US 20220414420A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2022/0414420 A1**

LUISE et al. (43) **Pub. Date: Dec. 29, 2022**

(54) **ULTRA-LOW-POWER AND LOW-AREA SOLUTION OF BINARY MULTIPLY-ACCUMULATE SYSTEM AND METHOD**

(71) Applicants: **STMICROELECTRONICS S.r.l.**, Agrate Brianza (IT); **STMicroelectronics International N.V.**, Geneva (CH)

(72) Inventors: **Loris LUISE**, Ornago (IT); **Surinder Pal SINGH**, NOIDA (IN); **Fabio Giuseppe DE AMBROGGI**, Biassono (IT)

(73) Assignees: **STMICROELECTRONICS S.r.l.**, Agrate Brianza (IT); **STMicroelectronics International N.V.**, Geneva (CH)

(21) Appl. No.: **17/360,986**

(22) Filed: **Jun. 28, 2021**

Publication Classification

(51) **Int. Cl.**

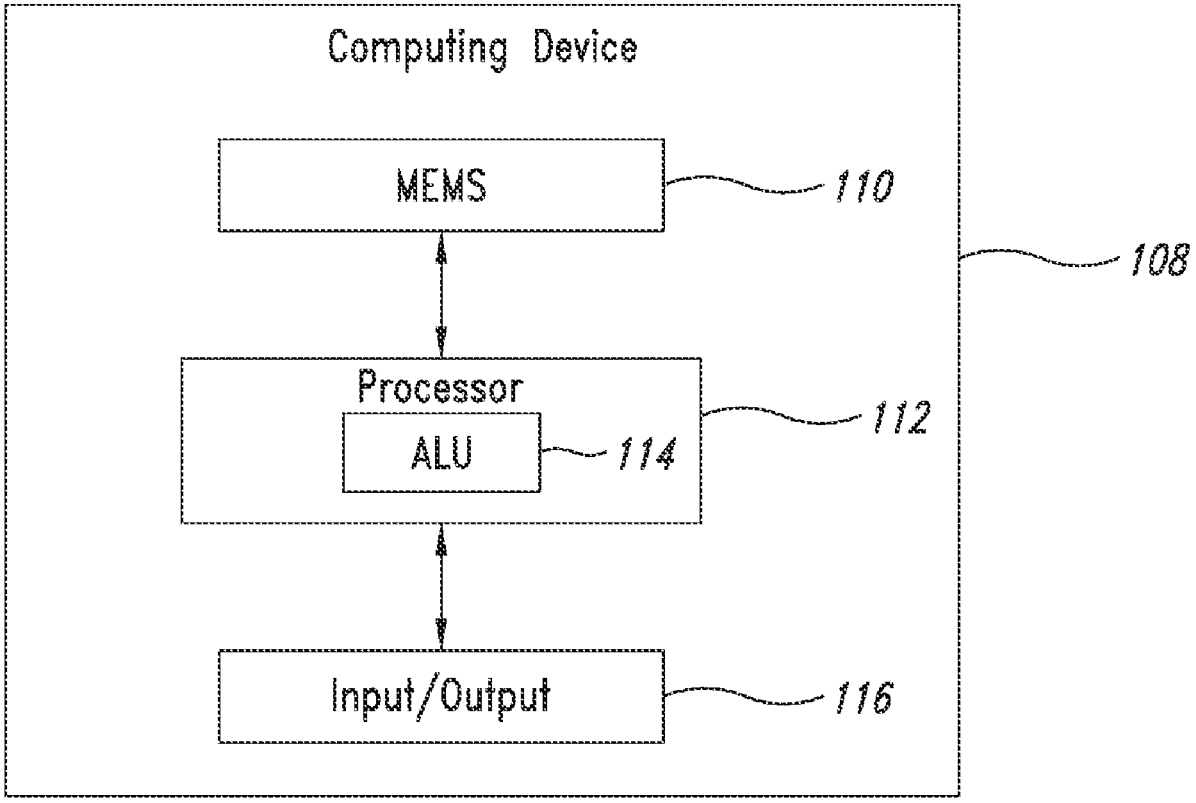
G06N 3/04	(2006.01)
G06F 9/30	(2006.01)
G06F 7/57	(2006.01)

(52) **U.S. Cl.**

CPC ***G06N 3/04*** (2013.01); ***G06F 9/30014*** (2013.01); ***G06F 7/57*** (2013.01); ***G06F 9/30101*** (2013.01)

(57) **ABSTRACT**

Data structure and microcontroller architecture performing binary multiply-accumulate operations using multiple partial copies of weights. Destination-register location, source-register location, and weight-register location are received. Using the weight-register location, a sub-set of the weight bits is copied a select number of times based on a filter index value that is received. Each copy of the sub-set of weights is executed in parallel. Using the source-register location, a sub-set of the input bits is selected based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of input bits. XOR operation is performed on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits. In a corresponding destination sub-location, output of each XOR operation is aggregated with each other and with current value of the corresponding destination sub-location.



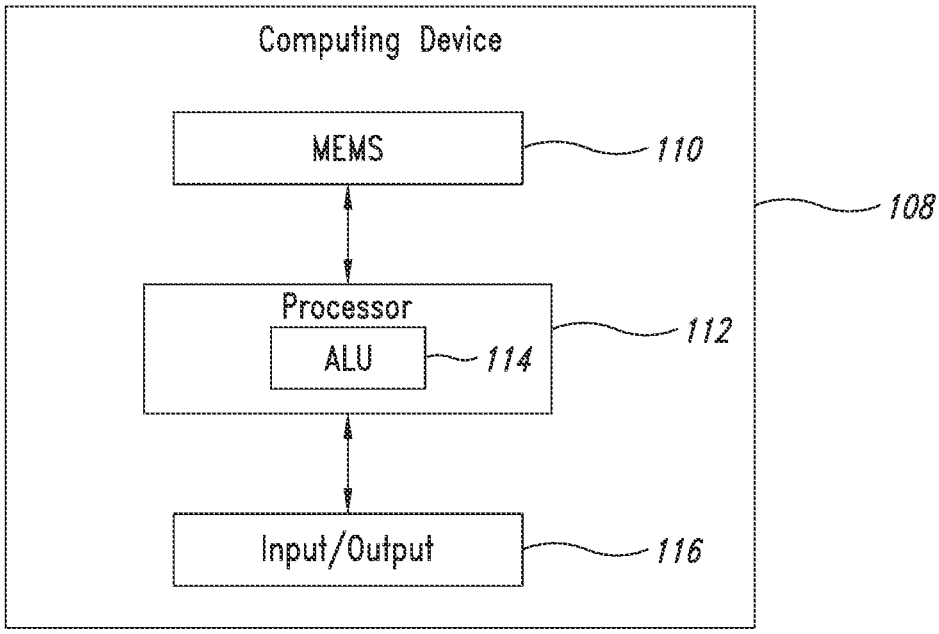

Computing Device
MEMS
110
108
Processor
ALU
114
112
Input/Output
116

FIG. 1

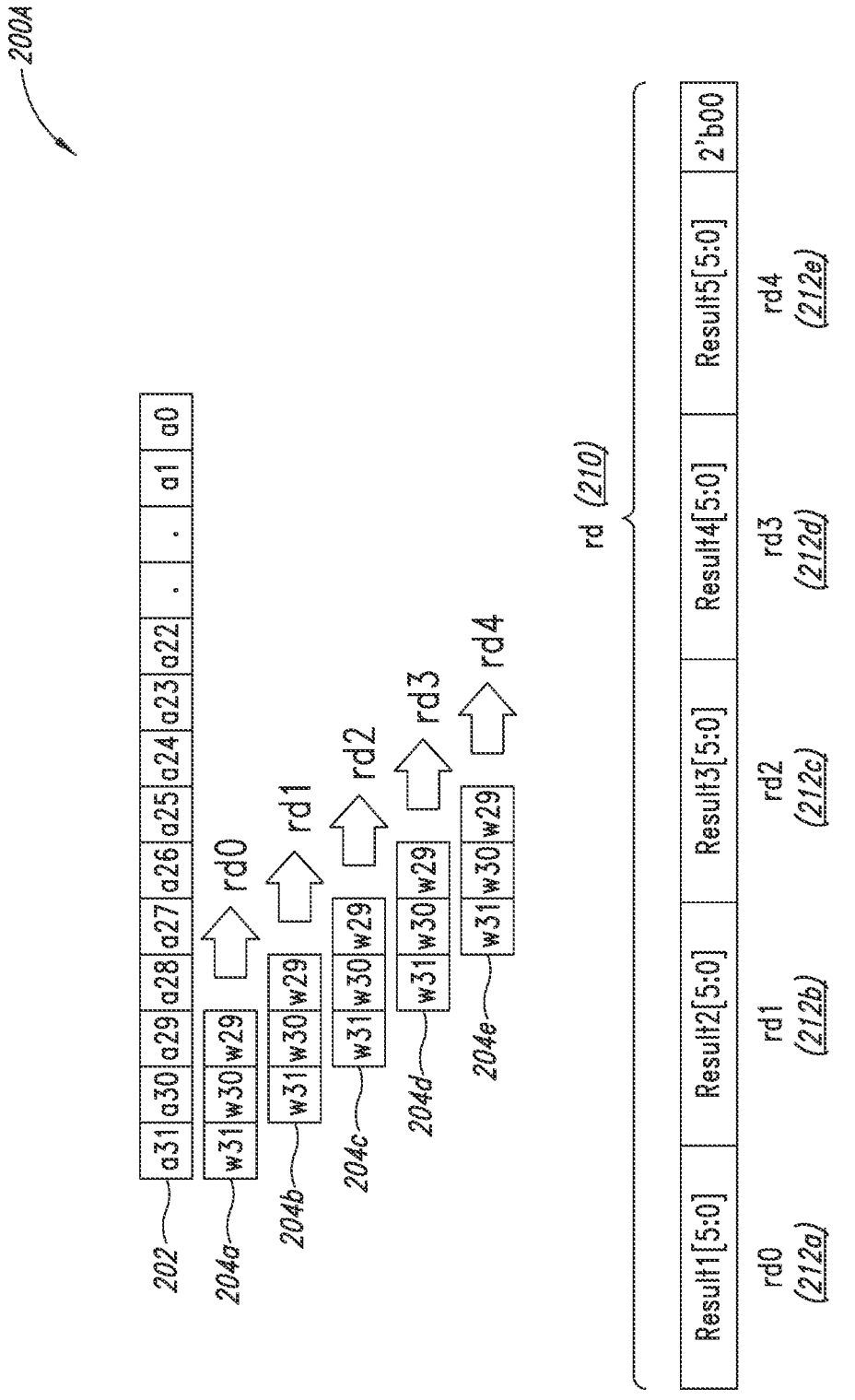

200A
a31 a30 a29 a28 a27 a26 a25 a24 a23 a22 . . a1 a0
202
w31 w30 w29
204a
rd0
w31 w30 w29
204b
rd1
w31 w30 w29
204c
rd2
w31 w30 w29
204d
rd3
w31 w30 w29
204e
rd4
rd (210)
Result1[5:0]
Result2[5:0]
Result3[5:0]
Result4[5:0]
Result5[5:0]
2'b00
rd0 (212a)
rd1 (212b)
rd2 (212c)
rd3 (212d)
rd4 (212e)

FIG. 2A

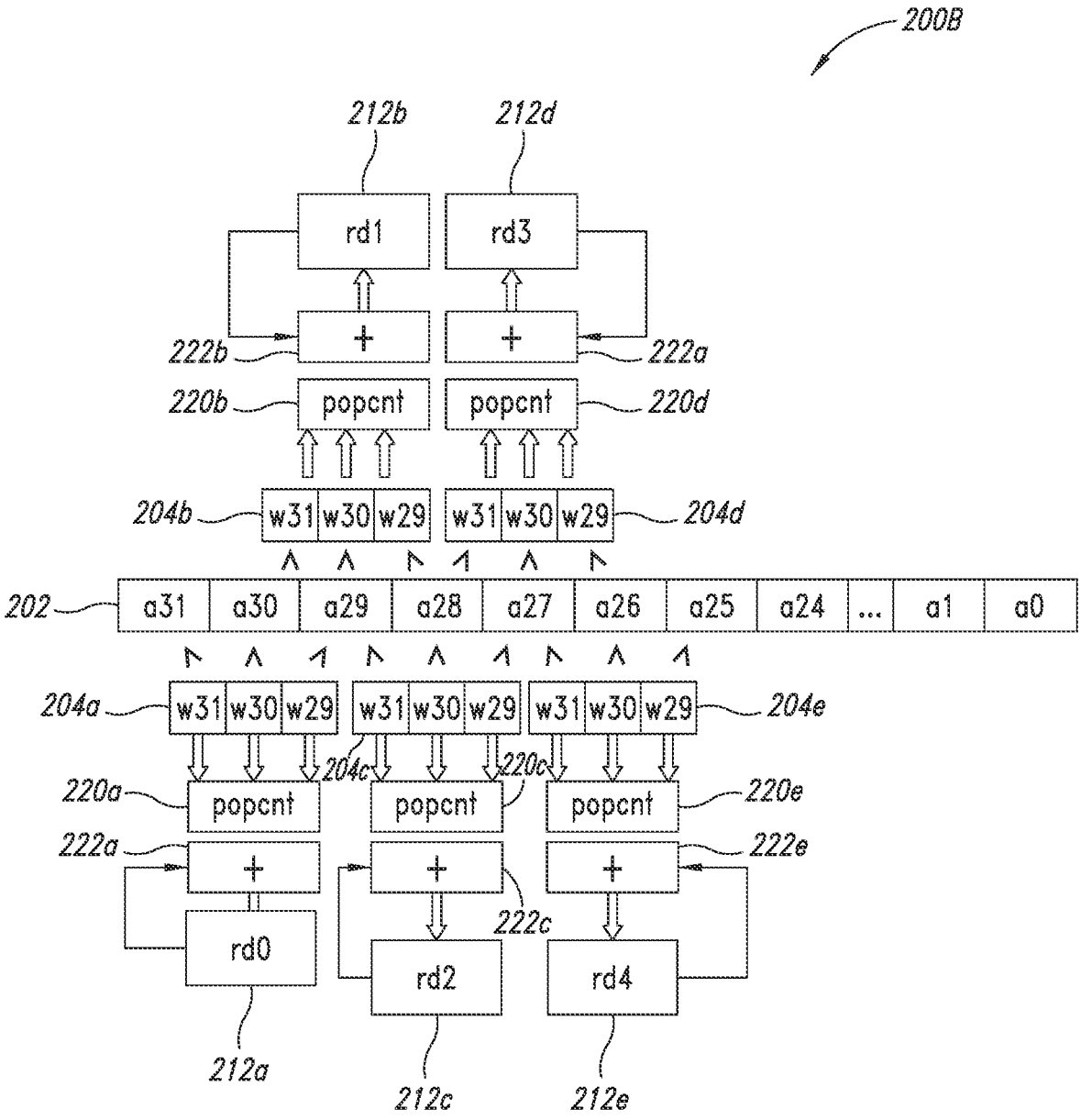

200B
212b
212d
rd1
rd3
+
+
222b
222a
220b
popcnt
popcnt
220d
204b
w31 w30 w29
w31 w30 w29
204d
202
a31 a30 a29 a28 a27 a26 a25 a24 ... a1 a0
204a
w31 w30 w29
w31 w30 w29
w31 w30 w29
204e
204c
220c
220a
popcnt
popcnt
popcnt
220e
222a
+
+
+
222e
222c
rd0
rd2
rd4
212a
212c
212e

FIG. 2B

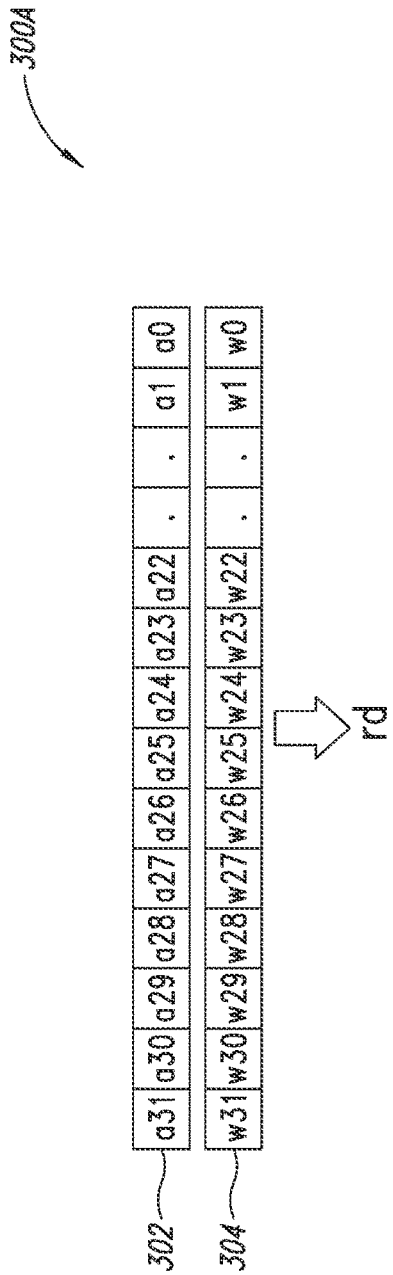

300A
a31 a30 a29 a28 a27 a26 a25 a24 a23 a22 . . a1 a0
w31 w30 w29 w28 w27 w26 w25 w24 w23 w22 . . w1 w0
rd
302
304

FIG. 3A

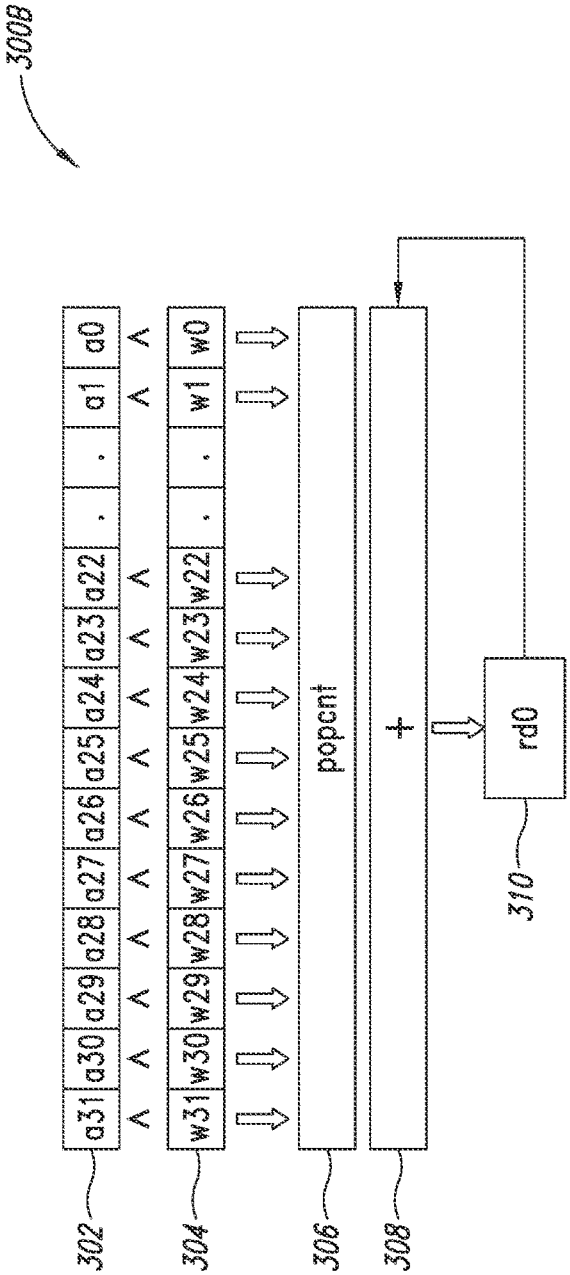

300B
a31 a30 a29 a28 a27 a26 a25 a24 a23 a22 . . a1 a0
w31 w30 w29 w28 w27 w26 w25 w24 w23 w22 . . w1 w0
popcnt
+
rd0
302
304
306
308
310

FIG. 3B

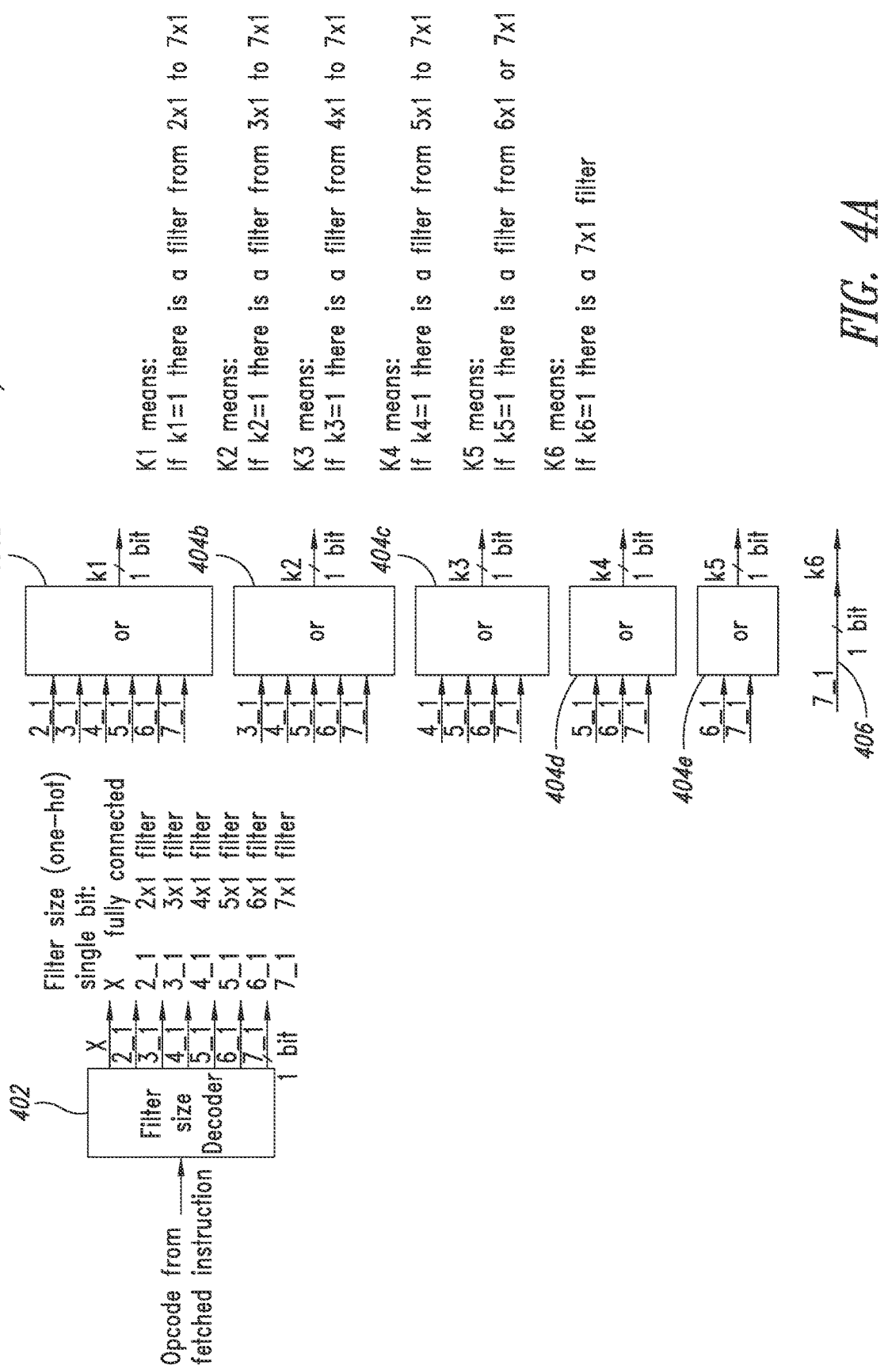

400A
K1 means:
If k1=1 there is a filter from 2x1 to 7x1
K2 means:
If k2=1 there is a filter from 3x1 to 7x1
K3 means:
If k3=1 there is a filter from 4x1 to 7x1
K4 means:
If k4=1 there is a filter from 5x1 to 7x1
K5 means:
If k5=1 there is a filter from 6x1 or 7x1
K6 means:
If k6=1 there is a 7x1 filter
Opcode from fetched instruction
402
Filter size Decoder
X
2_1
3_1
4_1
5_1
6_1
7_1
1 bit
Filter size (one-hot) single bit:
X fully connected
2_1 2x1 filter
3_1 3x1 filter
4_1 4x1 filter
5_1 5x1 filter
6_1 6x1 filter
7_1 7x1 filter
404a
404b
404c
404d
404e
406
or
k1
k2
k3
k4
k5
k6
1 bit

FIG. 4A

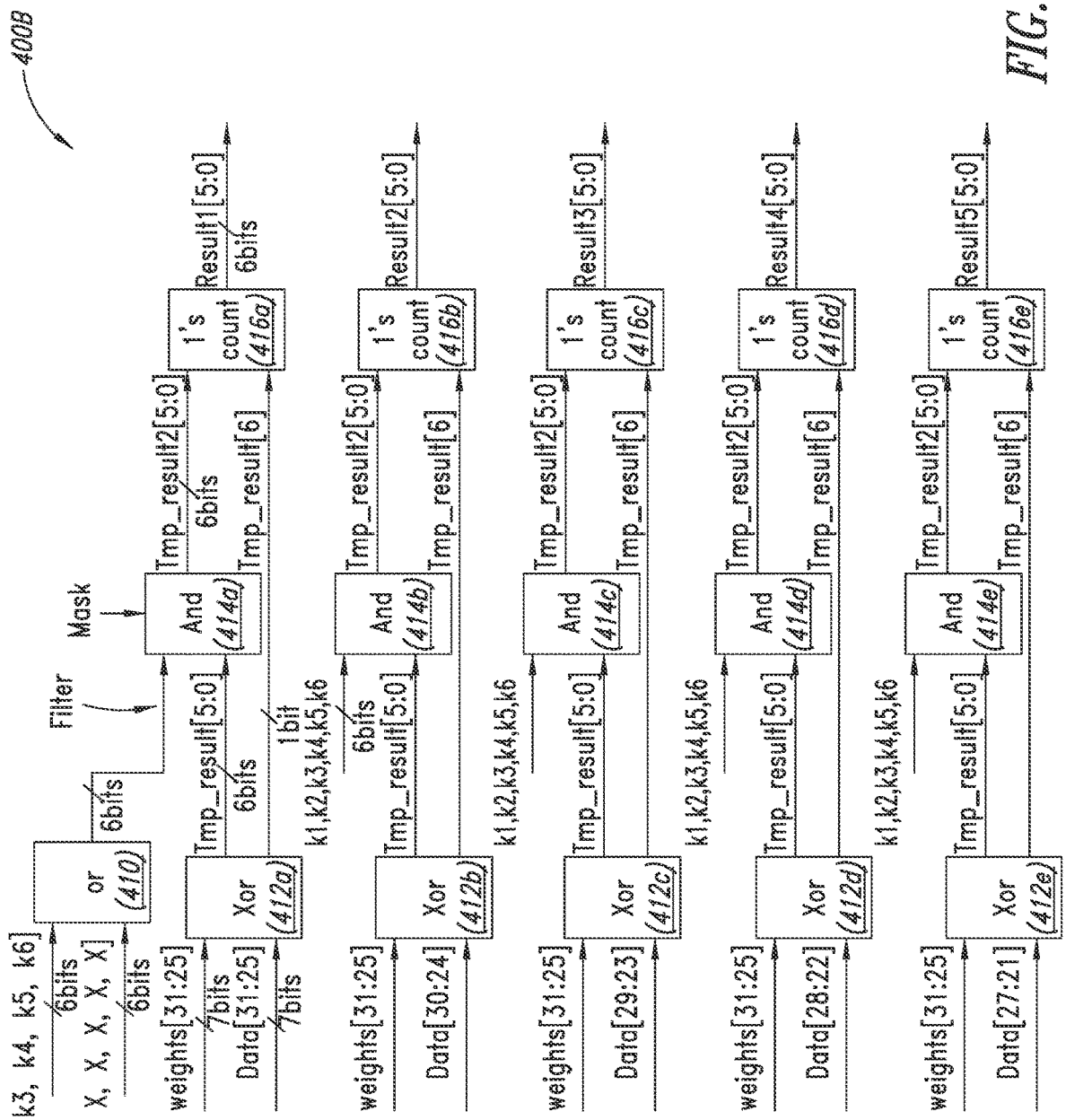

400B
[k1, k2, k3, k4, k5, k6]
6bits
[X, X, X, X, X, X]
6bits
or
(410)
6bits
Filter
Mask
And
(414a)
Tmp_result2[5:0]
6bits
1's
count
(416a)
Result1[5:0]
6bits
weights[31:25]
7bits
Data[31:25]
7bits
Xor
(412a)
Tmp_result[5:0]
6bits
Tmp_result[6]
1bit
k1,k2,k3,k4,k5,k6
And
(414b)
Tmp_result2[5:0]
1's
count
(416b)
Result2[5:0]
6bits
weights[31:25]
Data[30:24]
Xor
(412b)
Tmp_result[5:0]
Tmp_result[6]
k1,k2,k3,k4,k5,k6
And
(414c)
Tmp_result2[5:0]
1's
count
(416c)
Result3[5:0]
weights[31:25]
Data[29:23]
Xor
(412c)
Tmp_result[5:0]
Tmp_result[6]
k1,k2,k3,k4,k5,k6
And
(414d)
Tmp_result2[5:0]
1's
count
(416d)
Result4[5:0]
weights[31:25]
Data[28:22]
Xor
(412d)
Tmp_result[5:0]
Tmp_result[6]
k1,k2,k3,k4,k5,k6
And
(414e)
Tmp_result2[5:0]
1's
count
(416e)
Result5[5:0]
weights[31:25]
Data[27:21]
Xor
(412e)
Tmp_result[5:0]
Tmp_result[6]

FIG. 4B

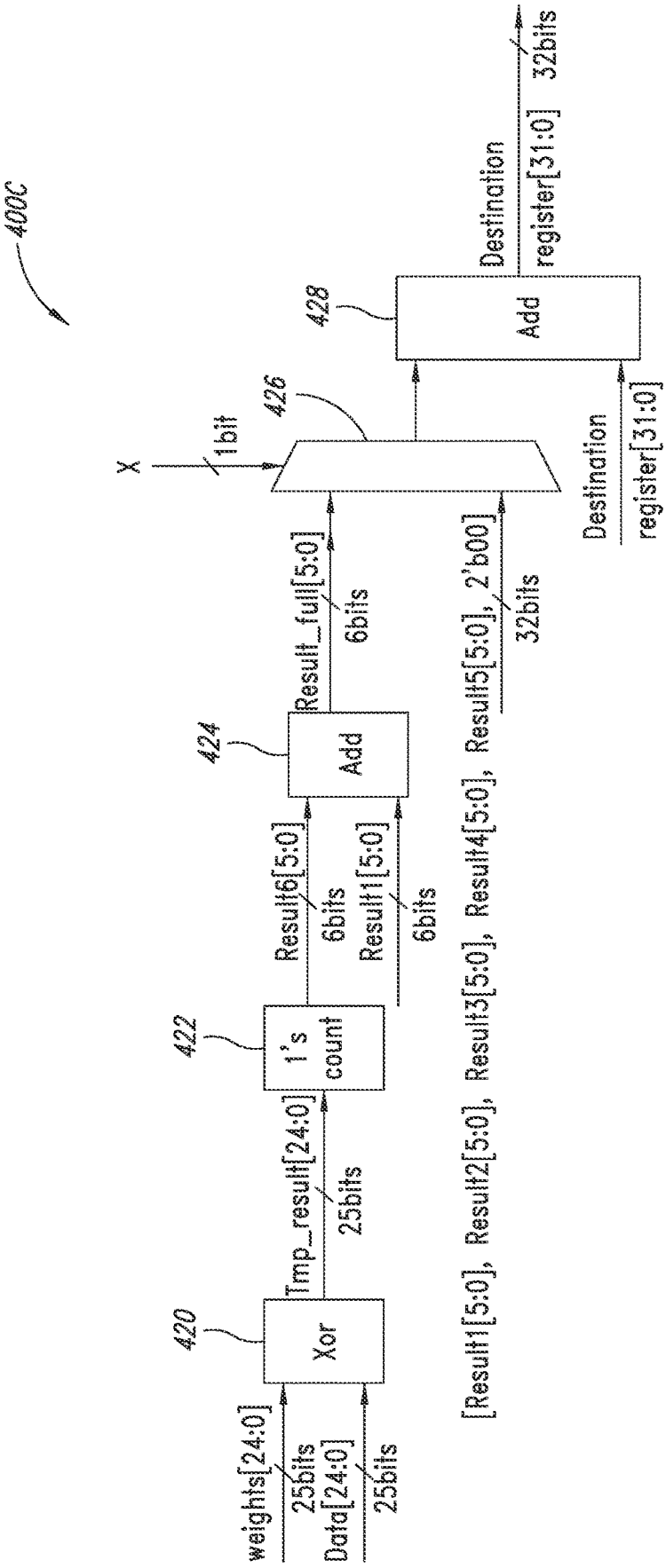

400C
weights[24:0]
25bits
Data[24:0]
25bits
420
Xor
Tmp_result[24:0]
25bits
422
1's count
Result6[5:0]
6bits
Result1[5:0]
6bits
424
Add
Result_full[5:0]
6bits
[Result1[5:0], Result2[5:0], Result3[5:0], Result4[5:0], Result5[5:0], 2'b00]
32bits
X
1bit
426
Destination
register[31:0]
428
Add
Destination
register[31:0]
32bits

FIG. 4C

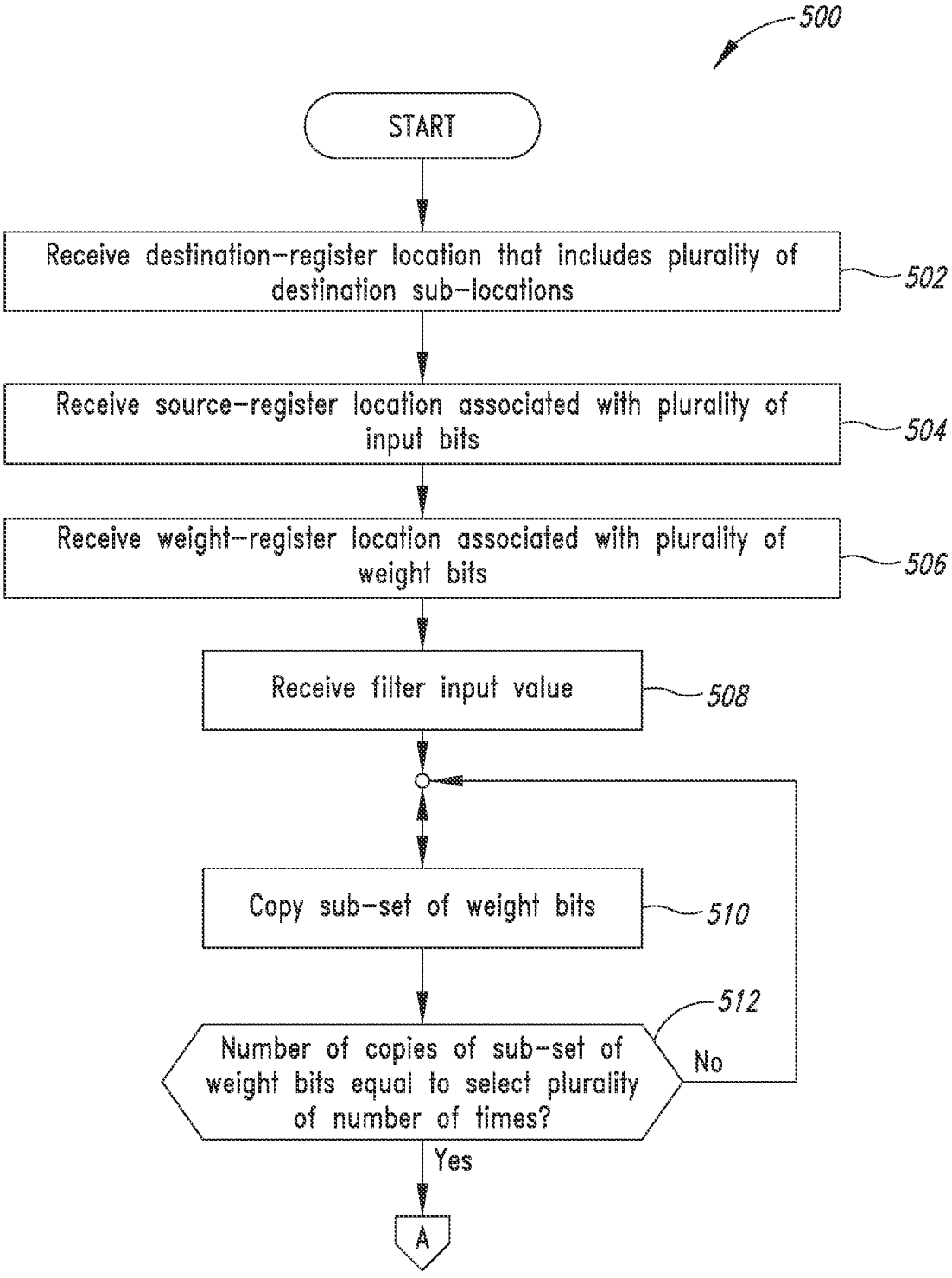

500
START
Receive destination-register location that includes plurality of destination sub-locations
502
Receive source-register location associated with plurality of input bits
504
Receive weight-register location associated with plurality of weight bits
506
Receive filter input value
508
Copy sub-set of weight bits
510
512
Number of copies of sub-set of weight bits equal to select plurality of number of times?
No
Yes
A

FIG. 5A

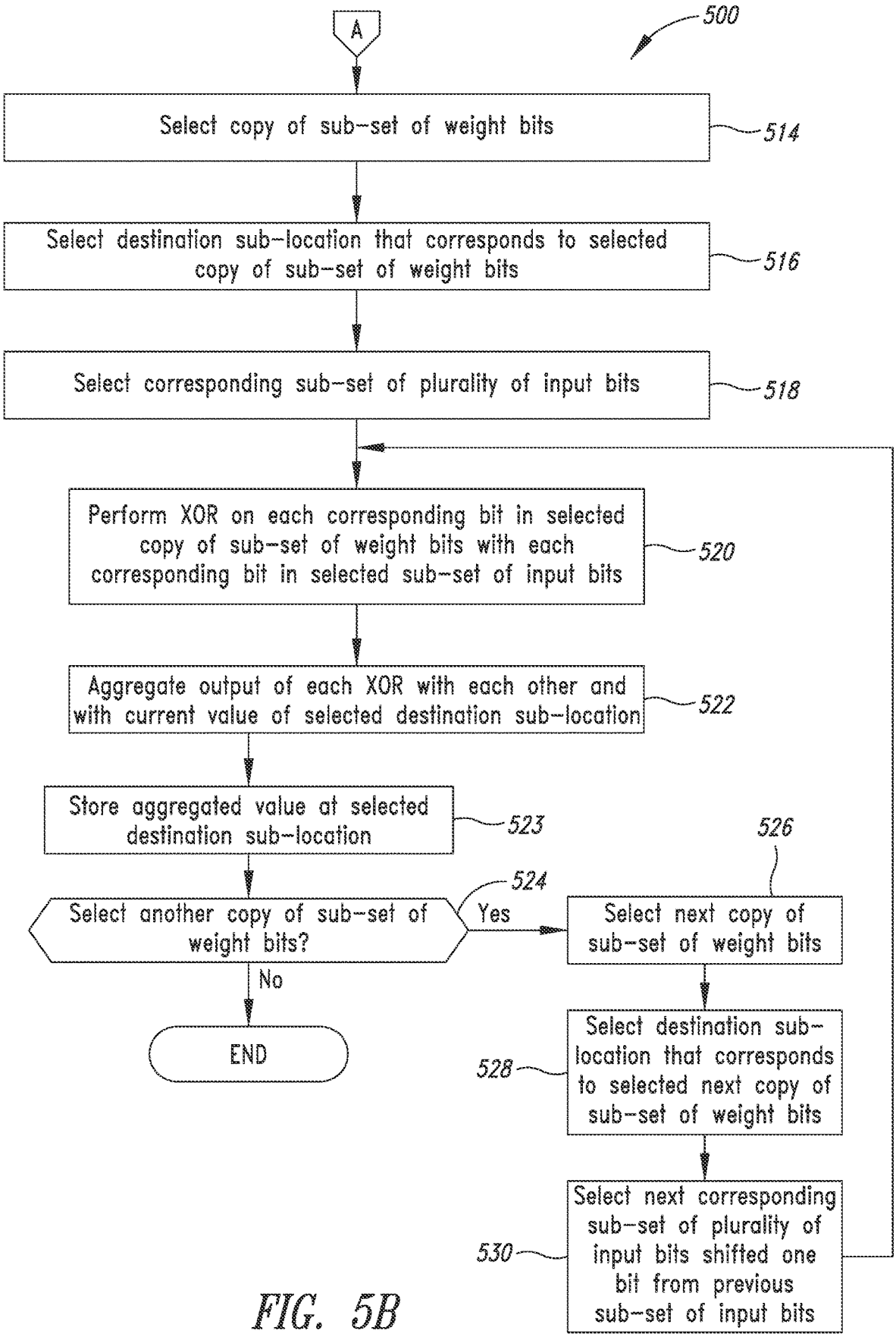

A
500
Select copy of sub-set of weight bits
514
Select destination sub-location that corresponds to selected copy of sub-set of weight bits
516
Select corresponding sub-set of plurality of input bits
518
Perform XOR on each corresponding bit in selected copy of sub-set of weight bits with each corresponding bit in selected sub-set of input bits
520
Aggregate output of each XOR with each other and with current value of selected destination sub-location
522
Store aggregated value at selected destination sub-location
523
Select another copy of sub-set of weight bits?
524
Yes
No
END
526
Select next copy of sub-set of weight bits
528
Select destination sub-location that corresponds to selected next copy of sub-set of weight bits
530
Select next corresponding sub-set of plurality of input bits shifted one bit from previous sub-set of input bits

FIG. 5B

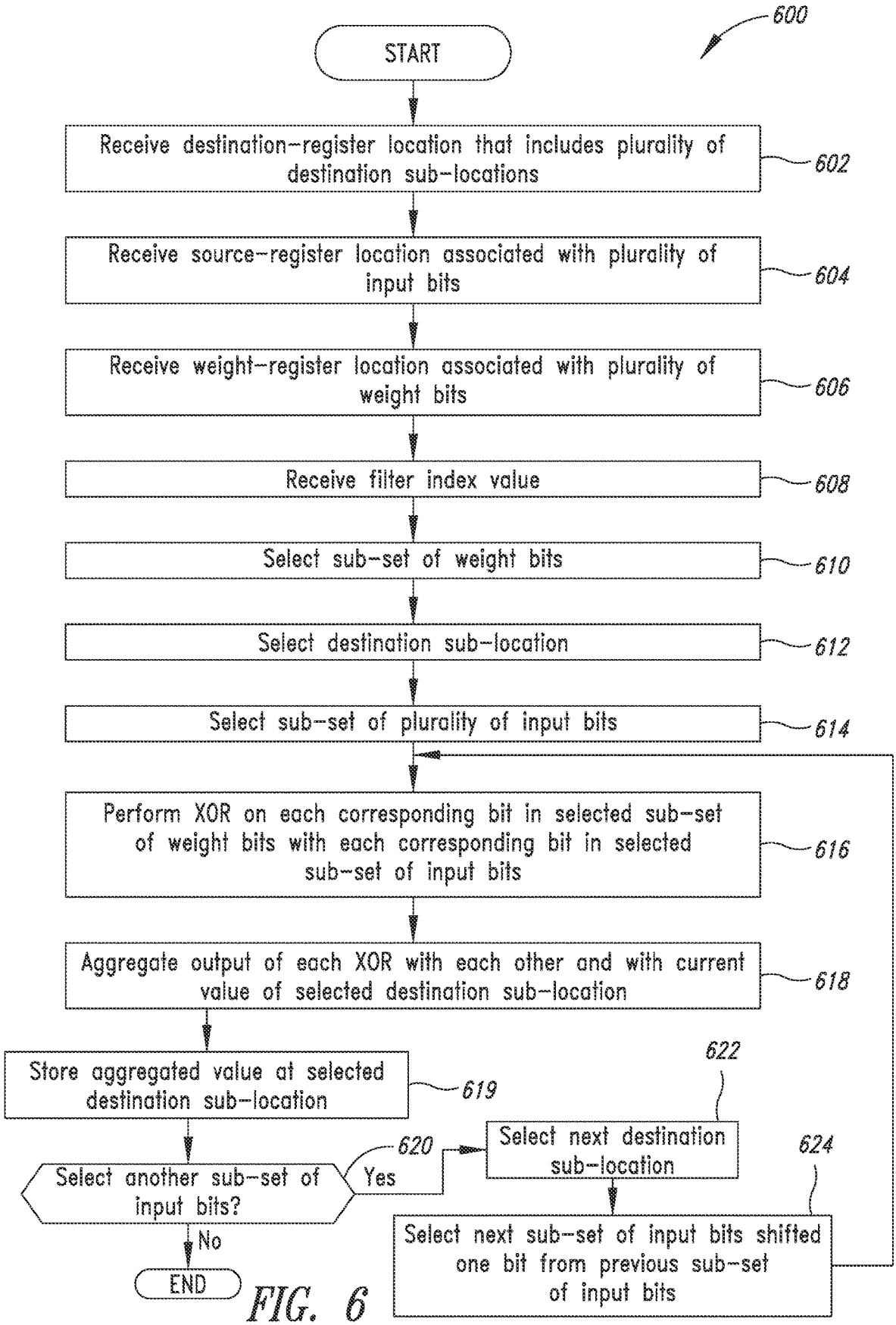

600
START
Receive destination-register location that includes plurality of destination sub-locations
602
Receive source-register location associated with plurality of input bits
604
Receive weight-register location associated with plurality of weight bits
606
Receive filter index value
608
Select sub-set of weight bits
610
Select destination sub-location
612
Select sub-set of plurality of input bits
614
Perform XOR on each corresponding bit in selected sub-set of weight bits with each corresponding bit in selected sub-set of input bits
616
Aggregate output of each XOR with each other and with current value of selected destination sub-location
618
Store aggregated value at selected destination sub-location
619
622
Select next destination sub-location
620
Yes
Select another sub-set of input bits?
624
Select next sub-set of input bits shifted one bit from previous sub-set of input bits
No
END

FIG. 6

ULTRA-LOW-POWER AND LOW-AREA SOLUTION OF BINARY MULTIPLY-ACCUMULATE SYSTEM AND METHOD

BACKGROUND

Technical Field

[0001] The present disclosure generally relates to electronic devices of the type often used in embedded applications. More particularly, but not exclusively, the present disclosure relates to utilizing multiple partial copies of weights to perform binary multiply-accumulate operations for deep neural networks.

Description of the Related Art

[0002] Many computer vision, speech recognition, and signal processing applications benefit from the use of various types of machine learning and artificial intelligence mechanisms. These mechanisms are arranged to quickly perform many hundreds or thousands of operations, often concurrently. One such mechanism is a deep neural network (DNN). A DNN is a computer-based tool that processes large quantities of data and adaptively "learns" by conflating proximally related features within the data, making broad predictions about the data, and refining the predictions based on reliable conclusions and new conflations. For example, a DNN can learn a variety of characteristics of faces such as edges, curves, angles, dots, color contrasts, bright spots, dark spots, etc. The DNN can use these initially learned characteristics to learn a variety of recognizable features of faces such as eyes, eyebrows, foreheads, hair, noses, mouths, cheeks, etc.; each of which is distinguishable from all of the other features. The DNN can then learn higher order characteristics such as a specific face, race, gender, age, emotional state, etc.

[0003] Traditionally, DNNs used floating point values—mostly 32-bit to perform various operations, including convolution. Convolution can be represented as a matrix multiplication operation, which is essentially computing the dot product of each row of matrix A with each column of matrix B. In these types of operations, computing the dot product translates to a Multiply-Accumulate (MAC) operation, which can be quite expensive to implement and generally utilizes many logic gates. Therefore, greater die area and more power consumption is utilized for floating point values and more complex convolution. It is with respect to these and other considerations that the embodiments described herein have been made.

BRIEF SUMMARY

[0004] A method may be summarized as including receiving a destination-register location configured to store accumulation results, wherein the destination-register location includes a plurality of destination sub-locations; receiving a source-register location configured to store a plurality of input bits; receiving a weight-register location configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an input length of the plurality of input bits; copying, using the weight-register location, a sub-set of the plurality of weight bits a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and for each copy of the sub-set of weights: selecting, using the source-register location, a sub-set of the plurality of input bits based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits; performing an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and aggregating, in a corresponding destination sub-location of the plurality of destination sub-locations, an output of each XOR operation with each other and with a current value of the corresponding destination sub-location.

[0005] The method may further include receiving the filter index value between 2 and 7. The method may further include receiving the filter index value of zero to indicate a fully connected layer. Copying the sub-set of the plurality of weight bits the select plurality of number of times may include copying the sub-set of the plurality of weight bits five times. The method may further include for each copy of the sub-set of weights: performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits.

[0006] The method may further include performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

[0007] The method may further include for each copy of the sub-set of weights: performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; generating a filtered output by concatenating outputs from the one's count operations for each copy of the sub-set of weights; performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; generating a fully connected output by adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits; selecting a final result between the filtered output and the fully connected output based on the filter index value; and combining the final result with a current value stored at the destination-register location.

[0008] A system may be summarized as including a memory that stores a destination register configured to store accumulation results, wherein the destination-register includes a plurality of sub-destinations; a source register configured to store a plurality of input bits; a weight register configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an

input length of the plurality of input bits; a microprocessor coupled to the memory, wherein the microprocessor, in operation copies a sub-set of the plurality of weight bits in the weight register a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and for each copy of the sub-set of weights selects a sub-set of the plurality of input bits from the source register based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits; performs an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and aggregates, in a corresponding sub-destination of the plurality of sub-destinations in the destination register, an output of each XOR operation with each other and with a current value of the corresponding sub-destination.

[0009] The microprocessor, in further operation, may receive the filter index value between 2 and 7. The microprocessor, in further operation, may receive the filter index value of zero to indicate a fully connected layer. The microprocessor, in further operation, may copy the sub-set of the plurality of weight bits five times. The microprocessor, in further operation, for each copy of the sub-set of weights may perform a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits.

[0010] The microprocessor, in further operation, may perform an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; performs a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and adds the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

[0011] The microprocessor, in further operation, for each copy of the sub-set of weights may perform a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; generates a filtered output by concatenating outputs from the one's count operations for each copy of the sub-set of weights; may perform an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; may perform a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; may generate a fully connected output by adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits; may select a final result between the filtered output and the fully connected output based on the filter index value; and may combine the final result with a current value stored at the destination register.

[0012] A non-transitory computer-readable medium having contents that configure a microcontroller to perform a method, the method may be summarized as including receiving a destination-register location configured to store accumulation results, wherein the destination-register location includes a plurality of destination sub-locations; receiving a source-register location configured to store a plurality of input bits; receiving a weight-register location configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an input length of the plurality of input bits; copying, using the weight-register location, a sub-set of the plurality of weight bits a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and for each copy of the sub-set of weights selecting, using the source-register location, a sub-set of the plurality of input bits based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits; performing an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and aggregating, in a corresponding destination sub-location of the plurality of destination sub-locations, an output of each XOR operation with each other and with a current value of the corresponding destination sub-location. Receiving the filter index value may include receiving the filter index value between 2 and 7. Receiving a filter index value may include receiving the filter index value of zero to indicate a fully connected layer. Copying the sub-set of the plurality of weight bits the select plurality of number of times may include copying the sub-set of the plurality of weight bits five times.

[0013] The non-transitory computer-readable medium, may further include for each copy of the sub-set of weights performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits. The non-transitory computer-readable medium, may further include performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0014] Non-limiting and non-exhaustive embodiments are described with reference to the following drawings, wherein like labels refer to like parts throughout the various views, unless the context indicates otherwise. The sizes and relative positions of elements in the drawings are not necessarily drawn to scale. For example, the shapes of various elements are selected, enlarged, and positioned to improve drawing legibility. The particular shapes of the elements as drawn have been selected for ease of recognition in the drawings. One or more embodiments are described hereinafter with reference to the accompanying drawings in which:

[0015] FIG. **1** is a block diagram showing an example computing device for implementing embodiments described herein;

[0016] FIGS. **2**A and **2**B are conceptual block diagrams showing example of bit and register structures in accordance with embodiments described herein;

[0017] FIGS. **3**A and **3**B are conceptual block diagrams showing another example of bit and register structures in accordance with embodiments described herein;

[0018] FIGS. **4**A-**4**C are conceptual block diagrams showing an example gate architecture in accordance with embodiments described herein;

[0019] FIGS. **5**A and **5**B show a logical flow diagram of a process for performing a new processor instruction to do binary multiply-accumulate operations in accordance with embodiments described herein; and

[0020] FIG. **6** shows a logical flow diagram of an alternative process for performing the new processor instruction to do binary multiply-accumulate operations in accordance with embodiments described herein.

DETAILED DESCRIPTION

[0021] In the following description, along with the accompanying drawings, certain details are set forth in order to provide a thorough understanding of various embodiments of devices, systems, methods, and articles. One of skill in the art, however, will understand that other embodiments may be practiced without these details. In other instances, well-known structures and methods associated with, for example, circuits, such as transistors, multipliers, adders, dividers, comparators, integrated circuits, logic gates, finite state machines, accelerometers, gyroscopes, magnetic field sensors, memories, bus systems, etc., have not been shown or described in in detail in some figures to avoid unnecessarily obscuring descriptions of the embodiments. Moreover, well-known structures or components that are associated with the environment of the present disclosure, including but not limited to the communication systems and networks, have not been shown or described in order to avoid unnecessarily obscuring descriptions of the embodiments.

[0022] Unless the context requires otherwise, throughout the specification and claims that follow, the word "comprise" and variations thereof, such as "comprising," and "comprises," are to be construed in an open, inclusive sense, that is, as "including, but not limited to."

[0023] Throughout the specification, claims, and drawings, the following terms take the meaning explicitly associated herein, unless the context clearly dictates otherwise. The term "herein" refers to the specification, claims, and drawings associated with the current application. The phrases "in one embodiment," "in another embodiment," "in various embodiments," "in some embodiments," "in other embodiments," and other variations thereof refer to one or more features, structures, functions, limitations, or characteristics of the present disclosure, and are not limited to the same or different embodiments unless the context clearly dictates otherwise. As used herein, the term "or" is an inclusive "or" operator, and is equivalent to the phrases "A or B, or both" or "A or B or C, or any combination thereof," and lists with additional elements are similarly treated. The term "based on" is not exclusive, and allows for being based on additional features, functions, aspects, or limitations not described, unless the context clearly dictates otherwise. In addition, throughout the specification, the meaning of "a," "an," and "the" include singular and plural references. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more embodiments to obtain further embodiments.

[0024] FIG. **1** is a block diagram showing an example computing device **108** for implementing embodiments described herein. Computing device **108** includes a MEMS **110**, processor **112**, and an input/output **116**. Although not illustrated, computing device **108** may have other computing components.

[0025] MEMS **110** obtain various sensor data that is provided to processor **112** for processing. MEMS **110** may include accelerometers or gyroscopes configured to sense movement or positional data associated with the computing device **108**. Although FIG. **1** shows the use of a MEMS, other sensing technologies or input sensors may also be used. Such other sensors may include, but are not limited to, a GPS system, a temperature sensor, a gas sensor, a pressure sensor, a magnetism sensor, imaging sensors, etc., or various combinations thereof.

[0026] Data obtained from MEMS **110** is provided to processor **112** for additional processing. The processor **112** includes one or more processing cores or circuits. The processor may comprise, for example, one or more processors, a state machine, a microprocessor, a programmable logic circuit, discrete circuitry, logic gates, registers, etc., and or various combinations thereof. The processor **112** may control overall operation of the computing device **108**, execution of applications programs by the computing device **108**, etc.

[0027] The processor **112** includes an arithmetic logic unit (ALU) **114**. The processor **112**, the ALU **114**, or some combination thereof, may perform embodiments described herein. Thus, in some embodiments where the processor **112** performs the embodiments described herein, the ALU **114** may not be present in the computing device **108**. Conversely, if the ALU **114** performs the embodiments described herein, the computing device **108** may still include the processor **112** to perform other actions associated with the functioning of the computing device **108**.

[0028] The computing device **108** also includes one or more memories (not shown), such as one or more volatile or non-volatile memories, or a combination thereof, which may store, for example, all or part of instructions and data related to applications and operations performed by the computing device **108**. For example, the memory may store computer instructions that when executed by the processor **108** perform the actions described herein. The memory also stores various information, including input data or weights, used to perform embodiments described herein.

[0029] The computing device **108** also includes input/output **116**. The input/output **116** may be configured to output information or results obtained or determined by processor **112** or ALU **114**, such as by performing embodiments described herein. In other embodiments, input/output **116** may be configured to receive input data from other computing devices or external sensors.

[0030] The computing device **108** may also include a bus system (not illustrated), which may be configured such that the processor, MEMS **110**, input/output **116**, memories, or other circuits or circuitry (not illustrated) **108** are communicatively coupled to one another to send or receive, or send and receive, data to or from other components. The bus system may include one or more of data, address, power, or

control busses, or some combination thereof, electrically coupled to the various components of the computing device **108**.

[0031] As described herein, ALU **114** may implement a new processor instruction and data structure to perform binary multiple-accumulate operations in neural network calculations. In various embodiments, this new processor instruction may take the form of:

[0032] stxcnt % rd, % rs, % rs0, filter_idx=#imm

where,

[0033] stxcnt is the calling operation for the instruction;

[0034] % rd is the destination register location, which keeps the accumulation results;

[0035] % rs is the source register location, which keeps 32 continuous input data bits A that may be represented as a[i,31]~a[i, 0];

[0036] % rs0 is the weights register location, which keeps the source operand of weight bits W that may be represented as w[31]~w[0] contains only one set without duplication; and

[0037] filter_idx is used to indicate which filter is implemented (from 2 to 7).

[0038] The following is an example demo code of the kernel loop for a two-dimensional 3×3 convolution filter, which utilize the new processor instruction data structure and logic architecture described herein:

```
  1dw %r0, [%r5]          ;load weights (w[31],w[30],w[29],w[28], ...,w[23],0,0...)
  1dw %r2, [%r6]+         ; load first row of input data(a[0,31]~a[0,0])
  1dw %r3, [%r6]+         ; load second row of input data (a[1,31]~a[1,0])
  1dw %r4, [%r6]+         ; load third row of input data (a[2,31]~a[2,0])
loop_2D:
  movw %r1, #0            ; reset accumulators
  stxcnt %r1, %r2, %r0, 3 ; new processor instruction to perform first convolutional part
rd0,
                            rd1, rd2, rd3, rd4 described herein
  rotlw %r0, #3           ; rotate left weights (w[28]~w[23],0,0,0...0,0,w[31]~w[29])
                            because first set of weights is used
  stxcnt %r1, %r3 %r0, 3  ; new processor instruction to perform second convolutional
part
                            rd0+rd0new, rd1+rd1new, rd2+rd2new, rd3+rd3new,
                            rd4+rd4new described herein because first input row has
                            been used
  rotlw %r0, #3           ; rotate left weights (w[25]~w[23],0,0,0...0,0,w[31]~w[26])
  stxcnt %r1, %r4, %r0, 3 ; new processor instruction to perform third convolutional
part
                            rd1+rd1new, rd2+rd2new, rd3+rd3new, rd4+rd4new
                            described herein because second input row has been used
  rotrw %r0, #6           ; rotate right weights (for initial phase)
                            (w[31]~w[23],0,0,0...) to reset for another loop
  stw [%r7]+, %r1         ; store result (r1) from running three new processor
instructions
                            3 consecutive rows in 3x3 2D convolutional filter
  sllw row1, row2,row3    ; shift left the input data for the next filters
jpia loop_2D
```

[0039] These instructions are for illustration purposes and could be different for different computer languages. However, this illustration demonstrates how the new processor instruction, data structure, and architecture described herein can be used to perform a two-dimensional 3×3 convolution filter. Similar embodiments can be used for other sizes of filters, for example, from size 2×2 to 7×7. Other filter sizes may also be considered by using additional bit lines, additional copies of weight bits, etc.

[0040] Similar embodiments can be utilized for a fully connected layer. The following is an example demo code of the kernel loop for a fully connected convolution layer, which utilize the new processor instruction data structure and logic architecture described herein:

```
movw %r1, #0              ; reset accumulator
loop_FC:
  1dw %r0, [%r5]+         ; load weights (w[31]~w[0])
  1dw %r2, [%r6]+         ; load row of input data (a[0,31]~a[0,0])
  stxcnt %r1, %r2, %r0, 0 ; 0 new processor instruction to perform fully connected
layer
  cmp end_fiter           ; compare the cycle loop
jpdne loop_FC
  stw [%r7]+, %r1           ; store result (r1) of the fully connected layer
```

[0041] These instructions are for illustration purposes and could be different for different computer languages. However, this illustration demonstrates how the new processor instruction, data structure, and architecture described herein can be used to perform a fully connected convolution layer.

[0042] FIGS. **2**A and **2**B are conceptual block diagrams showing example of bit and register structures in accordance with embodiments described herein. Convolution bit structure **200**A in FIG. **2**A illustrates a plurality of input bits **202** and multiple copies of weight bits **204***a*-**204***e*. The input bits **202** are obtained from an input register (not shown). In this example, the input bits **202** include 32 bits.

[0043] The number of copies of weight bits **204***a*-**204***e* is selected by an administrator or developer. In this example, there are five copies of weight bits **204***a*-**204***e*. Each copy of the weight bits **204***a*-**204***e* is a sub-set of weight bits obtained from a weight register (not illustrated). The number of bits in each copy of weight bits **204***a*-**204***e* is selected based on a filter input value that selects the size or type of filter to be employed. In this example, the filter input value is three, and thus each copy of weight bits **204***a*-**204***e* includes the same three bits obtained from the weight register.

[0044] Each copy of weight bits **204***a*-**204***e* is arranged to correspond to a separate sub-set of input bits **202**. For example, weight bits **204***a* correspond to input bits a**31**, a**30**, and a**29**; weight bits **204***b* correspond to input bits a**30**, a**29**, and a**28**; weight bits **204***c* correspond to input bits a**29**, a**28**, and a**27**; weight bits **204***d* correspond to input bits a**28**, a**27**, and a**26**; and weight bits **204***e* correspond to input bits a**27**, a**26**, and a**25**.

[0045] Each copy of weight bits **204***a*-**204***e* corresponds to a separate destination sub-location **212***a*-**212***e* (also referred to as destination sub-register) within destination register **210**. For example, copy of weight bits **204***a* corresponds to destination sub-location **212***a*, copy of weight bits **204***b* corresponds to destination sub-location **212***b*, copy of weight bits **204***c* corresponds to destination sub-location **212***c*, copy of weight bits **204***d* corresponds to destination sub-location **212***d*, and copy of weight bits **204***e* corresponds to destination sub-location **212***e*.

[0046] As described in more detail below, when the weight bits **204***a*-**204***e* are XOR'd with corresponding input bits **202** and aggregated together, the aggregate is combined with a current result or value stored in the corresponding destination sub-location **212***a*-**212***e*. The resulting combination is then re-stored in the corresponding destination sub-location **212***a*-**212***e*.

[0047] FIG. **2**B is a further conceptual block diagram of the bit and register structure discussed above in FIG. **2**A. Block structure **200**B includes input bits **202** and multiple copies of weight bits **204***a*-**204***e*. Structure **200** also includes popcount **220***a*-**220***e*, summation **222***a*-**222***e*, and destination sub-locations **212***a*-**212***e*.

[0048] With respect to copy of weight bits **204***a*, weight bit w**31** is XOR'd with input bit a**31**, weight bit w**30** is XOR'd with input bit a**30**, and weight bit w**29** is XOR'd with input bit a**29**. The results of these XOR operations is provided to popcount **220***a*, where the number of 1's bits from the XOR operations is calculated. The results from popcount **220***a* are provided to summation **222***a*, which is combined with a current value stored in destination sub-location **212***a*. The output from summation **222***a* is written to destination sub-location **212***a*.

[0049] Embodiments for copies of weight bits **204***b*-**204***e* are similarly employed but for shifted input bits. Details of each are provided for completeness.

[0050] With respect to copy of weight bits **204***b*, weight bit w**31** is XOR'd with input bit a**30**, weight bit w**30** is XOR'd with input bit a**29**, and weight bit w**29** is XOR'd with input bit a**28**. The results of these XOR operations is provided to popcount **220***b*, where the number of 1's bits from the XOR operations is calculated. The result from popcount **220***b* is provided to summation **222***b*, which is combined with a current value stored in destination sub-location **212***b*. The output from summation **222***b* is written to destination sub-location **212***b*.

[0051] With respect to copy of weight bits **204***c*, weight bit w**31** is XOR'd with input bit a**29**, weight bit w**30** is XOR'd with input bit a**28**, and weight bit w**29** is XOR'd with input bit a**27**. The results of these XOR operations is provided to popcount **220***c*, where the number of 1's bits from the XOR operations is calculated. The result from popcount **220***c* is provided to summation **222***c*, which is combined with a current value stored in destination sub-location **212***c*. The output from summation **222***c* is written to destination sub-location **212***c*.

[0052] With respect to copy of weight bits **204***d*, weight bit w**31** is XOR'd with input bit a**28**, weight bit w**30** is XOR'd with input bit a**27**, and weight bit w**29** is XOR'd with input bit a**26**. The results of these XOR operations is provided to popcount **220***d*, where the number of 1's bits from the XOR operations is calculated. The results from popcount **220***d* are provided to summation **222***d*, which is combined with a current value stored in destination sub-location **212***d*. The output from summation **222***d* is written to destination sub-location **212***d*.

[0053] With respect to copy of weight bits **204***e*, weight bit w**31** is XOR'd with input bit a**27**, weight bit w**30** is XOR'd with input bit a**26**, and weight bit w**29** is XOR'd with input bit a**25**. The results of these XOR operations is provided to popcount **220***e*, where the number of 1's bits from the XOR operations is calculated. The results from popcount **220***e* are provided to summation **222***e*, which is combined with a current value stored in destination sub-location **212***e*. The output from summation **222***e* is written to destination sub-location **212***e*.

[0054] FIGS. **3**A and **3**B are conceptual block diagrams showing another example of bit and register structures in accordance with embodiments described herein. Convolution bit structure **300**A in FIG. **3**A illustrates a plurality of input bits **302** and a plurality of weight bits **304**. In various embodiments, this bit structure is utilized when the filter input value is zero indicating a fully connected convolution layer.

[0055] The input bits **302** are obtained from an input register (not shown). In this example, the input bits **302** include 32 bits. The weight bits **304** are obtained from a weight register (not shown). In this example, the weight bits **304** include 32 bits. Each weight bit **304** corresponds to an input bit **302**. For example, weight bit w**31** corresponds to input bit a**31**, weight bit w**30** corresponds to input bit a**30**, and so on.

[0056] FIG. **3**B is a further conceptual block diagram of the bit and register structure discussed above in FIG. **3**A. Block structure **300**B includes input bits **302**, weight bits **304**, popcount **306**, summation **308**, and destination sub-location **310**.

[0057] Each corresponding weight bit **304** is XOR'd with a corresponding input bit **302**. For example, weight bit w**31** is XOR'd with input bit a**31**, weight bit w**30** is XOR'd with input bit a**30**, weight bit w**29** is XOR'd with input bit a**29**, weight bit w**28** is XOR'd with input bit a**28**, and so on. The results of these XOR operations is provided to popcount **306**, where the number of 1's bits from the XOR operations is calculated. The results from popcount **306** are provided to summation **308**, which is combined with a current value stored in destination sub-location **310**. The output from

summation **308** is written to destination sub-location **310**. In some embodiments, destination sub-location **310** uses the same memory as destination sub-location **212***a* in FIG. **2**A.

[0058] FIGS. **4**A-**4**C are conceptual block diagrams showing an example architecture in accordance with embodiments described herein. Architecture **400**A in FIG. **4**A includes a filter size decoder **402** and ORs **404***a*-**404***e*. The opcode from the fetched new processor instruction described herein is input into filter size decoder **402**. In some embodiments, this input may be a separate input associated with the new processor instruction. Each output from filter size decoder is a single separate bit. Each separate output line or output bit represents a different filter size, where output line **2_1** represents a 2×1 filter, output line **3_1** represents a 3×1 filter, output line **4_1** represents a 4×1 filter, output line 5_1 represents a 5×1 filter, output line **6_1** represents a 6×1 filter, output line **7_1** represents a 7×1 filter, and output line X represents a fully connected layer.

[0059] The output lines from filter size decoder **402** are input into ORs **404***a*-**404***e*. In particular, output line **2_1** is input into OR **404***a;* output line **3_1** is input into OR **404***a*-**404***b;* output line **4_1** is input into OR **404***a*-**404***c;* output line **5_1** is input into OR **404***a*-**404***d;* output line **6_1** is input into OR **404***a*-**404***e;* and output line **7_1** is input into OR **404***a*-**404***e*. Output line **7_1** is also a separate line **406**.

[0060] If the output, labeled k**1**, from OR **404***a* is "1," then the filter is a size from 2×1 to 7×1. If the output, labeled k**2**, from OR **404***b* is "1," then the filter is a size from 3×1 to 7×1. If the output, labeled k**3**, from OR **404***c* is "1," then the filter is a size from 4'1 to 7×1. If the output, labeled k**4**, from OR **404***d* is "1," then the filter is a size from 5×1 to 7×1. If the output, labeled k**5**, from OR **404***e* is "1," then the filter is a size from 6×1 to 7×1. If line **406**, labeled k**6**, is "1," then the filter is a size of 7×1.

[0061] Architecture **400**B in FIG. **4**B includes OR **410**, XOR **412***a*-**412***e,* AND **414***a,* and one's count **416***a*-**416***e*. The outputs from OR **404***a*-**404***e* and line **406** in FIG. **4**A are provided as a 6 bit input to OR **410** in FIG. **4**B. Likewise, the output line X from OR **402** in FIG. **4**A is provided as a 6 bit input into OR **410**. OR **410** performs a logical OR on the inputs and outputs a six bit result. This result identifies the convolution filter to be applied. Accordingly, the result from OR **410** is provided as input to each of AND **414***a*-**414***e*.

[0062] Each XOR **412***a*-**412***e* has two seven bit inputs, one seven bit weight input and one seven bit data input. Seven bits are used for each input because the filter size ranges from 2×1 to 7×1. The actual number of active bit lines would vary depending on the filter input value provided with the new processor instruction. The seven bit weight input is a copy of weight bits [**31**:**25**] from the 32 bit weight register described herein. The seven bit data input is obtained from the 32 bit data input register described herein, but each input is shifted one bit. For example, the inputs to XOR **412***a* include weights [**31**:**25**] and data [**31**:**25**]; the inputs to XOR **412***b* include weights [**31**:**25**] and data [**30**:**24**]; the inputs to XOR **412***c* include weights [**31**:**25**] and data [**29**:**23**]; the inputs to XOR **412***d* include weights [**31**:**25**] and data [**28**:**22**]; and the inputs to XOR **412***e* include weights [**31**:**25**] and data [**27**:**21**].

[0063] Each XOR **412***a*-**412***e* performs a logical exclusive OR operation on the two inputs. The corresponding first six bits output (shown as Tmp_results[**5**:**0**]) from the corresponding XOR **412***a*-**412***e* are provided to corresponding AND **414***a*-**414***e*. The corresponding seventh bit output (shown as Tmp_result[**6**]) from the corresponding XOR **412***a*-**412***e* are provided to corresponding one's count **416***a*-**416***e*.

[0064] Each AND **414***a*-**414***e* performs a logical AND operation on the corresponding six bit input (Tmp_results [**5**:**0**]) and the six bit filter output from OR **410**. The corresponding results (shown as Tmp_results**2**[**5**:**0**]) from corresponding AND **414***a*-**414***e* are provided to corresponding one's count **416***a*-**416***e*.

[0065] Each one's count **416***a*-**416***e* performs operations to count the number of ones bits between the results (Tmp_results**2**[**5**:**0**]) from corresponding AND **414***a*-**414***b* and the seventh bit output (Tmp_result[**6**]) from corresponding XOR **412***a*-**412***e*. The output of one's count **416***a* is shown as Result**1**[**5**:**0**]; the output of one's count **416***b* is shown as Result**2**[**5**:**0**]; the output of one's count **416***c* is shown as Result**3**[**5**:**0**]; the output of one's count **416***d* is shown as Result**4**[**5**:**0**]; and the output of one's count **416***e* is shown as Result**5** [**5**:**0**].

[0066] Architecture **400**C in FIG. **4**C includes XOR **420**, one's count **422**, adder **424**, MUX **426** and adder **428**. In general, the MUX **426** selects between using the outputs from filters 2×1 to 7×1 in FIG. **4**B or a fully connected layer.

[0067] XOR **420** has two 25 bit inputs, weights[**24**:**0**] and data[**24**:**0**]. Weights[**24**:**0**] are the remaining weight bits in the weight register that are not used in FIG. **4**B, and data[**24**:**0**] are obtained from the input register that also provided the data input bits used in FIG. **4**B. XOR **420** performs a logical exclusive OR operation on the inputs and outputs a 25 bit result (shown as Tmp_result[**24**:**0**]). The output from XOR **420** is provided to one's count **422**, where a total number of ones bits are counted. The output from one's count **422** is a six bit output (shown as Result**6**[**5**:**0**]) that is provided to adder **424**.

[0068] Adder **424** adds the result (Result**6**[**5**:**0**]) from one's count **422** with the output (Result**1**[**5**:**0**]) from one's count **416***a* in FIG. **4**B. This addition calculates the total result of a fully connected layer because Result**1**[**5**:**0**] is obtained from data input[**31**:**25**] and Result**6**[**5**:**0**] is obtained from data input[**24**:**0**], thus using all bits from the 32 bit input register.

[0069] The output from adder **424** is shown as Result full[**5**:**0**] and is provided as input to MUX **426**. The combined results from one's count **416***a*-**416***e* in FIG. **4**B are provided as a 32 bit input into MUX **426** in FIG. **4**C. MUX **426** also includes a one bit control line, whose input is the X output line from filer size decoder **402** in FIG. **4**A. MUX **426** selects between using the results from a fully connected layer or the results from a filter between 2×1 to 7×1.

[0070] The output from MUX **426** is provided to adder **428**. Adder **428** adds the result from MUX **426** with the current destination register value (shown as destination register[**31**:**0**]). The output from adder **428** is then written to the destination register. Therefore, in a non-fully connected layer, the outputs of each separate one's count **416***a*-**416***d* in FIG. **4**B are stored in the corresponding sub-locations of the destination register, without having to make multiple calls or writes to the destination register.

[0071] The components shown in FIGS. **4**A-**4**C may include or be made up of one or more logical gates.

[0072] The operation of one or more embodiments will now be described with respect to FIGS. **5**A, **5**B and **6**, and for convenience will be described with respect to the

embodiments of FIGS. **1-4** described above. In at least one of various embodiments, processes **500** and **600** described in conjunction with FIGS. **5**A-**5**B and **6**, respectively, may be implemented by or executed on one or more computing devices, such as computing device **108** in FIG. **1**.

[0073] FIGS. **5**A and **5**B show a logical flow diagram of a process **500** for performing a new processor instruction to do binary multiply-accumulate operations in accordance with embodiments described herein. Process **500** begins, after a start block, at block **502**, where a destination-register location is received. The destination-register location identifies a memory location of a destination register. In various embodiments, the destination register stores 32 bits in memory. The destination register is logically separated into a plurality of destination sub-locations. In at least one embodiment, the destination register is separated into at least five sub-locations. These destination register sub-locations are utilized as accumulators.

[0074] Process **500** proceeds to block **504**, where a source-register location is received. The source-register location identifies the memory location of a source register that includes a plurality of input bits. In at least one embodiment, the source register stores 32 bits in memory. In some embodiments, the source register is loaded with input data received from another process or sensor. For example, the input data may be a portion of an image that is being analyzed using a DNN.

[0075] Process **500** continues at block **506**, where a weight-register location is received. The weight-register location identifies the memory location of a weight register that includes a plurality of weight bits. In at least one embodiment, the weight register stores 32 bits in memory. In some embodiments, the weight register is loaded with weights for processing the input data. In at least one embodiment, the weights may be selected for employment during convolution of a DNN.

[0076] Process **500** proceeds next to block **508**, where a filter input value is received. In various embodiments, the filter input value identifies the type or size of filters to be employed during convolution of the DNN.

[0077] Process **500** continues next at block **510**, where a sub-set of the weight bits in the weight register are copied. In some embodiments, the size of the copied sub-set is equal to the filter input value. In other embodiments, the size of the copied sub-set is equal to the maximum number of weight bits when then filter input value is zero, such as in during processing of a fully connected convolution layer. In at least one embodiment, the sub-set of weight bits is selected from the highest ordered bits in the weight bits.

[0078] Process **500** proceeds to decision block **512**, where a determination is made whether the number of copies of the sub-set of weight bits equals a select plurality of number of times. In at least one embodiment, the selected plurality of number of times is five. Although embodiments described herein discuss copying the sub-set of weight bits five times, other numbers of times may also be used. The number of copies may be selected based on the number of bits in the source register, the filter input value, or other factors. If the number of copies of the sub-set of weight bits equals the selected plurality of number of times, then process **500** flows to block **514** in FIG. **5**B; otherwise, process **500** loops to block **510** in FIG. **5**A to make another copy of the sub-set of weight bits.

[0079] At block **514** in FIG. **5**B, a copy of the sub-set of weight bits is selected.

[0080] Process **500** proceeds to block **516**, where a destination sub-location of the plurality of destination sub-locations is selected. This selected destination sub-location corresponds to the selected copy of the sub-set of weight bits. For example, a first destination sub-location may be selected for a first copy.

[0081] Process **500** continues at block **518**, where a corresponding sub-set of the plurality of input bits is selected for the selected copy of the sub-set of weight bits. For example, a first sub-set of input bits may be selected for a first copy of the sub-set of weight bits. In various embodiments, the number of bits in the sub-set of input bits is equal to the number of bits in the copy of the sub-set of weight bits.

[0082] Process **500** proceeds next to block **520**, where an XOR (exclusive "OR") operation is performed on each corresponding bit in the selected copy of sub-set of weight bits with each corresponding bit in the selected sub-set of input bits. For example, if the selected sub-set of input bits includes three bits: a**31**, a**30**, and a**29**, and if the selected sub-set of weight bits includes three bits: w**31**, w**30**, and w**29**, then the following corresponding bit XOR operations are performed: a**31** XOR w**31**, a**30** XOR w**30**, and a**29** XOR w**29**.

[0083] Process **500** continues next at block **522**, where the output of each XOR operation in block **520** is aggregated with each other and with a current value stored in the selected destination sub-location. For example, if the output of a**31** XOR w**31** is 1, the output of a**30** XOR w**30** is 0, and the output of a**29** XOR w**29** is 1, then the aggregated XOR output value is 2. If the currently stored value in the selected destination sub-location is 3, then the total aggregated value is 5.

[0084] Process **500** proceeds to block **523**, where the total aggregated value is stored in the destination registration at the selected destination sub-location. In this way, the previously stored value in the selected destination sub-location is written over with the new total aggregated value.

[0085] Process **500** continues at decision block **524**, where a determination is made whether to select another copy of the sub-set of weight bits. In various embodiments, the determination to select another copy of the sub-set of weight bits will continue until all copies have been selected. If another copy of the sub-set of weight bits is to be selected, process **500** flows to block **526**; otherwise, process **500** terminates or otherwise returns to a calling process to perform other actions.

[0086] At block **526**, a next copy of the sub-set of weight bits is selected. In various embodiments, block **526** may include embodiments of block **514**, but to select another, non-processed copy of subset of weight bits.

[0087] Process **500** proceeds next to block **528**, where a destination sub-location that corresponds to the selected next copy of sub-set of weight bits is selected. For example, a second destination sub-location may be selected for a second copy. In various embodiments, block **528** may include embodiments of block **516**.

[0088] Process **500** continues next to block **530**, where a next corresponding sub-set of the plurality of input bits is selected for the selected next copy of sub-set of weight bits. The selected next sub-set of input bits are selected by shifting the sub-set one bit, such as one bit to the right, from

the previously selected sub-set of input bits. For example, if the input bits include a**31**, a**30**, a**29**, a**28**, a**27**, . . . , a**0**, and the previously selected sub-set of input bits includes a**31**, a**30**, and a**29**, then the next selected sub-set of input bits includes a**30**, a**29**, and a**28**.

[0089] After block **530**, process **500** loops to block **520** where an XOR operation is performed on each corresponding bit in the selected next copy of the sub-set of weight bits with each corresponding bit in the selected next sub-set of input bits.

[0090] Although process **500** is described as looping through the copies of the sub-set of weight bits, embodiments are not so limited. In various embodiments, separate copies of the sub-set of weight bits are utilized in parallel. Thus, the performance of blocks, **514**, **516**, **518**, **520**, **522**, and **523** for a first copy of the sub-set of weight bits, a first sub-set of input bits, and a first destination sub-location may be in parallel to the performance of blocks, **514**, **516**, **518**, **520**, **522**, and **523** for a second copy of the sub-set of weight bits, a second sub-set of input bits, and a second destination sub-location. In this way, multiple sub-set of input values are processed in parallel. In at least one embodiment, these parallel operations are being performed for five sub-sets of input values using five copies of the sub-set of weight bits, along with five corresponding destination sub-locations.

[0091] FIG. **6** shows a logical flow diagram of an alternative process **600** for performing the new processor instruction to do binary multiply-accumulate operations in accordance with embodiments described herein.

[0092] Process **600** begins, after a start block, at block **602**, where a destination-register location is received. In various embodiments, block **602** may perform embodiments similar to block **502** in FIG. **5**A.

[0093] Process **600** proceeds to block **604**, where a source-register location is received. In various embodiments, block **604** may perform embodiments similar to block **504** in FIG. **5**A.

[0094] Process **600** proceeds to block **606**, where a weight-register location is received. In various embodiments, block **606** may perform embodiments similar to block **506** in FIG. **5**A.

[0095] Process **600** proceeds to block **608**, where a filter index value is received. In various embodiments, block **608** may perform embodiments similar to block **508** in FIG. **5**A.

[0096] Process **600** continues next at block **610**, where a sub-set of the weight bits in the weight register is selected. In some embodiments, the size of the sub-set is equal to the filter input value. In other embodiments, the size of the sub-set is equal to the maximum number of weight bits when then filter input value is zero, such as in during processing of a fully connected convolution layer. In at least one embodiment, the sub-set of weight bits is selected from the highest ordered bits in the weight bits.

[0097] Process **600** proceeds to block **612**, where a destination sub-location of the plurality of destination sub-locations is selected. This selected destination sub-location corresponds to the selected sub-set of weight bits. For example, a first destination sub-location may be selected for a first selected sub-set of weight bits.

[0098] Process **600** continues at block **614**, where a corresponding sub-set of the plurality of input bits is selected for the selected sub-set of weight bits. In various embodiments, the number of bits in the sub-set of input bits is equal to the number of bits in the selected sub-set of weight bits.

[0099] Process **600** proceeds next to block **616**, where an XOR (exclusive "OR") operation is performed on each corresponding bit in the selected of sub-set of weight bits with each corresponding bit in the selected sub-set of input bits. In various embodiments, block **616** may perform embodiments similar to block **520** in FIG. **5**B.

[0100] Process **600** continues next at block **618**, where the output of each XOR operation in block **616** is aggregated with each other and with a current value stored in the selected destination sub-location. In various embodiments, block **618** may perform embodiments similar to block **522** in FIG. **5**B.

[0101] Process **600** proceeds to block **619**, where the total aggregated value is stored in the destination registration at the selected destination sub-location. In various embodiments, block **619** may perform embodiments similar to block **523** in FIG. **5**B.

[0102] Process **600** continues at decision block **620**, where a determination is made whether to select another sub-set of input bits. In various embodiments, the determination to select another sub-set of input bits is performed until a select number of sub-sets have been selected. If another sub-set of input bits is to be selected, process **600** flows to block **622**; otherwise, process **600** terminates or otherwise returns to a calling process to perform other actions.

[0103] At block **622**, a next destination sub-location is selected. For example, a second destination sub-location may be selected for a second sub-set of input bits. In various embodiments, block **622** may include embodiments of block **612**.

[0104] Process **600** continues next to block **624**, where a next corresponding sub-set of the plurality of input bits is selected. The selected next sub-set of input bits are selected by shifting the sub-set one bit, such as one bit to the right, from the previously selected sub-set of input bits. In various embodiments, block **624** may include embodiments of block **530** in FIG. **5**B.

[0105] After block **624**, process **600** loops to block **616** where an XOR operation is performed on each corresponding bit in the selected sub-set of weight bits with each corresponding bit in the selected next sub-set of input bits.

[0106] Although process **600** is described as looping through separate sub-sets of input bits, embodiments are not so limited. In various embodiments, separate sub-sets of input bits are processed in parallel. Thus, the performance of blocks, **612**, **614**, **616**, **618**, and **619** for a first sub-set of input bits, a first destination sub-location, and the selected sub-set of weight bits may be in parallel to the performance of blocks, **612**, **614**, **616**, **618**, and **619** for a second sub-set of input bits, a second destination sub-location, and the selected sub-set of weight bits. In this way, multiple sub-set of input values are processed in parallel. In at least one embodiment, these parallel operations are being performed for five sub-sets of input values, while reusing the sub-set of weight bits, along with five corresponding destination sub-locations.

[0107] In the foregoing description, certain specific details are set forth to provide a thorough understanding of various disclosed embodiments. However, one skilled in the relevant art will recognize that embodiments may be practiced without one or more of these specific details, or with other methods, components, materials, etc. In other instances, well-known structures associated with electronic and computing systems including client and server computing sys-

tems, as well as networks have not been shown or described in detail to avoid unnecessarily obscuring descriptions of the embodiments.

[0108] Unless the context requires otherwise, throughout the specification and claims which follow, the word "comprise" and variations thereof, such as, "comprises" and "comprising," are to be construed in an open, inclusive sense, e.g., "including, but not limited to."

[0109] The headings and Abstract of the Disclosure provided herein are for convenience only and do not limit or interpret the scope or meaning of the embodiments.

[0110] The various embodiments described above can be combined to provide further embodiments. Aspects of the embodiments can be modified, if necessary to employ concepts of the various patents, application and publications to provide yet further embodiments.

[0111] These and other changes can be made to the embodiments in light of the above-detailed description. In general, in the following claims, the terms used should not be construed to limit the claims to the specific embodiments disclosed in the specification and the claims, but should be construed to include all possible embodiments along with the full scope of equivalents to which such claims are entitled. Accordingly, the claims are not limited by the disclosure.

1. A method, comprising:

receiving a destination-register location configured to store accumulation results, wherein the destination-register location includes a plurality of destination sub-locations;

receiving a source-register location configured to store a plurality of input bits;

receiving a weight-register location configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an input length of the plurality of input bits;

copying, using the weight-register location, a sub-set of the plurality of weight bits a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and

for each copy of the sub-set of weights:

selecting, using the source-register location, a sub-set of the plurality of input bits based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits;

performing an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and

aggregating, in a corresponding destination sub-location of the plurality of destination sub-locations, an output of each XOR operation with each other and with a current value of the corresponding destination sub-location.

2. The method of claim **1**, further comprising:

receiving the filter index value between 2 and 7.

3. The method of claim **1**, further comprising:

receiving the filter index value of zero to indicate a fully connected layer.

4. The method of claim **1**, wherein copying the sub-set of the plurality of weight bits the select plurality of number of times comprises:

copying the sub-set of the plurality of weight bits five times.

5. The method of claim **1**, further comprising:

for each copy of the sub-set of weights:

performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits.

6. The method of claim **1**, further comprising:

performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and

adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

7. The method of claim **6**, further comprising:

for each copy of the sub-set of weights:

performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits;

generating a filtered output by concatenating outputs from the one's count operations for each copy of the sub-set of weights;

performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

generating a fully connected output by adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits;

selecting a final result between the filtered output and the fully connected output based on the filter index value; and

combining the final result with a current value stored at the destination-register location.

8. A system, comprising:

a memory that stores:

a destination register configured to store accumulation results, wherein the destination-register includes a plurality of sub-destinations;

a source register configured to store a plurality of input bits;

a weight register configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an input length of the plurality of input bits;

a microprocessor coupled to the memory, wherein the microprocessor, in operation:

copies a sub-set of the plurality of weight bits in the weight register a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and

for each copy of the sub-set of weights:

selects a sub-set of the plurality of input bits from the source register based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits;

performs an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and

aggregates, in a corresponding sub-destination of the plurality of sub-destinations in the destination register, an output of each XOR operation with each other and with a current value of the corresponding sub-destination.

9. The system of claim **8**, wherein the microprocessor, in further operation:

receives the filter index value between 2 and 7.

10. The system of claim **8**, wherein the microprocessor, in further operation:

receives the filter index value of zero to indicate a fully connected layer.

11. The system of claim **8**, wherein the microprocessor, in further operation:

copies the sub-set of the plurality of weight bits five times.

12. The system of claim **8**, wherein the microprocessor, in further operation:

for each copy of the sub-set of weights:

performs a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits.

13. The system of claim **8**, wherein the microprocessor, in further operation:

performs an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

performs a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and

adds the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

14. The system of claim **13**, wherein the microprocessor, in further operation:

for each copy of the sub-set of weights:

performs a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits;

generates a filtered output by concatenating outputs from the one's count operations for each copy of the sub-set of weights;

performs an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

performs a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

generates a fully connected output by adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits;

selects a final result between the filtered output and the fully connected output based on the filter index value; and

combines the final result with a current value stored at the destination register.

15. A non-transitory computer-readable medium having contents that configure a microcontroller to perform a method, the method comprising:

receiving a destination-register location configured to store accumulation results, wherein the destination-register location includes a plurality of destination sub-locations;

receiving a source-register location configured to store a plurality of input bits;

receiving a weight-register location configured to store a plurality of weight bits, wherein a weight length of the plurality of weight bits is equal to an input length of the plurality of input bits;

copying, using the weight-register location, a sub-set of the plurality of weight bits a select plurality of number of times, wherein a size of the sub-set of weights is based on a filter index value; and

for each copy of the sub-set of weights:

selecting, using the source-register location, a sub-set of the plurality of input bits based on the size of the sub-set of weights, wherein the sub-set of input bits is shifted one bit from a previous sub-set of the plurality of input bits;

performing an XOR operation on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits; and

aggregating, in a corresponding destination sub-location of the plurality of destination sub-locations, an output of each XOR operation with each other and with a current value of the corresponding destination sub-location.

16. The non-transitory computer-readable medium of claim **15**, wherein receiving the filter index value comprises:

receiving the filter index value between 2 and 7.

17. The non-transitory computer-readable medium of claim **15**, wherein receiving a filter index value comprises:

receiving the filter index value of zero to indicate a fully connected layer.

18. The non-transitory computer-readable medium of claim **15**, wherein copying the sub-set of the plurality of weight bits the select plurality of number of times comprises:

copying the sub-set of the plurality of weight bits five times.

19. The non-transitory computer-readable medium of claim **15**, further comprising:

for each copy of the sub-set of weights:

performing a one's count operation on an output from the XOR operations on each corresponding bit in the copy of the sub-set of weights with each corresponding bit in the selected sub-set of input bits.

20. The non-transitory computer-readable medium of claim **15**, further comprising:

performing an XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits;

performing a one's count operation on an output from the XOR operation on each corresponding remaining bit in the plurality of weights with each corresponding remaining bit in the input bits; and

adding the output of the one's count operation with another output from another one's count operation performed on an output from the XOR operation of each corresponding bit in a first copy of the sub-set of weights with each corresponding bit in the a first selected sub-set of input bits.

* * * * *