(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 December 2010 (09.12.2010)

PCT

(10) International Publication Number
WO 2010/141222 A1

(51) International Patent Classification:
G06F 15/78 (2006.01)

(21) International Application Number:
PCT/US2010/035450

(22) International Filing Date:
19 May 2010 (19.05.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
12/478,412 4 June 2009 (04.06.2009) US

(71) Applicant *(for all designated States except US)*: **MICRON TECHNOLOGY, INC.** [US/US]; 8000 S. Federal Way, Boise, Idaho 83707-0006 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **WALKER, Robert** [US/US]; 9000 Deerland Grove Drive, Raleigh, North Carolina 27615 (US).

(74) Agents: **MANWARE, Robert, A.** et al.; 7915 FM 1960 West, Suite 330, Houston, TX 77070 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

(54) Title: PARALLEL PROCESSING AND INTERNAL PROCESSORS

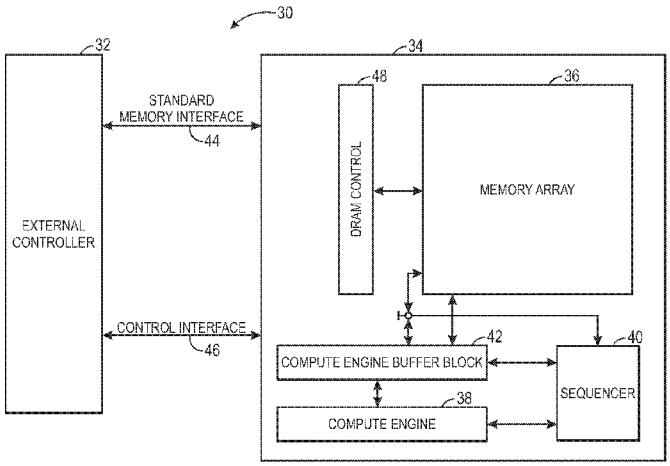

FIG. 2

(57) Abstract: Systems, internal processors (38), and methods of parallel data processing in an internal processor (38) are provided. In one embodiment, an external controller (32) sends instructions to a memory device (34), and the internal processor (38) on the memory device (34) executes the instructions on the data. The internal processor (38) may include one or more arithmetic logic units (ALUs) (50), and each ALU (50) may perform an operation on an entire operand (A, B), such that one or more operands (A, B) may be processed in parallel by one or more ALUs (50) in the internal processor (38). The operations may be completed on each operand (A, B) in one or more cycles through the circuitry of the ALU (50), and the path of the operands (A, B) through the ALU (50) may be based on the width of the ALU (50), the size of the operands (A, B), or the type of operation to be performed.

WO 2010/141222 A1

PARALLEL PROCESSING AND INTERNAL PROCESSORS

BACKGROUND

Field of Invention

[0001] Embodiments of the invention relate generally to memory systems, and more particularly, to memory systems having internal processors.

Description of Related Art

[0002] This section is intended to introduce the reader to various aspects of art that may be related to various aspects of the present invention, which are described and/or claimed below. This discussion is believed to be helpful in providing the reader with background information to facilitate a better understanding of the various aspects of the present invention. Accordingly, it should be understood that these statements are to be read in this light and not as admissions of prior art.

[0003] Electronic systems typically include one or more processors, which may retrieve and execute instructions, and output the results of the executed instruction, such as to store the results to a suitable location. A processor generally includes arithmetic logic unit (ALU) circuitry, which is capable of executing instructions such as arithmetic and logic operations on one or more operands. For example, the ALU circuitry may add, subtract, multiply, or divide one operand from another, or may subject one or more operands to logic operations, such as AND, OR, XOR, and NOT logic functions. The various arithmetic and logic operations may have different degrees of complexity. For example, some operations

may be performed by inputting the operand(s) through the ALU circuitry in one cycle, while other operations may utilize multiple clock cycles.

[0004] A number of components in the electronic system may be involved in directing a set of instructions to the ALU for execution. In some devices, the instructions and any corresponding data (e.g., the operands on which the instructions will be executed) may be generated by a controller, or some other suitable processor in the electronic system. As the time or number of clock cycles required for the execution of a set of instructions may vary depending on the type of operation, the instructions and/or data may be written to a memory device, for example, a memory array, before being executed by the ALU. The instructions and data may be retrieved and sequenced and/or buffered before the ALU begins to execute the instructions on the data.

[0005] To improve processing performance, the steps of writing, reading, sequencing, buffering, and executing instructions and/or data may occur substantially simultaneously on different instructions, or different parts of an instruction. This parallel processing may be referred to as "pipelining." The performance of the device may also be improved in a processor-in-memory (PIM) device, where the processor (e.g., the ALU) is implemented directly on the memory device, conserving power in processing. Further, processing performance of the electronic system may also be improved in the data processing of the ALU.

BRIEF DESCRIPTION OF DRAWINGS

[0006] Certain embodiments are described in the following detailed description and in reference to the drawings in which:

[0007] FIG. 1 depicts a block diagram of a processor-based system in accordance with an embodiment of the present technique;

[0008] FIG. 2 depicts a block diagram of a memory system with embedded arithmetic logic units interfaced with an external memory controller, in accordance with an embodiment of the present technique;

[0009] FIG. 3 depicts a block diagram of a compute buffer and a compute engine comprising ALUs embedded on a memory device, in accordance with one or more embodiments of the present technique;

[0010] FIG. 4 illustrates a basic adder in a typical ALU;

[0011] FIG. 5 illustrates an adder with logic gates for mathematical operations in a 1 byte ALU;

[0012] FIG. 6A and 6B depict a part of a compute engine having a 1 byte ALU, in accordance with one or more embodiments of the present technique;

[0013] FIG. 7 depicts a block diagram of an input component for a 1 byte ALU, in accordance with one or more embodiments of the present technique; and

[0014] FIG. 8 depicts a block diagram of a shift unit used for shifting data between ALUs in an 8 bit ALU, in accordance with one or more embodiments of the present technique.

DETAILED DESCRIPTION

[0015] Arithmetic logic unit (ALU) circuitry is generally used to process instructions in multiple stages. Processing the instructions may include executing the instructions, and storing the results of the executed instructions. More specifically, instructions, and the data on which the instructions will be executed, may be sent by a controller to the ALU, and may first be stored in a memory device to be retrieved when the ALU circuitry is available to execute the instructions. Once the instructions have been executed, the ALU may write the results of the operation to a memory component, or to any other suitable output.

[0016] In one or more embodiments of the present techniques, one or more processors, such as ALUs, may be packaged with a memory device. For example, the memory device may be a processor-in-memory (PIM), and may include embedded ALUs and a memory array, which may store instructions and data to be executed by the ALUs and the results from the completed instructions. In other embodiments, the ALUs and the memory array may be on unique dies in the same package. For example, the ALUs and the memory array may be arranged in a multi-chip package (MCP), and may be electrically connected by one or more through-silicon vias (TSVs). Processors which are embedded on a memory device, or packaged with a memory component in a memory device, may be referred to as "internal processors," as they are internal to the memory device. As used herein, a

"compute engine" may be an example of an internal processor, and may be embedded on or packaged in a memory device in accordance with the present techniques.

[0017] While a processor that is external to the memory device may require an external input/output (I/O) to transfer information (e.g., instructions and/or data) to and from the memory array of the memory device, a compute engine may conserve power consumption by allowing information to be transferred between the memory array and the compute engine without an external I/O. The memory device may also include components such as a sequencer to organize the instructions, and a memory component such as a buffer to hold data before the compute engine performs the operations.

[0018] As discussed, the compute engine may perform various mathematical and logical operations, and may also be referred to as an internal processor of the memory device. The compute engine may have a number of basic building blocks, which may be ALUs that are each one byte wide. The ALUs of the compute engine may be configured in a way to improve processing performance. One embodiment of the present technique involves a memory device having an embedded compute engine configured for parallel data processing. Parallel data processing in the compute engine may enable one ALU of the compute engine to operate on one operand. While each ALU may take more than one cycle to complete an instruction on an operand, each of the ALUs in the compute engine may process a different operand, allowing the compute engine to process multiple operands in parallel. Thus, in accordance with the present parallel processing techniques, a memory device having an embedded compute engine may process a larger amount of data within the same memory device.

[0019] Now turning to the figures, FIG. 1 depicts a processor-based system, generally designated by reference numeral 10. As is explained below, the system 10 may include various electronic devices manufactured in accordance with embodiments of the present technique. The system 10 may be any of a variety of types such as a computer, pager, cellular phone, personal organizer, control circuit, etc. In a typical processor-based system, one or more processors 12, such as a microprocessor, control the processing of system functions and requests in the system 10. As is explained below, the processor 12 and other subcomponents of the system 10 may include memory devices manufactured in accordance with one or more embodiments of the present technique.

[0020] The system 10 typically includes a power supply 14. For instance, if the system 10 is a portable system, the power supply 14 may advantageously include a fuel cell, a power scavenging device, permanent batteries, replaceable batteries, and/or rechargeable batteries. The power supply 14 may also include an AC adapter, so the system 10 may be plugged into a wall outlet, for instance. The power supply 14 may also include a DC adapter such that the system 10 may be plugged into a vehicle cigarette lighter, for instance.

[0021] Various other devices may be coupled to the processor 12 depending on the functions that the system 10 performs. For instance, an input device 16 may be coupled to the processor 12. The input device 16 may include buttons, switches, a keyboard, a light pen, a mouse, a digitizer and stylus, and/or a voice recognition system, for instance. A display 18 may also be coupled to the processor 12. The input device 16 and/or the display 18 may each or both form a user interface. The display 18 may include an LCD, an SED

display, a CRT display, a DLP display, a plasma display, an OLED display, LEDs, and/or an audio display, for example. Furthermore, an RF sub-system/baseband processor 20 may also be coupled to the processor 12. The RF sub-system/baseband processor 20 may include an antenna that is coupled to an RF receiver and to an RF transmitter (not shown). One or more communication ports 22 may also be coupled to the processor 12. The communication port 22 may be adapted to be coupled to one or more peripheral devices 24 such as a modem, a printer, a computer, or to a network, such as a local area network, remote area network, intranet, or the Internet, for instance.

[0022] The processor 12 generally controls the system 10 by processing software programs stored in the memory. The software programs may include an operating system, database software, drafting software, word processing software, and/or video, photo, or sound editing software, for example. The memory is operably coupled to the processor 12 to store and facilitate execution of instructions to implement various programs. For instance, the processor 12 may be coupled to the system memory 26, which may include dynamic random access memory (DRAM), and/or synchronous dynamic random access memory (SDRAM). The system memory 26 may include volatile memory, non-volatile memory, or a combination thereof. The system memory 26 is typically large so that it can store dynamically loaded applications and data.

[0023] The processor 12 may also be coupled to non-volatile memory 28, which is not to suggest that system memory 26 is necessarily volatile. The non-volatile memory 28 may include read-only memory (ROM), such as an EPROM, resistive read-only memory (RROM), and/or flash memory to be used in conjunction with the system memory 26. The

size of the ROM is typically selected to be just large enough to store any necessary operating system, application programs, and fixed data. Additionally, the non-volatile memory 28 may include a high capacity memory such as a tape or disk drive memory, such as a hybrid-drive including resistive memory or other types of non-volatile solid-state memory, for instance.

[0024] Some embodiments of the present technique involve communication between the processor 12 and components of the system memory 26. For example, the processor 12 may include a general purpose processor, a central processing unit, a processor core, an ASIC, a memory controller, and/or an ALU, for example, capable of sending and receiving signals from internal processors of memory devices in the system memory 26. Components of the system 10 involved in the communication between the processor 12 and the components of the system memory 26 may be generally referred to as a "memory system" 30, as illustrated in the block diagram of FIG. 2. In some embodiments, a memory system 30 may include a memory device 34, which may be part of the system memory 26 of the system 10 (as in FIG. 1) and may have an internal processor. The memory system 30 may also include an external processor 32, which may be in a system-on-a-chip (SOC) with a more general purpose processor to collectively form a processor 12 of a processor-controlled system 10 (as in FIG. 1). The external processor 32, which may also be an external memory controller, may communicate with and/or control certain components of a memory device 34.

[0025] The memory system 30 may include components which have functions that are not limited to the communication between the external processor 32 and the memory

device 32. For example, the external processor 32 may control devices in addition to the memory device 34. However, the external processor 32, as explained with respect to the memory system 30, may refer to one function of the external processor 32 which communicates with and/or controls certain components of the memory device 34. Likewise, not all parts of the system memory 26 may be part of the memory system 30. The "memory device" 34 may refer to components of the system memory 26 involved in the communication with the external processor 32, in accordance with the present techniques.

[0026] The external processor 32 and the memory device 34 may be operably coupled by a standard memory interface 44 (e.g., DDR, DDR2, DDR3, LPDDR, or LPDDR2), which may allow data transfer between the external processor 32 and the memory device 34, and may allow the external processor 32 to send (e.g., transfer) commands to the memory device 34. In one or more embodiments, the types of standard memory interface 44 may include DDR, DDR2, DDR3, LPDDR, or LPDDR2, for example. Further, in some embodiments, an additional interface(s) may be configured to allow the transfer of data, and also commands (e.g., requests, grants, instructions, etc.), between the memory device 34 and the external processor 32. For example, the external processor 32 and the memory device 34 may also be operably coupled by a control interface 46, which may allow the transfer of commands between the external processor 32 and the memory device 34, including commands from the memory device 34 to the external processor 32.

[0027] The memory device 34 may include a compute engine 38 and a memory array 36. The memory array 36 may refer to any suitable form of storage, and may include, for

example, a DRAM array or an SDRAM array. The memory controller 32 may have access to the memory array 36, and may be able to write data or instructions to be executed by the compute engine 38. The compute engine 38 may include one or more arithmetic logic units (ALUs).

[0028] The compute engine 38 may be embedded on the memory device 34 and capable of accessing the memory array 36, including retrieving information from, and storing information in the memory array 36. The process of retrieving and storing information between the compute engine 38 and the memory array 36 may involve a sequencer 40 and compute engine buffer block 42. The sequencer 40 may sequence the instructions sent by the controller 32 to the memory array 36 and store the data retrieved from the memory array 36 in a memory component such as the compute engine buffer block 42. Once the compute engine 38 has executed the instructions, the results may be stored in the compute engine buffer block 42 before they are written to the memory array 36. Further, as some instructions may require more than one clock cycle in the compute engine, intermediate results may also be stored in memory components in the memory device 34. For example, intermediate results may be stored in memory components such as the compute engine buffer block 42, other buffers, or registers coupled to the compute engine 38. In some embodiments, the compute engine buffer block 42 may include more than one layer of buffers. For example, the buffer block 42 may include a compute buffer, which may store operands, and an instruction buffer, which may store instructions. The buffer block 42 may also include additional buffers, such as a data buffer or a simple buffer, which may provide denser storage, and may store intermediate or final results of executed instructions. As

used herein, "buffer 42" may refer to any layer (e.g., a compute buffer, instruction buffer, data buffer, etc.) in the compute engine buffer block 42.

[0029] In a typical memory system 30, an external processor 32 may store data and instructions in the memory array 36 on the memory device 34. A sequencer 40 may access the memory array 36 to retrieve the instructions, and may copy the data from the memory array 36 to the buffer 42. The block diagram of FIG. 3 illustrates a compute engine 38 having a plurality of ALUs 50, and may be connected to the buffer 42. In one embodiment, the buffer 42 may be configured such that data may be written to and read from storage elements in the buffer 42 to allow savings in the number of compute cycles of the compute engine 38. Further, the compute engine 38 may be configured such that each ALU 50 may operate on one operand at a time. As will be further discussed with reference to FIGS. 6A and 6B, each ALU 50 in the compute engine 38 may operate on an operand, and multiple operands may be operated on in parallel to increase the efficiency of the compute engine 38.

[0030] An ALU 50 may operate on any size operand, and depending on the size of the operand, the operation may be performed through one or more cycles through an ALU 50. A basic building block of an ALU 50, a full adder 52, is depicted in the diagram of FIG. 4. Operands A and B may enter through the logic gates of the full adder 52. In some embodiments, the OR gate 54 may instead be an XOR gate. In some operations, the operands A and B may pass through one cycle of the full adder 52, and may be output through the carry-out output 56 to re-enter the full adder 52 through the carry-in input 58. The sum output 60 may output the completed operation (e.g., a summing operation) of the

operands A and B. The full adder 52 may be capable of performing mathematical operations, such as addition, subtraction, and multiplication.

[0031] The capabilities of a full adder 52 may be increased by adding additional logic gates, as depicted in the diagram of FIG. 5. In some embodiments, the additional logic gates may allow the ALU, and the compute engine, to perform a more complicated set of instructions without significantly adding to the size of the ALU, or the size of the compute engine. Adding gates to the full adder may form a 1 bit (1b) ALU 62. Multiple 1b ALUs may be connected to form the ALU circuitry of the compute engine 38. For example, in one embodiment, eight 1b ALUs may be placed in parallel to form an 8 bit (8b) ALU. In other embodiments, the compute engine 38 may comprise any number of ALUs, which may be connected in different ways, in accordance with the present techniques. Further, a compute engine 38 may comprise any number of 8b ALUs or different arrangements of 1b ALUs 62.

[0032] The diagram of FIGS. 6A and 6B depicts one embodiment of an 8b (i.e., single byte) ALU 50 in a portion of a compute engine 38. The illustration of the 8b ALU 50 has been split between FIGS. 6A and 6B, and the lettering (marked N-V) correspond to how each half is joined to form the 8b ALU 50. The 8b ALU 50 is configured to receive operands from a memory component such as the buffer block 42, and configured to operate on the received operands in parallel. Data may be input to each 1b ALU 62 of an 8b ALU 50 through an input multiplexer 64. In some embodiments, the sequencer 40 (FIG. 2) may select the data to be written to each 1b ALU 62, from five different inputs of the input mux 64, including constant register (labeled "const [0-7]"), array A ("A [0-7]"), array B ("B [0-

7]"), shift register ("shiftReg [0-7]"), and sum register ("SUM [0-7]"). For some operations, operands may cycle through one or more 1b ALUs 62 more than once, and the outputs of one cycle through a 1b ALU 62, which could be an intermediate result, may be carried in as an input for another 1b ALU 62 in the 8b ALU 50. Thus, the constant register may be input into one 1b ALU 62 from one cycle through another 1b ALU 62, for example, a summing operation from another 1b ALU 62. The arrays A and B may be input from different parts of the memory array 36. In some embodiments, the memory array may include banks A and B, which may be connected to one another, configured around the compute engine buffer block 42, or otherwise configured on the memory device 34. Further, intermediate results of operations may also be input to the input mux 64 through the sum register and shift register inputs. For example, in one embodiment, intermediate results may be output from the 1b ALUs 62 and stored in the sum register or the shift register until they are input back into the input mux 64. A more detailed diagram illustrating the circuitry in one embodiment of the input mux 64 is provided in FIG. 7. The input mux 64, as depicted in FIGS. 6A and 6B, is presented beside the detailed circuitry of the input mux 64 for comparison purposes. The circuitry of the input mux 64 may include logic gates and multiplexers, configured as illustrated.

[0033] The number of clock cycles through an ALU 50 to perform an operation on an operand may be based on the operation and the size of the operand to be operated on. For example, if the ALU 50 is 8 bits wide, and the operand size is 8 bits or less, the ALU 50 may receive operand A and operand B through the input, and may perform a logical operation on the operands in a single clock cycle. In some embodiments, the number of cycles needed to complete an operation on the operands may be based on the bit size of the

operand, divided by the bit size of the ALU. If the operand size is greater than 8 bits, an 8b ALU 50 may perform the operation on the first byte of the operands A and B. Once the first byte has been completed, the ALU may accept the next bits to perform the operation on a second byte.

[0034] Some operations, for example, multiplication operations, may require multiple cycles to complete, even when an operand is 8 bits or less. In operations which are executed through multiple cycles, an output of one cycle through a 1b ALU 62 may be an input into another 1b ALU 62 in the same 8b ALU 50. For example, each 8b ALU may have a shift unit 66 which may shift the results of a multiply operation to the left from SUM[0]. As seen in the diagram of a shift unit 66 in FIG. 8, when two 8b operands are multiplied together with an 8b ALU 50, the lower byte of the results may be formed in the shift registers, while the upper byte of the results may be formed in the sum registers. As illustrated in both FIG. 8 and FIGS. 6A and 6B, the shift unit 66 may comprise a plurality of multiplexers and flip flops to shift results from one 1b ALU 62 to another in the 8b ALU 50.

[0035] The use of the shift unit 66 in the 8b ALU 50 may be seen in a multiple cycle operation like a multiplication operation. In operations where the operands are smaller, a shift register may be sufficient to output the intermediate results to the input of the input mux 64. In operations where the operands are larger, the buffer 42 may be used to store intermediate results before the intermediate results are input to the input of the input mux 64.

[0036] Table 1 below provides one example, illustrating the multiplication of two 8 bit operands B x A to produce a 16 bit result.

		B							0	0	1	1	0	1	1	0
	X	A							1	1	0	0	1	0	0	1
		C							0	0	1	1	0	1	1	0
								0	0	0	0	0	0	0	0	
							0	0	0	0	0	0	0	0		
						0	0	1	1	0	1	1	0			
					0	0	0	0	0	0	0	0				
				0	0	0	0	0	0	0	0					
			0	0	1	1	0	1	1	0						
	0	0	0	1	1	0	1	1	0							
Result	0	0	1	0	1	0	1	0	0	1	1	0	0	1	1	0
Bit Number:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 1

[0037] The result, in the row marked "Result," includes two 8b bytes, and the byte from bit numbers 0 to 7 may be the least significant byte, which may be stored in the shift register of the 8b ALU 50. The byte from bit numbers 8 to 15 may be the most significant byte, which may be stored in the sum register of the 8b ALU 50.

[0038] Table 2 provides the operation of each clock cycle of the same multiplication of 8b operands A and B, and the bytes stored in the sum register and shift register at each cycle.

clock Cycle:	Carry	Sum Reg	ShiftReg	Operation:
0	0	8'b00000000	8'b11001001	Copy multiplier (A) in to shiftReg, set SUM and Carry registers to 0
1	0	8'b00011011	8'b01100100	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
2	0	8'b00001101	8'b10110010	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
3	0	8'b00000110	8'b11011001	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
4	0	8'b00011110	8'b01101100	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit

				right
5	0	8'b00001111	8'b00110110	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
6	0	8'b00000111	8'b10011011	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
7	0	8'b00011110	8'b11001101	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right
8	0	8'b00101010	8'b01100110	if(shiftReg[0]==1) SUM = SUM + multiplicand (B) ELSE SUM = SUM; shift {Carry,SUM, shiftReg} 1bit right

Table 2

[0039] As seen in Table 2, the sum register is initially set to 0 in clock cycle [0]. At each clock cycle, the least significant bit of the shift register determines whether the 8b ALU 50 will add the multiplicand or zero to the intermediate result. Further, after the addition operation is performed on each cycle, the concatenation of the bits in the carry-out, sum register, and shift register are shifted to the right by one bit. After the eight cycle, all 8 bits of the multiplier may have been evaluated as a 0 or a 1, and all addition operations have completed. The compute engine 38 (FIG. 2) may then write the lower byte contained in the shift register (least significant byte) into the buffer 42. On the next cycle, the compute engine may write the upper byte contained in the sum register (most significant byte) to the buffer 42.

[0040] An example of the multiplication of two 16b operands, producing a 32b result, may be provided in Table 3, showing the long hand format of the multiplication.

						B											1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0
					x	A											0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1
																	1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0
																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
														1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0			
													0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0					
											1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0						
										1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0							
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
								1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0									
							0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										
						1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0											
					1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0												
				1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0													
			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
Result		0	1	0	0	0	1	0	1	0	1	1	1	0	0	0	1	1	1	1	0	1	0	1	0	1	1	0	0	1	1	0
Bit Number:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 3

[0041] In some embodiments, the ALU may operate on byte wide boundaries, and the compute engine may store intermediate results in a buffer 42. For example, an 8b ALU 50 may be bound by one byte (or 8 bits), and may not be able to process more than 8 bits at once. As may be seen in Table 3, when multiplying operands greater than 8 bits, the compute engine 38 may repeat the 8b multiply operation n times, where n is the number of bytes of the multiplier times the number of bytes in the multiplicand. For example, the multiplication of a 16b multiplier by a 16b multiplicand may take 4 (2 bytes times 2 bytes) 8b multiplications. As the 8b ALU 50 is limited by the one byte boundary, the 8b ALU 50 may perform 4 8b multiplication operations, and the intermediate results of these operations may be stored in the buffer 42, or in some other memory component.

[0042] Tables 4-7 below, and the corresponding descriptions, may describe the multiplication of the 16b x 16b operands in more detail. As seen in Table 4 below, the first bytes of the operands (shaded) may be the most significant bytes. The lower bytes of the operands (unshaded) may be the least significant bytes, and the multiplication of the least significant bytes may take eight cycles.

		B	1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0								
	x	A	0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0
																			0	0	1	1	0	1	1	0
																		0	0	0	0	0	0	0	0	
																	0	0	0	0	0	0	0	0		
																0	0	1	1	0	1	1	0			
															0	0	0	0	0	0	0	0				
														0	0	0	0	0	0	0	0					
													0	0	1	1	0	1	1	0						
											0	0	0	1	1	0	1	1	0							
											0	**0**	**1**	**0**	**1**	**0**	**1**	**0**	0	1	1	0	0	1	1	0
Bit Number:											15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 4

[0043] The compute engine 38 may set the sum register to zero in the initial cycle zero, as there are no intermediate results (e.g., partial products in a multiplication operation) to add yet. The compute engine 38 may then multiply the lower byte of the multiplier (operand A) by the lower byte of the multiplicand (operand B). Once the multiplication of the two least significant bytes is complete, the lower byte of the results may be found in the shift register. The most significant byte of this multiplication (bit numbers 8-15, bolded), may be intermediate results stored in the sum register, while the least significant byte of this multiplication (bit numbers 0-7) may be part of the final results, and may be written to the buffer 42.

[0044] The compute engine 38 may then shift or copy the sum registers from the previous step (bolded in Table 4) to the shift register for the second part of the 16b x 16b multiplication operation, illustrated in Table 5.

	B		1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0								
x	A		0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
																			1	0	0	1	0	1	1	1
																		0	0	0	0	0	0	0	0	
																	0	0	0	0	0	0	0	0		
																1	0	0	1	0	1	1	1			
															0	0	0	0	0	0	0	0				
														0	0	0	0	0	0	0	0					
													1	0	0	1	0	1	1	1						
											0	1	0	0	1	0	1	1	1							
											0	**1**	**1**	**1**	**0**	**1**	**1**	**0**	1	0	1	1	1	0	0	1
Bit Number:											15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 5

[0045] The sum register of the first part of the operation in Table 4 may contain a partial product to be added to the second part of the operation in Table 5 (bolded, in the operand A row). As seen in Table 5, the most significant byte of the multiplicand (operand B) and the

least significant byte of the multiplier (operand A), both unshaded, are multiplied in long hand. After eight clock cycles for this multiplication step, the sum register and the shift register contain intermediate results. The sum register (bit numbers 8-15, bolded) is copied to the buffer 42 for use in a later step, and the shift register (bit numbers 0-7) is copied to the sum register for use in the third part of the multiplication process, as shown in Table 6.

	B		1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0								
x	A		0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1	1	0	1	1	1	0	0	1
																			0	0	0	0	0	0	0	0
																		0	0	1	1	0	1	1	0	
																	0	0	0	0	0	0	0	0		
																0	0	1	1	0	1	1	0			
															0	0	1	1	0	1	1	0				
														0	0	1	1	0	1	1	0					
													0	0	0	0	0	0	0	0						
											0	0	0	0	0	0	0	0	0							
											0	**0**	**0**	**0**	**1**	**1**	**0**	**0**	1	1	1	1	0	1	0	1
Bit Number:											15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 6

[0046] In Table 6, the compute engine 38 may perform the 8 cycle multiplication process once the shift register of the previous results have been copied to the sum register. As shown by the shaded boxes in Table 6, the least significant byte of the multiplicand (operand B) is multiplied by the most significant byte of the multiplier (operand A). The compute engine 38 may perform the 8b multiplication, and once the third part of the multiplication operation is complete, the shift register may contain first byte (bolded, bit numbers 8 to 15) of the final results.

[0047] In the fourth step of the 16b x 16b multiplication operation, the compute engine 38 may sum the results in the sum register from the third step (Table 6) with the intermediate results saved in the compute buffer during the second step (Table 5). In some

embodiments, the result of this addition is left in the sum register (bolded, in the operand A row) to be added to the final multiplication. As seen in Table 7 below, the most significant byte of the multiplier (operand A) and the multiplicand (operand B), both unshaded, are multiplied with the partial sum from the previous steps.

	B		1	0	0	1	0	1	1	1	0	0	1	1	0	1	1	0								
x	A		0	0	1	1	1	0	1	0	1	1	0	0	1	0	0	1	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
																			0	0	0	0	0	0	0	0
																		1	0	0	1	0	1	1	1	
																	0	0	0	0	0	0	0	0		
																1	0	0	1	0	1	1	1			
															1	0	0	1	0	1	1	1				
														1	0	0	1	0	1	1	1					
													0	0	0	0	0	0	0	0						
											0	0	0	0	0	0	0	0	0							
											0	**0**	**1**	**0**	**0**	**0**	**1**	**0**	1	0	1	1	1	0	0	0
Bit Number:											15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Table 7

[0048] The results of the multiplication are the final results for the most significant two bytes, and the second most significant byte of the result (bit numbers 0-7) is in the shift register while the most significant byte (bolded, bit numbers 8-15) is in the sum register. The shift register and the sum register are then copied to the buffer 42 with other results. In some embodiments, the final results stored in the buffer 42 may eventually be copied to the memory array 36.

[0049] In one embodiment, a compute engine 38 using the present techniques may also perform division operations using the restoring method of division. The restoring method of division may refer to the relationship of Quotient = Numerator / Denominator, and may operate on fixed-point fractional numbers based on the assumption that $N < D$; $0 < N$; and $D < 1$. During a division operation, the denominator may be copied to the shift register,

and the sum register may be set to zero. On each cycle of the division operation, the compute engine 38 may shift the shift register one bit into the sum register and subtract the numerator from the shifted sum register value. If the results are positive, then the carry-out value of the 1b ALU 62 may be "0," and the compute engine 38 may set the shift register to "1" while the sum register will be the result of the subtraction. If the results are negative, the carry-out value of the 1b ALU 62 may be "1," and the compute engine 38 may set the shift register to "0," and the sum register will be set to the shifted sum value prior to the subtraction.

[0050] The present techniques may also apply to division operations where operands greater than one byte are divided. In such operations, the compute engine 38 may start with the most significant byte (the last byte) of the denominator, and may perform the sequence as if it were only an 8b division operation. The subtraction process in each cycle may take n cycles, and once the 8b division of the most significant byte is complete, the compute engine 38 may store the shift register or the quotient of the operation in the buffer 42 at a designated address. The compute engine 38 may then perform the 8b division operation using the shifted most significant byte, and may continue the process until the least significant byte of the denominator has been divided. Thus, as for the multiplication operation previously discussed, the number of cycles taken to complete a division operation may also depend on the number of bytes in the operands to be divided.

[0051] The 8b ALU 50 may also be used to perform addition operations on operands of any size. When operands are 8 bits or less, the 8b ALU 50 may add operand A to operand B, and the addition operation may take one cycle. The results may be stored in the sum

register and the carry-out register of the 8b ALU 50, and in the following cycle of the next operand, the result may be stored into the buffer 42 at a designated address.

[0052] When an addition operation is to be performed on operands larger than 8 bits, the compute engine 38 may perform the addition one byte at a time, or one byte per clock cycle. When performing the operation with an 8b ALU 50, performing the addition operation one byte at a time may mean adding the operands 8 bits at a time. For cycle 0, the least significant byte may be added, and for cycle 1, the least significant byte + 1 is added, and the carry-out of cycle 0 may be used as the carry-in for cycle 1. After adding each byte, the results may be stored in the buffer 42 at a designated address. The process may be repeated until all bytes of the operands have been added.

[0053] An 8b ALU 50 may also be used to perform subtraction operations on operands of any size. For operands that are 8b or less, the compute engine 38 may subtract operand B from operand A. The subtraction operation may take one cycle, and the results may be stored in the sum registers and the carry-out register. The compute engine 38 may store the results to the buffer 42 at a destination address in a next cycle of the 8b ALU 50.

[0054] When a subtraction operand is to be performed on operands large than 8 bits, the compute engine 38 may perform the subtraction operation one byte at a time, or one byte per clock cycle. For cycle 0, the least significant byte may be subtracted, and for cycle 1, the least significant byte + 1 is subtracted, and the carry-out of cycle 0 may be used as the carry-in for cycle 1. After subtracting each byte, the results may be stored in the buffer 42 at a designated address. The process may be repeated until all bytes of the operands have been subtracted.

[0055] While the present disclosure provides examples of mathematical operations executed by an 8b ALU 50, an ALU of a different size may also be used. An ALU in may be composed of building blocks (e.g., adders, or 1b ALUs) which may enable the ALU to perform logic or mathematical operations on operands of any size. For operands that have a byte width greater than the size of the ALU, the compute engine 38 may operate on a byte of each operand at a time and store the results of each cycle of the operation in the compute buffer.

[0056] Furthermore, while the present disclosure provides examples of mathematical operations such as multiplication, division, addition, and subtraction, other operations may also be performed by the compute engine 38. The possible operations which may be performed by each 8b ALU 50, or by the compute engine, are not limited by the gates or structure of each 1b ALU 62. For example, there may be no explicit NAND functionality in the 1b ALU of FIG. 5. However, if the compute engine 38 is instructed to perform A NAND B, operands A and B may first be cycled through the 1b ALU as A AND B, and the results of that may be carried back in, and INV B may be selected on the second cycle to obtain results for A NAND B.

[0057] While the invention may be susceptible to various modifications and alternative forms, specific embodiments have been shown by way of example in the drawings and have been described in detail herein. However, it should be understood that the invention is not intended to be limited to the particular forms disclosed. Rather, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the following appended claims.

CLAIMS

1. A system comprising:

a memory device comprising:

a plurality of internal processors configured to execute instructions on data in parallel; and

a memory component configured to hold the data, intermediate results of the instructions on the data, and final results of the executed instructions on the data.

2. The system, as set forth in claim 1, comprising an external processor configured to communicate with the memory device to provide the instructions to be executed.

3. The system, as set forth in claim 1, wherein the memory device comprises a sequencer configured to sequence the instructions to be executed.

4. The system, as set forth in claim 1, wherein each of the plurality of internal processors are grouped into a larger internal processor with a fixed byte width, and wherein the memory device comprises one or more of the larger internal processors.

5. The system, as set forth in claim 4, wherein each of the plurality of internal processors is 1 bit wide, and the larger internal processor is 8 bits wide.

6. The system, as set forth in claim 4, wherein the larger internal processor is configured to execute the instructions on a data set of any size, and wherein each of the plurality of internal processors grouped in the larger internal processor is configured to execute the instructions on the data set in one or more cycles.

7. The system, as set forth in claim 1, wherein the memory component comprises one or more of:

a buffer configured to hold one or more of the data, the intermediate results, and the final results;

a memory array configured to store one or more of the data and the final results; and

a memory register configured to hold the intermediate results.

8. The system, as set forth in claim 1, wherein the plurality of internal processors is embedded on the memory device and configured to write and read from the memory component.

9. An internal processor comprising:

a plurality of arithmetic logic units (ALUs), wherein each of the plurality of ALUs comprises a plurality of building blocks;

a plurality of input multiplexers configured to select data to input into each of the plurality of building blocks; and

a shift unit coupled to each of the plurality of ALUs, wherein the shift unit is configured to shift an output from one of the plurality of building blocks to input into another one of the plurality of building blocks.

10. The internal processor, as set forth in claim 9, wherein each of the plurality of building blocks are one bit wide.

11. The internal processor, as set forth in claim 9, wherein each of the plurality of ALUs are configured to operate on one byte of the data selected by a corresponding one of the plurality of input multiplexers.

12. The internal processor, as set forth in claim 9, wherein each of the plurality of ALUs are configured to perform an operation on the data, and configured to process the data through one or more cycles until the operation is performed.

13. The internal processor, as set forth in claim 9, wherein the plurality of input multiplexers are configured to select the data to input from one or more of a constant register, a bank from a memory component coupled to the internal processor, or from an output of one of the plurality of building blocks.

14. The internal processor, as set forth in claim 9, wherein the internal processor is packaged with a memory device and configured to be controlled by a sequencer in the memory device.

15. A method of parallel processing by an internal processor, comprising:

selecting data from a memory component comprising intermediate results, wherein the intermediate results comprise results from a previous cycle of an instruction being executed by the internal processor;

inputting the selected data set to a unit in the internal processor; and

executing an instruction on the data in the unit based on a width of the unit and a size of the data.

16. The method, as set forth in claim 15, wherein the internal processor is embedded on a memory device, and wherein selecting the data is controlled by a sequencer of the memory device.

17. The method, as set forth in claim 15, wherein inputting the selected data to the unit comprises providing the selected data through a multiplexer coupled to the unit in the internal processor.

18. The method, as set forth in claim 15, wherein the unit comprises a plurality of 1 bit ALUs.

19. The method, as set forth in claim 15, wherein executing the instruction on the data comprises performing a first cycle of an operation on a first byte of the data and performing

a next cycle of the operation on a next byte of the data until the instruction is executed on the data.

20. The method, as set forth in claim 19, wherein executing the instruction comprises storing one or more of a result from the first cycle of the operation or a result from the next cycle of the operation as intermediate results.

21. The method, as set forth in claim 19, wherein one or more of a result from the first cycle of the operation and a result from the next cycle of the operation are stored as intermediate results in a register.

22. The method, as set forth in claim 21, wherein the register comprises different parts, and the intermediate results are stored based on the size of the data and a type of instruction being executed by the internal processor.

23. The method, as set forth in claim 15, wherein executing the instruction is based on a type of operation being executed by the internal processor.

24. The method, as set forth in claim 15, wherein executing the instruction is based on the size of the data divided by the width of the ALU.

25. An internal processor comprising:

a plurality of logic units;

a plurality of multiplexers, wherein each multiplexer is coupled to a respective logic unit of the plurality of logic units and configured to input data to the respective logic unit; and

a shift unit configured to shift data from one of the plurality of logic units to another one of the plurality of logic units.

26. The internal processor, as set forth in claim 25, wherein the plurality of logic units each comprise logic gates, and is configured to perform logical operations and mathematical operations.

27. The internal processor, as set forth in claim 25, wherein the data is stored in one or more of a constant register, a bank from a memory component coupled to the internal processor, or an output of one of the plurality of logic units.

28. The internal processor, as set forth in claim 25, wherein the data comprises the data shifted by the shift unit.

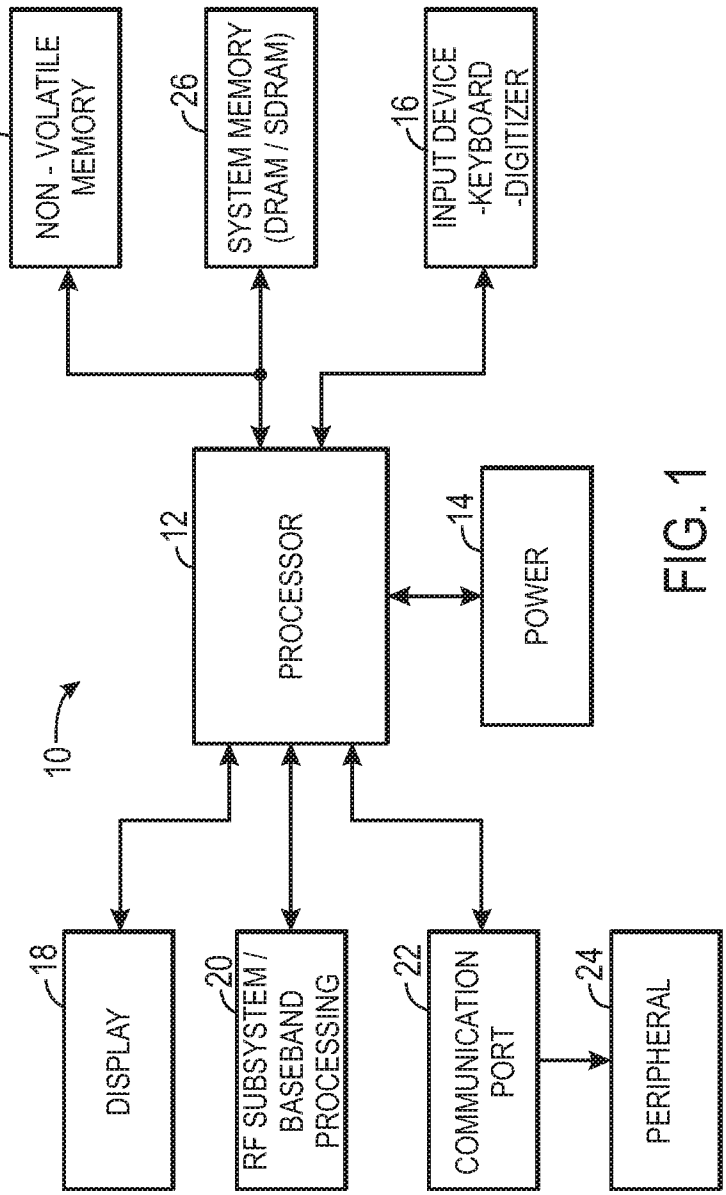
28
NON - VOLATILE MEMORY
26
SYSTEM MEMORY (DRAM / SDRAM)
16
INPUT DEVICE -KEYBOARD -DIGITIZER
12
PROCESSOR
14
POWER
10
18
DISPLAY
20
RF SUBSYSTEM / BASEBAND PROCESSING
22
COMMUNICATION PORT
24
PERIPHERAL

FIG. 1

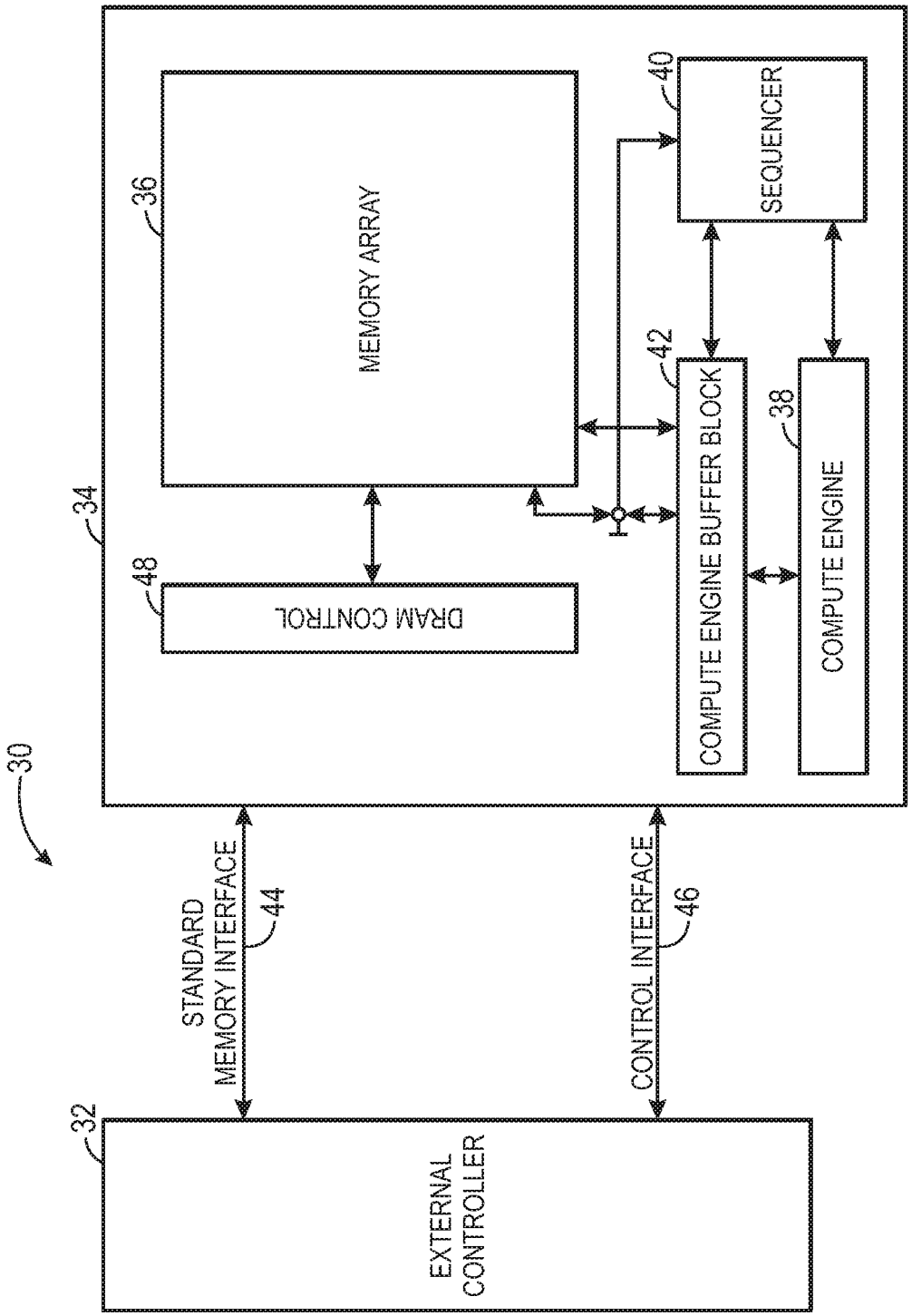
30
34
36
MEMORY ARRAY
40
SEQUENCER
42
COMPUTE ENGINE BUFFER BLOCK
38
COMPUTE ENGINE
48
DRAM CONTROL
STANDARD MEMORY INTERFACE
44
CONTROL INTERFACE
46
32
EXTERNAL CONTROLLER

FIG. 2

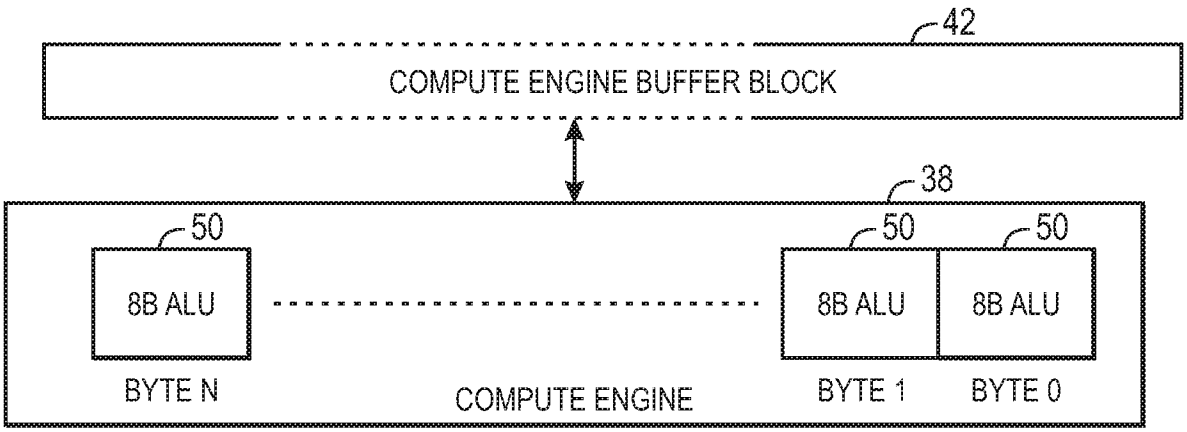
42
COMPUTE ENGINE BUFFER BLOCK
38
50
50
50
8B ALU
8B ALU
8B ALU
BYTE N
COMPUTE ENGINE
BYTE 1
BYTE 0

FIG. 3

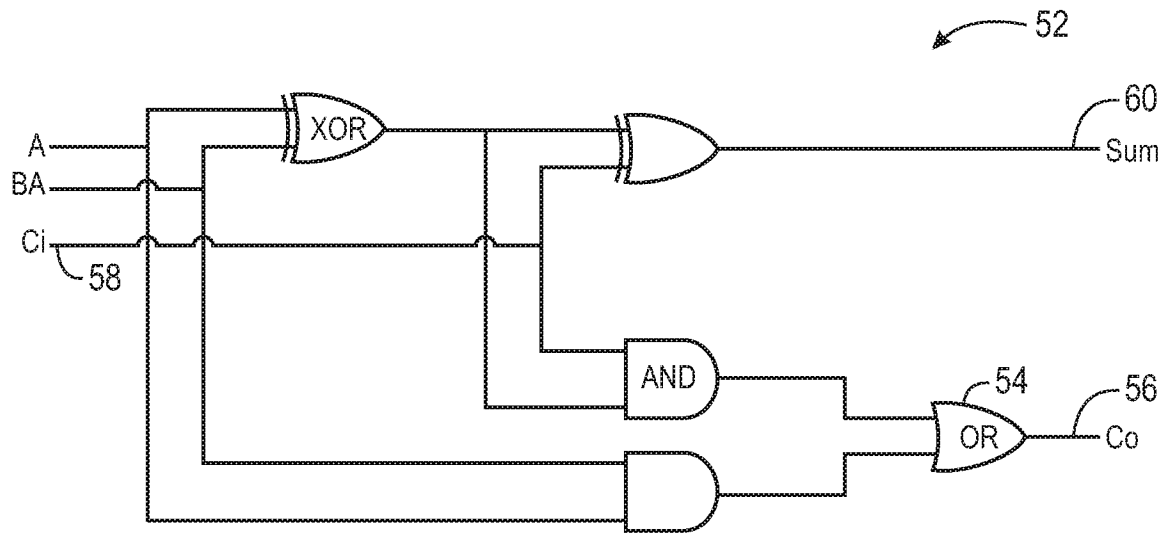
52
A
BA
Ci
58
XOR
60
Sum
AND
54
OR
56
Co

FIG. 4

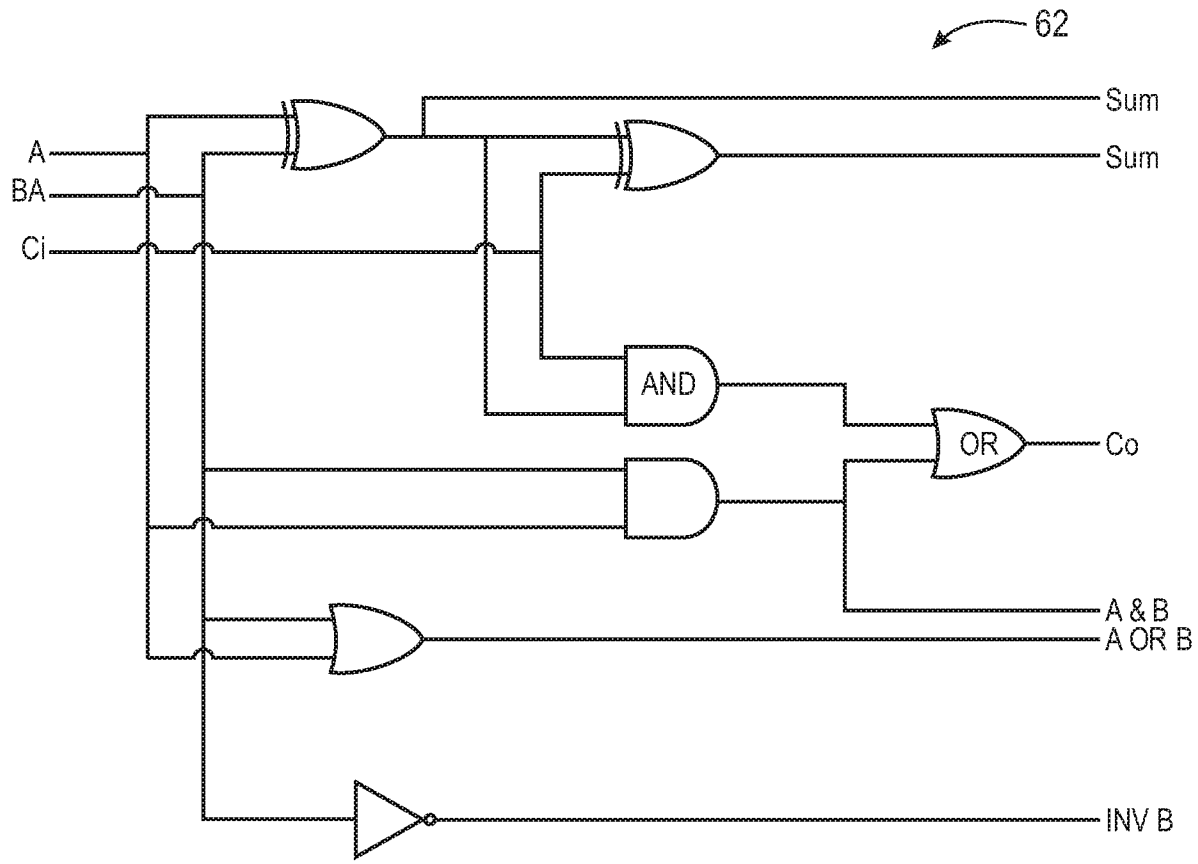
62
A
BA
Ci
Sum
Sum
AND
OR
Co
A & B
A OR B
INV B

FIG. 5

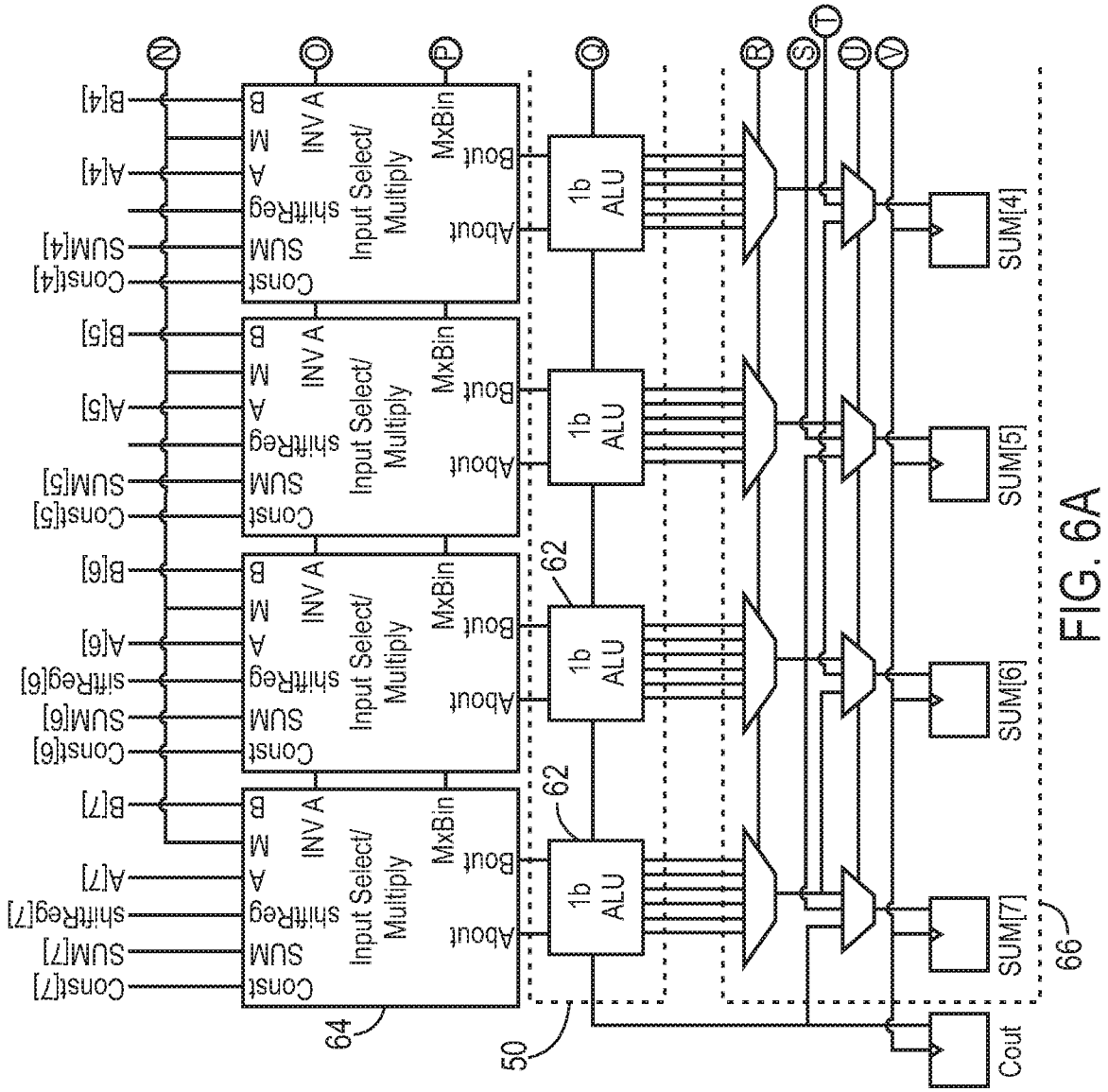
38
N
O
P
Q
R
S
T
U
V
B[4]
A[4]
SUM[4]
Const[4]
B[5]
A[5]
SUM[5]
Const[5]
B[6]
A[6]
shiftReg[6]
SUM[6]
Const[6]
B[7]
A[7]
shiftReg[7]
SUM[7]
Const[7]
INV A
B
M
A
shiftReg
SUM
Const
Input Select/
Multiply
MxBin
Bout
About
1b
ALU
62
64
50
SUM[4]
SUM[5]
SUM[6]
SUM[7]
66
Cout

FIG. 6A

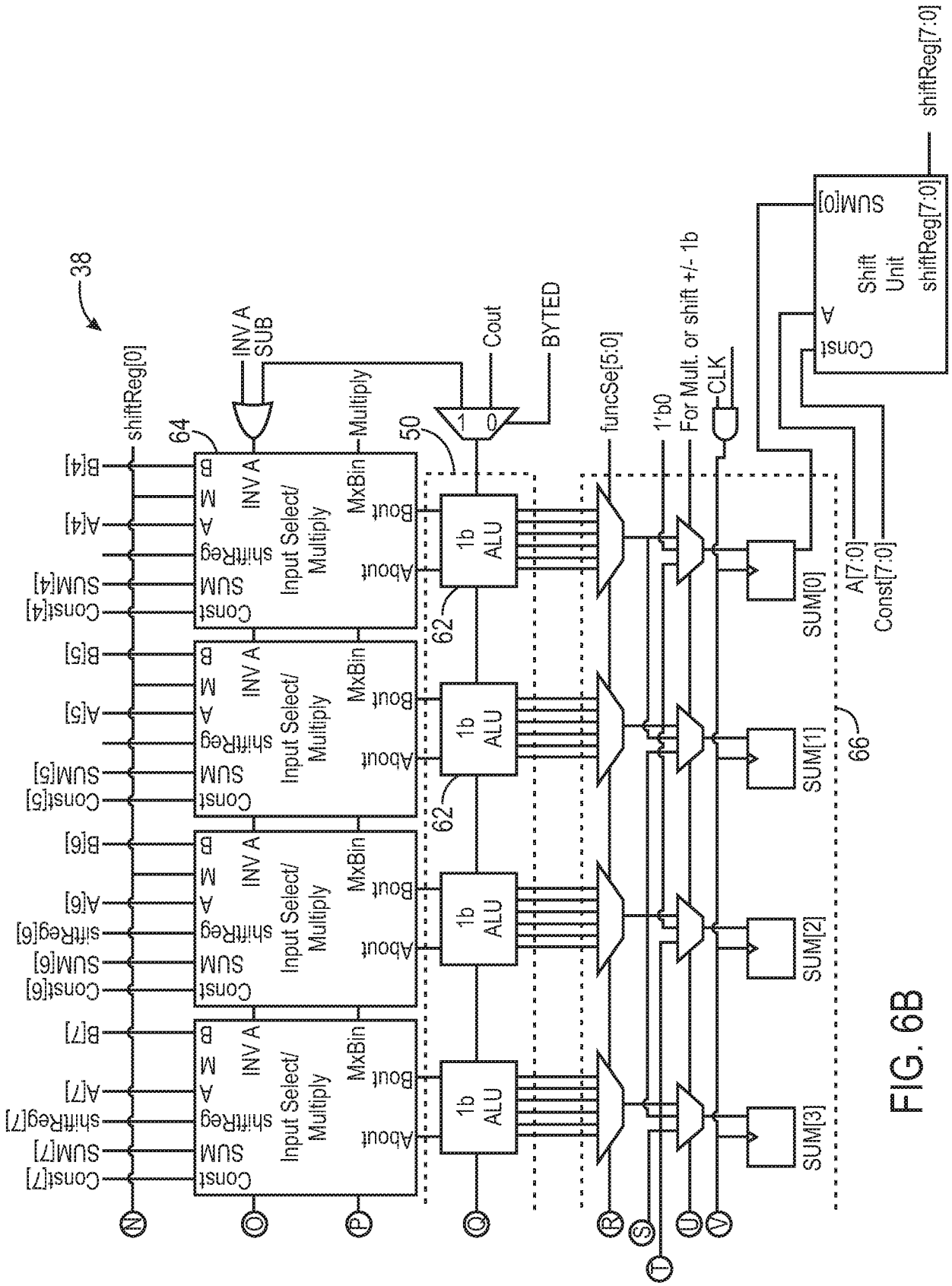
38
shiftReg[0]
INV A
SUB
Multiply
Cout
BYTED
funcSe[5:0]
1'b0
For Mult. or shift +/- 1b
CLK
shiftReg[7:0]
SUM[0]
Shift Unit
Const
A
shiftReg[7:0]
64
50
62
66
B[4]
A[4]
Const[4]
SUM[4]
B[5]
A[5]
Const[5]
SUM[5]
B[6]
A[6]
shiftReg[6]
Const[6]
SUM[6]
B[7]
A[7]
shiftReg[7]
Const[7]
SUM[7]
INV A
B
M
A
shiftReg
SUM
Const
Input Select/ Multiply
MxBin
Bout
Aout
1b ALU
SUM[0]
SUM[1]
SUM[2]
SUM[3]
A[7:0]
Const[7:0]
N
O
P
Q
R
S
T
U
V

FIG. 6B

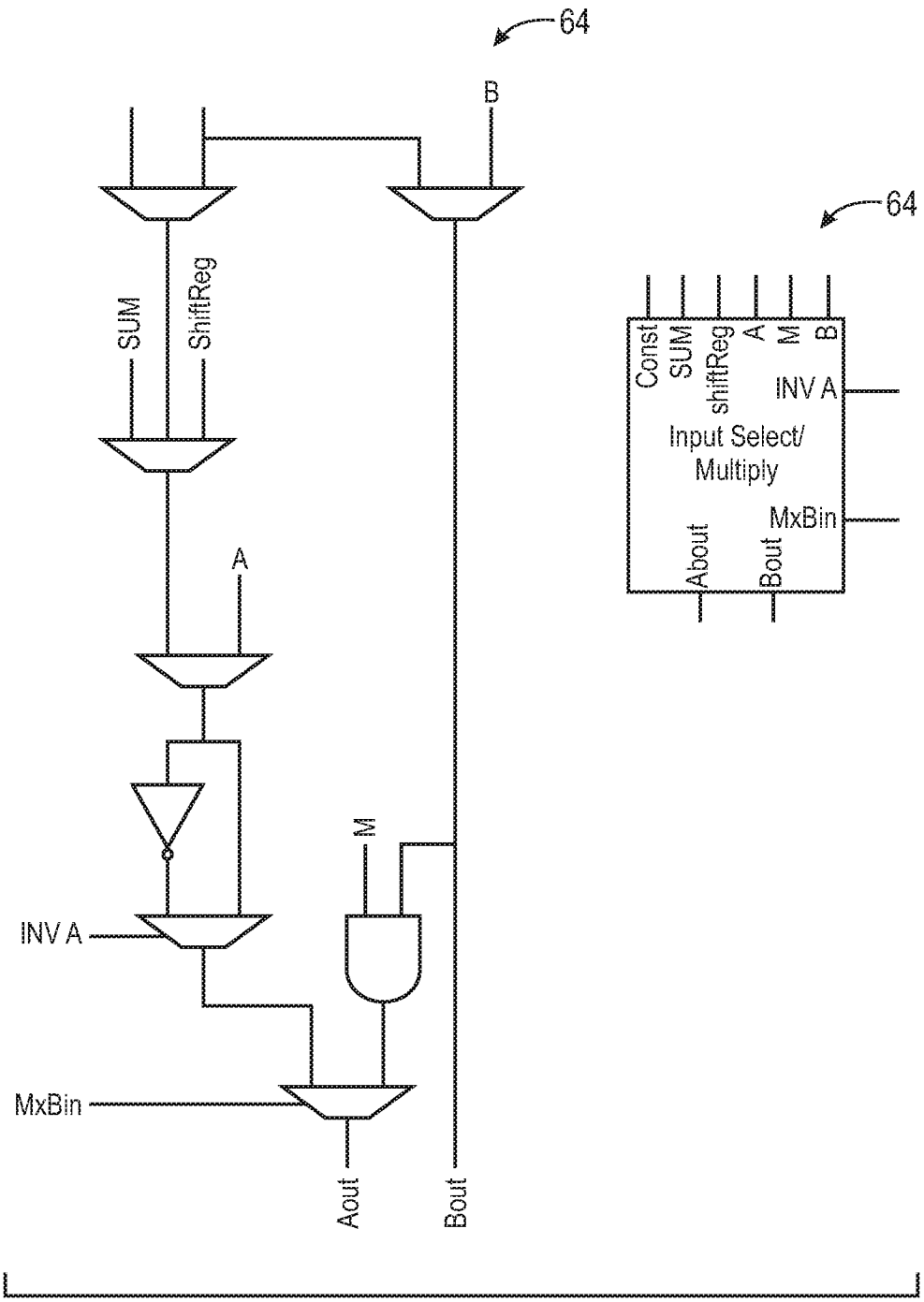
64
B
SUM
ShiftReg
A
INV A
M
MxBin
Aout
Bout
64
Const
SUM
shiftReg
A
M
B
INV A
Input Select/
Multiply
MxBin
Aout
Bout

FIG. 7

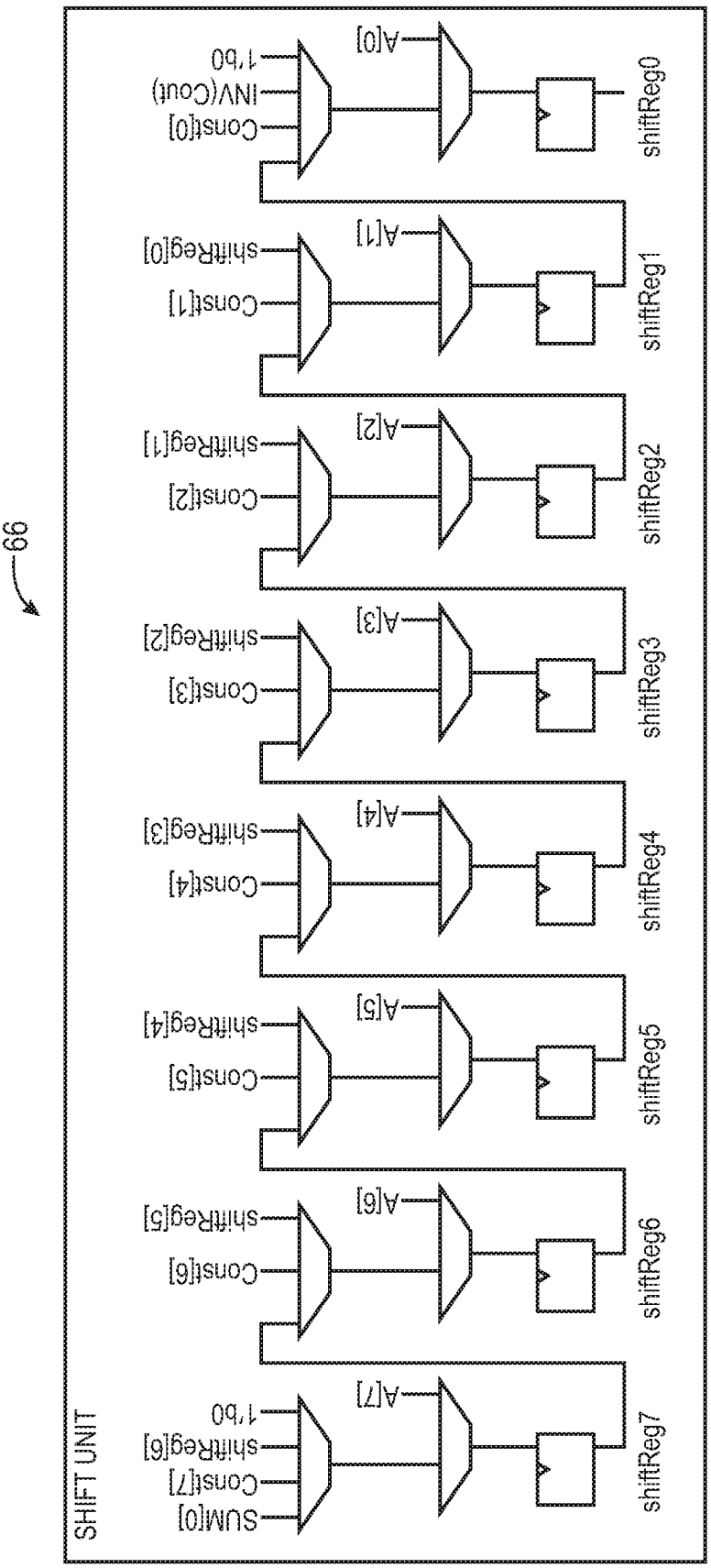
99
SHIFT UNIT
SUM[0]
Const[7]
shiftReg[6]
1'b0
A[7]
shiftReg7
Const[6]
shiftReg[5]
A[6]
shiftReg6
Const[5]
shiftReg[4]
A[5]
shiftReg5
Const[4]
shiftReg[3]
A[4]
shiftReg4
Const[3]
shiftReg[2]
A[3]
shiftReg3
Const[2]
shiftReg[1]
A[2]
shiftReg2
Const[1]
shiftReg[0]
A[1]
shiftReg1
Const[0]
INV(Cout)
1'b0
A[0]
shiftReg0

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No
PCT/US2010/035450

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F15/78
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	GOKHALE M ET AL: "PROCESSING IN MEMORY: THE TERASYS MASSIVELY PARALLEL PIM ARRAY" COMPUTER, IEEE SERVICE CENTER, LOS ALAMITOS, CA, US LNKD-DOI:10.1109/2.375174, vol. 28, no. 4, 1 April 1995 (1995-04-01), pages 23-31, XP000507857 ISSN: 0018-9162 figures 1-3 page 24, right-hand column, line 34 - line 39 page 24, right-hand column, line 49 - line 51 page 24, right-hand column, line 48 - line 58 page 25, right-hand column, line 5 - page 26, left-hand column, line 17 page 25, left-hand column, line 7 - line 12 -/--	1-28

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier document but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 August 2010	06/09/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Bosch Vivancos, P

Form PCT/ISA/210 (second sheet) (April 2005)

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
	page 25, left-hand column, line 1 - line 7 page 26, left-hand column, line 19 - line 36 -----	
X	DRAPER J ET AL: "Implementation of a 32-bit RISC processor for the data-intensive architecture processing-in-memory chip" APPLICATION-SPECIFIC SYSTEMS, ARCHITECTURES AND PROCESSORS, 2002. PROC EEDINGS. THE IEEE INTERNATIONAL CONFERENCE ON 17-19 JULY 2002, PISCATAWAY, NJ, USA,IEEE, 17 July 2002 (2002-07-17), pages 163-172, XP010601469 ISBN: 978-0-7695-1712-4 figure 2a figure 3 -----	1-28
X	NYASULU P M ET AL: "Minimizing the effect of the host bus on the performance of a computational RAM logic-in-memory parallel-processing system" CUSTOM INTEGRATED CIRCUITS, 1999. PROCEEDINGS OF THE IEEE 1999 SAN DIEGO, CA, USA 16-19 MAY 1999, PISCATAWAY, NJ, USA,IEEE, US LNKD-DOI:10.1109/CICC.1999.777360, 16 May 1999 (1999-05-16), pages 631-634, XP010340698 ISBN: 978-0-7803-5443-2 figure 1a figure 1b figures 2-4 page 631, left-hand column, line 15 - line 30 page 631, left-hand column, line 17 - line 30 ----- -/--	1-21,25, 27,28

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	YAMASHITA N ET AL: "A 3.84 GIPS INTEGRATED MEMORY ARRAY PROCESSOR WITH 64 PROCESSING ELEMENTS AND A 2-MB SRAM" IEEE JOURNAL OF SOLID-STATE CIRCUITS, IEEE SERVICE CENTER, PISCATAWAY, NJ, US LNKD-DOI:10.1109/4.328633, vol. 29, no. 11, 1 November 1994 (1994-11-01), pages 1336-1342, XP000483119 ISSN: 0018-9200 figure 1 figure 4 figure 6 figure 7c page 1339, left-hand column, line 21 - line 40 page 1336, right-hand column, line 1 - line 5 page 1336, right-hand column, line 1 - line 10 page 1337, right-hand column, line 1 - line 25	1-28