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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 September 2010 (10.09.2010)

PCT

(10) International Publication Number
WO 2010/101944 A1

(51) International Patent Classification:
H03M 13/00 (2006.01)

(21) International Application Number:
PCT/US2010/025956

(22) International Filing Date:
2 March 2010 (02.03.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/156,792 2 March 2009 (02.03.2009) US
61/293,999 11 January 2010 (11.01.2010) US

(72) Inventors; and
(71) Applicants : **REYNOLDS, David** [US/US]; 10 Starbird Road, Scarborough, Maine 04074 (US). **VIGODA, Benjamin** [US/US]; 44 Lloyd Street, Winchester, MA 01890 (US).

(74) Agent: **ROHLICEK, J., Robin**; Occhiuti Rohlicek & Tsao LLP, 10 Fawcett Street, Cambridge, MA 02138 (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, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (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).

Published:

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

WO 2010/101944 A1

(54) Title: BELIEF PROPAGATION PROCESSOR

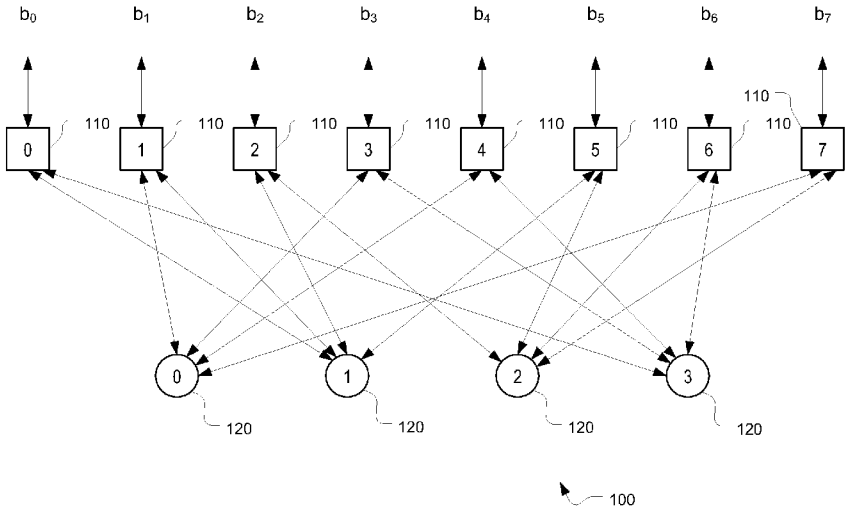

FIG. 1

(57) Abstract: A processor includes a first memory module for storing a first set of storage values each representing a respective input, and a second memory module for storing a second set of storage values in analog form. An analog module is coupled to the first and the second memory modules. The analog module is configured to, in each operation cycle of at least one iteration, update at least some of the second set of storage values based on the first and the second sets of storage values. An output module is for generating a set of outputs from at least some of the second set of storage values.

BELIEF PROPAGATION PROCESSOR

Cross-Reference to Related Applications

[001] This application claims the benefit of U.S. Provisional Application No. 61/156,792, titled "Belief Propagation Processor," filed March 2, 2009, and U.S. Provisional Application No. 61/293,999, filed January 11, 2010, which are incorporated herein by reference.

[002] This application is related to U.S. Provisional Patent Application Serial No. 61/156,794, titled "Circuits for Soft Logical Functions," filed March 2, 2009, and U.S. Provisional Patent Application Serial No. 61/156,721, titled "Signal Mapping", filed March 2, 2009, and U.S. Provisional Patent Application Serial No. 61/156,735, titled "Circuits for Soft Logical Functions," filed March 2, 2009. This application is also related to U.S. Application Serial No. 12/537,060, titled "Storage Devices with Soft Processing," filed August 6, 2009. The contents of the above applications are incorporated herein by reference.

Background

[003] This document relates to an analog belief propagation processor.

[004] "Belief Propagation" (BP) is an efficient approach to solving statistical inference problems. The approach exploits underlying structure of a network of stochastic elements and its constraints and Bayesian laws of probabilities to find the most optimal set of valid outputs that satisfy constrains and network structure requirements.

[005] Belief Propagation includes a class of techniques for performing statistical inference using a system model that is in the form of a graph. The term "graph" here refers to the mathematical definition of a graph, which represents the connectedness of a set of abstract objects. The objects are often referred to as "nodes" and the

connections between objects are often referred to as "edges." One common type of graph used in such models is referred to as a "factor graph." In a factor graph (a particular style of factor graph called a "Forney factor graph") the nodes represents statistical relationships between values, which are represented as edges. Other types of graphs, such as Bayesian networks, and Markov random fields are also commonly used for statistical inference.

[006] Examples of Belief Propagation approaches operate by passing messages between nodes in the graph, where each message represents a summary of the information known by that node through its connections to other nodes. Such approaches are known by various names, including belief propagation, probability propagation, message passing, and summary-product algorithms, among others. Particular forms of these approaches include sum-product, max-product, and min-sum.

[007] A large variety of approaches to coding, signal processing, and artificial intelligence may be viewed as instances of the summary-product approach (or belief/probability propagation approach), which operates by message passing in a graphical model. Specific instances of such approaches include Kalman filtering and smoothing, the forward backward algorithm for hidden Markov models, probability propagation in Bayesian networks, and decoding algorithms for error correcting codes such as the Viterbi algorithm, the BCJR algorithm, and the iterative decoding of turbo codes, low-density parity check codes, and similar codes.

[008] Graphs on which belief propagation may operate include two types: graphs with loops (cyclic graphs) and graphs with no loops (acyclic graphs). Graphs with no loops are also known as "trees." Belief propagation procedures differ fundamentally between these two types of graphs. For a tree, belief propagation approach can proceed in a well-defined order with a well-defined number of steps to compute the result. And assuming ideal computation, this result is always known to be correct. For

a graph with loops, on the other hand, belief propagation approaches are generally iterative, meaning the same set of computations must be repeated successively until a result is reached. In this case, the computation typically converges to a useful result, but does not always do so. In some cases, the computation may not converge to a single result, or if it does, the result in some cases is inaccurate. For a cyclic graph, the performance of belief propagation can depend on the order in which the computations are performed, which is known as the message passing 'schedule.'

[009] In one particular application mentioned above, Belief Propagation has been adopted as an efficient method of implementing decoders for various forward error correcting codes. In this case BP uses structure of the code and constraints to infer the correct valid codeword from the input codeword that contains noise, for instance, with each element (e.g., bit) of the input codeword being represented as a distribution rather than a discrete value. In some implementations of Belief Propagation for forward error correction a Digital Signal Processor is used to perform various arithmetic computations required by the algorithm with all the statistical data being processed in digital format.

[010] Observing the fact that "soft"- probabilistic data is continuous in nature, i.e., represented by real values in a finite interval, it is possible to implement belief propagation algorithm using analog electrical circuits. Since only one signal is associated with the unit of statistical data rather than multiple signals for different digits (e.g., binary digits, bits) of the digital signal representing the same data, the savings in hardware and power dissipation can be very significant.

[011] Several architectures had been proposed that utilize analog circuits to perform efficient decoding of various codes, including convolutional codes, Low Density Parity Check Codes (LDPC) or linear block codes. These include analog implementations that use a so-called full flat architecture, where each input data symbol is associated with dedicated computing element.

Summary

[012] In one aspect, in general, an analog processor has a first memory module and a second memory module. The first memory module is for storing a first set of storage values in respective storage elements each representing a respective input to the processor. The second memory module is for storing a second set of storage values in analog form in respective storage elements. The second set of storage values includes intermediate values determined during operation of the processor. The analog processor also includes an analog computation module coupled to the first and the second memory modules. This processor is configurable such that in each of a set of operation cycles the analog module determines values for at least some of the second set of storage values based on at least some of the first and the second sets of storage values. An output module is use for generating a set of outputs from at least some of the second set of storage values.

[013] Aspects may include or more of the following features.

[014] The first storage module is configured to store the first set of storage values in analog form.

[015] The analog computation module is linked to the first and the second memory modules via analog signal paths. For example, the analog signal paths are each configured to carry a value on a conductor represented as at least one of a voltage and a current proportional to the value.

[016] The analog module is configurable to determine values for a different subset of the second set of storage values in each of a plurality of operation cycles.

[017] The processor includes input selection circuitry configurable to couple the analog computation module to outputs of selected memory elements of the first and the second memory modules.

[018] The processor further includes, for each analog computation module, a plurality of signal busses, each bus providing an input value to the analog computation module and being switchably coupled to a plurality of the storage elements of the second memory module.

[019] The storage elements are coupled to switchably provide a current representation of a storage value stored in the storage element such that the input value provided to the analog computation module is represented as a current that is substantially proportional to a sum of the currents representations provided by the storage elements.

[020] The processor further include output section circuitry configurable to accept outputs of selected memory elements of the first and the second memory modules, and to determine outputs of the analog processor.

[021] The processor includes multiple analog computation modules being concurrently operable to determines values for different subsets of the second set of storage values in each operation cycle.

[022] The second memory module includes a plurality of section, each associated with a corresponding different one of the analog computation modules for storing values determined by the associated computation module.

[023] The second memory module is configured such that in a single operation cycle, each storage element can provide a storage value to one or more of the analog computation modules and can accept a determined value to storage in the storage element for providing in a subsequent operation cycle.

[024] Each storage element is associated with two storage locations such that in any one cycle, one storage location is used for accepting a determined value and one storage location is used for providing a value.

[025] Tthe second memory module includes multiple memory sections. Groups of the sections form banks, wherein for each of the analog computation modules each of a set of inputs to the module is associated with a different bank of the memory sections.

[026] The processor is configurable such that in each cycle, each memory section includes memory elements that either provide values to one ore more analog computation modules or memory elements that are updated with values from one or more analog computation modules.

[027] The processor is configured to implement a belief propagation computation.

[028] The processor of claim is configured to implement a factor graph computation.

[029] The analog computation module is configured to accept and provide analog signals that are substantially logarithmic representations of at least one of probabilities, likelihoods, and likelihood ratios.

[030] The processor is configured to implement a decoder for a low density parity check (LDPC) code.

[031] The processor further includes a controller configured to control operation of the processor to perform a set of iterations of computation, each iteration comprising a set of computation cycles.

[032] The set of computation cycles is substantially the same in each iteration, each cycle being associated with a configuration of the first and the second storage modules to provide inputs and output to one or more analog computation modules.

[033] The processor is configured and/or configurable to implement a decoder parity check code, and each cycle is associated with one or more parity check constraints, and wherein the cycles of each iteration are together associated with all the parity check constraints of the code.

[034] The analog computation module implements a network of analog processing elements.

[035] The analog processing elements include elements that represent soft logical operations. For example, the soft logical operations include soft XOR operations.

[036] The network of elements is acyclic.

[037] The network of elements includes at least one cycle of elements, the analog computation module being configured to implement an relaxation computation.

[038] The analog computation module includes inputs for configuring one or more gain characteristics in the network of processing elements.

[039] In another aspect, in general, a decoder includes a first memory for storing code data having a length in bits, and a second memory for storing intermediate data in analog form. The decoder includes an analog decoder core coupled to the first memory and to the second memory. The decoder core has an input length less than the length of the code data and an output length less than a number of constraints represented in the code data. The decoder further includes a controller for, in each of a set of cycles, coupling the inputs of the decoder code to selected values from the first and the second memories, and coupling outputs of the decoder core for storage in the second memory. An output section of the decoder is coupled to the second memory for providing decoded data based on values stored in the second memory.

[040] In another aspect, in general, a method is used for forming a data representation of an analog processor. The method includes forming: a data representation of a first memory module for storing a first set of storage values in respective storage elements each representing a respective input to the processor; a data representation of a second memory module for storing a second set of storage values in analog form in respective storage elements, the second set of storage values including intermediate values determined during operation of the processor; a data

representation of an analog computation module coupled to the first and the second memory modules, the processor being configurable such that in each of a set of operation cycles the analog module determines values for at least some of the second set of storage values based on at least some of the first and the second sets of storage values; and a data representation of an output module for generating a set of outputs from at least some of the second set of storage values.

[041] In some examples, forming the data representations includes forming Verilog representations of the processor.

[042] The method can further include fabricating a integrated circuit implementation of the analog processor according to the formed data representation.

[043] In some examples, the method further includes accepting a specification of a parity check code and forming the data representations to represent an implementation of a decoder for the code.

[044] In another aspect, in general, software stored on a computer readable medium includes instructions for and/or data imparting functionality when employed in a computer component of an apparatus for forming an integrated circuit implementation of any of the analog processor described above.

[045] In another aspect, in general, decoding method includes, in each of a series of cycles of a decoding operation, applying a portion of code data and a portion of an intermediate value data to an analog decoder core, and storing an output of the decoder coder in an analog storage for the intermediate data. Data, including intermediate value data from the analog storage, are combined to form decoded data representing an error correction of the code data.

[046] In some examples, each of the series of cycles is associated with a corresponding subset of less that all of a plurality of parity-check constraints of the

code. The intermediate value data may include values each associated with a different one of the parity check constraints of the code.

[047] In another aspect, in general, a processor includes a first memory module for storing a first set of storage values each representing a respective input, and a second memory module for storing a second set of storage values in analog form. An analog module is coupled to the first and the second memory modules. The analog module is configured to, in each operation cycle of at least one iteration, update at least some of the second set of storage values based on the first and the second sets of storage values. An output module is for generating a set of outputs from at least some of the second set of storage values.

[048] The analog module may be configured for updating a different subset of the second set of storage values in each of at least two operations cycles of an iteration.

[049] The analog module may include a set of distributed components each configured to update a different subset of the second set of storage values using a different subset of the first set of storage values and the second set of storage values.

[050] In another aspect, in general, a decoder includes a first memory for storing code data having a length in bits, and a second memory for storing intermediate data in analog form. An analog decoder core is coupled to the first memory and to the second memory, the decoder core having an input length less than the length of the code data and an output length less than a number of constraints represented in the code data. A controller in the decoder is for, in each of a plurality of cycles, coupling the inputs of the decoder code to selected values from the first and the second memories, and coupling outputs of the decoder core for storage in the second memory. An output section is coupled to the second memory for providing decoded data based on values stored in the second memory.

[051] In another aspect, in general, a decoding method includes, in each of a number of cycles of a decoding operation, applying a portion of code data and a portion of an intermediate value data to an analog decoder core, and storing an output of the decoder coder in an analog storage for the intermediate data. Data, including intermediate value data from the analog storage, is then combined to form decoded data representing an error correction of the code data.

[052] Advantages of one or more aspects may include the following:

[053] Use of analog computations and/or analog storage of intermediate values provides lower power and/or smaller circuit area implementations as compared to a digital implementations, for instance in applications of iterative decoding or error correcting codes.

[054] Iterative use of one or more analog computation cores provides lower power and/or smaller circuit area as compared to fully parallel relaxation implementations of similar decoding algorithms. In some examples, a partially relaxation implementation in which parts of a computation are implemented in relaxation from in each of a succession of cycles may also provide similar advantages over a fully parallel relation implementation.

[055] Approaches are applicable to decoding of block codes without requiring that the size and/or power requirements of an implementation grow substantially with the length of the code.

[056] Other features and advantages of the invention are apparent from the following description, and from the claims.

Description of Drawings

[057] FIG. 1 is an example factor graph for a length 8 LDPC code;

[058] FIG. 2A is a diagram that illustrates transformation of a variable node with bidirectional links to a set of variable nodes with directed links, and FIG. 2B is a diagram that illustrates a similar transformation for a constraint node;

[059] FIG. 3 is a portion of the graph shown in FIG. 1;

[060] FIG. 4 is a portion of a directed graph corresponding to the portion of the bidirectional graph shown in FIG. 3;

[061] FIG. 5 is a diagram illustrating a module implementation corresponding to the portion of the graph shown in FIG. 4;

[062] FIG. 6 is a diagram illustrating output calculation;

[063] FIG. 7A is a diagram that shows a relationship between input and outputs or a module, and FIG. 7B illustrates the corresponding code matrix;

[064] FIG. 8 is a diagram of an implementation of a decoder for a length 8 LDPC code using a shared module;

[065] FIG. 9 is a table that specifies inputs and outputs for the shared module shown in FIG. 8;

[066] FIG. 10 is a block diagram of a decoder with two shared modules;

[067] FIG. 11 is a tabular representation of a parity matrix for a (1056, 352) LDPC code;

[068] FIG. 12 is a diagram of a shared module for use with the code shown in FIG. 11

[069] FIG. 13 is a block diagram of a decoder for a (1056, 352) LDPC code with eight shared modules (of which two are illustrated);

[070] FIG. 14 is a circuit implementation of a variable node;

[071] FIG. 15A is a circuit implementation of a constraint node;

[072] FIG. 15B is an alternative implementation of a constraint node;

[073] FIG. 16 is a diagram that illustrates a distributed bus implementation of a variable node;

[074] FIG. 17 is a diagram of an alternative shared module;

[075] FIG. 18 is a block diagram of a decoder that uses distributed bus implementations of variable nodes;

[076] FIG. 19 is a block diagram of an alternative architecture for a decoder for a (1056, 352) LDPC code;

[077] FIG. 20 is a block diagram that shows multiple update modules in the architecture shown in FIG. 17; and

[078] FIGS. 21A-B are a block diagram of an implementation of a decoder.

Description

[079] Referring to FIG. 1, in one example of an analog-based implementation of a belief propagation processor, a decoder for a Low Density Parity Check (LDPC) code is based on a factor graph 100 in which one variable node 110 is associated with each different input bit (b_j), and one check (constraint) node 120 is associated with each constraint. In FIG. 1, an example with eight input bits with four checks (constraints) on the input bits is shown. The code can be represented in matrix form in which each column is associated with a different input bit, and each row is associated with a different check or constraint. An (i, j) entry is 1 if the j^{th} input is used in the i^{th}

constraint and 0 otherwise. In the LDPC example, the constraint is that the XOR of the inputs for a constraint is 0. This example length 8 LPDC code can be represented according to the following check matrix (note that the rows are dependent modulo 2 in this illustrative example, which is not necessarily true in general):

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

[080] In FIG. 1, each edge is bidirectional. Referring to FIGS. 2A-B, an equivalent directed (unidirectional) graph can be formed by replacing each n-edge node with n separate nodes, each of the n nodes having n-1 inputs and one output, and forming unidirectional edges between the nodes to achieve the connectivity of the original graph. Referring to FIG. 2A, for instance, each 3-edge variable node 110 can be replaced with three 2-input/1-output variable nodes 210, 212. Referring to FIG. 2B, each 4-edge check node 120 can be replaced with four 3-input/1-output check nodes 220.

[081] One approach to analog implementation of a decoder corresponding to the factor graph shown in FIG. 1 is to implement a circuit element for each node of the equivalent unidirectional graph. Referring to FIG. 3, a portion of the graph shown in FIG. 1 is illustrated showing check node 0 (120), the bidirectional edges and variable nodes 1, 3, 4 and 7 (110) linked to that check node, as well as the other check nodes 1, 2 and 3 (120) linked to those variable nodes. Referring to FIG. 4, a portion of the corresponding directed graph is shown in which check node 0 (120) is expanded as four 3-input/1-output check nodes 220, for instance, labeled "0/1" to indicate that this is part of the expansion of check node 0 with the output link coupled to variable node 1. Similarly variable node 1 (110) is shown in its expansion into three 2-input/1-output nodes 210, 212, for instance, labeled "1/0" to indicate that this is part of the

expansion of variable node 1 with the output link coupled to check node 0, or labeled "1/out" to indicate that the output link provides an output of the factor graph.

[082] In the example, which is partially illustrated in FIG. 4, a full implementation has four circuit elements for each check node (i.e., 16 total expanded unidirectional check nodes 220), and three circuit elements for each variable node (i.e., 24 total expanded unidirectional variable nodes 210, 212). Out of the three circuit elements for a variable node, two (i.e., 16 total expanded variable nodes 210 for all variable nodes) are used for message passing in an iterative stage of decoding operation, and one (i.e., 8 total expanded variable nodes 212 for all variable nodes) is used for generating the decoder output (i.e., the "belief") in an output stage of decoding operation, as will be described further below.

[083] In operation, input signals y_i are used to determine corresponding analog representations of input messages, which may be determined in a signal mapping circuit. In some examples, the inputs messages form representations of the probabilities corresponding to bits b_i, but the reader should recognize that the discussion below with respect to computations involving representations of bit probabilities is illustrative of a particular form of input and internal messages that are stored or passed during computation. These messages are provided to the inputs of the variable nodes 210, for example, as outputs of analog input registers 260. As discussed further below, in some embodiments the representations of the bit probabilities are provided as analog signals from the input registers 260 encoding a (prior) log likelihood ratio (LLR) which is typically of the form

$$\log\left(\frac{\Pr(b_i = 0 \mid y_i)}{\Pr(b_i = 1 \mid y_i)}\right),$$

In the case of equal prior bit probabilities $\mathrm{P}(b_i = 0) = \mathrm{P}(b_i = 1)$ is equal to

$$\log\left(\frac{\mathrm{P}(y_i \mid b_i = 0)}{\mathrm{P}(y_i \mid b_i = 1)}\right).$$

In some examples, these bit probabilities are encoded as voltage or current in single-

ended or differential form (e.g., using a pair of conducting paths for each unidirectional signal).

[084] The approach partially illustrated in FIG. 4 is one of a number of approaches to implementation of a decoder corresponding to the graph shown in FIG. 1 that involve introducing an analog memory element 230 to break some or all cycles in the directed graph. In the approach shown in FIG. 4, the memory elements are introduced at the outputs of the check nodes. Other versions have such memory elements introduced at the output of the variable nodes instead of or in addition to the memory elements at the outputs of the check nodes. Note that in yet other embodiments, some or all cycles remain without memory elements, and operation is at least partially based on a "relaxation" form of computation as signals propagate through the cycles. In some embodiments, as combination of relaxation and memory based computation is used.

[085] As illustrated in the example partially illustrated in FIG. 4, memory elements 230 in this embodiment store values in analog form, and are introduced at each output of the check nodes 220; that is, 16 memory elements are introduced. For notational simplicity, these locations are indexed as (i, j) and labeled "Ci,j", for the output from check node i that is linked to variable node j. Note that each location corresponds to one of the non-zero entries in the check matrix of the code. The (i, j) memory location corresponds to the row i, column j, non-zero entry of the check matrix of the code.

[086] In a number of approaches that make use of analog memory elements, the memory is introduced in the circuit implementation of the graph such there remain no cycles in the directed graph by breaking all cycles in the directed graph. The circuit implementation is then operated in a series of clocked cycles, such that at each cycle analog values read from some or all of the analog memory elements are propagated through analog circuit elements to inputs of some or all of the memory elements

where they are stored at the end of the clock cycle. As discussed in detail below, such clocked ("discrete time") implementation can be used to decode with a result that is similar to that which would result from a relaxation ("continuous time") implementation.

[087] Referring to FIG. 5, another partial illustration of the example shown in FIG. 4 includes outputs of the four expanded check nodes 220 associated with the original check node 0 (120). A circuit block 390 forms an analog computation module that includes implementations of the expanded variable nodes 1/0, 3/0, 4/0, and 7/0 (210) which have outputs to the four expanded check nodes 220. Note that check node 0 corresponds to row 0 of the matrix representation of the code, which is reproduced in FIG. 7B. Note that the outputs of the circuit block 390 correspond to the memory locations row 0 of the matrix representation, as illustrated in FIG. 7A. The inputs of the circuit block 390 correspond to the non-zero entries in each column of the matrix representation that has a non-zero entry in row 0, omitting those entries in row 0. In this illustration, the inputs correspond to the non-zero entries in columns 1, 3, 4 and 7 in rows 1, 2 and 3. This results in four memory cell inputs, C1,1, C3,3, C3,4 and C2,7, in addition to the inputs from the input bit probabilities, B1, B3, B4, and B7.

[088] An example of a full clocked circuit implementation of a decoder for the length 8 LDPC has a memory element 230 at the output of each unidirectional check node 220, and four copies of the circuit block 390, one corresponding to each row of the code matrix. In the first stage of decoding operation, each unidirectional variable node 210 (i.e., a total of 16 circuit elements) takes its input from an output of a memory element 230, and one of the input bit probabilities 260. (Note that in general for other size codes, the variable nodes are associated with more than two check nodes, and therefore variable nodes would take as input values from multiple memory elements). The memory cells 230 as a whole form a memory that is configured so that effectively all the values are updated at once at the end of each clock cycle. One

implementation of such a memory uses a "double buffering" approach in which two banks of memory are used, and in each clock period, one bank is read from and the other bank is written to, with the banks switching role between each clock period.

[089] In some examples, the decoder may perform memory updates in successive clock cycles, each clock cycle corresponding to a full update of all memory cells of the memory 250. The number of clock cycles to be performed in the first stage of decoding operation may be pre-determined, for example, based on design preference, or depend upon the satisfaction of certain convergence conditions, for example, satisfaction of the code constraints (i.e., full error correction) or a condition based on a rate of change of output values between iterations..

[090] Referring to FIG. 6, in some examples, once the iterations of memory updates are completed, the decoder proceeds to the output stage of decoding operation to generate decoder outputs representing bit estimates. Here, the decoder outputs are denoted as $\hat{b}_j$, each being an estimate of a corresponding input bit (b_j) based on the entire input signal. In some examples, as illustrated in FIG. 6, the variable node 212 outputs a message that includes a representation of the bit probability after decoding, for example, as an LLR, which can be considered to approximate

$$\log\left(\frac{\mathrm{P}(b_i = 0 \mid y_{\setminus i})}{\mathrm{P}(b_i = 1 \mid y_{\setminus i})}\right)$$

where $y_{\setminus i}$ denotes the observations not including y_i. The output of variable node 212 is combined in a combination element 312 with the input bit probability representation from input register 260 to form the representation of the bit probability based on all the inputs and the constraints between the decoded bits. Recall that the output of input register 260 can be considered to represent

$$\log\left(\frac{\mathrm{Pr}(b_i = 0 \mid y_i)}{\mathrm{Pr}(b_i = 1 \mid y_i)}\right)$$

and therefore the combined probability output from combination element 312 is computed as a sum approximates

$$\log\left(\frac{\Pr(b_i = 0 \mid y)}{\Pr(b_i = 1 \mid y)}\right)$$

where y represents all the input values. Optionally the combined bit probability is passed through a hard decision, which in the case of binary outputs and logarithmic representations determines $\hat{b}_j$ to take on the value of either 0 or 1 based on a thresholding of the combined log likelihood ratio as either greater or less than zero. For example, the output element that uses memory elements C0,1 and C1,1 and the input B1 to generate bit estimate $\hat{b}_1$. In some implementations, the set of eight output elements may be configured to operate in a parallel fashion to generate the full set of bit estimates $\hat{b}_j$ in a single clock cycle. Note that as illustrated in FIG. 6, elements 212 and 312 are drawn as separate. However, each effectively computes a sum of its inputs, and the two summations may be combined into a single circuit element 315.

[091] Referring to FIG. 8, in another example of a clocked circuit implementation each of the nodes of the directed graph is not required to correspond to a different circuit element. That is, certain circuit elements form analog computation modules ("cores") that are reused multiple times with different input and output connections (i.e., shared) within each iteration. The functions performed by multiple modules 390 in one clock cycle in the previous example are carried out in a series of clock cycles such that at in each of the series of clock cycles, only some of the memory elements 230 are updated, with all the memory elements being updated at the end of the series of clock cycles. Similarly, in the output stage of decoding operation, one or more shared circuit elements (e.g., element 315) may be reused in an output section 395 for generating one or more bit estimates in each of a series of clock cycles. In the discussion below, the entire series of clock cycles that updates all the memory elements in FIG. 3 is referred to as an "iteration."

[092] Continuing to refer to FIG. 8, a shared module 390 is coupled to input selection circuitry 370 and output circuitry 380, which together provide interfaces to the memory elements 230 in the memory 250. For example, the input circuitry 370

couples each input of a variable node 210 to the output of an appropriate memory cell 230 and to an appropriate input register 260, which collectively form an input memory module 265, and the output circuitry 380 passes the outputs of the check nodes 220 to the inputs of appropriate memory cells 230, which collectively form an intermediate memory module 250. In this example, the shared module 390 includes all the variable nodes 210 and check nodes 220 needed to compute all the outputs corresponding to one of the bidirectional check nodes 120 in the factor graph illustrated in FIG. 1. During each successive clock cycle of an iteration, the input circuitry 370 and the output circuitry 380 is effectively reconfigured to change the connection of the variable nodes 210 and check nodes 220 to the memory 250 and the input bits.

[093] As an example of a multiple cycle iteration using the shared module 390 illustrated in FIG. 8, the table shown in FIG. 9 illustrates the configurations during the four clock cycles of an iteration. Note that the configuration indicated for cycle 0 corresponds to the configuration illustrated in FIG. 5.

[094] In some examples, multiple shared modules 390 are implemented in a single integrated circuit. For example, the example shown in FIG. 8 may be modified to have two shared modules, thereby providing eight new values for memory cells 230 in each clock cycle, with the iteration to update all the memory cells taking a total of two cycles (i.e., four constraints per iteration divided by two constraints per cycle yielding two cycles per iteration). Similarly, in some examples, a shared module may update fewer cells, for example, updating only a single cell in each clock cycle (i.e., using a single check node 220 and three variable nodes 210).

[095] In the example illustrated above in FIG. 8, the updated values to the memory 250 are not passed through to the outputs of the memory until after the entire iteration is completed. In some examples, the updated values determined in one clock cycle may be presented at the output of the memory during subsequence clock cycles within

the same iteration. In such examples, the order in which the outputs of the check nodes are computed (the "schedule") may be significant. Examples of schedules include a sequential updating of the outputs associated with each of the check nodes 120 (see the factor graph in FIG. 1), and random updating in which different nodes are updated at each clock cycle.

[096] Referring to FIG. 10, in some examples, multiple modules 390 are used (but not a sufficient number so that an iteration may be completed in a single cycle), and the input selection circuitry 370, output selection circuitry 380, and memory 250, are distributed among a set of local processing elements 490, and each local processing element 490 has one shared module 390. Each local processing element has a local output circuitry 480 and a local input selection circuitry 470. The memory is distributed such that the memory cells 230 in the memory 450 of a local processing element are those cells that are updated by the shared module 390 in the various clock cycles of an iteration. As illustrated, each row of memory cells is updated in one clock cycle. A control input controls the configuration of the input and output circuitry according to the cycle in the iteration being performed. Note that in general, a shared module 390 at one local processing element 490 requires outputs of memory cells 230 in a local memory 470 of its own local processing element and/or another (or more generally, one or more other) local processing element. The local input selection circuitry 470 selects the memory cells required by each of the local processing elements and passes those values onto a global selection unit 440, which then determines the proper subsets of the memory values to be passed onto each one of those local processing elements. In the output stage, the memory cells are coupled through the selection circuitry 470 to the output section 495 to determine the outputs. The configuration shown in FIG. 10 can also be understood as the function of input selection logic 370 shown in FIG. 8 is distributed among blocks 470 and 442, and the output logic 380 is distributed among the blocks 480.

[097] In some examples, the global selection unit 440 may include a set of selection units 442, each coupled to inputs of a respective local processing element to provide the corresponding subset of memory values to the shared module 390. For example, one selection unit 442 may receive 8 signals representing memory values provided by the two local input selection circuitries 470 to generate four output signals representing the memory values to be provided to the local processing element shown on the left of FIG. 10.

[098] Referring again to FIG. 8, in some examples, the memory 250 as a whole is configured such that effectively all the values are updated exactly once in an iteration. In one implementation of such a memory using a "double buffering" approach two banks of memory are used. In iteration k, the write circuitry always writes into memory bank #1, and the read circuitry always reads from memory bank #2. By the end of iteration k, memory bank #1 has achieved a full update. In the next iteration $k+1$, the write circuitry switches to write into memory bank #2, and the read circuitry reads from memory bank #1 which was just updated in the last iteration. In this case, the memory 250 would need a capacity twice the amount of the outputs from the local check nodes to keep two different copies for read and write operations respectively.

[099] In some examples, system is configured using an incremental "scheduled" approach such a subset of values is updated and available for use at the end of each cycle rather than at the end of an entire iteration composed of multiple cycles. In some such examples, for example using a single block 390 as shown in FIG. 8, there is no conflict between memory cells that are read in a cycle and locations that are being written in a cycle. The reason that there is no conflict can be understood with reference to FIGS. 7A-B. For example, applying a constrain corresponding to row 0, the memory cells corresponding to non-zero entries in that row are written, while

memory cells in other rows, specifically in columns that have non-zero entries in row 0, are read.

[0100] Referring again to FIG. 10, when two or more update modules 490 are used, it is possible that a memory cell being updated in one module is to be read in the same cycle in another module. This can be understood again with reference to FIGS. 7A-B, for instance, in a case where one module is applying the constraint associated with row 0 and another is applying a constrain associated with row 1. The application of the row 0 constraint writes C0,1 and reads C1,1, while application of the row 1 constraint writes C1,1 and reads C0,1.

[0101] In another implementation a memory 250 for an incremental approach does not required twice the capacity of the entire memory. Here, in the first half of clock cycle k, the read circuitry retrieves some values from the memory, some of which may have been updated in a preceding clock cycle $k-1$. The outputs of the XOR gates are written back into the same memory during the 2nd half of clock cycle k. In some examples, by using edge-sensitive (instead of level-sensitive) memory, the read and write operation in the same clock cycle would not interfere with each other.

[0102] The approaches described above in the context of a length 8 code is applicable to a larger example of an (1056,352) LDPC code, such as is used in IEEE 802.16 based communication. The check matrix of the code can be represented in tabular form breaking the 0,1 matrix into 8 rows by 24 columns of 44 by 44 blocks, with each block being either all zero, or being an shifted diagonal with one non-zero entry in each row and in each column. This tabular representation of the code is shown in FIG. 11. The upper-left (0,0) block (showing the number of "0") in the tabular representation is a diagonal matrix. The (0,2) block shown as a "8" is a off-diagonal block $M=\left[m_{i,j}\right]$ such that $m_{i,j}=1$ if $j=i+8 \pmod{44}$ and 0 otherwise. The full factor graph is not illustrated, but can be derived from the matrix representation in the same manner as the example illustrated in FIG. 1.

[0103] Referring to FIG. 12, a module 590 is configured to include variable nodes 510 and constraint nodes 520 for the code shown in FIG. 11. Note that the nodes illustrated in FIG. 12 are unidirectional nodes in which links are either input or output links. Module 590 is analogous to module 390 for the length 8 code discussed above. Note that each row in the code matrix shown in FIG. 11 has ten non-zero entries in all row blocks, except row block 6, which has eleven non-zero entries per block. In order to implement constraints outside row block 6, the module 590 has ten (unidirectional) variable nodes 510 and ten (unidirectional) constraint nodes 520, and for rows in row block 6, eleven (unidirectional) variable nodes 510 and eleven (unidirectional) constrain nodes 520. Each variable nodes accepts inputs for memory cells corresponding to non-zero entries in a particular column of the code matrix. Therefore, variable nodes corresponding to columns in the range 0 through 15 have four inputs (three inputs for memory cells corresponding to entries in the code matrix and one input for the bit probability) and one output. Variable nodes for columns 16 through 23 have two or three inputs depending on the column and the block row. In some examples, the module 590 has the maximum number of variable nodes and inputs necessary, and is configurable during different cycles to accommodate the specific number of variable nodes and inputs needed, for instance, by ignoring certain inputs.

[0104] FIG. 13 illustrates one type of implementation of a decoder operable to perform the iterative stage of decoding operation for use with the (1056, 352) LDPC code shown in FIG. 11. In such an implementation, the decoder includes an analog input memory 660 that stores representations of the input bit probabilities (e.g., as voltages encoding log likelihood ratios) corresponding to the 1056 bits (i.e., 24 blocks of 44) of the LDPC code illustrated in FIG. 11. These input bits are then distributed by memory selection circuitry 672 to be processed in a set of local processing elements 690. Each local processing element 690 has a shared module 590 that includes the variable nodes and check nodes needed to compute all the outputs

corresponding to one of the check nodes of the full factor graph. The structure of each processing element 690 is similar to each processing element 490 shown in FIG. 10 to implement the decoder for a length 8 code.

[0105] Each local processing element 690 also includes a local output circuitry 680, which directs the output of the local check nodes into appropriate cells 630 of a memory 650. In this example, the memory is distributed among the set of local processing elements 690 as a set of local memories 650, each of which includes memory cells 630 updated by the shared module 590 of its local processing element 690 (not other local processing elements) in the various clock cycles of an iteration. As described before, in general, each shared module 590 at one local processing element 690 requires outputs of memory cells in a local memory 650 of its own local processing element and/or one or more of other local processing elements. These outputs are obtained by a set of local read circuitry 670 that retrieve values from the local memory 650 and send them to a global selection unit 640, which then determines the appropriate combinations of output values to be sent to the individual local processing elements 690 at various clock cycles. The global selection unit 640 includes a separate input selection unit 642 associated with each of the local processing elements, and provides as outputs the values stored in the memories 650 required for input that that unit on each iteration.

[0106] Implementations of the type illustrated in FIG. 13 can have different numbers of processing elements, and use different schedules of applying each of the 352 constraints in different cycles of a decoding iteration. Referring back to FIG. 11 in the matrix representation of the (1056, 352) LDPC, out of the total 8 blocks of rows, rows in 7 of the blocks of rows (i.e., row blocks, 0-5, and 7) contains 10 non-zero entries, and row in one block (row block 6) each contains 11 non-zero entries. Therefore, of the 192 (i.e., 24*8) blocks of entries, only 81 are non-zero, and each of

the non-zero blocks has exactly 44 non-zero entries, for a total of 3564 (81 times 44) entries.

[0107] The exemplary arrangement shown in FIG. 13 uses a set of 8 local processing elements 690 each configurable to perform the computation associated with one check node (i.e., one row), of a corresponding block of rows of the check matrix. In other words, each local processing element 690 is used repeatedly in 44 cycles to compute the outputs for the 44 constraints represented by the 44 rows of the code matrix in the corresponding block. For example, the 8 elements apply constraints 0, 44, 88, ... , 308, respectively, on the first cycle, constraints 1, 45, 89, ... , 309, respectively, and finally 43, 87, 131, .., 351, respectively, on the last cycle of an iteration.

[0108] Note that because of differences in each of the row blocks in the code matrix, the shared module 590 in each local processing element 690 may be have differences. Consider a shared module 590 for performing the computation associated with a row in the first (row block 0) block. The corresponding check node in the bidirectional graph has 10 edges linked to variable nodes. Each of the first 8 variable node has five edges, four to check nodes and one to a bit input; the 9^{th} variable node has four edges, three to check nodes and one to a bit input, and the 10^{th} variable node has three edges, two to check nodes and one to a bit input. The shared module 590 therefore has circuits for 10 (directional) check nodes, each with 9 inputs and one output. The 10 outputs of the check nodes update 10 locations in the local memory. The local processor has circuits for 10 (directional) variable nodes 510, each with four, three, or two inputs and one output. Each node 510 provided an input to 9 of the 10 (directional) check nodes 520. Of the inputs for each variable, one input is for an input bit probability and the remainder are for values from the local memories.

[0109] Shared modules 590 in the other local processing elements have the same structure as that associated with row block 0, with differences including the shared module 590 for row block 6 having 11 check nodes, and 11 variable nodes, and the

share module 590 for blocks 1 through 5 each having two variable nodes with two inputs and the other variable nodes having four inputs.

[0110] In operation, at each clock cycle, the variable nodes of shared module 590 for row block 0 reads 10 sets of inputs from the input memory 660, one set for each variable node and updates 10 locations of the local memory 650. The values from memory 650 are passed through the blocks 670 of multiple of the local processing elements 690 and through the control unit 642 associated with the destination processing element. Over 44 clock cycles of an iteration, the shared module 590 provides updated values for all 440 (44 times 10) locations in the local memory.

[0111] As outlined above, in some embodiments, each one of the shared modules 590 may be implemented as a combination of 10 variable nodes and 10 check nodes (also referred to as a 10x10 shared module), except for the shared module 590 for row block 6 which is implemented as a 11x11 module.

[0112] A number of different circuit arrangements and signal encodings can be used within the approaches described above. For certain soft decoding applications, each variable node circuit can be formed using a soft Equals gate, and each check node circuit can be formed by a soft XOR gate. In the example of FIG. 13, each variable node takes the form of a 4-input (or 3-input or 2-input) soft Equals gate and each check node takes the form of a 9-input (or 10-input) soft XOR gate. Therefore, for each shared module 590, besides reading the 10 (or 11) of input bits (one each soft Equals gate) from the input memory 660, it also requires 10 (or 11) sets of values from the local memories 650 (one set for each soft Equals gate). Note that in this example these values come from the memory cells in the other local processing element(s), and not from the memory 650 in the same processing element.

[0113] One approach for providing the proper combinations of memory values needed as input to the shared modules 590 includes forming, in the global control unit 640, a set of 8 individual selection units 642, each of which selects or combines the

outputs of the local processing elements 690 as needed for the input values for a corresponding shared module 590. In some examples, each one of the read circuitry 670 is selectively coupled to the set of 8 selection units, for example, using a set of 8 buses with each bus containing 10 (or 11) wires for sending a total of 10 (or 11) output values to an individual selection unit in one clock cycle. The selection unit 642 then chooses a set of 10x3 (or 11x3) output values for input to the shared module 690.

[0114] By arranging the decoder into local processing elements, in some embodiments, all of the XOR signals become local to the local processing elements in which they are formed. The inputs to the Equals gates become globally routed signals that come from multiple local processing elements. In some examples, the local processing elements 690 can be configured in a way such that each shared module 690 requires only output values from a pre-defined set of three other local processing elements. As a result, the coupling between each local processing element to the global control unit 640 can be reduced, for example, with read circuitry 670 now being coupled to only 3 (instead 8) selection units. In some examples, the local processing elements 690 can be further arranged such that all of the even-numbered (i.e., 0, 2, 4, and 6) local processing elements communicate with each other but not with the odd-numbered (i.e., 1, 3, 5, and 7) local processing elements (except for the last eight block columns of the check matrix).

[0115] Note that, in some applications relating to soft decoding, the decoder described above is used for converting input "soft" bits based on individual measurements of each bit to soft bits each based on the entire block of soft bits, taking into account the constraints that the original bits of the block satisfied. These output soft bits can then be further processed, or converted by hard decision into output "hard" bits taking values 0 or 1. The input soft bits may be provided in the probability domain, for example, as the probability of a bit being value of 1 or 0.

Alternatively, the input soft bits may be provided in the log domain, for example, as the log likelihood ratio of a bit (e.g., as defined by

$$\log\left(\frac{p(b_i = 0 \mid y_i)}{p(b_i = 1 \mid y_i)}\right).$$

In either case, the shared module 590 can be implemented using a set of analog circuit components that perform analog computation functions appropriate for the particular application. Implementations of some of these analog circuit components (such as soft Equals and soft XOR) are illustrated in detail in U.S. Patent Application Serial No. 61/156,794, titled "Circuits for Soft Logical Functions," filed March 2, 2009.

[0116] Referring to FIG. 14, in some implementations, the soft equals makes use of differential voltage inputs, each representing a log likelihood ratio to produce a voltage that is proportional to the sum of the inputs. Each differential voltage input is passed through a voltage to current converter 712, and the resulting currents are summed on a bus 714. The current on the bus is passed through a current to voltage converter 716. The output voltage then branches to the soft XOR circuits that require the output of this equal node. Exemplary circuit implementations are shown in the figure. A variety of alternative circuits can be used, including alternative soft Equals circuits described in U.S. Patent Application Serial No. 61/156,794.

[0117] Referring to FIG. 15A, in some implementations, the soft XOR circuits make use of log domain differential voltages as produced by the circuit shown in FIG. 14. In the exemplary implementation of the soft XOR circuit shown in FIG. 15A, which approximates an ideal soft XOR function for log domain processing, one differential voltage input is passed to a circuit 812. The second and further inputs to the soft XOR circuit are passed to circuits 814, each of which performs an analog computation that approximates multiplication of the current provided by the previous element according to that input. The resulting current approximates the ideal soft XOR function each of the inputs and is passed through a current to voltage converter 816, to provide the differential voltage output of the soft XOR. Note that unlike the soft

Equals circuit shown in FIG. 14, the output of the overall soft XOR circuit does not fan out on any particular cycle, because the output of the soft XOR circuit provides the input to only a single memory cell. The circuit parameters, for instance, resistance values, transistor dimensions, and voltage scaling, are chosen to best approximate the ideal function of a soft XOR and/or to optimize higher level (e.g., overall decoding) system performance.

[0118] Referring to FIG. 15B, an alternative arrangement 820 of circuit elements to the soft XOR circuit 520 shown in FIG. 15A implements the directional soft XOR circuit using a branching tree structure, optionally sharing signals between different trees associated with a same bidirectional check node. Specifically, groups of circuit elements 818 effectively form two-input, one-output, voltage based soft XOR circuits using the circuit elements 812, 814, and 816 introduced with reference to FIG. 15A. These groups of circuit elements 818 are then arranged in a tree structure, preferably a binary tree structure that is as balanced as possible to form the circuit arrangement 820 shown in the figure. In some implementations, the branching structure shown in FIG. 15B may have preferable characteristics, for instance, providing a better approximation of the ideal soft XOR function with LLR representations. Furthermore, when multiple modules 820 are implemented for a set of unidirectional XOR circuits, certain computations can be shared, for example, by passing a signal 825 from one module to another where a portion of the tree in that other module can be eliminated.

[0119] Referring to FIG. 16, in some implementations, the bus 714 associated with each equal gate is distributed. Each memory cell 230, which stores its value as a differential voltage, has at its output a corresponding voltage to current converter 912. These voltage to current converters are similar to the converters 712 shown in FIG. 14. The current output is passed to a set of switches 913, at most one of which is enabled if the corresponding cell's value is to be injected as a current on a current bus

714 corresponding to that switch 913. Each bus 714 similarly includes a portion onto which current associated with an appropriate input bit is injected at each cycle to account for the input to the equal gate corresponding to the bit input. Note that the bus 714 for each equal node may have a complex structure, for example, having numerous branches. Nevertheless, all the current injected onto the bus passes to the current to voltage converter 716 for the equal node, from where it branches to the unidirectional XOR circuits that require that output.

[0120] Referring to FIG. 17, in some implementations that make use of a distributed bus 714 to perform a current summation function, the module 590 illustrated in FIG. 12 is replaced by a module 592 in conjunction with distributed busses 714 and voltage-to-current converters 712. In module 592, each variable node corresponds to a current-to-voltage converter 716, which outputs a voltage proportional to the total injected current on the corresponding bus 714, and then that voltage branches to the appropriate check node circuits 520. Note that in yet other implementations, the current-to-voltage converters 716 are themselves distributed, and a module 593 (i.e., a portion of module 592) receives voltage inputs, which are internally distributed to the appropriate check node circuits.

[0121] Referring to FIG. 18, a second example of an implementation of a decoder operable to perform the iterative stage of decoding operation for use with the (1056, 352) LDPC code shown in FIG. 11 provides the same or similar functionality to the implementation shown in FIG. 13. In this example, modules 592 are used, as illustrated in FIG. 17. The circuitry that implements each soft Equals circuit associated with a variable node includes a current-to-voltage converter in the module 592, with the distributed busses corresponding to the 10 unidirectional variable nodes of the module 592 being distributed. Each memory 652 includes circuitry to inject current to the appropriate busses corresponding to the soft Equal circuits for different variable nodes via read switching circuits 671. The bus section 644 effectively

includes 81 busses, each associated with a different current-to-voltage converter 716 at the input of a module 592. Therefore, the soft Equal circuit is distributed in a manner effectively forms interconnection paths between the memories 652 and the analog computation modules 592.

[0122] In another implementation of a decoder for the (1056, 352) LDPC code uses a different arrangement of memory cells, and takes advantage of the distributed current summing approach to implementing the log likelihood ratio based equals gates. The implementation makes use of a schedule in which multiple constraints are applied at each cycle of an iteration, and the updated values are used in the next cycle of the iteration. Referring to FIG. 11, a previously described approach applies eight constraints in each cycle, with each constraint being associated with a different row block. As discussed above, there are situations in which the same cell 230 is both written and read in the same cycle, and therefore approaches such as double buffering and multiple phases (update, write, etc.) per cycle are used. In the present implementation, multiple constraints are applied in each cycle, and these constraints are selected such that there is no read and write conflict on the memory cells, permitting a scheduled approach without the need for approaches such as double buffering. One way to avoid the conflict is for each of the constraints being applied in a cycle to come from the same row block. The reason this avoids conflicts may be understood with reference to FIG. 7A, which relates to a simpler length 8 code. As illustrated, a constraint corresponds to a row in the code matrix, and the cells that are updated are associated with the non-zero entries in that row. The cells that are read are in the columns in which that the constraint's row has non-zero entries. If another constraint is applied in the same cycle, but the row associated with that constraint as no non-zero entries in the same columns as the row for the first constraint, then the cells updated for the second constraint are not read for applying the first constraint, and the cells read to apply the second constraint are disjoint from those read for the first constraint. Turning back to the code matrix illustrated in FIG. 11, it is evident

due to the offset diagonal structure of each non-zero block that multiple rows from the same row block have the property that no two rows have ones in any of the same columns. Therefore, multiple constraints corresponding to multiple rows in one row block can be applied in one cycle without read/write conflicts. As another observation, two rows one in an even block 2n and one in an odd block 2n+3 mod 8 (or odd block 2n+5 mod 8) similarly cannot have any columns with non-zero entries in common due to the overall block structure of the code matrix.

[0123] Referring to FIG. 19, a partial illustration of an instance of this approach arranges the memory cells 230 of the memory 950 into groups 955 of 44 memory cells. Each of these groups 955 corresponds to a different non-zero block of the code matrix shown in FIG. 11. Further, these groups are arranged into banks 958 each corresponding to a column in the same way that they are arranged in the code matrix. For the first 24 columns, there are exactly four memory groups 955 corresponding to the four non-zero blocks in each column block. Each of the last eight columns have two or three memory groups each.

[0124] In FIG. 19, a single module 590 is shown, representing the update logic to apply a single constraint in a cycle. As discussed previously, each equal node in the module receives up to four inputs, up to three from the memory cells within the memory 950 and one from the input registers in memory 960, which is also arranged into groups 965, each associated with a different column block. The equal node uses a distributed current bus approach, as described above, onto which the current encoding log likelihood ratios are injected. FIG. 19 illustrates application of a constraint in the first row block. In the first bank of memory groups 955, one memory cell in the first memory group 955 is updated, and one memory cell from each of the other memory groups 955 is read as well as one input from an input register in a register group 965 associated with that column block. Each of the 10 (or 11

depending on the row block) inputs to the module 590 comes from one bank of memory groups 955.

[0125] As is discussed further below, the off-diagonal structure of the blocks constrains which combinations of memory cells are coupled to the current bus in any one cycle. Specifically, the combinations of memory cells must be found in a same column of the check matrix. Therefore, if the memories 955 are indexed by column within the block column, then in any particular cycle a same index is applied to each memory 955 in a bank 958 and the input memory group 965. Note also that with an arrangement indexed by column, contiguous columns are naturally indexed contiguously, treating wrapping from the highest index (43) to zero as contiguous.

[0126] In some examples, multiple constraints in a row block are applied in each cycle. For example, four constraints corresponding to four contiguous rows may be applied in each cycle. Referring to FIG. 20, as compared to FIG. 19, such an example has four copies of the module 592, and each distributed equal node has as a separate input bus. For updates of four contiguous constraint rows within a row block, for each block column, four contiguous columns are read or written in each bank 958 of the memory groups 955. As in the example shown in FIG. 19, the memory locations in a same column are coupled to a same bus, and therefore outputs of four contiguous columns for a memory 955 are each output to a different of the four busses. Similarly, the updates to the updates of the memory group 955 in contiguous groups of four memory cells in one of the groups 955. Each memory group 955 is arranged in four groups 966 of eleven memory cells each, with each group 966 being coupled to a corresponding one of the four busses. For instance, one memory group 966 has memory cells for the within-block column indices 0, 3, 7, ... 40 (i.e., columns indices 0 mod 4), a second for column indices 1, 4, ... 41 (i.e., 1 mod 4), and so forth. In any cycle, one cell from a group 966 is coupled to the bus or one cell from the group is written to. In each block column, the column that is applied associated with the first

module 590 depends on the offset of the block diagonal in the block being updated. Therefore, if the first constraint row has a non-zero entry in that block column at an index *i*, a shifter 970 "rotates" current bus *i* mod 4 to the 0^{th} module 590, bus (*i*+1) mod 4 to the 1^{st} module, (*i*+2) mod 4 to the 2^{nd}, and (*i*+3) mod 4 to the 3^{rd}.

[0127] In some examples, the function of a soft Equal circuit is distributed such that the output of a memory group 966 is a current, which is summed on a bus as illustrated in FIG. 20. Then the shifter 970 includes a current to voltage conversion circuit for each bus at its input, and then shifts the resulting voltages to modules when each voltage encoded input is applied to the inputs of appropriately selected soft XOR circuits. The module 592 is then replaced with a module 593, which accepts voltage rather than current inputs.

[0128] Note that a number of further optimizations can be used to take advantage of structure of the code. For example, four constraints from row block 2n can be applied at the same time as four constrains from row block (2n+3 mod 8), thereby allowing eight constraints to be applied in each cycle. Note that of the first 16 banks of memory groups 955, memories in an even block row are updated based on an update based only on even block rows in the first 16 block columns, and odd block rows in the last 8 block columns. Similarly, memories in an odd block row are updated based only on odd block rows in the first 16 block columns and even blocks rows in the last 8 block columns (with the exception of block (6,16) which is used to update block (0,16) and vice versa.

[0129] In various examples, different schedules for associating sets of constraints with cycles can be used. For example, with four modules 590, the constraints applied in each cycle can proceed as {0,1,2,3}, {4,5,6,7}, ... {40,41,42,43}, {44, 45,46,47}, ... {348,349,350,351}, in 88 cycles for each iteration. As another example using eleven modules, the cycles can proceed as {0,1,...10}, {11,12, ..., 21}, {341,342,...,351} in 32 cycles in each iteration. As another example, with two sets of four modules 590

(i.e., eight total, four for block 2n and four for block 2n+3), the constraints applied can proceed as {0,1,2,3,132,133,134,135}, {4,5,6,7,136,137,138,139}, ... , {40,41,42,43,172,172,173,174},{88,89,90,91,220,221,222,223}, ... in 44 cycles.

[0130] Referring to FIGS. 21A-B, an overall diagram for an implementation of a decoder for the code with check matrix shown in FIG. 11 is shown. An input memory 1660 holds the 1056=24x44 input values, represented as voltages proportional to the LLR inputs. In some examples, these values are loaded, for example, four at a time, in a serial manner before the iterative stage begins. In some examples, the LLR values are obtained from a signal mapping circuit that accepts one or more of the signal values y_i and outputs one or more of the LLR values. In some examples, one signal input generates a set of LLR inputs, for example, when multi-level encoding is used, such as when four bit values are encoded in a sixteen-level analog signal. Generally, the architecture shown in FIGS. 21A-B has separate parts for the "even" and the "odd" block rows, which as introduced above allows concurrent processing of constraints in different block rows without read-write contention for memory cells. A set of memory banks 1650 include memories 1652, each holding 44 entries corresponding to a corresponding block of the check matrix. Each bank includes only either odd or even block rows in a particular block column of the check matrix. Together, these banks correspond to the memories 955 shown in FIG. 19. Two parity check processors 1590 are illustrated. Each implements four analog computation modules generally of the form of module 593 that is illustrated in FIG. 17. The memory banks 1650 are coupled to the parity check processors 1590 via rotator modules 1970. Generally, each rotator module 1970 includes a set of four rotators each associated with one of the four analog computation modules that are in the corresponding parity check processor 1590. Note that these rotator modules include current to voltage conversion circuits such that the links between the memory banks 1650 and the rotator modules 1970 are differential current encoded, while the links between the rotator modules 1970 and the parity check processors 1590 are

differential voltage encoded. The outputs of the parity check processors 1590 are passed to the appropriate memory banks 1590 where the determined values are stored. An output section 1400 is coupled to the memory banks 1650 and to the input memory 1660, and includes the circuitry for computing the output values, which may be computed multiple outputs at a time an multiplexed. Not shown is the control logic that coordinates the configurations of the rotators and read and write control logic within the memory banks.

[0131] It should be understood that the decoder applications described above are only one example of an application of an analog belief propagation processor. The techniques employed in these examples are applicable to other uses of belief propagation.

[0132] Implementations of the belief propagation processors may have different degrees of customization to particular applications. Generally, a controller (not shown) sequences the application of different constraints in different cycles of an iteration, and sequences the series of iterations to complete a computation. The controller can be a special-purpose controller or sequencer that is configured for a particular code, for a particular class of codes, or to some other class of computations. The controller may also be a general purpose controller that may be used to implement a wide variety of computations. In some implementations, the modules that include the variable and constraint nodes may have fixed structures, or may be configurable. Circuit configuration may occur through the operation of the controller and/or the configuration may be implemented through a field programmable approach in which certain connections between circuit elements are enabled using personality data that is applied to the processor. In some cases the controller is implemented on the same device that implements the analog circuit portions of the processor, while in other cases, the controller is fully or partially implemented in a separate device that is in communication with the device implementing the analog processing.

[0133] The belief propagation processor may be used in many applications. One application relates to communication systems, where the belief propagation processor is used as a soft decoder subsequent to a soft demapper that converts signals transmitted over a noisy communication channel into soft bits. The soft bits may be represented in the probability domain (e.g., as probabilities or differential probabilities), or alternatively, in the log domain (e.g., as log likelihood ratios or log-odds). Examples of soft demappers are described in detail in U.S. Patent Application Serial No. 61/156,721, titled "Signal Mapping".

[0134] One application of a decoder implemented in using an analog belief propagation processor is in a communication system in which blocks of transmitted information are received, and the decoder performs an error correction of the received information. Another application of such a decoder is as a component of a data storage system, for instance a semiconductor memory (e.g., flash memory, which may include multilevel cell storage) or a disk storage memory, in which the decoder performs an error correction of the information retrieved in the storage system, for example, retrieved as storage levels in memory cells or as degrees of magnetization in a disk storage system. In some examples, the decoder in integrated on the same device as a semiconductor memory, while in other examples, the decoder is implemented in a separate device (e.g., on a separate integrated circuit). Some examples of memory systems that can make use of decoder implementations as described in the present application are found in co-pending U.S. Application No. 12/537,060, titled "STORAGE DEVICES WITH SOFT PROCESSING," filed August 6, 2009, which is incorporated herein by reference.

[0135] It should also be understood that various modifications of the approaches described are possible. For example, memory elements are not necessarily analog and/or continuous valued. For example, digital (e.g., storage for binary (base 2) digits, possibly in "soft" forms) and/or quantized storage may be used. Other

representations of the values passing between nodes can be used. In some examples, log likelihood ratios are used. Other possibilities include passing of linear probabilities. With different representations, different circuit implementations of the equal and constraint nodes would be used. Various encodings of the representations may be use. In some examples described above, differential voltage and differential current encodings are used. Alternatives include stochastic (e.g., dithered) representation, digital or quantized representations.

[0136] Examples described above are implemented in various ways. In one example, a particular code, for example, represented as a check matrix, is transformed using a computer implemented (optionally human assisted) technique that produces a data representation of switching and interconnect circuitry, and optionally of the circuit implementations of soft Equals and soft XOR circuits. In some examples, this data representation (e.g., data structures or instructions) is stored on a machine readable medium and is later used to impart functionality when employed in a computer-based device layout and fabrication system. In other words, when employed in such a system, the data representation is read and determines the physical circuit implementation. An example of a data representation includes a representation that follows Verilog-A or Verilog-AMS specifications. In other examples, a specific code is not specified and the resulting device is configurable to accommodate a variety of different codes. Furthermore, the approach is not limited to devices used to decode codes, as many other analog graph-based computations may be implemented using the techniques described above.

[0137] In some examples, the device may have somewhat unpredictable characteristics, for instance, that depend on particular fabrication time (process) or run-time conditions. For example, gains of transistors may depend on process characteristics or on operating temperature of the device. Some designs include selectable and/or configurable gain elements, for example, to adjust the gains of soft

XOR circuits, gains of current to voltage conversion circuits, gains of current outputs of memory cells, etc. In some examples, gain is adjusted by controlling resistance elements, such as but not limited to, resistive elements in current to voltage converters 716 (see FIG. 14) or circuits 812 and 814 (see FIG. 15A) or circuits 816 (see FIG. 15B), for example, at the last stage of the tree of two-input XOR stages. The gain control can be provide in digital form, and passed through a digital-to-analog converter to an analog-controllable resistive element (e.g., an appropriately biased transistor). In some examples, these gains are adjusted to optimize overall system performance, for instance, in a system calibration stage before operational use or in an ongoing (e.g., feedback) adaptation of the elements to maintain optimal or near optimal performance.

[0138] It is to be understood that the foregoing description is intended to illustrate and not to limit the scope of the invention, which is defined by the scope of the appended claims. Other embodiments are within the scope of the following claims.

What is claimed is:

1. An analog processor comprising:

a first memory module (265; 960) for storing a first set of storage values in respective storage elements (260) each representing a respective input to the processor;

a second memory module (250; 450; 950) for storing a second set of storage values in analog form in respective storage elements (230), the second set of storage values including intermediate values determined during operation of the processor;

an analog computation module (390; 590; 592; 593) coupled to the first and the second memory modules, the processor being configurable such that in each of a plurality of operation cycles the analog module determines values for at least some of the second set of storage values based on at least some of the first and the second sets of storage values; and

an output module (395; 495) for generating a set of outputs from at least some of the second set of storage values.

2. The processor of claim 1 wherein the first storage module is configured to store the first set of storage values in analog form.

3. The processor of claim 1 wherein the analog computation module is linked to the first and the second memory modules via analog signal paths.

4. The processor of claim 3 in which the analog signal paths are each configured to carry a value on a conductor represented as at least one of a voltage and a current proportional to the value.

5. The processor of claim 1, wherein the analog module is configurable to determine values for a different subset of the second set of storage values in each of a plurality of operation cycles.

6. The processor of claim 1 further comprising input selection circuitry (370; 470, 440) configurable to couple the analog computation module (390) to outputs of selected memory elements (260, 230) of the first and the second memory modules.

7. The processor of claim 1 further comprising, for each analog computation module (592), a plurality of signal busses (714), each bus providing an input value to the analog computation module and being switchably coupled to a plurality of the storage elements (230) of the second memory module.

8. The processor of claim 7 wherein the storage elements are coupled to switchably provide a current representation of a storage value stored in the storage element such that the input value provided to the analog computation module is represented as a current that is substantially proportional to a sum of the currents representations provided by the storage elements.

9. The processor of claim 1 further comprising output section circuitry (395; 495) configurable to accept outputs of selected memory elements (260, 230) of the first and the second memory modules, and to determine outputs of the analog processor.

10. The processor of claim 1 comprising a plurality of analog computation modules (390), said modules being concurrently operable to determines values for different subsets of the second set of storage values in each operation cycle.

11. The processor of claim 10 wherein the second memory module includes a plurality of section (450), each associated with a corresponding different one of the analog computation modules (390) for storing values determined by the associated computation module.

12. The processor of claim 10 wherein the second memory module (250) is configured such that in a single operation cycle, each storage element (230) can provide a storage value to one or more of the analog computation modules (390) and can accept a determined value to store in the storage element for providing in a subsequent operation cycle.

13. The processor of claim 12 wherein each storage element is associated with two storage locations such that in any one cycle, one storage location is used for accepting a determined value and one storage location is used for providing a value.

14. The processor of claim 10 wherein the second memory module includes a plurality of memory sections (955), groups of the sections forming banks (958), wherein for each of the analog computation modules each of a plurality of inputs to the module is associated with a different bank of the memory sections.

15. The processor of claim 14 wherein the processor is configurable such that in each cycle, each memory section includes memory elements that either provide values to one ore more analog computation modules or memory elements that are updated with values from one or more analog computation modules.

16. The processor of claim 1 configured to implement a belief propagation computation.

17. The processor of claim 1 configured to implement a factor graph computation.

18. The process of any of claims 16 or 17 wherein the analog computation module is configured to accept and provide analog signals that are substantially logarithmic representations of at least one of probabilities, likelihoods, and likelihood ratios.

19. The processor of claim 17 wherein the processor is configured to implement a decoder for a low density parity check (LDPC) code.

20. The processor of claim 1 further comprising:

a controller configured to control operation of the processor to perform a plurality of iterations of computation, each iteration comprising a plurality of computation cycles.

21. The processor of claim 20 wherein the plurality of computation cycles is substantially the same in each iteration, each cycle being associated with a configuration of the first and the second storage modules to provide inputs and output to one or more analog computation modules.

22. The processor of claim 20 wherein the processor is at least one of configured and configurable to implement a decoder parity check code, and each cycle is associated with one or more parity check constraints, and wherein the cycles of each iteration are together associated with all the parity check constraints of the code.

23. The processor of claim 1 wherein the analog computation module implements a network of analog processing elements.

24. The processor of claim 24 wherein the analog processing elements include elements that represent soft logical operations.

25. The processor of claim 24 wherein the soft logical operations include soft XOR operations.

26. The processor of claim 23 wherein the network of elements is acyclic.

27. The processor of claim 23 wherein the network of elements includes at least one cycle of elements, the analog computation module being configured to implement an relaxation computation.

28. The processor of claim 23 wherein the analog computation module includes inputs for configuring one or more gain characteristics in the network of processing elements.

29. A decoder comprising:

a first memory for storing code data having a length in bits;

a second memory for storing intermediate data in analog form;

an analog decoder core coupled to the first memory and to the second memory, the decoder core having an input length less than the length of the code data and an output length less than a number of constraints represented in the code data;

a controller for, in each of a plurality of cycles, coupling the inputs of the decoder code to selected values from the first and the second memories, and coupling outputs of the decoder core for storage in the second memory; and

an output section coupled to the second memory for providing decoded data based on values stored in the second memory.

30. The decoder of claim 29, wherein the first memory is configured for storing code data in analog form.

31. A method for forming a data representation of an analog processor, the method comprising:

forming a data representation of a first memory module (265; 960) for storing a first set of storage values in respective storage elements (260) each representing a respective input to the processor;

forming a data representation of a second memory module (250; 450; 950) for storing a second set of storage values in analog form in respective storage elements (230), the second set of storage values including intermediate values determined during operation of the processor;

forming a data representation of an analog computation module (390; 590; 592; 593) coupled to the first and the second memory modules, the processor being configurable such that in each of a plurality of operation cycles the analog module determines values for at least some of the second set of storage values based on at least some of the first and the second sets of storage values; and

forming a data representation of an output module (395; 495) for generating a set of outputs from at least some of the second set of storage values.

32. The method of claim 31 wherein forming the data representations includes forming Verilog representations of the processor.

33. The method of claim 31 further comprising fabricating a integrated circuit implementation of the analog processor according to the formed data representation.

34. The method of claim 31 further comprising:

accepting a specification of a parity check code; and

forming the data representations to represent an implementation of a decoder for the code.

35. A method comprising forming a data representation for the analog processor or any one of claims 1 through 27.

36. Software stored on a computer readable medium comprising instructions for and/or data imparting functionality when employed in a computer component of an apparatus for forming an integrated circuit implementation of the analog processor or any one of claims 1 through 27.

37. A decoding method comprising:

in each of a plurality of cycles of a decoding operation, applying a portion of code data and a portion of an intermediate value data to an analog decoder core, and storing an output of the decoder coder in an analog storage for the intermediate data; and

combining data, including intermediate value data from the analog storage, to form decoded data representing an error correction of the code data.

38. The method of claim 37 wherein each of the plurality of cycles is associated with a corresponding subset of less that all of a plurality of parity-check constraints of the code.

39. The method of claim 38 wherein the intermediate value data includes values each associated with a different one of the parity check constraints of the code.

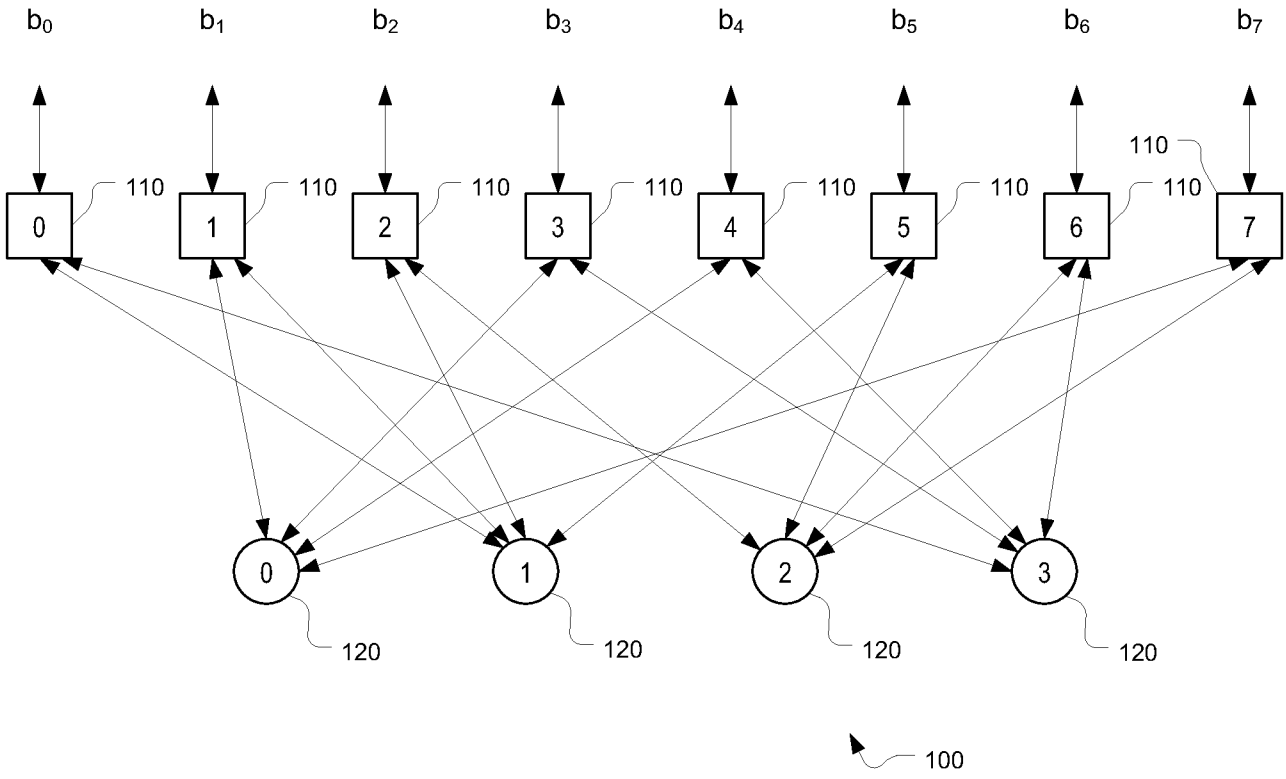

FIG. 1

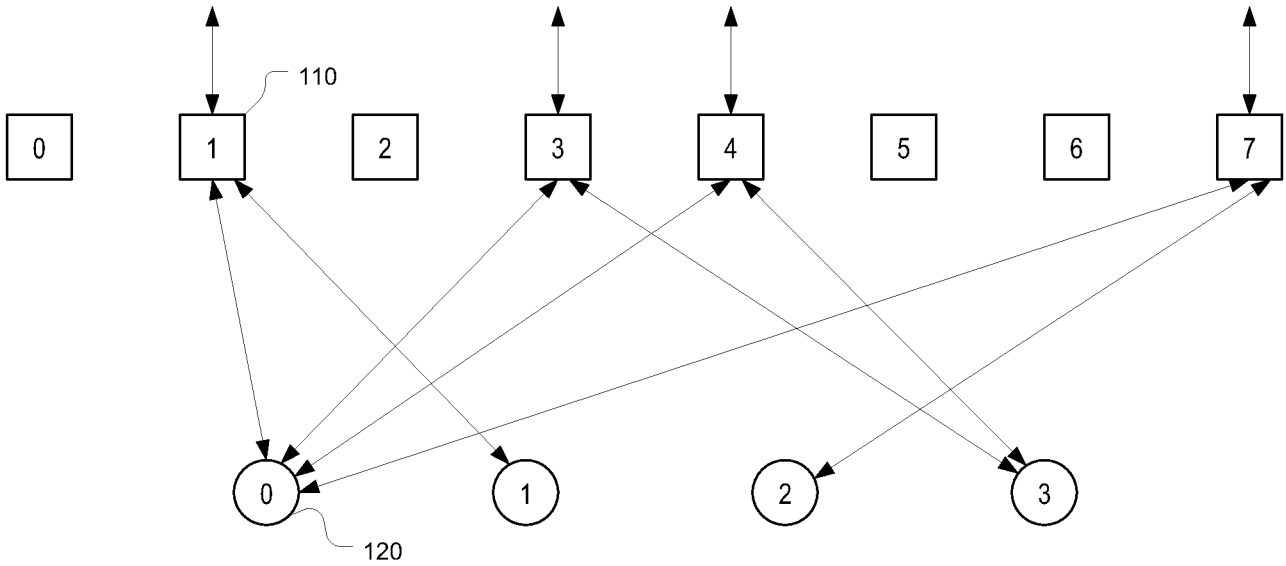

FIG. 3

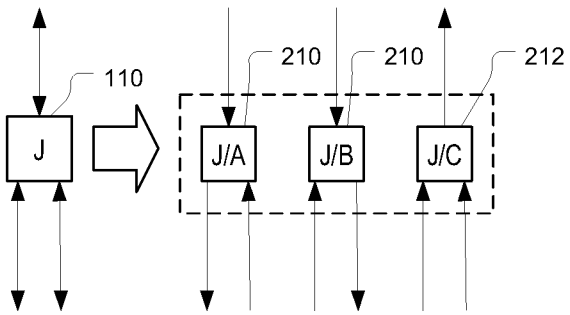

FIG. 2A

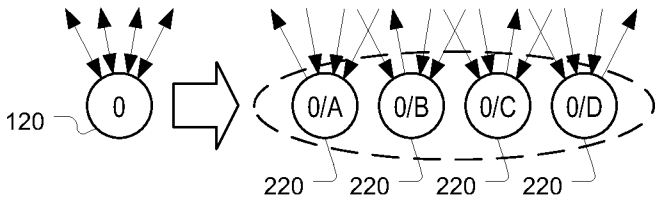

FIG. 2B

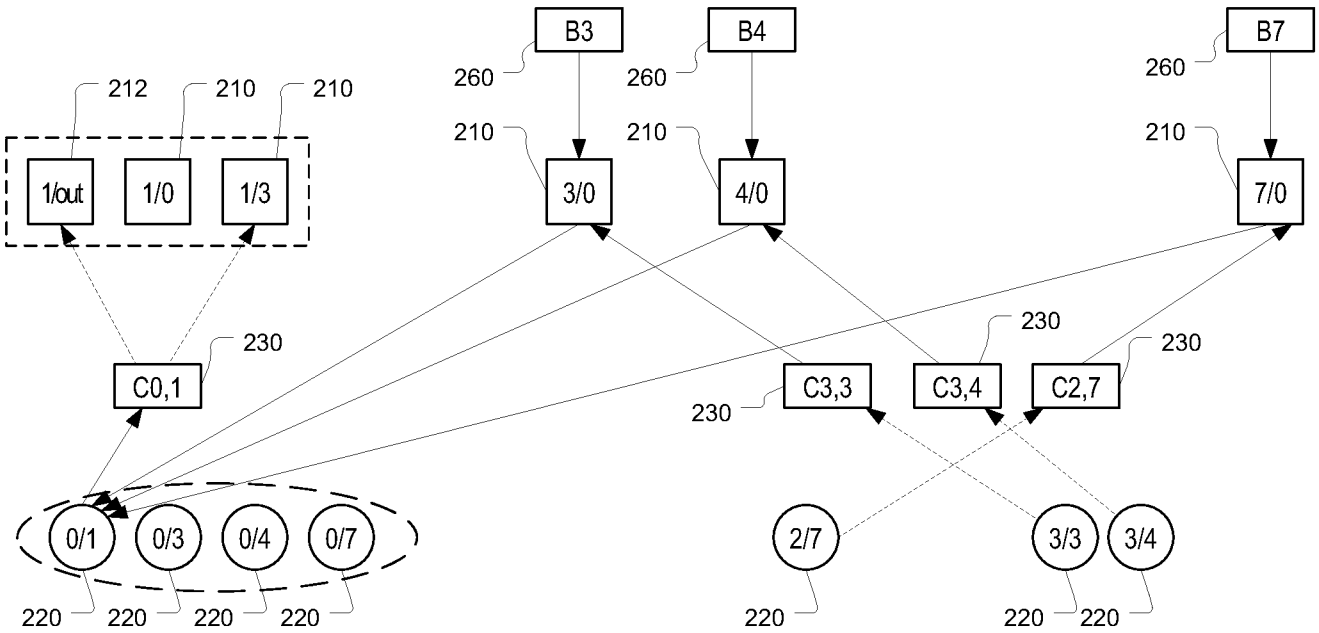
B3
B4
B7
260
260
260
212
210
210
210
210
1/out
1/0
1/3
3/0
4/0
7/0
230
230
230
230
C0,1
C3,3
C3,4
C2,7
0/1
0/3
0/4
0/7
2/7
3/3
3/4
220
220
220
220
220
220
220

FIG. 4

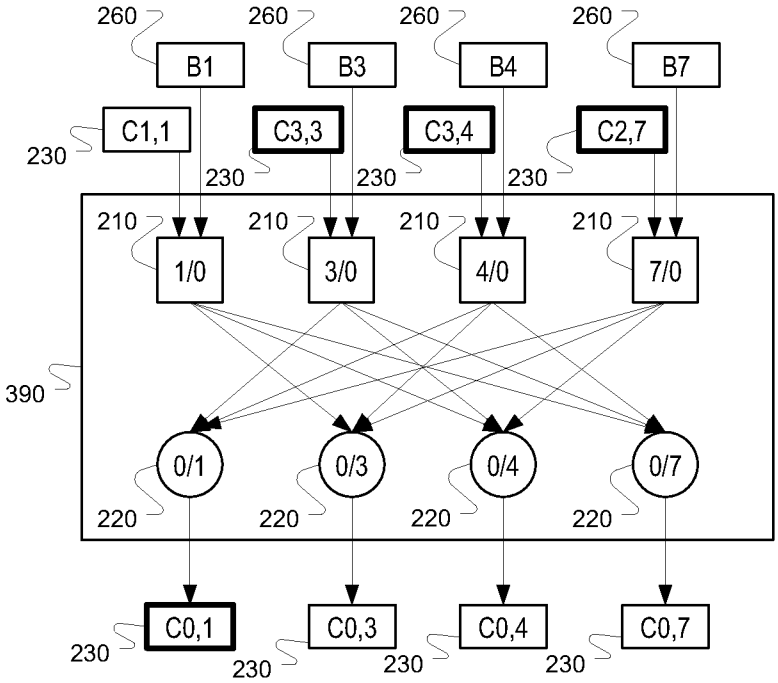

FIG. 5

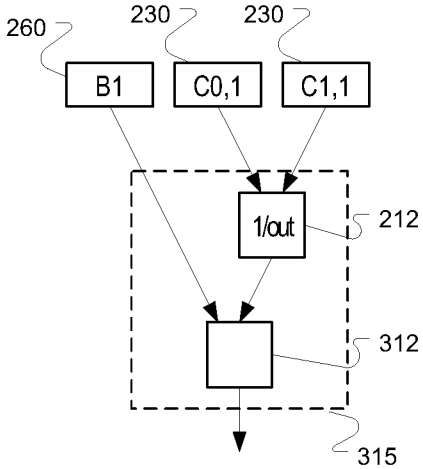

FIG. 6

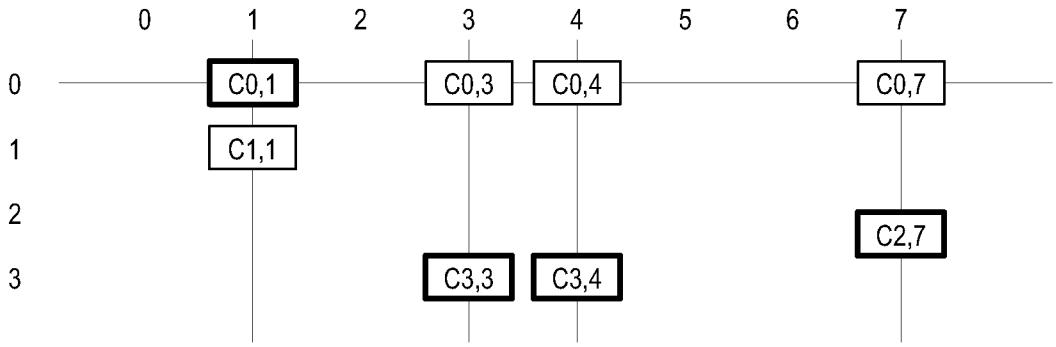

FIG. 7A

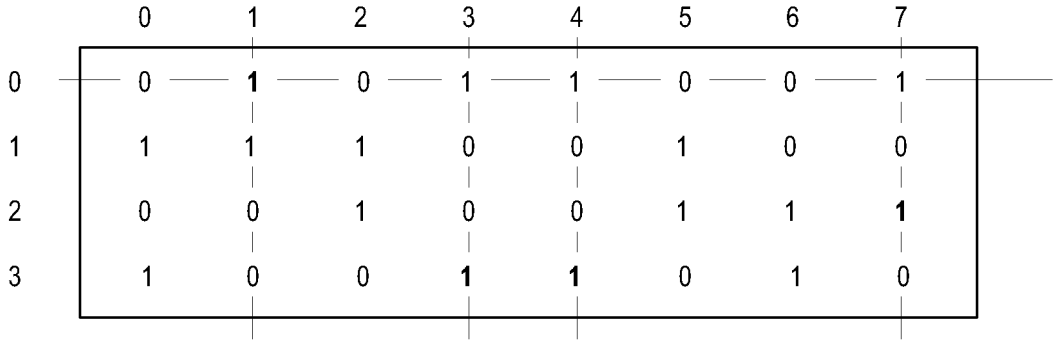

FIG. 7B

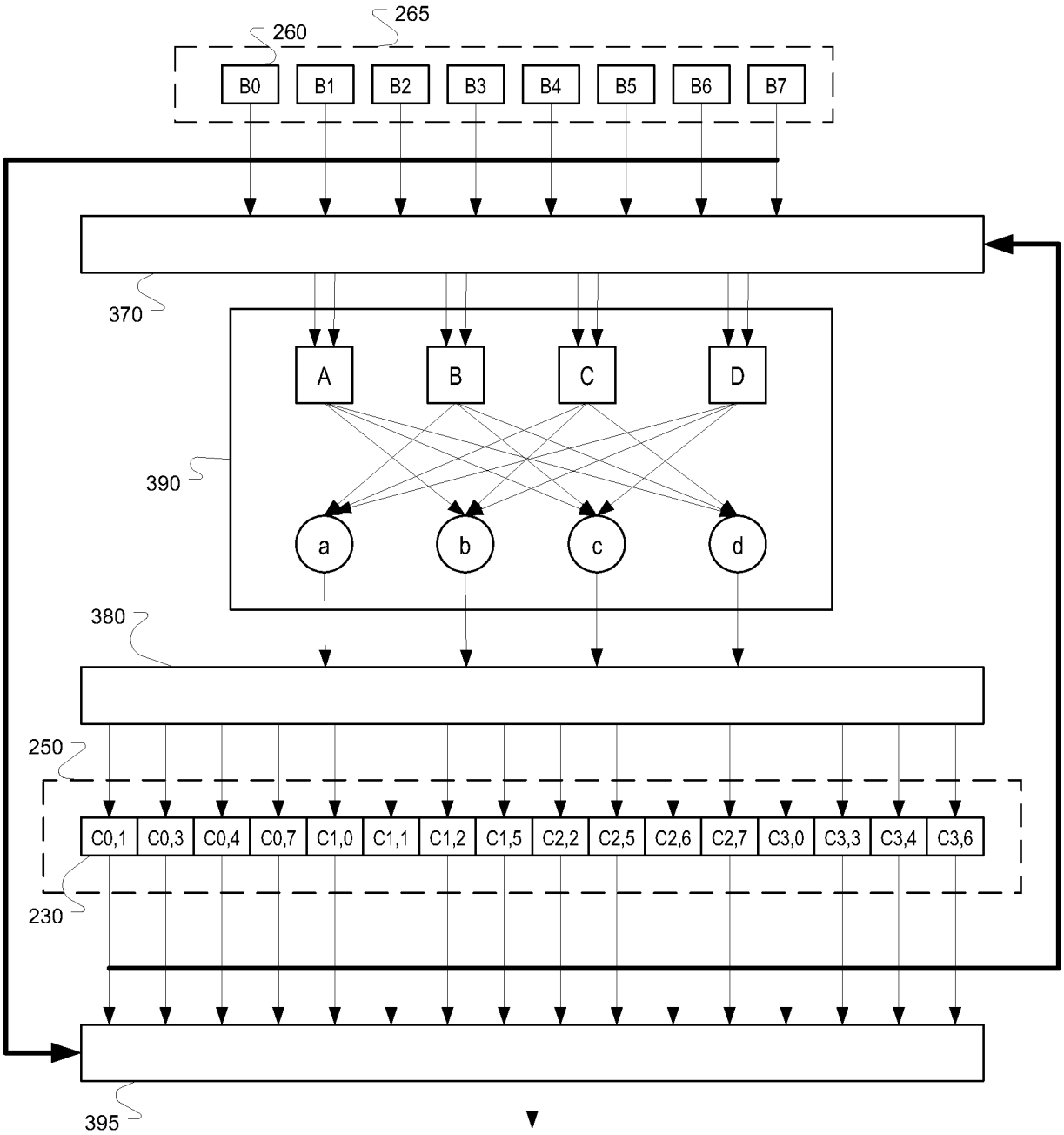
265
260
B0
B1
B2
B3
B4
B5
B6
B7
370
A
B
C
D
390
a
b
c
d
380
250
C0,1
C0,3
C0,4
C0,7
C1,0
C1,1
C1,2
C1,5
C2,2
C2,5
C2,6
C2,7
C3,0
C3,3
C3,4
C3,6
230
395

FIG. 8

	Clock Cycle			
	0	**1**	**2**	**3**
variable node A inputs	B1	B0	B2	B0
	C1,1	C3,0	C1,2	C1,0
variable node B inputs	B3	B1	B5	B3
	C3,3	C0,1	C1,5	C0,3
variable node C inputs	B4	B2	B6	B4
	C3,4	C2,2	C3,6	C0,4
variable node D inputs	B7	B5	B7	B6
	C2,7	C2,5	C0,7	C2,6
check node a output	C0,1	C1,0	C2,2	C3,0
check node b output	C0,3	C1,1	C2,5	C3,3
check node c output	C0,4	C1,2	C2,6	C3,4
check node d output	C0,7	C1,5	C2,7	C3,6

FIG. 9

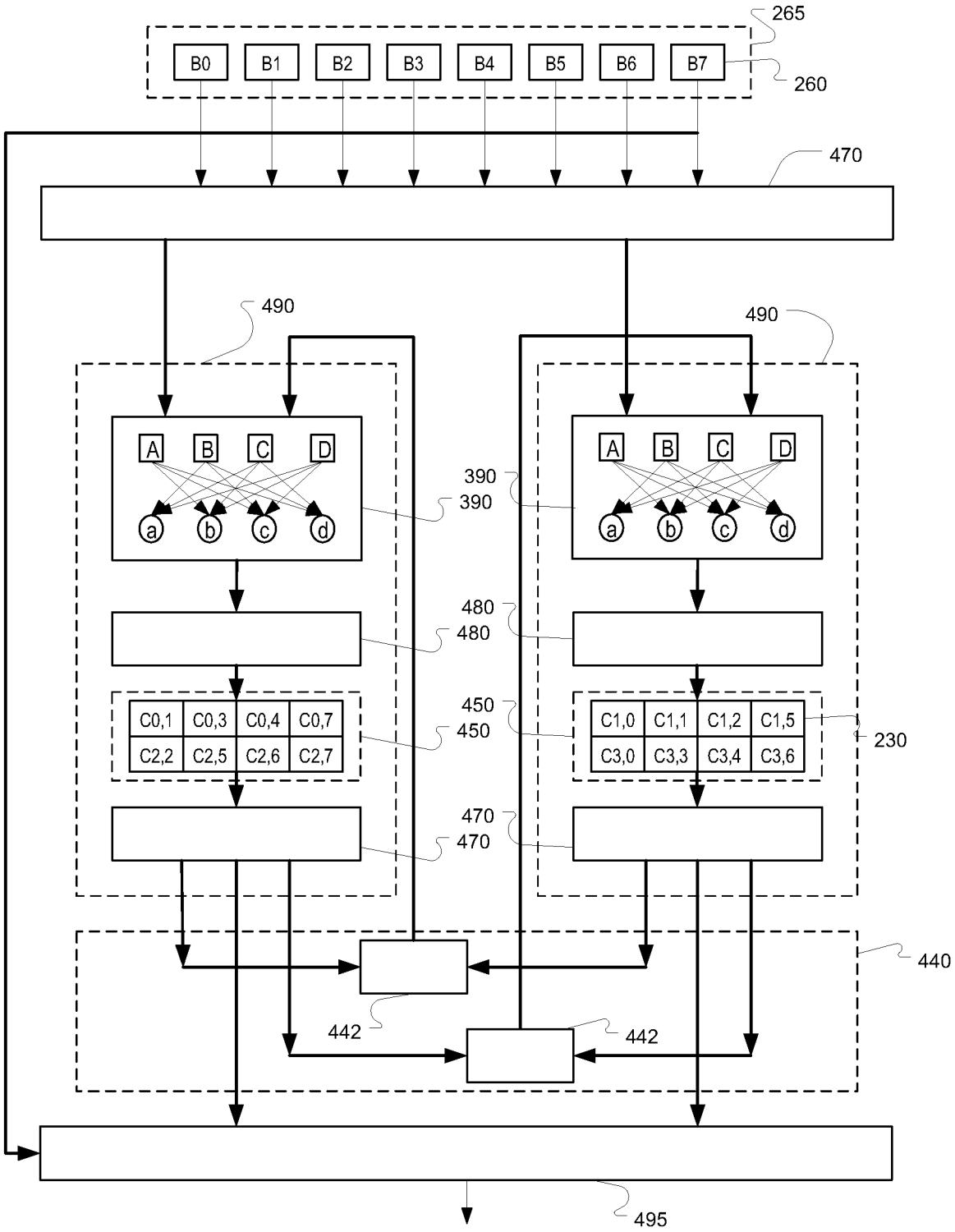
265
B0
B1
B2
B3
B4
B5
B6
B7
260
470
490
490
A
B
C
D
a
b
c
d
390
390
480
480
C0,1
C0,3
C0,4
C0,7
C2,2
C2,5
C2,6
C2,7
450
450
C1,0
C1,1
C1,2
C1,5
C3,0
C3,3
C3,4
C3,6
230
470
470
440
442
442
495

FIG. 10

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0		8		21		22		16		37		21		6		43	0						
1		31		40		15		1		7		16		18		22		0	0					
2	4		39		28		12		38		7		15		33				0	0				
3		12		14		37		12		40		2		25		16				0	0			
4	10		13		6		13		30		11		22		28						0	0		
5		13		29		24		6		0		13		33		0						0	0	
6	14		0		6		25		38		2		2		23		0						0	0
7		0		21		5		27		38		25		35		18	43							0

FIG. 11

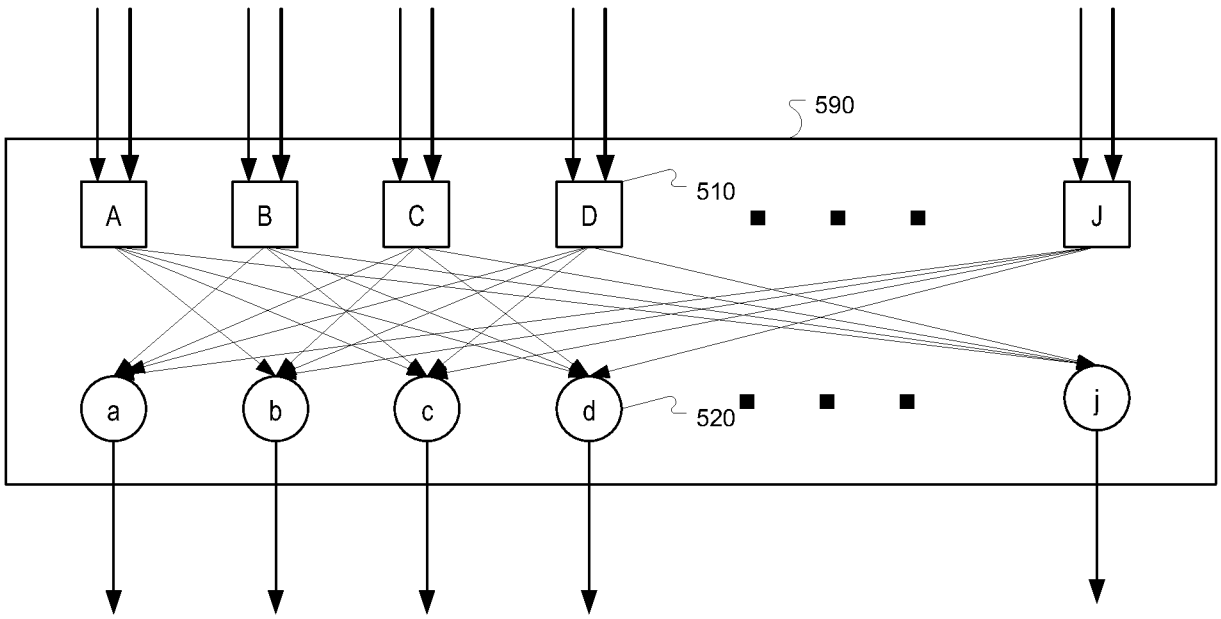

FIG. 12

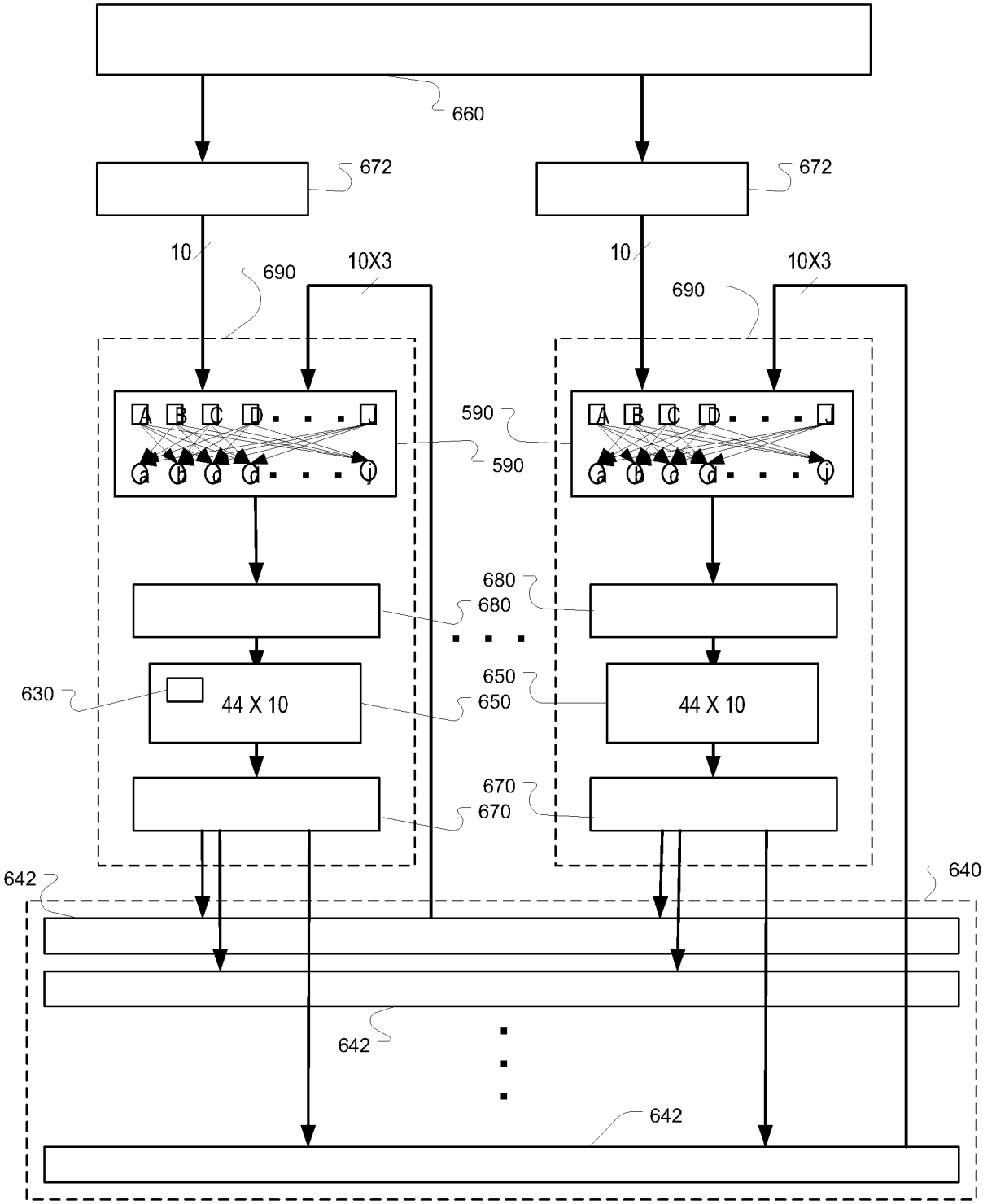
660
672
672
10
10
690
10X3
690
10X3
590
590
680
680
630
44 X 10
650
650
44 X 10
670
670
642
640
642
642

FIG. 13

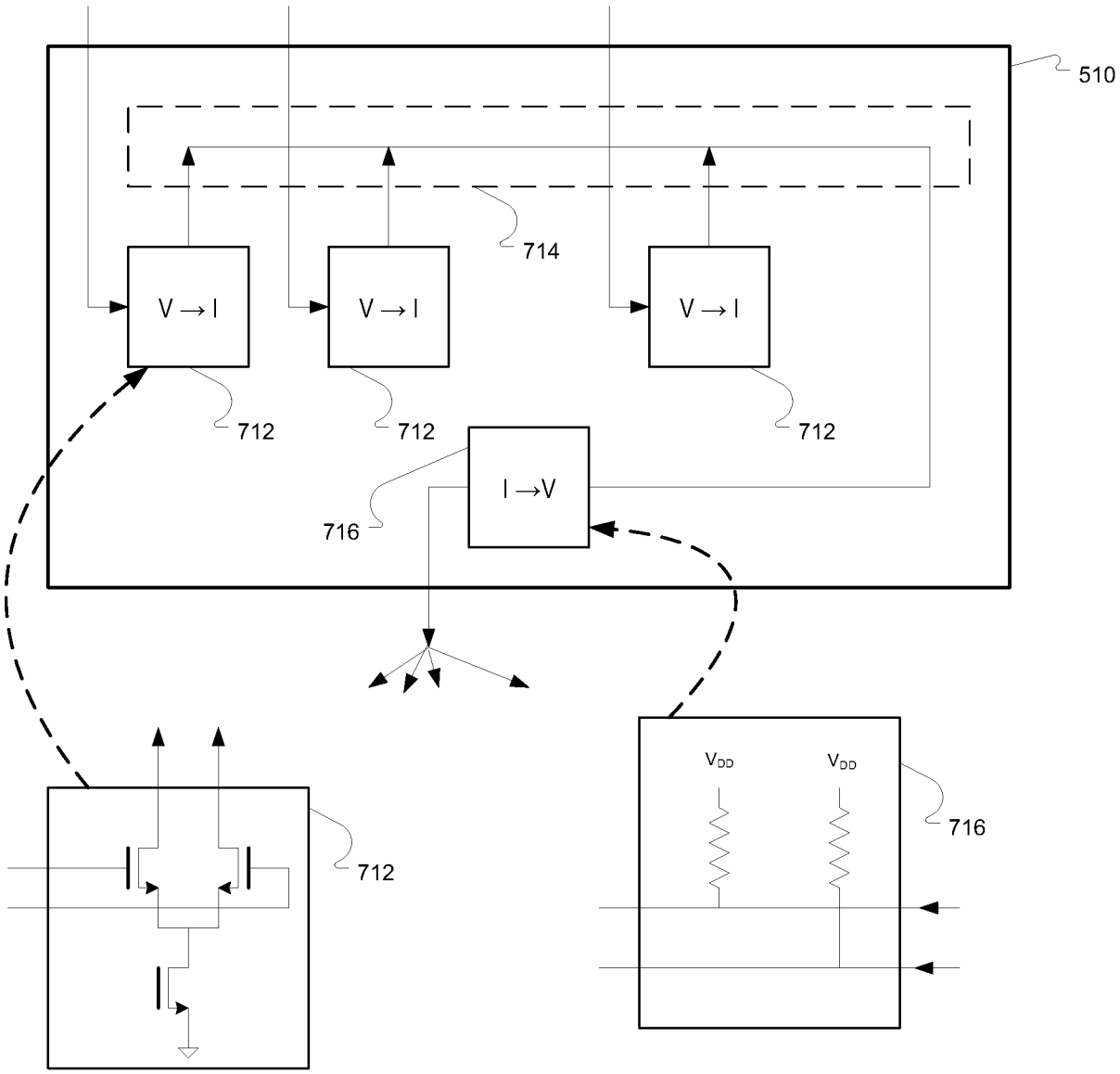

FIG. 14

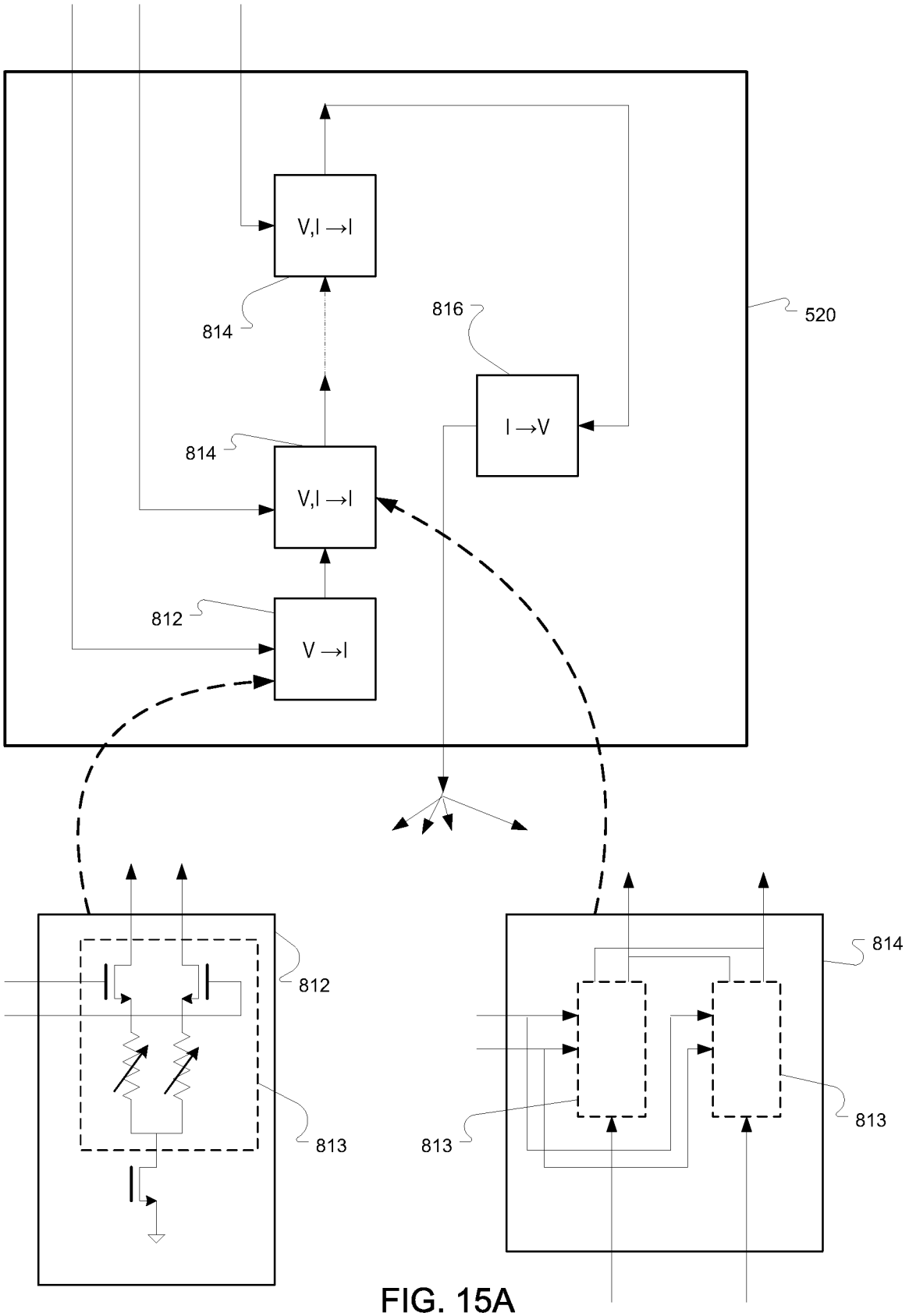

FIG. 15A

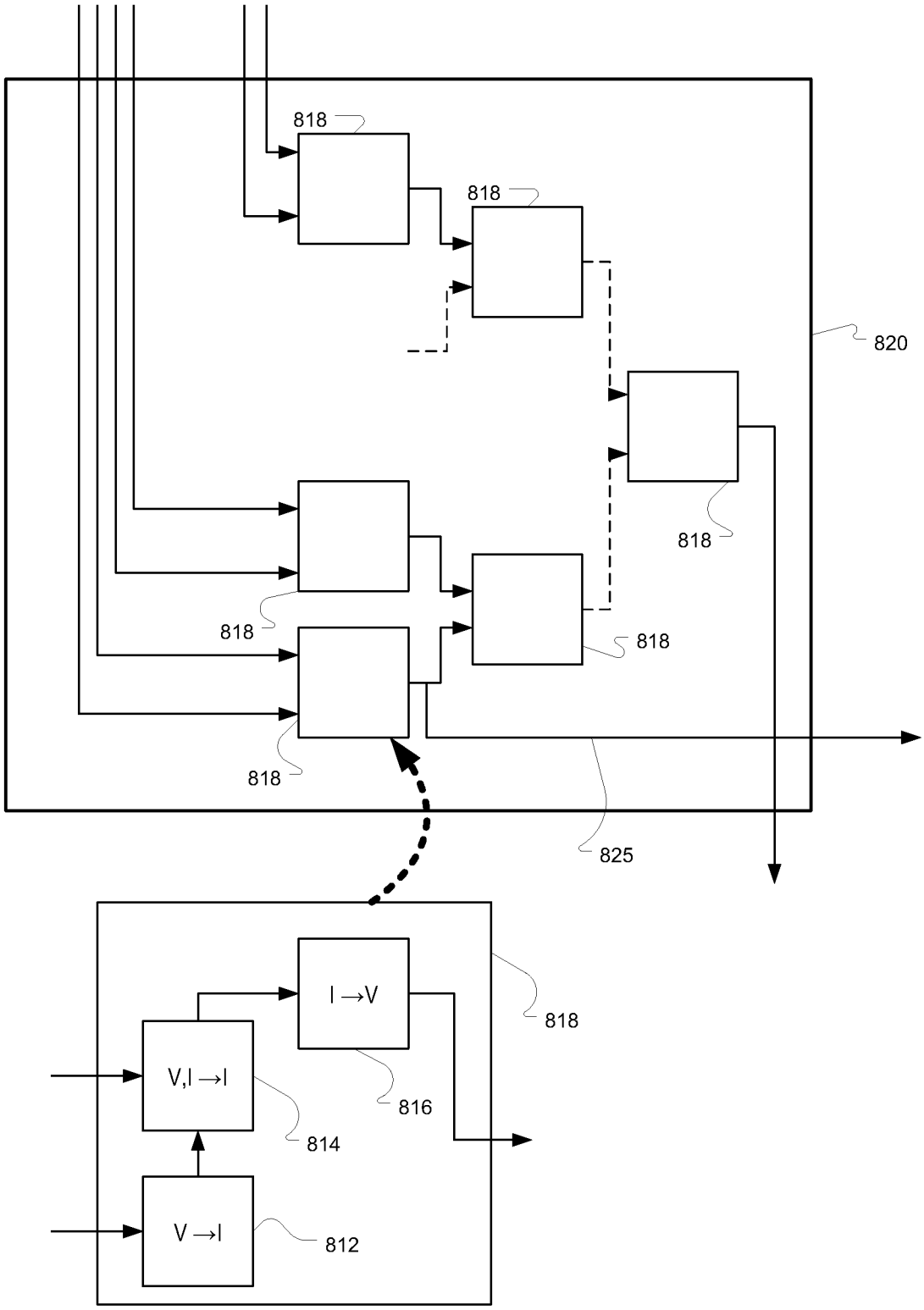

FIG. 15B

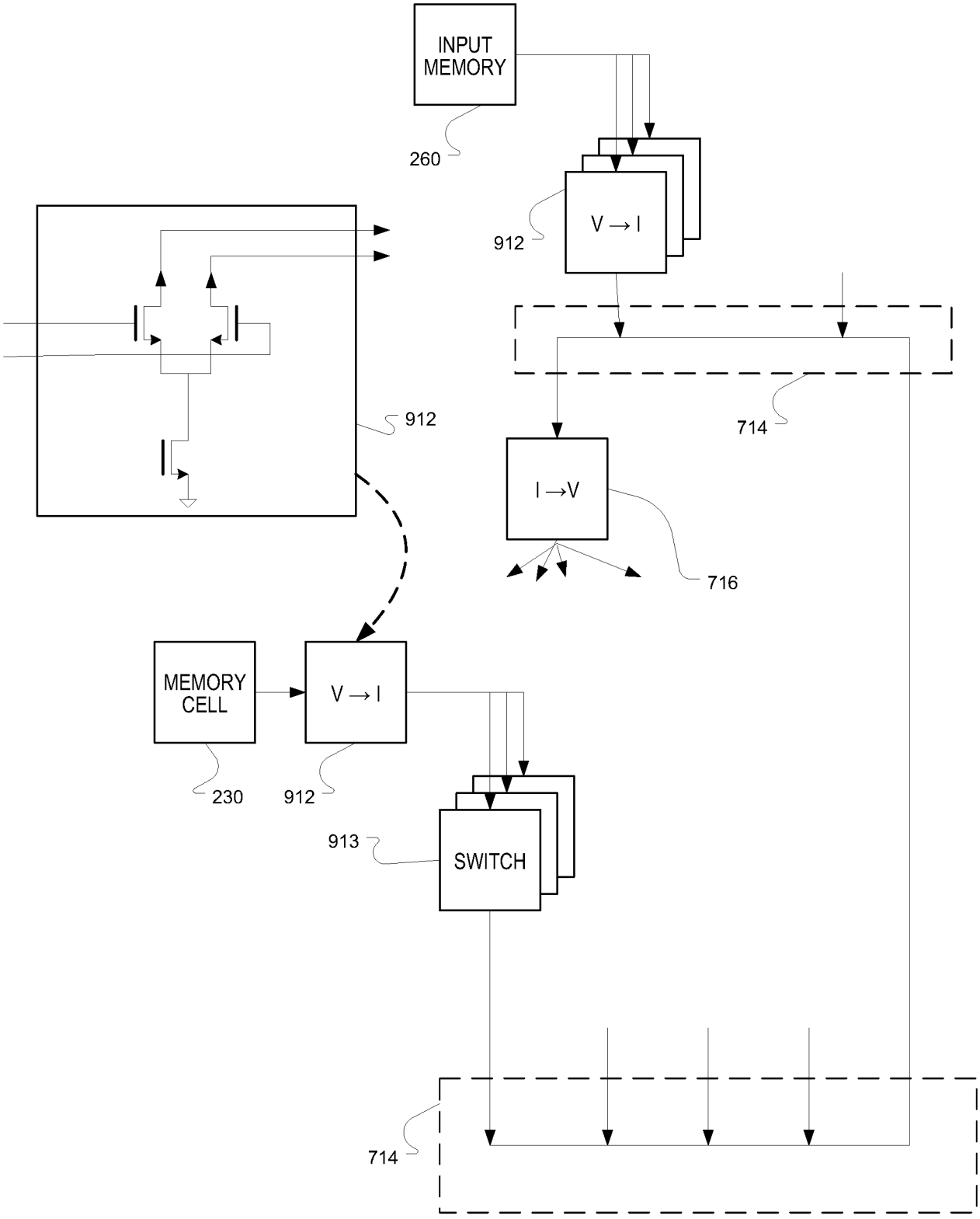
INPUT MEMORY
260
912
V → I
912
714
I →V
716
MEMORY CELL
V → I
230
912
913
SWITCH
714

FIG. 16

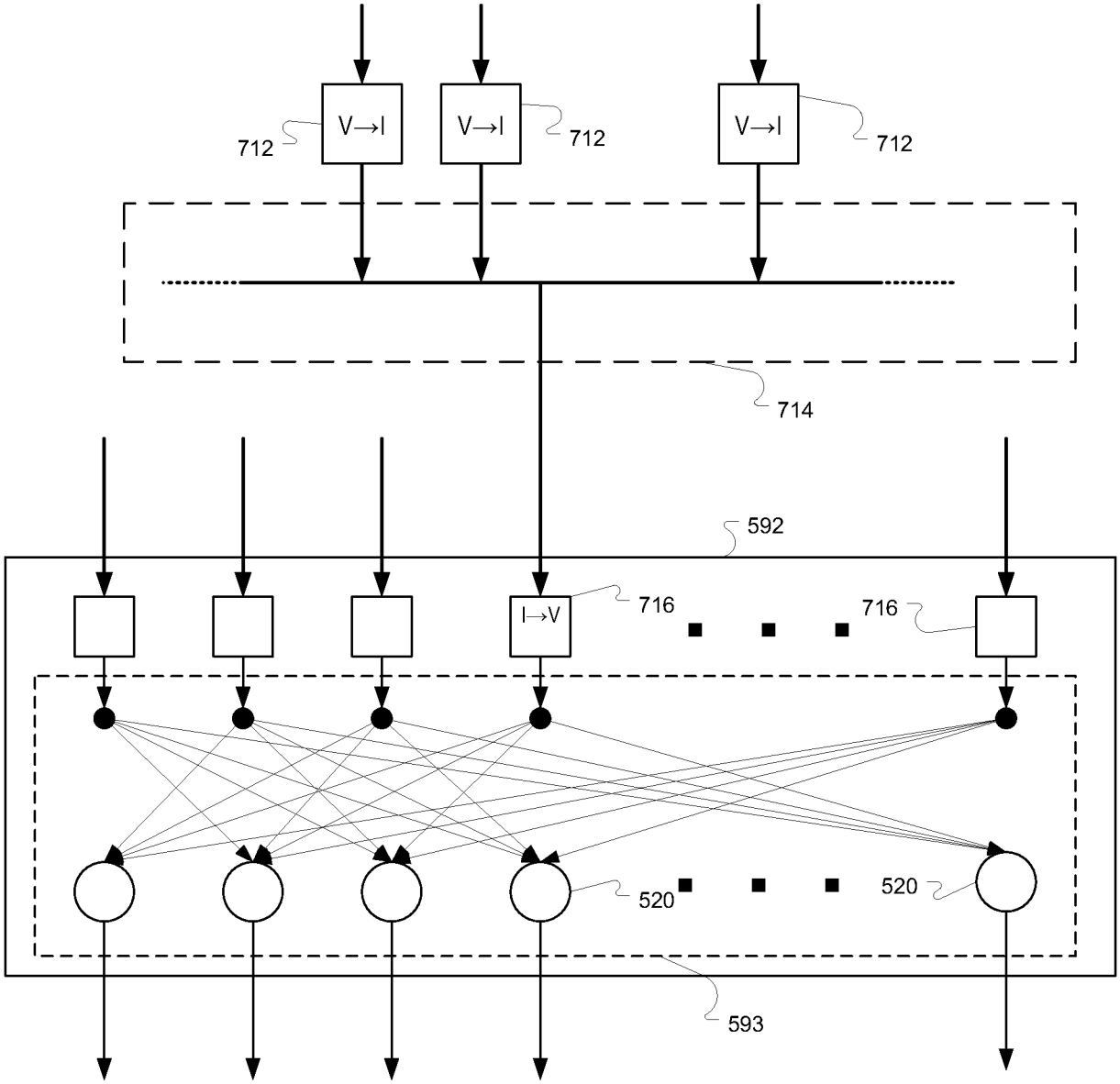

FIG. 17

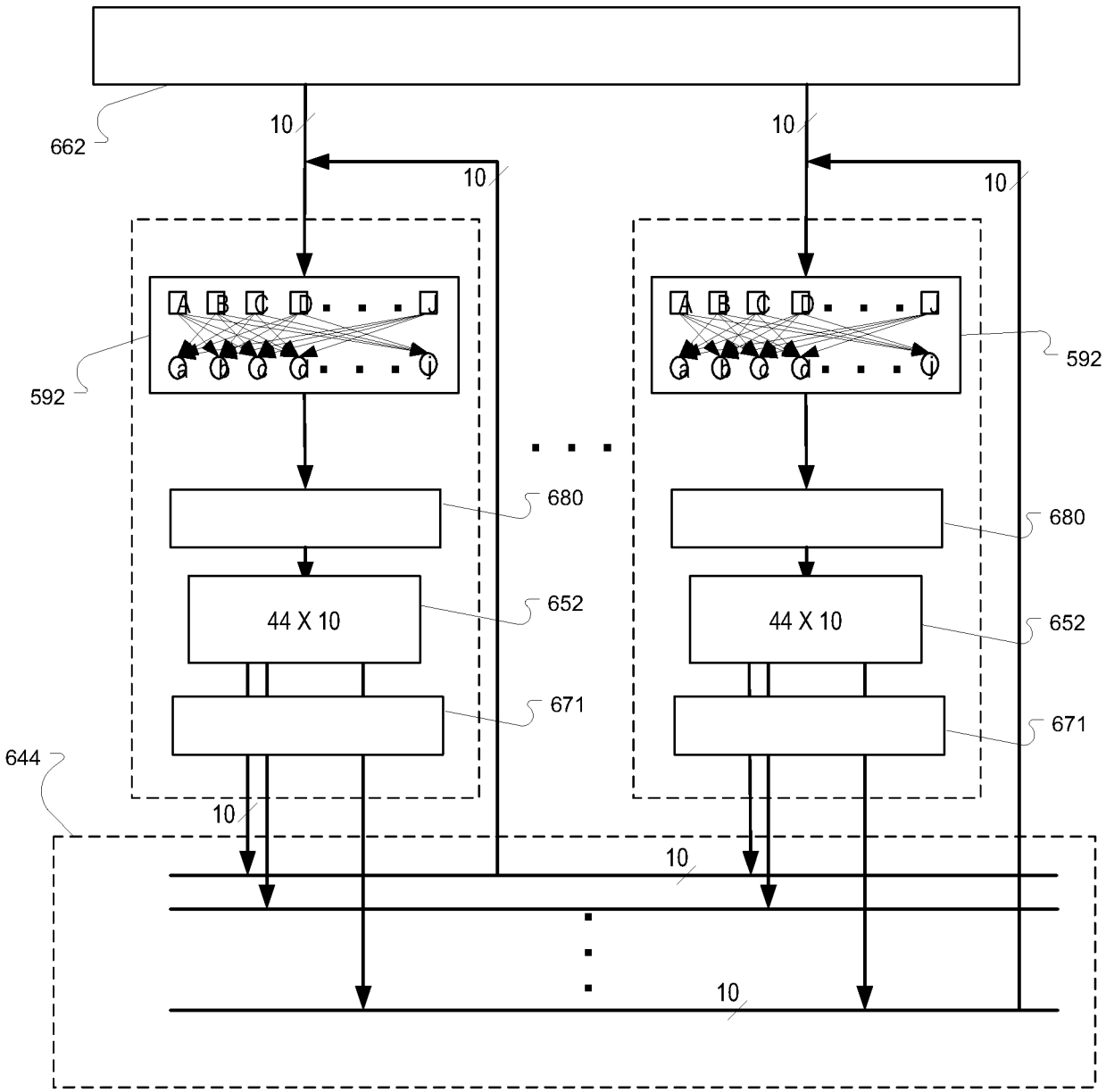
662
10
10
10
10
592
592
680
680
44 X 10
44 X 10
652
652
671
671
644
10
10
10

FIG. 18

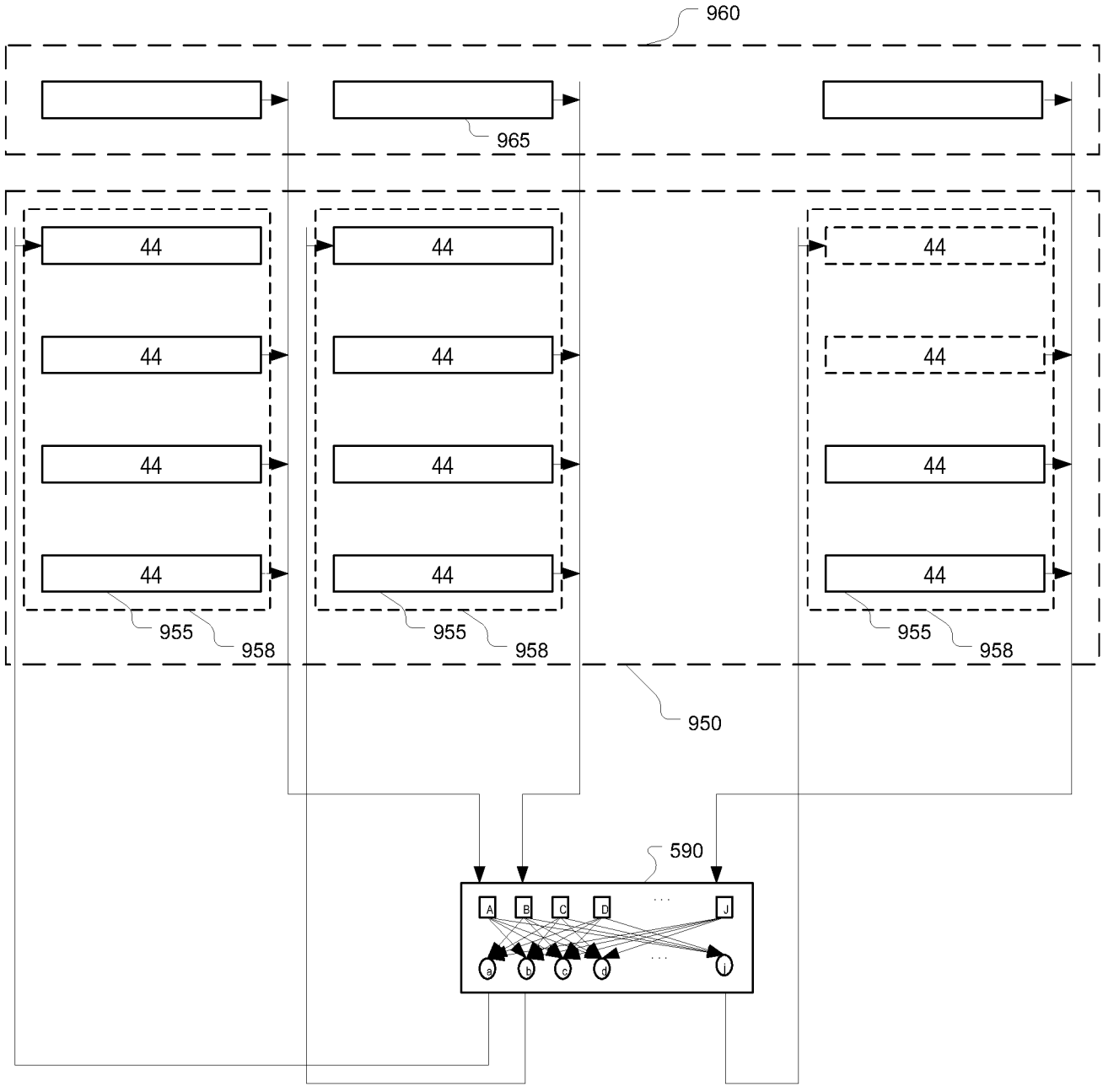

FIG. 19

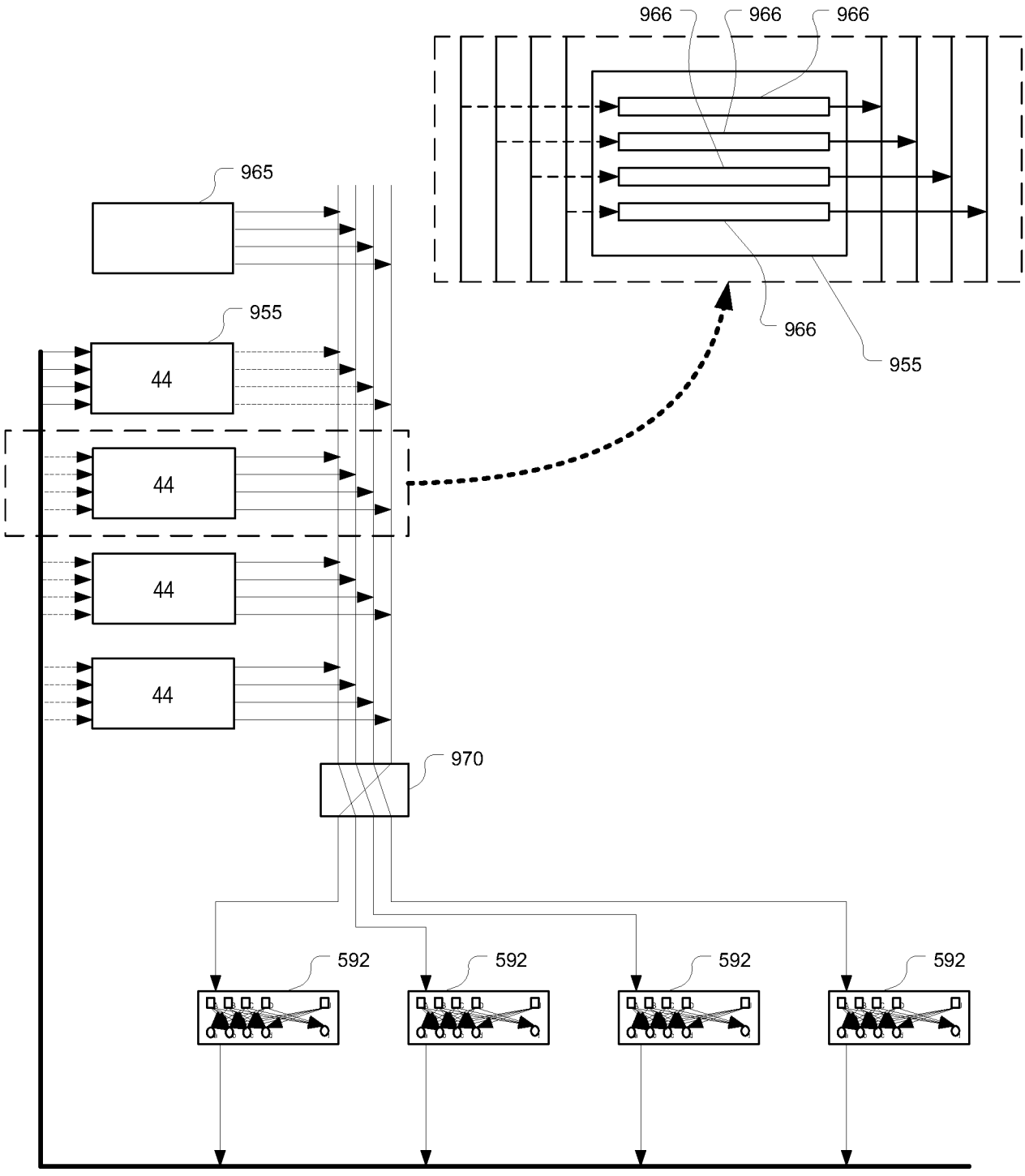

FIG. 20

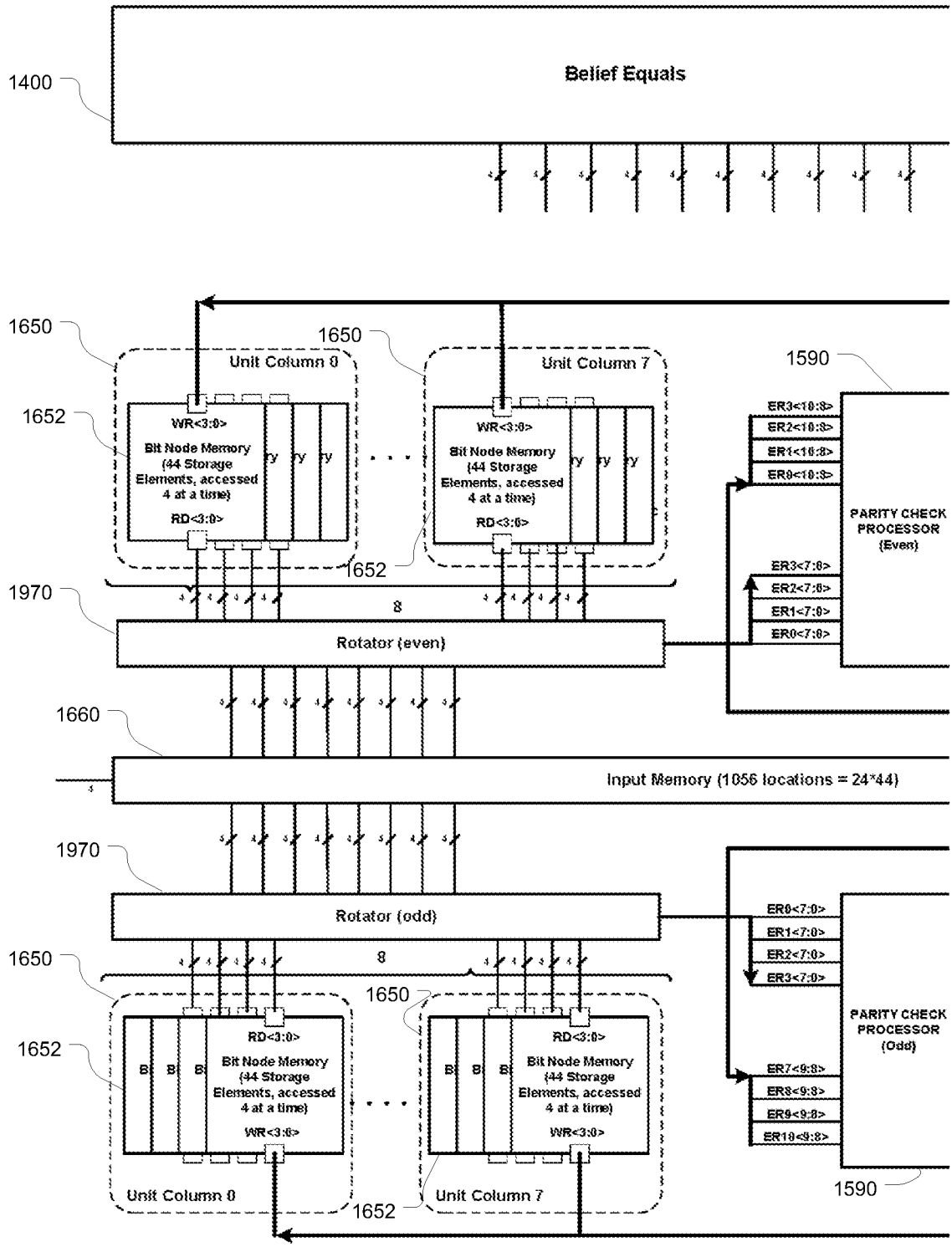
1400
Belief Equals
1650
1650
Unit Column 0
Unit Column 7
1590
1652
WR<3:0>
Bit Node Memory
(44 Storage
Elements, accessed
4 at a time)
RD<3:0>
ER3<10:8>
ER2<10:8>
ER1<10:8>
ER0<10:8>
PARITY CHECK
PROCESSOR
(Even)
1652
1970
8
Rotator (even)
ER3<7:0>
ER2<7:0>
ER1<7:0>
ER0<7:0>
1660
Input Memory (1056 locations = 24*44)
1970
Rotator (odd)
ER0<7:0>
ER1<7:0>
ER2<7:0>
ER3<7:0>
1650
8
1650
1652
RD<3:0>
Bit Node Memory
(44 Storage
Elements, accessed
4 at a time)
WR<3:0>
PARITY CHECK
PROCESSOR
(Odd)
ER7<9:8>
ER8<9:8>
ER9<9:8>
ER10<9:8>
Unit Column 0
Unit Column 7
1652
1590

FIG. 21A

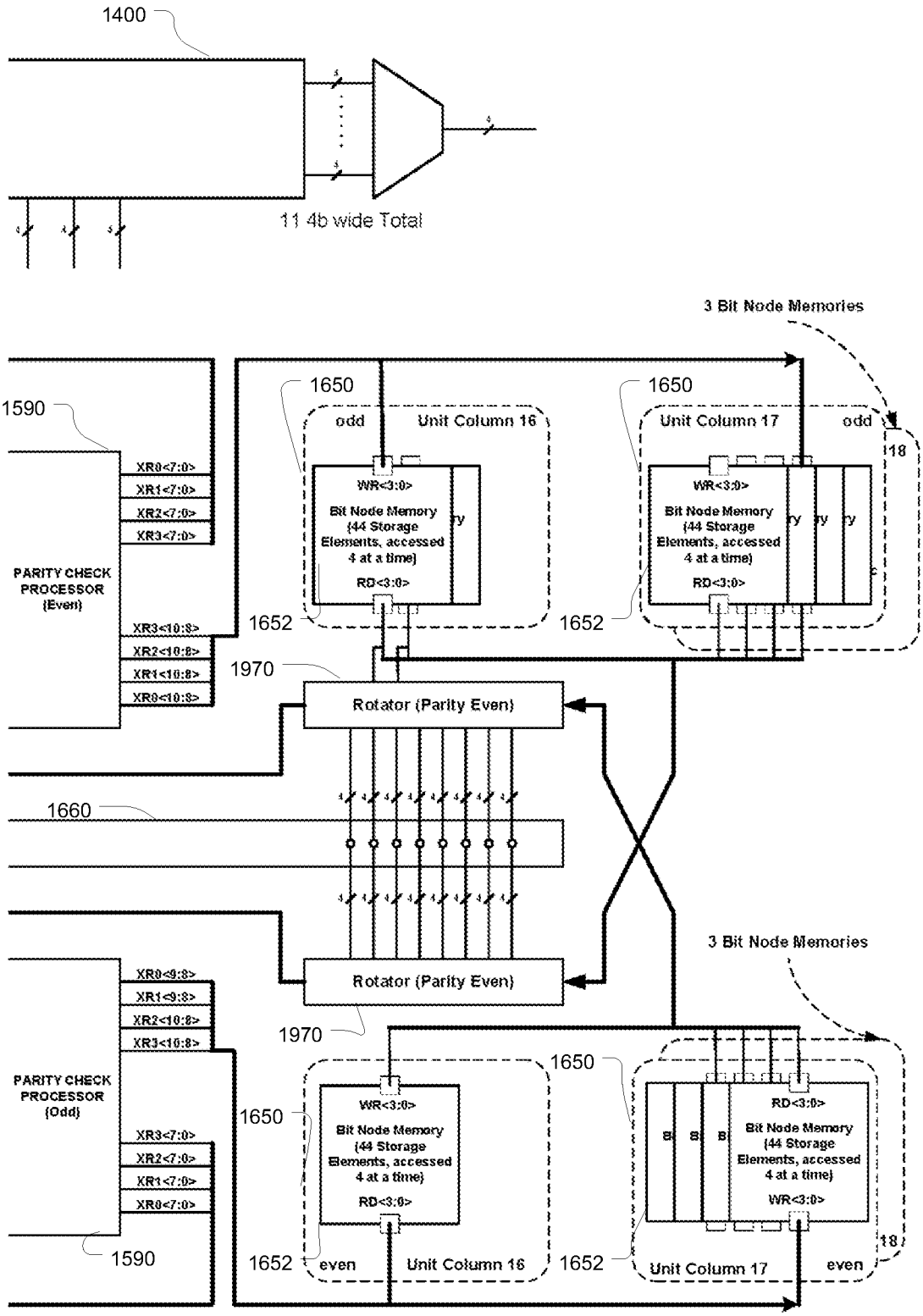
1400
11 4b wide Total
3 Bit Node Memories
1590
1650
1650
odd
Unit Column 16
Unit Column 17
odd
18
WR<3:0>
Bit Node Memory
(44 Storage
Elements, accessed
4 at a time)
RD<3:0>
1652
1652
XR0<7:0>
XR1<7:0>
XR2<7:0>
XR3<7:0>
PARITY CHECK
PROCESSOR
(Even)
XR3<10:8>
XR2<10:8>
XR1<10:8>
XR0<10:8>
1970
Rotator (Parity Even)
1660
Rotator (Parity Even)
3 Bit Node Memories
XR0<9:8>
XR1<9:8>
XR2<10:8>
XR3<10:8>
1970
PARITY CHECK
PROCESSOR
(Odd)
1650
1650
XR3<7:0>
XR2<7:0>
XR1<7:0>
XR0<7:0>
1590
1652
even
Unit Column 16
1652
Unit Column 17
even
18

FIG. 21B

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 10/25956

A. CLASSIFICATION OF SUBJECT MATTER
IPC(8) - H03M 13/00 (2010.01)
USPC - 714/763
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)
USPC: 714/763 IPC: H03M 13/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
USPC: 714/763; 706/521; 711/154; H03M 13/00 (keyword limited; terms below)

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
PubWEST; Dialog: Google, Google Scholar, Google Patents;
Search Terms: Statistical, inference, stochastic, probability, probabilities, bayesian, belief, propagation, graph, abstract, nose, processor, cpu, microcontroller, analog, memory, value, node, subset, group, first, 1st, second, 2nd, output, voltage, current, LDPC, Low, density

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	US 2008/0174460 A1 (Vigoda) 24 July 2008 (24.07.2008), entire document, especially Abstract, Fig 1A-1B and 21, para [0008]-[0119]	1-34 and 37-39
Y	US 4,649,515 A (Thompson et al) 10 March 1987 (10.03.1987), entire document, especially Fig 1-2, col 4, ln 28-32 and col 5, ln 4-6)	1-34 and 37-39
Y	US 2004/0136472 A1 (Vigoda et al.) 15 July 2004 (15.07.2004), entire document, especially para [0035]	24-25
A	US 6,938,196 B2 (Richardson et al.) 30 August 2005 (30.08.2005), entire document	1-34 and 37-39
A	Vigoda, B., Analog Logic: Continuous-Time Analog Circuits for Statistical Signal Processing, Massachusetts Institute of Technology [online], September 2003 [retrieved on 15 April 2010 (15.04.2010)] Retrieved from the Internet: <URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.3503&rep=rep1&type=pdf.>, entire document	1-34 and 37-39
A	Luckenbill, S., Building Bayesian Networks with Analog Subthreshold CMOS Circuits, Yale University [online], May 2002 (retrieved on 15 April 2010 (15.04.2010)] Retrieved from the Internet: <URL: http://www.eng.yale.edu/pjk/EESrProj_02/luckenbill.pdf>, entire document	1-34 and 37-39

☐ Further documents are listed in the continuation of Box C. ☐

* 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 application or patent 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
15 April 2010 (15.04.2010)	11 MAY 2010
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US, Commissioner for Patents P.O. Box 1450, Alexandria, Virginia 22313-1450 Facsimile No. 571-273-3201	Authorized officer: Lee W. Young PCT Helpdesk: 571-272-4300 PCT OSP: 571-272-7774

Form PCT/ISA/210 (second sheet) (July 2009)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US 10/25956

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☒ Claims Nos.: 35-36
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.