S(3,3)
=
A
A0 ** A0 A0 ** A1 A0 ** A2 A0 ** A3
A1** A0 A1 ** A1 A1 ** A2 A1 ** A3
A2 ** A0 A2 ** A1 A2 ** A2 A2 ** A3
A3 ** A0 A3 ** A1 A3 ** A2 A3 ** A3
+
B
B0 ** B0 B0 ** B1 B0 ** B2 B0 ** B3
B1** B0 B1 ** B1 B1 ** B2 B1 ** B3
B2 ** B0 B2 ** B1 B2 ** B2 B2 ** B3
B3 ** B0 B3 ** B1 B3 ** B2 B3 ** B3
+
C
C0 ** C0 C0 ** C1 C0 ** C2 C0 ** C3
C1** C0 C1 ** C1 C1 ** C2 C1 ** C3
C2 ** C0 C2 ** C1 C2 ** C2 C2 ** C3
C3 ** C0 C3 ** C1 C3 ** C2 C3 ** C3
+
D
D0 ** D0 D0 ** D1 D0 ** D2 D0 ** D3
D1** D0 D1 ** D1 D1 ** D2 D1 ** D3
D2 ** D0 D2 ** D1 D2 ** D2 D2 ** D3
D3 ** D0 D3 ** D1 D3 ** D2 D3 ** D3

FIG. 7

BANDWIDTH EFFICIENT INSTRUCTION-DRIVEN MULTIPLICATION ENGINE

CROSS REFERENCE TO RELATED APPLICATION

[0001] This application claims priority based on Provisional Application Ser. No. 60/879,760, filed Jan. 10, 2007, which is hereby incorporated by reference in its entirety.

FIELD OF THE INVENTION

[0002] This invention relates to digital signal processors and, more particularly, to a software programmable complex multiplication engine.

BACKGROUND OF THE INVENTION

[0003] Advanced wireless networks require significant hardware acceleration in order to perform functions such as beamforming and path searching. To address these data processing requirements, CDMA systems often implement these algorithms directly with a dedicated ASIC or an on-chip coprocessor unit. Although this approach offers the highest potential performance, it carries significant design risks and is very inflexible to changes in standards and algorithms.

[0004] These and other algorithms usually involve multiplication operations. One of the limiting factors in a high performance multiplication engine is the rate at which data can be supplied to the engine from a register file or a memory. The speed and width of data buses can be increased, but at the expense of chip area and power dissipation. A further factor is that data is often reused in digital signal processing algorithms, such as FIR digital filters.

[0005] Accordingly, there is a need for improved multiplication engines and multiplication methods.

SUMMARY OF THE INVENTION

[0006] According to a first aspect of the invention, a multiplication engine is provided for a digital processor. The multiplication engine comprises a plurality of multipliers, each receiving a first operand and a second operand; a local operand register having a plurality of locations to hold the first operands for respective ones of the multipliers; a first operand bus coupled to the local operand register to supply the first operands from a compute register file to the local operand register; a second operand bus coupled to the plurality of multipliers to supply one or more of the second operands from the compute register file to respective ones of the multipliers; and a control unit responsive to a digital processor instruction to supply the first operands from the local operand register to respective ones of the multipliers, to supply the second operands from the compute register file to respective ones of the multipliers on the second operand bus and to multiply the first operands by the respective second operands in the respective multipliers, wherein one or more of the first operands in the local operand register are reused by the plurality of multipliers in two or more multiplication operations.

[0007] According to a second aspect of the invention, a method is provided for performing multiplication in a digital processor. The method comprises providing a plurality of multipliers; providing a local operand register having a plurality of locations to hold first operands for respective ones of the multipliers; supplying the first operands from a compute register file to the local operand register on a first operand bus; supplying second operands from the compute register file to respective ones of the multipliers on a second operand bus; controlling operation of the multipliers and the local operand register in response to a digital processor instruction by supplying the first operands from the local operand register to respective ones of the multipliers, supplying the second operands from the compute register file to respective ones of the multipliers on the second operand bus and multiplying the first operands by the respective second operands in respective ones of the multipliers; and reusing one or more of the first operands in the local operand register in two or more multiplication operations by the plurality of multipliers.

[0008] According to a third aspect of the invention, a multiplication engine is provided for a digital processor. The multiplication engine comprises a multiplier circuit, a register file to supply data to the multiplier circuit, the register file including a plurality of register locations in a shift configuration, and a control circuit to load data from a data source into the is register file and to shift data in the register file after a multiply instruction is executed, wherein data in the register file is used by two or more multiply instructions.

[0009] According to a fourth aspect of the invention, a method is provided for digital processing in a multiplication engine including a multiplier circuit and a register file to supply data to the multiplier circuit. The method comprises loading data from a data source into the register file, executing a first multiply instruction using data in the register file, shifting data in the register file from current register locations to next register locations after execution of the first multiply instruction, and executing a second multiply instruction using the shifted data in the register file.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] For a better understanding of the present invention, reference is made to the accompanying drawings, which are incorporated herein by reference and in which:

[0011] FIG. **1** is a block diagram showing an example of a digital signal processor architecture;

[0012] FIG. **2** is a block diagram showing an example of a compute block in the digital signal processor;

[0013] FIG. **3** is a block diagram that illustrates a multiplication engine in accordance with an embodiment of the invention;

[0014] FIG. **4** is a schematic block diagram that illustrates the multiplication engine of FIG. **3** in greater detail;

[0015] FIG. **5** is a schematic block diagram that illustrates the registers involved in the operation of the multiplication engine of FIG. **3**;

[0016] FIG. **6** is a schematic diagram that illustrates an FIR filter calculation in accordance with an embodiment of the invention; and

[0017] FIG. **7** is a schematic diagram that illustrates a matrix multiplication calculation in accordance with an embodiment of the invention.

DETAILED DESCRIPTION

[0018] A block diagram of an example of a digital signal processor (DSP) **10** suitable for incorporation of the present invention is shown in FIG. **1**. The digital signal processor may be the TigerSharc digital signal processor manufactured and sold by Analog Devices, Inc., Norwood, Mass., for example. The digital signal processor **10** may include a compute X

block **12**, a compute Y block **14**, an instruction sequencer **16**, memory blocks **20**, **22**, **24**, an integer ALU **30** and an I/O processor or DMA controller **32**. The elements of DSP **10** are interconnected by data and address buses **40***a*, **40***b*, **40***c* and **40***d*.

[0019] An example of compute blocks **12** and **14** is shown in FIG. **2**. The compute block includes a compute register file **50** and several computation units. The computation units include an ALU **52**, a multiplier **54**, a shifter **56** and an accelerator **58**. Compute register file **50** receives data on buses **40***a* and **40***b* and supplies operands to the computation units on operand buses **64** and **66**. The results of the computations are supplied on result buses **70**, **72**, **74**, **76** and **78** to compute register file **50**. The results may be written to memory from compute register file **50** or supplied to the computation units for subsequent computations.

[0020] A multiplication engine **100** in accordance with an embodiment of the invention is shown in FIGS. **3-5**. Multiplication engine **100** may correspond to multiplier **54** shown in FIG. **2**, may be used in accelerator **58**, or both. The multiplication engine **100** includes multiplier units **110**, **112**, . . . **118**. In the embodiment of FIGS. **3-5**, multiplication engine **100** includes 16 multiplier units. Multiplication engine **100** further includes a local operand register **130** coupled to each of multiplier units **110**, **112**, . . . **118**. The outputs of multiplier units **110**, **112**, . . . **118** are supplied to accumulators **150**, **152**, . . . **158**, respectively. Each accumulator may include a summing unit and an accumulation register. In some embodiments, each of accumulators **150**, **152**, . . . **158** includes two accumulation registers for enhanced performance. Thus, accumulator **150** includes a summing unit **160** and an accumulation register **170**, and may include a second accumulation register **171** (FIG. **4**). Multiplication engine **100** further includes a control unit **180** that controls the components of multiplication engine **100** in response to instructions being executed.

[0021] Multiplication engine **100** receives operands from compute register file **50** (FIG. **2**) on a first operand bus **190** and a second operand bus **192**. Results are returned to compute register file **50** on a result bus **196**. First operand bus **190** is coupled to local operand register **130** and to each of multiplier units **110**, **112**, . . . **118**. Second operand bus **192** is coupled to each of multiplier units **110**, **112**, . . . **118**.

[0022] Each of the multiplier units **110**, **112**, . . . **118** can be configured in response to an instruction being executed. In one configuration, each of the multiplier units is configured as eight multipliers of 16 bits by 2 bits. In another configuration, each of the multiplier units is configured as a single multiplier of 16 bits by 16 bits.

[0023] Local operand register **130** provides local storage of operands used by multiplier units **110**, **112**, . . . **118**. Local operand register **130** is useful where operands are reused by the multiplier units for two or more calculations. In some configurations, the same operands are used for two or more consecutive computations by the same multiplier units. In other configurations, operands are reused by different multiplier units for consecutive computations and the operands in local operand register **130** are shifted after completion of a computation. By holding operands that are reused in local operand register **130**, transfer of data on operand buses **190** and **192** is reduced and operating efficiency is increased.

[0024] As shown in FIG. **5**, the multiplier units of multiplication engine **100** may be configured as complex multipliers **510**, each of which receives a first operand from local operand register **130** (THR register) and real and imaginary parts of a second operand from an Rms register **512** in compute register file **50**. For example, the second operand may be an input data value and the first operand may be a coefficient. The multiplication engine **100** further includes complex summing units **520**, each of which performs complex addition of a value output by complex multiplier **510** and a previous value. In particular, the output of each multiplier **510** is summed with a previous value in an accumulation register **522** to provide a current value that is placed in the accumulation register. In the embodiment of FIG. **5**, multiplication engine **100** includes eight complex multipliers **510** and eight complex summing units **520**. The complex multipliers correspond to the multiplier units **110**, **112**, . . . **118** shown in FIG. **3**, the complex summing units correspond to summing units **160**, **162**, . . . **168** shown in FIG. **3**, and the accumulation registers **522** correspond to accumulation registers **170**, **172**, . . . **178** shown in FIG. **3**.

[0025] In the embodiment of FIG. **5**, Rms registers **512** may be located in compute register file **50** (FIG. **2**). Local operand register **130** and accumulation registers **522** may be located in close proximity to complex multipliers **510**. Local operand register **130** is configured to perform shift operations as described below.

[0026] In the embodiment of FIG. **5**, local operand register **130** includes eight operands, each having 16 bits, including 8 bits real and 8 bits imaginary. Control circuit **180** (FIG. **3**) causes operands to be loaded into local operand register **130** from compute register file **50** when operands are needed for multiply instructions. The operands in local operand register **130** are used to execute a multiply instruction as shown in FIG. **5**. After the multiply instruction has been executed, control circuit **180** causes the operands in local operand register **130** to be shifted to the right. In the case of 16-bit operands, the operands are shifted 16 bits to the right by control circuit **180**. In addition, a new operand value is loaded from compute register file **50** to local operand register **130**. Then, a second multiply instruction is executed with the shifted operands in local operand register **130**. This process can be repeated until all computations have been completed. In each calculation, the operands contained in local operand register **130** are multiplied by operands supplied from the Rms registers **512** in compute register file **50**. If necessary, a new set of operands can be loaded into local operand register **130** by control circuit **180**. Thus, two or more multiply instructions can be executed without reloading local operand register **130** with a complete set of new operands after each multiply instruction.

[0027] The multiplication engine can be used to execute an FIR filter instruction as follows.

TRsh+=Rms**THRo, THR7h=Rss (j)(clr)(sho)(mhl))(shl)

Where:

[0028] Rms—is an input single short coefficient.

[0029] THRo—is an octal register data—inhabits 16 data numbers.

[0030] THR7h—is the msb short in the THR7:0 registers.

[0031] Rss—is a short operand which is loaded into the msb THR.

[0032] j—for conjugate multiplication option.

[0033] clr—clears the TR accumulators.

[0034] sho—for real 8 bit multiplications.

[0035] mhl—high/low Rms

[0036] shl—high/low Rss

[0037] The instruction makes 16 complex multiplications and afterwards shifts the contents of local operand register **130** (THR 7:0) by 16 bits to the right and updates the THR7h location by the new short word data from the compute register file.

[0038] Example (from FIG. **6**):

[0039] Step1: TR15:0=c0**[d15 . . . d0]=c0**d15, c0**d14, c0**d13, . . . , c0**d3, c0**d2, c0**d1, c0**d0

[0040] The data d15:d0 is stored in the THR7:0 (each data number is 16b only). The coefficient c0 is being loaded by Rms and multiplied by all the data numbers. Then the data in the THR is shifted to the right and d16 is loaded to the THR7h. Thus, the data in the THR7:0 inhabits data numbers d16:d1 and is ready for step 2.

FIR Calc. 16 Taps

[0041]

	Calculation	Res. to RF in R23:20	RF to Memory
Data in R15:0			
R31:24 Coef. Permanent values			
1	2	3	4	5

```
R31:28=q[k2+=0x4]; R27:24=-q[j2+=0x4]; // 1,2.load coefficients to R31:24
xR3:0=[k0+=0x4]; yR3:0 = [j0+=0x4];; // 1.load d7:d0 to xR3:0
   // 2.load d39:d32 to yR3:0
xR7:4=[k0+=0x4]; yR7:4 = [j0+=0x4];; if 1.load d15:d8 to xR7:4
   // 2.load d47:d40 to yR7:4
xR3:0=[k0+=0x4]; yR3:0 = [j0+=0x4];       THR3:0 = R3:0;; // 1.load
d23:d16 to xR7:4
         // 2.load d55:d48 to yR7:4
         // 3.load data to THR3:0
   THR7:4 = R7:4;; // 3.load data to THR3:0
xR7:4=[k0+=0x4]; yR7:4 = [j0+=0x4];          TR15:0+= THR7:0**R24l,
THR7h = R0l (bc)(clr);;
   // Octal_data**single_tap (broadcast)
xR11:8=[k0+=0x4]; yR11:8 = [j0+=0x4];       TR15:0+= THR7:0**R24h,
THR7h = R0h (bc);;
xR15:12=[k0+=0x4];yR15:12 = [j0+=0x4];       TR15:0+= THR7:0**R25l,
THR7h = R1l (bc);;
      TR15:0+= THR7:0**R25h, THR7h = R1h (bc);;
      TR15:0+= THR7:0**R26l, THR7h = R2l (bc);;
      TR15:0+= THR7:0**R26h, THR7h = R2h (bc);;
      TR15:0+= THR7:0**R27l, THR7h = R3l (bc);;
      TR15:0+= THR7:0**R27h, THR7h = R3h (bc);;
       TR15:0+= THR7:0**R28l, THR7h = R4l (bc);;
      TR15:0+= THR7:0**R28h, THR7h = R4h (bc);;
       TR15:0+= THR7:0**R29l, THR7h = R5l (bc);;
      TR15:0+= THR7:0**R29h, THR7h = R5h (bc);;
      TR15:0+= THR7:0**R30l, THR7h = R6l (bc);;
      TR15:0+= THR7:0**R30h, THR7h = R6h (bc);;
      TR15:0+= THR7:0**R31l, THR7h = R7l (bc);;
      TR15:0+= THR7:0**R31h, THR7h = R7h (bc);;
           R23:20 = TR3:0;;
q[k1+=0x4] = xR23:20; q[j1+=0x4] = yR23:20;
R23:20 = TR7:4;;
q[k1+=0x4] = xR23:20; q[j1+=0x4] = yR23:20;
R23:20 = TR11:8;;
q[k1+=0x4] = xR23:20; q[j1+=0x4] = yR23:20;
R23:20 =TR15:12;;
q[k1+=0x4] = xR23:20; q[j1+=0x4] = yR23:20;;
```

[0042] The multiplication engine can be used to execute a matrix instruction (broadcast) as follows.

[0043] 1) TRsh+=Rmd**Rnd (j)(clr)(sho)(f)

[0044] 2) TRsh+=Rmd**THRd (j)(clr)(sho)(f)

[0045] 3) TRsh+=Rmq**Rn (j)(clr)(sho)(f)

[0046] 4) TRsh+=Rmq**THR (j)(clr)(sho)(f)

[0047] 5) TRsh+=(Rmq,Rnq)**THRs (j)(clr)(sho)(f)(ns)

Where:

[0048] Rmd—holds 4 data numbers, in total 64 bits operand.

[0049] Rnd,THRd—holds 4 coefficients numbers, in total 64 bits operand.

[0050] From FIG. **7** one may observe that in order to calculate the 4×4 matrix multiplication we need to make a 4 matrix accumulations. While each one of those matrix is the dot products of:

<Ai, Ai>+<Bi, Bi>+<Ci, Ci>+<Di, Di>

[0051] That means that in order to calculate matrix A we don't need to load 16 data numbers and 16 coefficients and

then to multiply them, but we can bring only 4 data numbers and 4 coefficients and multiply each of the data numbers by each of the coefficients to accept the dot product. That method utilizes all the 16 multipliers and saves bus bandwidth.

General Matrix Calc. 4*([4*4]**[4*4])—Continuous Matrix Calculation.

[0052]

```
| Data in R15:8  | Coef. In R7:0  |      Calculation      | Res. to RF in R27:24 |
| RF to Memory  |
|--------1--------|---------2--------|-------------3-----------|-----------4----------|---------5--------

xyR9:8 = q[k0+=0x4]; xyR11:10 = q[j0+=0x4];; // 1,2. xR11:8 =
S(3,A1),S(2,A1),S(1,A1),S(0,A1),S(3,A0),S(2,A0),S(1,A0),S(0,A0)
      //       yR11:8 =
S(7,A1),S(6,A1),S(5,A1),S(4,A1),S(7,A0),S(6,A0),S(5,A0),S(4,A0)
xyR1:0 = q[k1+=0x4]; yR3:2 = q[j1+=0x4];;   // 1,2. xr3:0 =
h(1,3),h(1,2),h(1,1),h(1,0),h(0,3),h(0,2),h(0,1),h(0,0)
      //       yr3:0 =
h(5,3),h(5,2),h(5,1),h(5,0),h(4,3),h(4,2),h(4,1),h(4,0)
xyR13:12 = q[k0+=0x4]; xyR15:14 = q[j0+=0x4];; // 1,2. xR15:12 =
S(3,A3),S(2,A3),S(1,A3),S(0,A3),S(3,A2),S(2,A2),S(1,A2),S(0,A2)
      //       yR15:12 =
S(7,A3),S(6,A3),S(5,A3),S(4,A3),S(7,A2),S(6,A2),S(5,A2),S(4,A2)
      // Start 2 Matrix calculation
xyR7:4 = q[k1+=0x4]; xyR7:4 = c[j1+=0x4];   TR15:0+= R17:16 **
R1:0 (clr)(bc);;
xyR11:8 = q[k0+=0x4]; xyR11:8 = q[j0+=0x4];   TR15:0+= R19:18 **
R3:2 (bc);;
xR3:0 = q[k1+=0x4]; yR3:0 = q[j1+=0x4];      TR15:0+= R21:20 ** R5:4
(bc);;
xR15:12 = q[k0+=0x4]; yR15:12 = q[j0+=0x4];      TR15:0+= R23:22 **
R7:6 (bc);;
   // End of first 2 Start next 2 Matrix calculation
xR7:4 = q[k1+=0x4]; yR7:4 = q[j1+=0x4];   TR31:16+= R17:16 ** R1:0
(clr)(bc); R27:24= TR3:0;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;      TR31:16+= R19:18 **
R3:2 (bc);      R27:24= TR7:4;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;      TR31:16+= R21:20 **
R5:4 (bc);      R27:24= TR3:0;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;      TR31:16+= R23:22 **
R7:6 (bc);      R27:24= TR7:4;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;
R27:24= TR3:0;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;
R27:24= TR7:4;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;
R27:24= TR3:0;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;
R27:24= TR7:4;;
[k2+=0x4] = xR27:24; [j2+=0x4] = yR27:24;;
      // End of 4 Matrix calculation
```

[0053] Having thus described several aspects of at least one embodiment of this invention, it is to be appreciated various alterations, modifications, and improvements will readily occur to those skilled in the art. Such alterations, modifications, and improvements are intended to be part of this disclosure, and are intended to be within the spirit and scope of the invention. Accordingly, the foregoing description and drawings are by way of example only.

What is claimed is:

1-17. (canceled)

18. A digital processor for performing a finite impulse response (FIR) filter operation on data numbers comprising:

a plurality of multipliers, each receiving a first operand and a second operand, wherein the first operands correspond to the data numbers, and the second operands correspond to coefficients for the FIR filter operation;

a local operand register having a plurality of locations to hold the first operands corresponding to a subset of the data numbers for the plurality of multipliers, wherein the plurality of locations correspond to respective ones of the multipliers;

a control unit responsive to a digital processor instruction to supply the first operands from the plurality of locations the local operand register to respective ones of the multipliers, to supply the second operands from a compute register file to respective ones of the multipliers and to multiply the first operands by the respective second operands in the respective multipliers,

wherein the control unit, upon detecting a first multiplication operation being executed by a first multiplier:

(1) causes the first operands corresponding to the subset of the data numbers to shift in the local operand register by a number of bits equal to a width of the first operands to respective next locations in the plurality of locations of the local operand register;

(2) if available, loads a next data number from the compute register file to the local operand register; and

(3) supplies the shifted first operands and the next data number to the plurality of multipliers, thereby causing

one or more of the first operands in the local operand register used by the first multiplier in the first multiplication operation to be reused by the at least one subsequent multiplier in at least one subsequent multiplication operation, and enabling two or more multiply instructions to be executed without reloading the local operand register with a complete set of new operands after each multiplication operation.

19. The digital processor of claim **18**, wherein the compute register file stores the data numbers and the coefficients for the FIR filter operation.

20. The digital processor of claim **19**, further comprising:

a first operand bus coupled to the local operand register to supply the first operands corresponding to the subset of data numbers from the compute register file to the local operand register.

21. The digital processor of claim **19**, further comprising:

a second operand bus coupled to the plurality of multipliers to supply the second operands corresponding to one of the coefficients for the FIR filter operation from the compute register file to each of the multipliers.

22. The digital processor of claim **18**, wherein a number of taps for the FIR filter correspond to a number of the plurality of multipliers used in the multiplication operation.

23. The digital processor of claim **18**, wherein each successive multiplication operations multiplies a shifted window of data numbers with one of the coefficients for the FIR filter.

24. The digital processor of claim **18**, wherein the first operands corresponding to the subset of the data numbers shift to the right by the number of bits equal to the number of bits for each data number, and the next data number is loaded to a most significant bit location of the local operand register.

25. The digital processor of claim **21**, wherein the second operand bus is configured to broadcast one of the coefficients of the FIR filter from the compute register file to each of the multipliers.

26. The digital processor of claim **21**, wherein the control unit, upon detecting the first multiplication operation being executed, further causes the second operand bus to broadcast a next one of the coefficients of the FIR filter from the compute register file to each of the multipliers.

27. The digital processor for claim **18**, wherein each of the first operands comprises a data number having 16-bits.

28. The digital processor for claim **27**, wherein the data number comprises 16 bits and the data number is a complex number having 8-bits real and 8-bits imaginary.

29. The digital processor for claim **27**, wherein the first operands corresponding to the subset of the data numbers shift by 16-bits to the right in the local operand register respective next locations in the plurality of locations of the local operand register.

30. A method for performing in a digital processor a finite impulse response (FIR) filter operation on data numbers, comprising:

loading first operands for a plurality of multipliers from a compute register file to a plurality of locations in a local operand register using a first operand bus, wherein a plurality of locations are configured to hold the first operands for respective ones of the multipliers, and the first operands stored in the local operand register correspond to a subset of the data numbers,

supplying second operands for the plurality of multipliers from the compute register file to respective ones of the plurality of multipliers using a second operand bus, wherein the second operands correspond to coefficients for the FIR filter operation;

controlling operation of the multipliers and the local operand register in response to a digital processor instruction by supplying the first operands corresponding to the subset of the data numbers from the plurality of locations of the local operand register to respective ones of the multipliers, supplying the second operands from the compute register file to respective ones of the multipliers on the second operand bus and multiplying the first operands by the respective second operands in respective ones of the multipliers;

shifting the local operand register corresponding to the subset of the data numbers a number of bits equal to a width of the first operands to respective next locations in the plurality of locations of the local operand register upon detecting a first multiplication operation being executed by a first multiplier;

loading if available, a next data number to the local operand register; and

supplying the shifted first operands and the next data number to the plurality of multipliers, thereby causing one or more of the first operands in the local operand register used by the first multiplier in the first multiplication operation to be reused by the at least one subsequent multiplier in at least one subsequent multiplication operation and enabling two or more multiply instructions to be executed without reloading the local operand register with a complete set of new operands after each multiplication operation.

31. The method of claim **30**, wherein the steps for shifting the local operand register corresponding to the subset of data numbers causes successive multiplication operations to multiply a shifted window of data numbers with one of the coefficients for the FIR filter.

32. The method of claim **30**, wherein the first operands corresponding to the subset of the data numbers in the local operand register are shifted to the right by the width of each data number, and the next data number is loaded to a most significant bit location of the local operand register.

33. The method of claim **30**, wherein the second operand bus is configured to broadcast one of the coefficients of the FIR filter from the compute register file to each of the multipliers.

34. The method of claim **30**, wherein the controlling operation of the multipliers comprises broadcasting a next one of the coefficients of the FIR filter from the compute register file to each of the multipliers in response to the digital processor instruction.

35. The method of claim **30**, wherein each of the first operands comprises a data number having 16-bits.

36. The method of claim **35**, wherein the data number comprises 16 bits and the data number is a complex number having 8-bits real and 8-bits imaginary.

37. The method of claim **35**, wherein the first operands corresponding to the subset of the data numbers in the local operand register are shifted by 16-bits to the right in the local operand register respective next locations in the plurality of locations of the local operand register.

* * * * *