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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 November 2007 (01.11.2007)

PCT

(10) International Publication Number
WO 2007/124048 A2

(51) International Patent Classification:
*H01L 51/00* (2006.01)

(21) International Application Number:
PCT/US2007/009658

(22) International Filing Date: 19 April 2007 (19.04.2007)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/793,665 19 April 2006 (19.04.2006) US
60/919,225 21 March 2007 (21.03.2007) US

(71) Applicant (*for all designated States except US*): **TRUSTEES OF PRINCETON UNIVERSITY** [US/US]; 4 New South Building, Princeton, New Jersey 08540 (US).

(72) Inventors; and
(75) Inventors/Applicants (*for US only*): **ZHANG, Wei** [CN/US]; 220A Eisenhower Street, Princeton, New Jersey 08540 (US). **JHA, Niraj K.** [IN/US]; 10 Floral Court, Westfield, New Jersey 07090 (US). **SHANG, Li** [CN/CA]; 310 Bath Road, Apt. 703, Kingston, Ontario K7M 9H1 (CA).

(74) Agents: **GEORGE, Bruce D.** et al.; Blank Rome LLP, One Logan Square, Philadelphia, Pennsylvania 19103 (US).

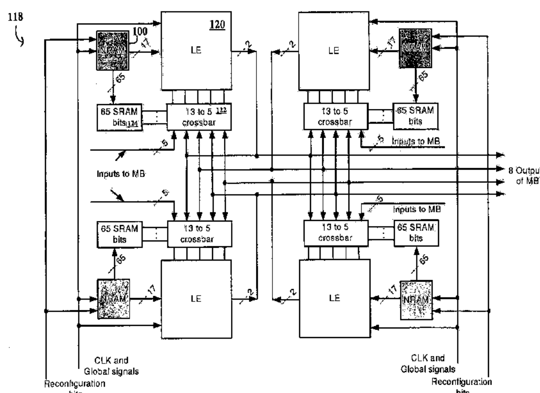
(81) Designated States (*unless otherwise indicated, for every kind of national protection available*): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, 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, HU, IE, IS, IT, LT, LU, LV, MC, MT, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

**Published:**
— *without international search report and to be republished upon receipt of that report*

*[Continued on next page]*

(54) Title: A HYBRID NANOTUBE/CMOS DYNAMICALLY RECONFIGURABLE ARCHITECTURE AND AN INTEGRATED DESIGN OPTIMIZATION METHOD AND SYSTEM THEREFOR




(57) Abstract: A hybrid nanotube, high-performance, dynamically reconfigurable architecture, NATURE, is provided, and a design optimization flow method and system, NanoMap. A run-time reconfigurable architecture is provided by associating a non-volatile universal memory to each logic element to enable cycle-by-cycle reconfiguration and logic folding, while remaining CMOS compatible. Through logic folding, significant logic density improvement and flexibility in performing area-delay tradeoffs are possible. NanoMap incorporates temporal logic folding during the logic mapping, temporal clustering and placement steps. NanoMap provides for automatic selection of a best folding level, and uses force-direct scheduling to balance resources across folding stages. Mapping can thereby target various optimization objectives and user constraints. A high-density, high-speed carbon nanotube RAM can be implemented as the universal memory, allowing on-chip multi-context configuration storage, enabling fine-grain temporal logic folding, and providing a significant increase in relative logic density.

WO 2007/124048 A2

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

# A HYBRID NANOTUBE/CMOS DYNAMICALLY RECONFIGURABLE ARCHITECTURE AND AN INTEGRATED DESIGN OPTIMIZATION METHOD AND SYSTEM THEREFOR

## RELATED APPLICATIONS

**[0001]** This application claims benefit of U.S. Provisional Application Serial Nos. 60/793,665, filed April 19, 2006, entitled "A Hybrid Nanotube/CMOS Dynamically Reconfigurable Architecture;" and 60/919,225, filed March 21, 2007, entitled "NanoMap: An Integrated Design Optimization Flow for a Hybrid Nanotube/CMOS Dynamically Reconfigurable Architecture. Each of the above-identified related applications are incorporated herein by this reference.

## GOVERNMENT RIGHTS

**[0002]** The present invention was made with Government support and the Government has certain rights in the invention.

## FIELD OF THE INVENTION

**[0003]** This invention relates generally to reconfigurable computer architectures, and particularly to CMOS compatible field-programmable gate arrays (FPGAs) having non-volatile universal memories supporting fine-grain reconfiguration to enable temporal logic folding, along with an RTL/gate-level automatic design optimization method and system.

## BACKGROUND OF THE INVENTION

**[0004]** After consistently providing large improvements in productivity and performance for more than two decades, CMOS is expected to approach its physical limits in the coming decade. To enable future technology scaling, intensive research is being directed towards the development of nanoscale molecular devices, such as carbon nanotube and nanowire. Such nanodevices demonstrate superior characteristics over MOSFET in terms of integration density, performance, power consumption, etc. However, lack of a mature fabrication process is a roadblock in implementing chips using these nanodevices. If photo-lithography could be used to implement structures made from these nanodevices, then such structures could be combined with CMOS logic to create hybrid CMOS/nanochips, which could leverage the beneficial aspects of both technologies.

[0005] Motivated by the impressive potential of nanotechnologies, researchers are investigating nanoelectronic circuits and architectures. If such circuits/architectures are implemented using bottom-up chemical self-assembly techniques, then the chip defect levels are expected to be high (between 1% and 10%). To be able to deal with such high defect levels, regular architectures are favored. Reconfigurable architectures, in addition to being regular, allow reconfiguration around fabrication defects as well as run-time faults. Thus, both regular and reconfigurable architectures have found popularity.

## SUMMARY OF THE INVENTION

[0006] The present invention provides a hybrid CMOS/non-volatile universal memory reconfigurable architecture, referred to as NATURE. In one embodiment, the present invention is based on CMOS logic and high-density high-speed non-volatile nanotube random-access memory. In one instance, NRAM® of Nantero, Inc., identifies a source of nanotube random-access memory chips. Nanotube random-access memory chips can be fabricated using CMOS-compatible manufacturing processes. Thus, architectures of the present invention can also be fabricated with currently-available processes.

[0007] The present invention exploits the excellent properties of non-volatile universal memories, including NRAM® chips, and distributes them in a reconfigurable fabric to act as on-chip storage for multi-context reconfiguration bits. Non-volatile memories include the emerging technologies of carbon nanotube RAMs, phase change RAMs, magnetoresistive RAMs, and ferroelectric RAMs (FRAMs). Although certain illustrated embodiments of the present invention describe implementation using NRAM® chips, the present invention is not limited to such use. All alternative emerging non-volatile technologies could be implemented and are contemplated in the present invention.

[0008] The logic implemented in the logic elements of the reconfigurable architecture of the present invention can be changed every few cycles, making both coarse-grain and fine-grain dynamic reconfiguration possible. The present invention thereby addresses two primary challenges in existing CMOS-based FPGAs: logic density and efficiency of run-time reconfiguration. Traditional reconfigurable architectures only allow partial dynamic reconfiguration, (i.e., only a part of the architecture can be reconfigured at run-time) due to the area overhead associated with SRAMs that store the reconfiguration bits and the long latency of reconfiguration due to the accessing of off-chip storage. Since the access latency of on-chip

storage is small, on-chip storage provides an opportunity to store multiple logic designs in the on-chip storage, and to invoke different designs through fine-grain dynamic reconfiguration.

**[0009]** Moreover, the ability to reconfigure the architecture of the present invention every few cycles provides for temporal logic folding, (i.e., the possibility of folding the logic circuit in time and mapping each fold to the same logic elements in the architecture). This provides significant gains (an order of magnitude or more for larger circuits) in the area-time product (where time refers to circuit delay, or latency) compared to traditional reconfigurable architectures, while allowing the flexibility of trading area for performance. For instance, a large logic circuit can be partitioned into a sequence of logic stages and stored in the on-chip configuration memory. At run-time, stage-by-stage, the logic circuit can be configured into the same hardware and executed in different clock cycles. Logic folding increases logic elements utilization, providing high logic density and a capability of using cheaper chips, having smaller capacities, to execute similar applications, hence, making them attractive for use in cost-conscious embedded systems.

**[0010]** In one exemplary realization, the architecture of NATURE includes island-style logic blocks, connected by a hierarchical reconfigurable interconnect fabric, where each logic block contains a super-macroblock (SMB) and a local switch matrix. The SMB includes a two-level logic cluster. The first level consists of a set of macroblocks (MBs). Each MB is composed of a set of logic elements (LEs). Low-latency reconfigurable crossbars are used to form local inter-MB and inter-LE connections. In NATURE, LE is the atomic functional element, and includes look-up tables (LUTs) and flip-flops. Each m-input LUT can realize any m-variable Boolean functions. Flip-flops are used to hold computation results which are used by subsequent cycles.

**[0011]** In this exemplary realization, support for reconfiguration is provided by using nanotube random-access memories as on-chip configuration storage, distributed within each level of logic and interconnect hierarchy. Each individual logic or interconnect element is associated with, or physically adjacent and connected to, a $k$-set nanotube random-access memory storage. Therefore, $k$ different logic functions can be realized within the same hardware resource without accessing off-chip storage, thereby providing significant improvement in logic density with only moderate area cost and delay overhead. Also, since logic folding results in most communication being local, the need for global interconnect is greatly reduced.

**[0012]** Temporal logic folding enables a realization of different Boolean functions within the same LE in different clock cycles. For instance, traditionally a logic circuit consisting of $n$ serially-connected LUTs requires $n$ LUTs. With the temporal logic folding support of the

present invention, all $n$ LUTs can be potentially mapped to a single LE, via $n$ configuration sets stored in a respective nanotube random-access memory. The subject logic circuit can then be executed cycle-by-cycle through run-time on-chip reconfiguration.

[0013] Different folding levels result in different circuit performance and area efficiency. Given a logic circuit, increasing the folding level leads to a higher clock period, but smaller cycle count, since a larger number of logic operations need to be performed within a single clock cycle. Since a constant latency is associated with each run-time reconfiguration, the overall circuit latency decreases as the folding level increases. On the other hand, increasing the folding level can result in much higher LE resource requirements. Accordingly, design flexibility is provided, and balancing performance capabilities with area efficiencies is always a consideration.

[0014] The present invention also provides an integrated design optimization platform for NATURE, referred to as NanoMap. NanoMap conducts design optimization from the RTL down to the physical level. Given an input design specified in RTL and/or gate-level VHDL, NanoMap optimizes and implements the design on NATURE through logic mapping, temporal clustering, placement, and routing. The design optimization techniques of the present invention exploit the design flexibilities enabled by fine-grain temporal logic folding. Given user-specified area and performance constraints, the mapping method and system of NanoMap can automatically explore and identify the best logic folding configuration, and make appropriate tradeoffs between performance and area efficiency. The present invention uses a force-directed scheduling (FDS) technique to balance resource use across different logic folding cycles. Combining NanoMap with existing commercial architectural synthesis tools provides a complete design automation flow for NATURE.

[0015] Accordingly, aspects of the present invention will be seen variously to:

- provide a high-performance non-volatile memory-based reconfigurable architecture enabling run-time coarse-grain to fine-grain (i.e., cycle-by-cycle) reconfiguration and temporal logic folding;
- be reliably fabricated using CMOS-compatible manufacturing processes;
- provide flexibility in achieving different optimization objectives based upon user specified constraints;
- provide flexibility in selecting best temporal folding levels and to perform area-delay trade-offs;
- be capable of reaching LE utilization of nearly 100%;
- reduce by 50% or more a need for deep interconnect hierarchy when using level-1 folding;

- provide an order of magnitude increase in logic density relative to current technologies; and
- significantly improve area/execution time features of FPGAs.

[0016] In one aspect of the invention, a reconfigurable computer architecture, or field-programmable gate array, is provided that includes a plurality of programmable elements and at least one, separate random access memory associated with, or physically adjacent and connected to, each programmable element. The reconfigurable architecture could equally include a separate random access memory associated with each of a plurality of logic elements, or reconfigurable blocks. The random access memory is a non-volatile memory such as a carbon nanotube RAM, phase change RAM, magnetoresistive RAM, or ferroelectric RAM (FRAM). The random access memory can store run-time reconfiguration bits of the respective programmable element/logic element/reconfigurable block, or could store data on-chip, or could store both run-time reconfiguration bits and data on-chip. Further, data storage could be distributed across the respective RAM chip.

[0017] In a further aspect, n-programmable elements and n-random access memories comprise a macro-block (MB), m-macro-blocks and m-random access memories comprise a super macro-block (SMB), and one SMB and one local switch matrix comprise a logic block (LB). A plurality of LBs could be included in the architecture. In one embodiment of the invention, the value of m and n is four (4).

[0018] In another instance, the reconfigurable architecture could include a plurality of logic elements; and an equal number of random access memories, where one random access memory is physically adjacent and connected to each logic element. The random access memory stores run-time reconfiguration bits of the respective logic element. The logic element further includes two flip-flops, where different computation values are stored in each of the two flip-flops at any point in time. In an alternative embodiment, switch blocks replace the logic elements.

[0019] The present invention also provides a method of run-time reconfiguration, where reconfiguration bits are written into a first random access memory at a time of initial configuration from off-chip storage, and reconfiguration bits are placed into a second random access memory during run-time reconfiguration to configure one or more logic elements or switches to implement different logic functionality or interconnections. In one instance, reconfiguration commences at one edge of clock signal, followed by computation at another edge of the clock signal. The method could provide that the first random access memory is nanotube

random access memory, and the second random access memory is a static random access memory.

**[0020]** In another method of run-time reconfiguration, a series of n- serially connected look-up tables (LUT1, LUT2, ..., LUTn) are mapped to a logic element (LE), and the LE is configured to implement LUT1 in a first cycle, to implement LUT2 in a second cycle, and continuing until configuring the LE to implement LUTn in nth cycle, wherein n cycles are needed for execution. Moreover, the LE could be configured to implement LUT1 in a first cycle, wherein LUT1 is then executed in the first cycle, the LE is then configured to implement LUT2 in a second cycle, wherein the LUT2 is then executed in the second cycle, with the method continuing until the LE is configured to implement LUTn in nth cycle, and LUTn is executed in the nth cycle. In certain embodiments, all communications between the LUTs could be local. As a variation to the method, a second LE could execute a LUT in the first cycle using output from the execution of the first LUT by the LE in the first cycle.

**[0021]** An alternative method maps one or more of a series of look-up tables (LUTs) to one or more logic elements (LEs), each LE is configured to implement a LUT in a first cycle, and after implementation of two sequential LUT computations, each LE is reconfigured to implement a LUT in a second cycle.

**[0022]** In a method for determining a logic folding configuration, and for balancing resource use across the logic folding configuration, an input circuit design specified in register-transfer level or gate-level VHDL is provided, and a folding level us determined by: 1) identifying each plane of the input circuit design; 2) obtaining circuit parameters within each plane; and 3) and by obtaining a user optimization objective. The register-transfer level or gate-level VHDL module is then partitioned into LUTs and LUT clusters, which are then assigned to a folding stage. The LUTs and LUT clusters are then mapped to a super-macroblock (SMB), and are then placed to specific macroblocks (MB) and logic elements (LE). Intra-SMB and inter-SMB routing is determined, then a layout generated for each folding stage and a configuration bitmap for each folding cycle of the reconfigurable architecture.

## BRIEF DESCRIPTION OF THE DRAWINGS

**[0023]** For the purpose of illustrating the invention, there is shown in the drawing(s) a form that is presently preferred; it being understood, however, that this invention is not limited to the precise arrangements and instrumentalities shown.

Figure 1 illustrates structure of a nanotube random access memory;

Figure 2 illustrates a high level view of the architecture of the present invention;

Figure 3 illustrates an architecture of a lower level macro-block (MB), in accordance with the present invention;

Figure 4 illustrates an architecture of a higher level super macro-block (SMB), in accordance with the present invention;

Figure 5a illustrates a connection block for one input of a MB, and Figure 5b a connection block for one output from a MB, in accordance with aspects of the present invention;

Figure 6 illustrates a switch block in accordance with the present invention;

Figure 7a illustrates level-1 temporal logic folding, and Figure 7b level-2 temporal logic folding, in accordance with aspects of the present invention;

Figure 8 illustrates experimental circuit mapping results of one instance of a reconfigurable architecture of the present invention;

Figure 9 illustrates an alternative architecture of a lower level MB of the present invention;

Figure 10 illustrates an alternative architecture of a higher level SMB of the present invention;

Figure 11a illustrates a logic element (LE) architecture of the present invention having one flip-flop, and Figure 11b illustrates another logic element (LE) architecture having two flip-flops;

Figure 12 illustrates still another architecture of a lower level MB of the present invention;

Figure 13a illustrates a high-level view of a logic block (LB) architecture where a SMB has 4 MBs, and Figure 13b illustrates a high-level view of a logic block (LB) architecture where the SMB has 6 MBs;

Figure 14 illustrates still another architecture of a higher level SMB of the present invention;

Figure 15 illustrates a further architecture of a lower level MB of the present invention, where the number of inputs vary for any given LUT of each LE;

Figure 16 illustrates a SMB architecture with one level of folding in accordance with the present invention;

Figure 17 illustrates routing about a general SMB structure including from one to *n* MBs;

Figure 18a illustrates an example Register Transfer Level (RTL) circuit, Figure 18b a module partition, and Figure 18c a mapping result thereof, to demonstrate a design optimization method and system of the present invention;

Figure 19 is a flow diagram illustrating a design optimization method and system of the present invention, referred to as NanoMap;

Figure 20 illustrates delay optimization procedure under area constraint, assuming across-plane resource sharing, to choose folding level;

Figure 21a illustrates an ASAP schedule, and Figure 21b a ALAP schedule, for LUTs and LUT clusters in a plane, for distribution graph (DG) creation during a force-directed scheduling (FDS) implementation of the present invention;

Figure 22a illustrates a storage lifetime for the ASAP schedule of Figure 21a, Figure 22b illustrates a storage lifetime for the ALAP schedule of Figure 21b, and Figure 22c illustrates a maximum storage lifetime for source distribution computations during Distribution Graph (DG) creation;

Figure 23a illustrates a LUT computation Distribution Graph (DG), and Figure 23b illustrates a register storage DG for the ongoing example of Figure 18, demonstrating a design optimization method and system of the present invention;

Figure 24a illustrates clustering, and Figure 24b placement, in an example of temporal logic folding in accordance with a design optimization method and system of the present invention; and

Figure 25 illustrates experimental circuit mapping results of instances of the design optimization method and system of the present invention.

## DETAILED DESCRIPTION OF ILLUSTRATIVE EMBODIMENTS

**[0024]** A high-performance run-time reconfigurable architecture is provided, along with a design optimization method and system to efficiency balance performance and area considerations of the architecture. A high-density, high-speed non-volatile memory is implemented in the architecture to enable cycle-by-cycle reconfiguration and logic folding. Choice of different folding levels allows the designer flexibility in performing area-performance trade-offs. The significant increase in relative logic density (more than an order of magnitude for larger circuits) made possible by the present invention can allow the use of cheaper reconfigurable architectures

with smaller logic capacities to implement the same functionality, thus giving a boost to such use in cost-conscious embedded systems.

**[0025]** One embodiment of the invention implements a non-volatile nanotube random-access memory, that is considerably faster and denser than DRAM, has much lower power consumption than DRAM or flash, has similar speed to SRAM and is highly resistant to environmental forces (temperature, magnetism). Use of highly-dense nanotube random-access memories, such as a NRAM® chip, or of other emerging non-volatile memory technologies, including Phase Change RAMs, Magnetoresistive RAMs, and Ferroelectric RAMs (FRAMs), allows on-chip multi-context configuration storage, thereby enabling fine-grain temporal logic folding of a circuit before mapping to the architecture.

**[0026]** Reconfigurable architectures do exist in the art. However, their teachings are limited to allowing later stages of a pipeline to be executed in a same set of logic blocks that executed an earlier stage of the pipeline. This can be regarded as coarse-grain temporal folding. However, such architectures are largely limited to stream media or DSP applications. The present invention, on the other hand, supports fine-grain temporal folding, and is without the application limitations present in current reconfigurable architectures. Current reconfigurable architectures are described in the following, which is incorporated herein by reference for its useful background information:

- S.C. Goldstein, H. Schmidt, M. Budiu, S. Cadambi, M. Moe, and R.R. Taylor, "PipeRench: A Reconfigurable Architecture and Compiler," *Computer*, vol. 33, pp 70-77, Apr. 2000.

*Carbon Nanotube & NRAM® Chips*

**[0027]** Carbon nanotubes are hollow cylinders composed of one or more concentric layers of carbon atoms in a honeycomb lattice arrangement. The diameter of a nanotube is usually a few nanometers and length up to millimeters. Nanotubes exhibit unique electronic, mechanical and chemical properties. For example, carrier transport in nanotube is ballistic in the micrometer range and allows current densities as high as $10^9 A/cm^2$. These properties of nanotubes make them very attractive building blocks for molecular electronics.

**[0028]** Carbon nanotube random-access memories are described in the following, which is incorporated herein by reference for its useful background information:

- J.W. Ward, M. Meinhold, B.M. Segal, J. Berg, R. Sen, R. Sivarajan, D.K. Brock, and T. Rueckes, "A Non-Volatile Nanoelectromechanical Memory Element Utilizing a Fabric of

Carbon Nanotubes," in *Proc. Non-Volatile Memory Technology Symp.*, pp 15-17, Nov. 2004.

[0029] FIG. 1 shows a basic structure of a carbon nanotube random-access memory 100, such as a NRAM® chip. Memory cells are fabricated in a two-dimensional array using photo-lithography. Each memory cell comprises multiple suspended nanotubes, support and electrode. The memory state is determined by the state of the suspended nanotubes--whether they are bent or not leads to well-defined electrostatically switchable ON/OFF states. When opposite voltages are applied to the support and electrode of a memory cell, the suspended nanotubes are bent due to VanderWals forces, reducing the resistance between the nanotubes and electrode to as low as several hundred ohms, corresponding to the "1" state. On the other hand, when the same high voltage is applied to the support and electrode, the nanotube remains straight or returns from the "1" state, resulting in a resistance of several Gigaohms, which is defined as the "0" state. Such ON/OFF states have been shown to be both electrically and mechanically very stable.

***Phase Change RAMs, Magnetoresistive RAMs, and Ferroelectric RAMs (FRAMs)***

[0030] Phase Change RAMs, Magnetoresistive RAMs, and Ferroelectric RAMs (FRAMs) are each respectively detailed in the following, each of which are incorporated herein by reference for their useful background information:

- S. Lai, "Current status of the phase change memory and its future," in Proc. Int. Electron Devices Meeting, Dec. 2003, pp. 10.1.1–10.1.4.;
- S. Tehrani, J. M. Slaughter, M. Deherrera, B. N. Engel, and N. D. Rizzo, "Magnetoresistive random access memory using magnetic tunnel junctions," Proc. IEEE, vol. 91, pp. 703–714, 2003;
- G. R. Fox, F. Chu, and T. Davenport, "Current and future ferroelectric non-volatile memory technology," J. Vaccum Science Technology B., vol. 19, pp. 1967–1971, 2001.

*NATURE Architecture*

[0031] A high-level view of the architecture of the present invention is shown in FIG. 2. In this embodiment, island-style logic blocks 102 (LBs) are illustrated and are connected by various levels of interconnect. Several types of wire segments are used to support local and global communications among LBs 102. S1 104 and S2 106 refer to switch boxes 108 that connect wire segments. Connection blocks 110 and switch blocks 112 are as indicated in FIG. 2. An LB 102 contains a super-macroblock (SMB) 114 and a local switch matrix 116. The inputs/outputs of an SMB 114 are connected to the interconnection network through a switch matrix 116 and neighboring SMBs 114 are also connected through direct links.

***Super-Macroblock (SMB) Architecture***

**[0032]** The embodiment of the invention illustrated in FIGs 2-4 present two levels of logic clusters in an LB 102 to facilitate temporal logic folding of circuits, and enable most inter-block communications to be local. The first (i.e., lower) level, called the macroblock (MB) 118 level, is shown in FIG. 3.

**[0033]** An MB 118 contains $n_1$ m-input reconfigurable logic elements (LEs) 120 (in this figure, $n_1 = 4$). In the second level, $n_2$ MBs 118 comprise an SMB 114, as shown in FIG. 4 (in this figure, $n_2 = 4$). In the embodiments illustrated in FIG. 3 and 4, each LE 120 and MB 118 is associated with (physically adjacent and connected to) a nanotube RAM 100, perhaps a NRAM® chip. Alternative embodiments of the invention include other emerging non-volatile universal memories, such as phase change RAMs, magnetoresistive RAMs, and/or ferroelectric RAMs. Any could be implemented instead of an NRAM® chip.

**[0034]** Within an MB 118 or SMB 114, communications among various components can take place through a local crossbar 122. In this embodiment, a crossbar 122 is selected instead of a multiplexer at this level to speed up local communications. Since a crossbar 122 requires more SRAM 124 control bits, a slight price in area is exchanged for faster speed. However, since logic folding enables significant area savings, this area penalty is negligible. As shown in FIG. 3, the m inputs of a LE 120 can arrive from the outputs of other LEs 120 in the MB 118 or from the inputs to the MB 118. Similarly, the inputs of an MB 118 can arrive from the outputs of other MBs 118 or from the inputs to the SMB 114 through the switch matrix 116. The outputs (two in this embodiment) from an LE 120 can be used within the MB 118 or can go to the upper level SMB 114 or go to other SMBs 114 through the routing network. This logic/interconnect hierarchy maximizes local communications and provides a high level of design flexibility for mapping circuits to the architecture.

**[0035]** An LE 120 implements a basic computation. The LE 120 can include an m-input look up table (LUT) 126 and a flip-flop 128 (see FIG. 11(a), detailed below). The m-input LUT 126 can implement any m-variable boolean function. The flip-flop 128 stores the internal results for future use (when a circuit is temporally folded, the result of a previous stage is often needed by a subsequent stage). A pass transistor can be used to decide if the internal result will be stored or not.

***Run-time Reconfiguration***

[0036] Run-time reconfiguration is mainly enabled by the carbon nanotube RAM 100 (or phase change RAMs, magnetoresistive RAMs, or ferroelectric RAMs) distributed throughout the architecture. The structure and operation of a carbon nanotube RAM are similar to those of a traditional memory. One minor difference is that in a carbon nanotube RAM, counters can be used instead of decoders as periphery circuits since reconfiguration bits for different logic contexts are read out in order.

[0037] A carbon nanotube RAM 100 is associated with each reconfigurable block (e.g., LE 120 or switch block 112, etc.,) to store its run-time reconfiguration bits. Reconfiguration commences at one edge of the clock signal CLK, followed by computation at another edge of CLK. Reconfiguration bits are written into the carbon nanotube RAMs 100 at the time of initial configuration from off-chip storage. During run-time reconfiguration, reconfiguration bits are placed into SRAM 124 cells to configure the LE 120 or switch block 112 to implement different logic functionality or interconnections. For example, if $k$ configuration sets are stored in a carbon nanotube RAM 100, then the associated components can be reconfigured $k$ times during execution. As an example, for the MB 118 architecture embodiment shown in FIG. 3, 65 reconfiguration bits are required for a complete configuration set (when m = 4). In this set, 16 bits are required for each 4-input LUT, and one bit for determining whether to store the internal result or not. Hence, when $n_1 = 4$, $m = 4$, and $k$ configuration sets are used, the total number of carbon nanotube RAM bits required for one MB is $65kn_1$.

[0038] Inclusion of carbon nanotube RAMs 100 (or phase change RAMs, magnetoresistive RAMs, or ferroelectric RAMs) in the LB 102 incurs area overhead. Assuming a 100nm technology for implementing CMOS logic, 100nm nanotube length, and $k$=16, the carbon nanotube RAMs 100 occupy roughly 10.6% of the LB 102 area. However, through carbon nanotube RAM-enabled logic folding, the number of LBs 102 required to implement a circuit is reduced nearly $k$-fold. To account for these facts, the concept of relative logic density is introduced, and is defined as the ratio of the amount of logic that architectures of the present invention can implement in a given amount of area compared to the amount of logic a traditional reconfigurable architecture can implement in the same amount of area. When $k$=16 and assuming the circuit being implemented can use 16 configurations (as most large circuits would), the relative logic density can be calculated as 16(1-0.106) = 14.3. This means that in the same area, architectures of the present invention can implement roughly 14 times more logic than a

traditional architecture, or equivalently needs 14 times less area to implement the same functionality.

[0039] It can be seen that both the carbon nanotube RAM size and relative logic density vary with the value of $k$. If $k$ is too small, more global communication may be needed. If $k$ is too large, it may not be possible to make use of the extra configurations, thus leading to wasted carbon nanotube RAM area that could have been put to other use. Since the best $k$ value varies with the specific design, the value of $k$ can be obtained through a design optimization technique, NanoMap, introduced below, or through design space exploration of the architecture with various values of $k$ and mapping a large number of circuits to that instance of the architecture. In many instances, $k = 16$ is a preferred value.

[0040] To further improve the performance of the architecture at the expense of increased area, one can use a shadow reconfiguration SRAM to hide the reconfiguration latency for transferring bits from the carbon nanotube RAMs to the SRAMs. This allows one group of SRAM bits to load reconfiguration bits from nanotube NRAMs, while another SRAM group supports the current computation. The performance improvement due to this feature will depend on the level of logic folding.

***Interconnect Design***

[0041] Reconfigurable interconnect resources are provided in reconfigurable architectures to enable communication between programmable LBs 102. Interconnect design is very important for reconfigurable architectures because routing delays can be quite large, and most of the chip area is devoted to programmable routing. Consequently, the routing architecture must be designed to be both fast and area-efficient, and to aid logic folding and local communication.

[0042] There are primarily two methods for providing both local and global routing resources: segmented routing and hierarchical routing. One embodiment of the present invention uses a hybrid of segmented and hierarchical routing. In this embodiment, within the SMB 114, the interconnect is hierarchical to aid the logic clusters and local communication. To connect SMBs 114, wire segments of various lengths are used. In segmented routing, short wires accommodate local traffic. Such wires are connected together using switch boxes to emulate long wires.

[0043] The following routing architecture features address an interconnect structure of the present invention:

- The length of each routing wire segment (i.e., how many LBs a routing wire spans before terminating);

- The number of wires (tracks) in each routing channel;
- The type of each routing switch: pass transistor, tri-state buffer or multiplexer (MUX);
- Location of routing switches and which routing wires they connect; and
- Size of transistors in the switches and the metal width and spacing of the routing wires.

[0044] For the length of each routing wire segment, since too many short wires decrease circuit performance, and too many long wires provide little routing flexibility and may waste area, one embodiment of the present invention implements a mixed wire segment scheme including length-1 130, length-4 132, and long wires 134. Length-1 130 (length-4 132) wire segments span one (four) LB(s) 102 before connecting to a switch block 112, while long wires 134 traverse the chip horizontally and vertically, connecting to each LB 102 along the way. Besides these wire segments, there are also direct links 136 from the outputs of one LB to its four neighboring LBs, further facilitating local communications.

[0045] To address the number of wires (tracks) in each routing channel, for the architecture instance in which $m=n_1=n_2=4$, I=64, and O=32 (where I/O refers to the number of inputs/outputs of an SMB), one embodiment of the invention implements 128 horizontal and vertical tracks and assume a 25%, 50%, and 25% distribution for length-1 130, length-4 132, and long wires 134, respectively, among the 128 tracks in each direction. In addition, 32 tracks are used for direct links 136 between adjacent SMBs (since O=32). FIG. 5 illustrates one embodiment of how the inputs/outputs of an SMB 114 are connected to the routing network.

[0046] Next is a consideration of the design of the connection block 110, characterized by $F_c$, and switch block 112, characterized by $F_s$ ($F_c$ refers to the number of adjacent tracks a pin of an LB can connect to and $F_s$ the number of tracks to which each track entering the switch block can connect). Higher values of $F_c$ and $F_s$ result in higher routing flexibility, however, at the expense of a higher number of switches and hence more routing area. For a cluster of N LUTs, $F_c$ can be chosen as 1/N of the total number of tracks and $F_s$ should be greater than three in order to achieve routing completion while maintaining area efficiency. In one embodiment of the invention, $F_c$=1/N and $F_s$=6 is used. Another related and important issue is whether or not the internal connection blocks or switch blocks should be populated (such a block is said to be populated if it is possible to make connections from the middle of the block to LBs or to other blocks). When both are fully populated, the number of routing tracks required to achieve routing completion can be reduced, at the expense of a larger number of switches attached to a wire

(resulting in more capacitance and, hence, decrease in speed). In one embodiment of the invention, the connection blocks are depopulated and the switch blocks are populated to provide the best performance-area advantage. FIG. 6 shows an example connection of disjoint switch blocks 112.

[0047] The third feature considers the type of switch. There are typically three types of switches: pass transistor, multiplexer and tri-state buffer. Since a pass transistor has the shortest switching time, pass transistors are implemented in one embodiment of the invention for the local crossbars within the MB and SMB. A multiplexer has longer delay, but needs fewer reconfiguration bits. Therefore, a multiplexer 138 is implemented in one embodiment of the invention to connect to the inputs of a SMB 114 (e.g., see FIG. 10, detailed below). The outputs of an SMB can be connected to long interconnects through tri-state buffers in the switch matrix.

[0048] For the last feature, one embodiment of the invention uses pass transistors that are 10 times the size of a minimum-sized transistor and five times the size of a minimum-sized transistor for tri-state buffers and multiplexers. Minimum width and spacing are used for the metal wires.

*Temporal Logic Folding*

[0049] Temporal logic folding provides design flexibility and benefits in the present invention. The basic idea behind logic folding is that one can use run-time reconfiguration, and in one embodiment of the invention nanotube RAM-enabled run-time reconfiguration, to realize different Boolean functions in the same LE every few cycles. For example, suppose a subcircuit can be realized as a series of *n* serially connected LUTs. Traditional reconfigurable architectures will need *n* LUTs to implement the subcircuit. However, using run-time reconfiguration, at one extreme all these LUTs can be mapped to a single LE, which is configured to implement LUT1 in the first cycle, LUT2 in the second cycle, and so on, requiring *n* cycles for execution. Traditional reconfigurable architectures only support partial dynamic reconfiguration and do not allow such fine-grain temporal logic folding. Moreover, all communications between the LUTs mapped to the same LE are local. Hence, global communication is reduced, and routing delay is significantly reduced as well.

[0050] Logic folding occurs at the expense of reconfiguration time. However, results reveal that the time required to output the reconfiguration bits from an carbon nanotube RAM to the SRAM (i.e., the reconfiguration time to switch from one LUT to another), is only around 160ps. This is

small compared to routing delay saved. Also, by allowing use of shadow SRAM, the reconfiguration time can be hidden by overlapping computation.

[0051] Logic folding can be performed at different levels of granularity, providing flexibility to enable area-performance trade-offs. As an example, consider the LUT graph (in which each node denotes a LUT) shown in FIG. 7(a), which denotes level-1 folding. Such a folding implies that the LEs are reconfigured after the execution of each LUT mapped to it. On the other hand, FIG. 7(b) shows level-2 folding, implying reconfiguration of the LE after execution of two LUT computations. A level-p folding can be similarly defined. The case of no folding corresponds to mapping of circuits to a traditional reconfigurable architecture in which spatial partitioning of the LUT graph is feasible, but not temporal folding.

[0052] There are various trade-offs involved in the choice of the folding level. First, when the folding level is large, the cycle period increases because a larger amount of computation is executed in one cycle. The number of LEs needed also increases since they are not fully time-shared. However, the total number of cycles decreases. This fact coupled with the reduction in reconfiguration time may reduce total circuit delay. However, this would generally be true when communications between LEs are still local in the folded circuit, usually within the range of several SMBs. If the area required for implementing the subcircuit is out of this range and long global communication is required in one cycle, then a small folding level may give better performance.

[0053] Another important advantage of logic folding occurs when the circuit is too large to fit into a traditional reconfigurable architecture; it could then be mapped into the architecture of the present invention with logic folding. In a situation where the number of available LEs is limited, factors considered for obtaining the best folding level may differ from those mentioned above. In such a case, the number of cycles required to execute the whole computation will be dependent on the number of computation nodes in the LUT graph divided by the number of available LEs. Hence, the best folding level might be one that best uses the available LEs. A smaller folding level will use LEs less efficiently, and require more cycles, while a larger folding level will increase the cycle period and result in time inefficiencies.

*Experimental Results-NATURE*

[0054] Various MCNC benchmarks and arithmetic circuits illustrate the benefits of the run-time reconfiguration and logic folding features of the present invention. Architectures of the present invention present a family of carbon nanotube RAM-based (and phase change RAM-based,

magnetoresistive RAM-based, and ferroelectric RAM-based) reconfigurable architectures at different levels of granularity in terms of the number of LEs in an MB ($n_1$), number of MBs in an SMB ($n_2$), number of inputs per LE (m), number of configuration sets stored in the NRAM (k), etc. Accordingly, different architecture instances may be best suited for different circuit types. Since it appears that a cluster of four 4-input LUTs provides one of the best area-delay trade-offs, one embodiment of the present invention (for experimental purposes) uses an architecture instance corresponding to $n_1$=4, $n_2$=4, and m=4. Parameter $k$ is varied in order to compare implementations corresponding to selected folding levels: level-1, level-2, level-4 and no logic folding (note that the number of carbon nanotube RAM bits increases as we go from no folding to level-4 folding and towards level-1 folding since the number of LE configurations increases).

**[0055]** Several small/middle sized benchmarks were manually mapped to the underlying architecture instance. The depth of the circuit LUT graph, number of LEs, circuit delay, product of number of LEs and delay (this is a proxy for the area-time product, which is reasonable since the present invention is a regular architecture), and frequency are shown, for different levels of folding, in Table I of FIG. 8. These results are based on 100nm CMOS technology parameters.

**[0056]** Area/performance trade-offs that become possible because of use of logic folding are observed. Consider the 64-bit ripple-carry adder. Its LUT graph has 64 LUTs on the critical path. Using level-1 logic folding, the complete adder can be mapped to only two LEs. This, of course, requires reconfiguration of the LEs from the local carbon nanotube RAMs at each cycle. If more LEs are allowed (as in level-2, level-4 and no folding cases), the execution time goes down because fewer reconfigurations are required (note that, in this instance, the presence of a shadow SRAM is not assumed to overlap the reconfiguration and computation times of an LE -- if assumed, the execution time for level-1 folding would go down by roughly 1.6X at the expense of a doubling of SRAM area). Traditional reconfigurable architectures will require 128 LEs for such an adder (some architectures incorporate a carry generation circuit with each LE; in such a case, they will require 64 LEs although each LE will be larger due to the carry generation circuit overhead) because they cannot perform any temporal logic folding. As the number of required LEs increases, the need for using higher-level (i.e., more global) interconnects to connect them also increases. This is one of the reasons traditional reconfigurable architectures are not competitive with ASICs in terms of performance.

[0057] Next, consider the area-time product. For larger, more serially-connected circuits of larger depth, the area-time product advantage of level-1 folding relative to no folding is typically larger. For example, for the 64-bit ripple-carry adder, it is observed that the advantage is about 34X. This results from a large saving in area while maintaining competitive performance.

[0058] Table I of FIG. 8 also illustrates that the present invention can operate at high frequency. Peak frequency is around 3.3GHz. From level-1 folding to no-folding, the frequency decreases because increasingly more computation is included in one cycle period.

[0059] In spite of the fact that traditional reconfigurable architectures devote a vast majority of their area to interconnects, their LE utilization may not be high (an extremely large number of routing tracks may be needed to approach 100% LE utilization). Because of the cycle-by-cycle reconfiguration features of the architecture of the present invention, the LE utilization and relative logic density can be very high, with a reduced need for a deep interconnect hierarchy. Thus, architectures of the present invention suggest an evolutionary path for existing reconfigurable architectures, where fewer levels of interconnect hierarchy will be used and the area saved can provide for distribution of emerging non-volatile universal memories, such as carbon nanotube RAMs, throughout the chip.

*A Discussion of Some Alternative NATURE Architectures*

[0060] As discussed, NATURE can be characterized along a large number of varying dimensions, all of which are contemplated in the present invention. A non-exclusive list of exemplary characterizations are: 1) number of logic elements (LEs) per logic block; 2) number of inputs per LE; 3) size of carbon nanotube RAMs supporting each LE (this determines the granularity of reconfiguration); 4) depth of the FPGA interconnect hierarchy (localized communications can help drastically reduce this depth); 5) mix of different types of interconnects (much fewer longer interconnects are necessary); 6) number of registers per LE (because of the success of logic folding in reducing the number of LEs required for implementing the combinational logic by an order of magnitude, implementing sequential blocks now becomes the bottleneck for further area reduction); etc.

[0061] For instance, as an extension of the high level architecture view of FIG. 2, consider an alternative embodiment SMB 114 architecture, as illustrated in FIGs. 9 and 10. The alternative SMB 114, in this embodiment, again includes two levels of logic. The first (i.e., lower) level, called the macroblock (MB) 118, is shown in FIG 9. The MB 118 contains $n_1$ reconfigurable LEs 120 (in this embodiment, $n_1$ = 4). A 13 to 5 crossbar 122 is used to speed up the local

communication. In the second (i.e, higher) level, $n_2$ MBs 118 comprise an SMB 114, as shown in FIG. 10 (in this embodiment, $n_2 = 4$). In this SMB 114, since many reconfiguration bits are necessary to configure a full crossbar 122, a multiplexer 138 is instead used for local communication. This architecture facilitates temporal logic folding of circuits and enables most inter-block communications to be local.

**[0062]** In the FIG. 9 and 10 embodiments, the inputs of an MB 118 can arrive from other MBs 118 or the switch matrix 116. Similarly, the inputs of an LE 120 can arrive from other LEs 120 or MBs 118 or the switch matrix 116. The outputs from an LE 120 can be used within the MB 118 or go to the upper level SMB 114 or go to other SMBs 114 through the switch matrix 116.

**[0063]** The inputs to the LE 120 include $m$ inputs to a look-up table (LUT) 126 and one to a flip-flop 128, as shown in FIG. 11(a). In this embodiment, $m = 4$. The flip-flop 128 can store the computation result from the LUT 126 (when a circuit is temporally folded, the result of a previous stage is often needed by a subsequent stage), or the value of a primary input. This gives the flexibility of storing a LUT computation result in the flip-flops 128 of other LEs 120. The $m$-input LUT 126 can implement any $m$-variable Boolean function.

**[0064]** To realize cycle-by-cycle logic reconfiguration capability, an carbon nanotube RAM 100 is again associated with each reconfigurable block (i.e., LE 120 or crossbar 122), to store the run-time reconfiguration bits. During reconfiguration, the reconfiguration bits are placed in the SRAM 124 cells to reconfigure the LE 120 or crossbar 122 to implement different logic functionality and interconnections. For example, if $k$ configuration sets are stored in the carbon nanotube RAM 100, then $k$ different logic functions can be realized within the same hardware resource without the need to access off-chip storage. For the MB 118 architecture shown in FIG. 9, 82 reconfiguration bits are required for a complete configuration set (when $m = 4$). In this set, 16 bits are required for each 4-input LUT, and one bit for determining whether to store the internal result or not. Hence, when $n_1 = 4$, $m = 4$, and $k$ configuration sets are used, the total number of carbon nanotube RAM bits required for one MB 118 is $82kn_1$. A detailed layout and SPICE simulation show that a 16-set carbon nanotube RAM storage (i.e., $k = 16$) introduces 10.6% area overhead with 160ps on-chip reconfiguration time (i.e., the access latency of on-chip carbon nanotube RAM). Using this setup, the logic density is improved by 14$x$ on average. In addition, logic folding constrains most communication to be local, which greatly reduces the need for global interconnect.

[0065] As a basis for relative discussions concerning other alternative NATURE architectures, the embodiment of FIGs 9, 10 and 11(a) will be hereinafter referred to as the baseline design. That is, as detailed below, the baseline design describes an FPGA instance where the number of inputs per LE $m = 4$, number of LEs per MB $n_1 = 4$, and number of MBs per SMB $n_2 = 4$, one LUT and one flip-flop per LE, and number of reconfiguration sets $k = 16$.

[0066] **Number of LEs $n_1$ per MB:** Changing the value of $n_1$ leads to area-delay trade-offs. For example, consider $n_1 = 6$, as shown in the exemplary embodiment of FIG. 12. This configuration leads to larger crossbars 122 within the MB 118, and a larger carbon nanotube RAM 100 to reconfigure it since more LE 120 outputs need to be connected to the crossbars 122. At the same time, more LEs 120 in an MB 118 increases the number of outputs from the MB 118. This also results in increases to both the size of the input multiplexers 138 to the MB 118, and the amount of interconnects associated with the MB 118, as shown in FIG. 13(a). In all, the area of an SMB increases by 1.9*X* for $n_1 = 6$, compared with $n_1 = 4$. Thus, relative area per LE goes up by 1.9/1.5 = 1.27*X*. The level of folding desired in a given application, and other area-delay constraints, will determine whether the increase in relative area per LE is advantageous.

[0067] **Number of MBs $n_2$ per SMB:** Varying $n_2$ will also result in area/delay trade-offs. Increasing $n_2$ allows more logic to be implemented in an SMB 114, and more local communications between MBs 118 within the SMB 114. Hence, circuit delay may be reduced. However, the area of the SMB 114 will increase correspondingly. Consider the case of $n_2 = 6$, as shown in the exemplary embodiment of FIG. 14, and for which the high-level LB 102 view is shown in FIG. 13(b). Since there are 1.5*X* more MBs, and the MB architecture is unchanged, the number of outputs of the SMB increases by 1.5*X*, in turn resulting in a 1.5*X* increase in the number of interconnect tracks necessary to connect all the SMB outputs while maintaining the same $F_c$ as the baseline design. Consequently, the size of the switch matrix will also increase since the inputs of the SMB will be selected from more interconnect tracks. When $n_2$ increases from four to six, the area of the LB again increases to 1.9*X*, and relative area per LE increases by 1.9/1.5 = 1.27*X*. Depending on the particular application, and respective level of folding, the corresponding reduction in circuit delay would need to be evaluated against the above-identified increase in relative area to determine if this embodiment is desirable.

[0068] **Number of inputs *m* per LUT:** The number of inputs *m* for each LUT is a very important consideration for any FPGA architecture. If *m* is too large, and the application cannot always make use of all the inputs of each LUT, area is wasted. If *m* is too small, a larger number

of LUTs are required and, therefore, more MBs, SMBs and more interconnect communications. For example, if $m = 5$, the SMB area increases to 1.25*X*. In an instance where most LUTs only require four inputs, the mapped number of SMBs remains nearly the same. Hence, the mapped area increases by 1.25*X*. However, random logic (such as a controller) may benefit from a larger *m*. Because of the ability of FPGAs in the present invention to implement temporal logic folding, the value of *m* most suitable to conventional FPGAs may not be the same as in the present invention. Further, depending on the application, and desired folding level, the present invention contemplates that different inputs can exist for any given LUT 126 for each LE 120 of a MB 118. An exemplary embodiment is shown in FIG. 15.

**[0069] Number of flip-flops per LE:** Since temporal logic folding may reduce the combinational logic by more than an order of magnitude, the number of registers in the circuit may now become the bottleneck of further area reduction. Thus, as opposed to traditional LEs that include only one flip-flop, the present invention includes embodiments having more flip-flops per LE to further reduce the number of LEs required. However, if the inputs to the flip-flops are separately accessed, the number of inputs/outputs of an LE will increase as the number of flip-flops in an LE increases. Then, as discussed above, the communication network within and outside the SMB may grow very fast due to the increase in the number of inputs/outputs per LE, MB and SMB. Hence, the SMB size may increase significantly. If flip-flops in each LE are not used efficiently, area may be wasted.

**[0070]** For example, assume two flip-flops 128 per LE 120 as shown in FIG. 11(b). The input for each flip-flop 128 is distinct in this embodiment, providing that different values are stored in each of the two flip-flops 128 at the same time. This arrangement results in an increase: 1) in the size of the crossbar 2 in an MB 118; in the size of the input multiplexer 138 in an SMB 114; and 3) in the number of inputs to each SMB 114. The area of the SMB thereby increases by 1.5*X*.

**[0071]** In an instance of level-1 folding with configuration sets $k = 16$, significant area savings were realized (i.e., reduced number of LEs). However, increasing the number of flip-flops to three per LE could result, in the same instance, in an area increase. Since area saving depends on the value of *k*, simultaneously consideration of these two parameters are necessary.

**[0072] Number of reconfiguration sets *k*:** The value of *k* determines the amount of logic folding possible. If *k* is too small, more LEs are needed to perform a mapping. If *k* is too large, use of the extra configurations may not be possible, thus resulting in wasted carbon nanotube

RAM area that could have been put to other use. Complicating this fact is that the best value of $k$ varies with a change in the optimization objective (e.g., area, delay or area-delay product).

[0073] **Number of logic levels per SMB:** In the baseline design, two levels of logic are used in an SMB (i.e., SMB→MB and MB→LE) to facilitate local communication. However, since any communication between two LEs in different SMBs has to traverse two levels of interconnect, the communication delay is larger compared with that within just one level of logic. In addition, a two-level logic structure requires more implementation area than a one-level logic structure. Hence, a one-level structure has an advantage in area and inter-SMB delay, but a disadvantage in intra-SMB delay. FIG. 16 shows the structure of a flattened SMB 114 with one level of logic. An input of an LE 120 is now directly selected from the inputs from the switch matrix 116 and the outputs of other LEs 120. In the FIG. 16 embodiment, area is reduced by $1.1X$.

[0074] **Interconnect parameters:** In the carbon nanotube RAM-based FPGAs of the present invention, inter-LE communications become much more local. Hence, the interconnect hierarchy can be sharply reduced. Currently, the baseline sets $Fc$ = W/N and $Fs$ = 6, where $N$ is the number of LEs in an SMB and $W$ is the number of interconnect tracks per channel. A larger $Fc$ and $Fs$ can provide more routing flexibility, but at the cost of more routing area. The values for $Fc$ and $Fs$ can also be varied to achieve an optimal trade-off between routability and area efficiency.

[0075] Moreover, in most embodiments of the present invention, every input in the SMB 114 is accessible from the interconnect, with full routability within an SMB 114. However, complete routability within an SMB 114 may not be necessary. Both $I$ and $M$ (see FIG. 17) could potentially be reduced while keeping LE usage high, to thereby reduce the size of input multiplexers 138 in an SMB 114. Since multiplexers 138 contribute most to the area of an SMB 114, multiplexer size could result in a reduction in the size of an LB 102. For example, if $I = 0.6$ and $M = 1$ (i.e., full routability assumed within the SMB), the size of the LB 102 reduces to 76%.

*NanoMap Design Optimization*

[0076] The present invention also provides an integrated design optimization platform for NATURE, referred to as NanoMap. NanoMap conducts design optimization from the RTL down to the physical level. Given an input design specified in RTL and/or gate-level VHDL, NanoMap optimizes and implements the design on NATURE through logic mapping, temporal clustering, placement, and routing. The design optimization techniques of the present invention exploit the design flexibilities enabled by fine-grain temporal logic folding. Given user-specified

area and performance constraints, the mapping method and system of NanoMap can automatically explore and identify the best logic folding configuration, and make appropriate tradeoffs between performance and area efficiency. The methods of the present invention can be implemented as software running on a general-purpose computer, such as an INTEL® PENTIUM® based personal computer running a MICROSOFT® WINDOWS® operating system, although the invention is not limited to that particular implementation.

**[0077]** To demonstrate the design optimization flow of NanoMap, an example RTL circuit 140 will be provided, and concepts associated therewith are first introduced for ease of exposition. Given an RTL circuit 140, the registers contained therein are first levelized. The logic between two levels of registers is referred to as a *plane*. The registers associated with the plane are called *plane registers*. The propagation cycle of a plane is called *plane cycle*. Using temporal logic folding, each plane is further partitioned into *folding stages*. Resources can be shared among different folding stages within a plane or across planes. The propagation cycle of a single folding stage is defined as *folding cycle*. Note that different planes should consist of the same number of folding stages to guarantee global synchronization. Thus, the key issue is to determine how many planes are folded together and to determine the appropriate folding level (i.e., the number of folding stages in one plane necessary to achieve the best area-performance tradeoff under specified design constraints).

**[0078]** FIG. 18(a) shows an example comprising a four-bit controller-datapath consisting of a single plane. The controller consists of flip-flops s0 and s1, and LUTs LUT1-LUT4. The datapath consists of registers reg1-reg3, a ripple-carry adder and parallel multiplier module, requiring in all 100 LUTs and 14 flip-flops. The ripple-carry adder consists of eight LUTs with a logic depth (i.e., the number of LUTs along the critical path) of four. The parallel multiplier consists of 38 LUTs with a logic depth of seven. The control logic consists of four LUTs. Suppose the optimization objective is to minimize circuit delay under a total area constraint of 20 LEs. We assume each LE contains one LUT and two flip-flops. Hence, 20 LEs equal 20 LUTs along with 40 flip-flops. Since the number of available flip-flops is more than required, we concentrate on the LUT constraint.

**[0079]** The present invention uses an iterative optimization flow. As a smaller number of folding stages leads to better performance, NanoMap starts with a guessed folding level, resulting in a minimal number of folding stages under the given area constraint, and gradually refines it. In the FIG. 18 example, the minimal number of folding stages is equal to the total

number of LUTs divided by the LUT constraint, $\left\lceil 50/20 \right\rceil = 3$ (i.e., at least three folding stages are required to meet the LUT constraint). The folding level is obtained by the maximum logic depth divided by the number of folding stages, which equals $\left\lceil 4 + 7/3 \right\rceil = 4$.

[0080] Next, based on the chosen folding level, the adder and multiplier modules are partitioned into a series of connected LUT clusters in a way that if the folding level is *p*, then all the LUTs at a depth less than or equal to *p* in the module are grouped into the first cluster, all the LUTs at a depth larger than *p* but less than or equal to *2p* are grouped into the second cluster, and so on. The LUT cluster can be considered in its entirety with its logic depth being less than or equal to the folding level. This implies that one LUT cluster can be executed within one folding cycle, thereby being contained in one folding stage. By dealing with LUT clusters instead of a group of single LUTs, the logic mapping procedure can be greatly sped up. FIG. 18(b) shows the partition for the multiplier module 142 with level-4 folding. However, note that the first LUT cluster of the multiplier already needs 32 LUTs, exceeding the area constraint. Thus, the folding level has to be further decreased to level-2 to guarantee that each LUT cluster can be accommodated within the available LEs. Correspondingly, the number of folding stages increases to six.

[0081] Next, after choosing a suitable folding level, Force Directive Scheduling (FDS) is used to determine the folding cycle assignment of each LUT and LUT cluster to balance the resource usage across the six folding stages. If the number of LUTs and flip-flops required by every folding stage is below the area constraint (i.e., 20 LEs) the solution is valid and offers the best possible performance. Otherwise, the folding level is reduced by one, followed by another round of optimization. This process continues until the area constraint is met, assuming the area constraint can be satisfied.

[0082] FIG. 18(c) illustrates the mapping result 144 for level-2 folding for the first three folding stages of the total of six folding stages. Note that plane registers, which provide inputs to the plane, need to exist through all the folding stages in the plane. The first folding cycle requires 14 LEs. Four LEs are required for mapping LUT cluster 1 of the adder, which is depicted as *add: c1* in FIG. 18(c). Flip-flops s0 and s1 are mapped to the available flip-flops inside the LEs assigned to adder cluster 1. Four LEs are also required for LUT1-LUT4 computation, and to store the respective computation results. The four-bit registers, reg1, reg2 and reg3; need two LEs each to accommodate their four flip-flops. Similarly, in folding cycle 2, four LEs are needed for adder cluster 2 computation and resulting storage. Four LEs are required for maintaining the LUT1-

LUT4 computation results, which need to be preserved until folding cycle 6 to control the loading of registers, and six LEs for reg1-reg3. Folding cycle 3 requires the maximum number of LEs, since multiplier cluster 1 needs 16 LUTs, which occupy 16 LEs. The number of LEs needed by the last three folding levels (not shown), are 16, 12 and 12, respectively. Hence, the number of LEs for mapping this RTL circuit is the maximum required across all the folding cycles (i.e., 16). This is within the area constraint.

**[0083]** Next, clustering, which groups LEs into SMBs, placement and routing are performed to produce the final layout of the implementation and obtain the best possible circuit delay under the given constraint. When performing clustering, inter-stage relationships are honored, since some computation results need to be preserved through several folding cycles. Once the results are assigned to some flip-flops in an SMB, they are not assigned to other SMBs in other folding cycles. In the FIG. 18 example, assume there are four LEs in an MB and four MBs in an SMB. Thus, the 14 LEs in folding cycle 1 can be accommodated into one SMB. Suppose LUT1-LUT4 are assigned to MB1. Then their computation results *storage1-4* will be present in MB1 through all the folding cycles before being overwritten by new results.

***Automated Optimization Flow of NanoMap***

**[0084]** FIG. 19 illustrates an integrated design optimization flow for NATURE. Given an input design 201 specified in mixed RTL and gate-level VHDL, NanoMap conducts logic mapping, temporal clustering, temporal placement and routing, and produces a configuration bitmap for NATURE.

**[0085] Logic Mapping**: (Steps 202–206) Steps 202-206 of FIG. 19 use an iterative approach to identify the best folding level based on user-specified design constraints, optimization objectives, and input circuit structure. FDS techniques, detailed below, are used to assign LUTs and LUT clusters to folding stages and balance inter-folding stage resource usage, and to produce the LUT network of each temporal folding stage.

**[0086] Temporal Clustering**: (Steps 207–208) Steps 207-208 of FIG. 19 take the flattened LUT network as input, and cluster the LUTs into MBs and SMBs to minimize the need for global interconnect, and to simplify placement and routing. As opposed to the traditional clustering problem, each hardware resource (i.e., LE, MB, or SMB) is temporally shared by logic from different temporal folding stages. Temporal folding necessitates that both intrastage and inter-stage data dependencies be jointly considered during LUT clustering. Folding stages need not be limited to one plane; temporal clustering can span planes. After clustering, verifying satisfaction

of the area constraint is performed. If the area constraint is satisfied, placement is invoked. Otherwise, NanoMap returns to logic mapping.

**[0087] Temporal Placement**: (Steps 209–214) Steps 209-214 of FIG. 19 perform physical placement and minimize the average length of inter-SMB interconnects. Physical placement and interconnect minimization is implemented on top of VPR, an FPGA place-and-route tool, detailed and referenced below, to provide inter-folding stage resource sharing. Placement is performed in two steps. First, a fast placement is used to derive an initial placement. A low-precision routability and delay analysis is then performed. If the analysis indicates success, a detailed placement is invoked to derive the final placement. Otherwise, several attempts are made to refine the placement and if the analysis still does not indicate success, NanoMap returns to logic mapping.

**[0088] Routing:** (Step 215) Step 215 of FIG. 19 uses the VPR router to generate intra-SMB and inter-SMB routing. After routing, the layout for each folding stage is obtained and the configuration bitmap generated 216 for each folding cycle.

**[0089]** The following details the above steps. For logic mapping, focus is provided on folding level determination and FDS technique.

**Choosing the Folding Level**

**[0090]** The folding level choice is critical to achieving the best area-performance tradeoff. As previously noted, the best folding level depends on input circuit structure, obtained by identifying each plane and obtaining the circuit parameters within each plane. The following outlines the necessary circuit parameters:

- Number of planes in input circuit: $num_plane$
- Number of LUTs in plane $i$: $num_LUT_i$
- Logic depth of plane $i$: $depth_i$
- Maximum number of LUTs among all the planes:
  $LUT_max = max\{num_LUT_i\}$ for $i = 1, ..., num_plane$
- Maximum logic depth among all the planes:
  $Depth_max = max\{depth_i\}$ for $i = 1, ..., num_plane$
- Area constraint, e.g., the available number of LEs: $available_LE$
- Number of reconfiguration copies in each carbon nanotube RAM: $num_reconf$

**[0091]** Given the specified optimization objective and constraint (e.g., circuit delay minimization under area constraint or area minimization under delay constraint, etc.), the best folding level is

computed using above parameters. The following details a targeting of one of the design objectives. Similar procedures can target other objectives.

**[0092]** Suppose the optimization goal is to minimize circuit delay. If there is no area constraint, we can use no-folding to obtain the shortest delay. If an area constraint is given, it is satisfied first, then the best possible delay obtained. There are two scenarios considered:

**[0093]** ***1) Multiple planes are allowed to share resources:*** Since circuit delay is equal to plane cycle times the number of planes in the circuit, plane cycle has to be minimized under the area constraint. First, all the planes together are stacked (i.e., resources are shared across all planes, since this does not increase circuit delay but reduces area). Suppose the area used up at this point is *LUT_max*. If *LUT_max* is larger than *available_LE*, logic folding is required to reduce the area within each plane. The minimum required number of folding stages within each plane is given by:

$$\#folding_stages = \left\lceil \frac{LUT_\max}{available_LE} \right\rceil \qquad (1)$$

Since the number of folding cycles should be kept the same in each plane, maximum logic depth is used to compute the folding level:

$$folding_level = \left\lceil \frac{depth_\max}{\#folding_stages} \right\rceil \qquad (2)$$

Using the chosen folding level, the present invention uses FDS and temporal clustering to obtain the area required. If the area constraint is not satisfied, the folding level is decreased by one. NanoMap then iterates until the area constraint is met or the folding level reduces to the minimum allowed, *min_level*, which is limited by *num_reconf*:

$$\min_level = \left\lceil \frac{depth_\max * num_plane}{num_reconf} \right\rceil \qquad (3)$$

FIG. 20 illustrates the optimization procedure.

**[0094]** ***2) Multiple planes are not allowed to share resources:*** Such a scenario is possible if the RTL circuit is pipelined and, hence, the different pipeline stages need to be resident in the FPGA simultaneously. In this scenario, temporal logic folding can only be performed within each plane. Then the folding level requested can be directly computed by the following equation:

$$folding_level = \left\lceil \frac{depth_\max * available_LE}{\sum_i num_LUT_i} \right\rceil \qquad (4)$$

After an appropriate folding level is chosen, the RTL module is partitioned into LUT clusters accordingly. The original mixed module/LUT network is transformed to an equivalent LUT/(LUT cluster) network which is fed to FDS.

**Force-Directed Scheduling (FDS)**

[0095] Different folding stages share the same set of LEs temporally. Overall LE use is then determined by the folding stage using the maximum number of LEs. To optimize overall resource use in each plane, a modified Force-Deflected Scheduling (FDS) method is implemented to assign the LUT or LUT cluster to folding stages and balance the resource use of the folding stages.

[0096] Force-Deflected Scheduling (FDS) is described in the following, which is incorporated herein by reference for its useful background information:

- P.G. Paulin and J.P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's," *IEEE Trans. Computer-Aided Design,* vol. 8, pp. 661-679, June 1989.

[0097] FDS is a popular scheduling technique in high-level synthesis. However, the present invention uses FDS in another scenario. FDS uses an iterative approach to determine the schedule of operations, to minimize overall resource use. The resource use is modeled as a force. The scheduling of an operation to some time slot, which results in the minimum force, indicates a minimum increase in resource use. The force is calculated based on distribution graphs (DGs), which describe the probability of resource use for a type of operation in each time slot.

[0098] In the present invention, since the LE use in each folding cycle is dependent on both the LUT computations and register storage operations conducted in parallel, two DGs must be assembled: one describing the resource use of the LUT computation; and another for register storage use. The following details: 1) how DGs are created; and 2) how forces are calculated based on the two created DGs.

[0099] 1) ***Creation of DGs:*** First, to build the LUT computation DG, the time frame of each LUT or LUT cluster needs to be determined. For a LUT or LUT cluster $i$, its time frame $time_frame_i$, or feasible time interval, is defined as the span from the folding cycle it is assigned to in the ASAP schedule to the folding cycle it is assigned to in the ALAP schedule. From the ASAP/ALAP schedules shown in FIG. 21 for the ongoing example, we can see that $time_frame_{LUT2}$ spans folding cycles 1 to 3. Here, $clus_i$ denotes LUT cluster $i$. If a uniform probability

distribution is assumed, the probability that this computation is assigned to a feasible folding cycle $j$ within its time frame equals 1 /|*time_frame*$_i$| for $j \in time_frame_i$.

[00100] Following a definition similar to that given by P.G. Paulin and J.P. Knight, above, a LUT computation DG models the aggregated probability distribution of the potential concurrency of $N$ LUT/(LUT cluster) computations within each folding cycle $j$, whose value $LUT_DG(j)$ is the sum of the probabilities of all the computations assigned to this folding cycle, as follows:

$$LUT_DG(j) = \sum_{i=1}^{N} \frac{1}{|\,time_frame_i\,|} * weight_i, j \in time_frame_i$$

where $weight_i$ is one for a LUT and equal to the number of LUTs in a LUT cluster.

[00101] To build the register storage DG, which models the distribution of register storage usage, a procedure similar to that of P.G. Paulin and J.P. Knight, above, is adopted. A storage operation is created at the output of every source computation that transfers a value to one or more destination computations in a later folding cycle. If both the source and destinations of a storage operation are scheduled, the distribution of the storage operation equals its *lifetime*, which begins from the folding cycle of the source and ends at the folding cycle of the last destination. Here, it is assumed the results are stored at the begining of each folding cycle. If one or more of the source or destinations are not scheduled, a probabilistic distribution is obtained.

[00102] The following heuristic is used to quickly estimate the resulting storage distribution. First, *ASAP_life* and *ALAP_life* of a storage operation are defined as its lifetime in the ASAP and ALAP schedules, respectively. For example, in FIG. 22, the output of source computation LUT2 is denoted as storage S. S transfers the value to destination computation LUT3 and LUT4. In the ASAP schedule, S begins at folding cycle 2 and ends at folding cycle 3. Hence, $ASAP_life_S$ = [2, 3] and the length of *ASAP_life*: $|ASAP\ life_S| = 2$. Similarly, in the ALAP schedule, S begins at folding cycle 4 and ends at folding cycle 4, which results in $|ALAP_life_S| = 1$.

[00103] The longest possible lifetime *max_life* for the storage operation is the union of its *ASAP_ life* and *ALAP_life*, whose length is obtained as:

$$|max_life| = (ALAP_life\ end - ASAP_life_begin + 1) \qquad (6)$$

For the ongoing example, S begins in folding cycle 2 in the ASAP schedule (i.e., $ASAP_life_begin_S = 2$). Its lifetime ends in cycle 4 in the ALAP schedule (i.e., $ALAP_life_end_S = 4$). Thus, the length of the maximum lifetime for S (e.g., $|max_life_S| = 3$).

**[00104]** If *ASAP_life* overlaps with *ALAP_life*, the overlap time, *overlap*, is the intersection of *ASAP_life* and *ALAP_life*, whose length is similarly obtained as:

$$|overlap| = (ASAP\ life\ end - ALAP\ life\ begin + 1) \qquad (7)$$

Within the overlap time, a storage operation must exist with probability 1. For the example, there is no overlap time for S. Then an estimate of the average length of all possible lifetimes can be obtained by:

$$avg_life = \frac{|ASAP_life| + |ALAP_life| + |\max_life|}{3} \qquad (8)$$

Next, the probability of a storage operation performed for a LUT or LUT cluster computation $i$ in folding cycle $j$ can be calculated as follows:

- when $j$ is outside of $overlap_i$ and $j \in \max_life_i$:

$$storage_i(j) = \frac{avg_life_i - |overlap_i|}{|\max_life_i| - |overlap_i|} * weight_i \qquad (9)$$

- when $j$ is within $overlap_i$, which means a storage operation must be performed:

$$storage_i(j) = weight_i \qquad (10)$$

The process is carried out for all the storage operations, and the separate probabilities due to $N$ LUTs and LUT clusters in folding cycle $j$ are added to obtain a single storage DG as follows:

$$storage_DG(j) = \sum_{i=1}^{N} storage_i(j) \quad j \in \max_life_i \qquad (11)$$

The two DGs obtained for the example in FIG. 21 are shown in FIG. 23.

**[00105]** 2) ***Calculation of Forces:*** In the FDS algorithm, force is used to model the impact of scheduling operations on resource use. A higher force implies higher concurrency of run-time operations, which requires more resources in parallel. The force is calculated based on DGs, which present the probability of resource usage concurrency. For a given computation with time frame spanning folding cycles $a$ to $b$, the force in cycle $j$ is calculated by:

$$force(j) = DG(j) * x(j) \qquad (12)$$

where $DG(j)$ is either $LUT_DG(j)$ or $storage_DG(j)$ in our case, and $x(j)$ is the increase (or decrease) in the probability of computation in cycle $j$ due to the scheduling of the computation. For example, before scheduling, the computation has a uniform probability of being scheduled in each folding cycle in its time frame. If in a scheduling attempt, the computation is scheduled in folding cycle $a$, the probability of the computation being scheduled in folding cycle $a$ will increase to 1 and the probability of the computation being scheduled in other folding cycles will

decrease to 0. The self-force associated with the assignment of a computation $i$, whose time frame spans folding cycles $a$ to $b$, to folding cycle $j$ is defined as the sum of all the resulting forces in each folding cycle in its time frame:

$$self_force_i(j) = \sum_{k=a}^{b} force(k) = DG(j) * x(j) + \sum_{k=a,k \neq j}^{b} | DG(k) * x(k) | \quad , j \in | a,b |$$

$$x(j) = (| time_frame_i | -1) / | time_frame_i |$$

$$x(k) = -1 / | time_frame_i | \qquad (13)$$

**[00106]** In the approach of the present invention, the resource use can be dictated by either LUT computations or storage operations. Assume there are $h$ LUTs and $l$ flipflops in one LE, then the self-force for scheduling a LUT or LUT cluster $i$ in folding cycle $j$ is determined by

$$\max\{\frac{LUT_self_force_i(j)}{h}, \frac{storage_self_force_i(j)}{l}\}$$

where $LUT_self_force_i(j)$ and $storage_self_force_i(j)$ are computed using Equation (13) based on the LUT computation and storage DGs.

**[00107]** Assigning a LUT computation to a specific folding cycle will often affect the time frame of its predecessors and successors, which in turn creates additional forces affecting the original move. Equation (13) is used to compute the force exerted by each predecessor or successor. The overall force is then the sum of the self-force and the forces of predecessors and successors. Then the total forces under each schedule for a computation are compared and the computation is scheduled into the folding cycle with the lowest force, which will result in the least concurrency.

**[00108]** 3) ***Summary of the FDS algorithm:*** The pseudo-code of the proposed FDS technique is shown in Algorithm 1. Algorithm 1 uses an iterative approach to schedule one computation in each iteration. In each iteration, the LUT computation and register storage DGs are obtained. The LUT or LUT cluster with the minimum force is chosen, and assigned to the folding cycle with the minimum force. This procedure continues until all the LUT or LUT cluster computations are scheduled.

**<u>Algorithm 1 - Force-Directed Scheduling (FDS)</u>**

1: **for** LUT/(LUT cluster) computations to be scheduled **do**
2: evaluate its time frame using ASAP and ALAP scheduling
3: create the LUT computation distribution graph and storage operation distribution graph
4: **for** each unscheduled LUT/(LUT cluster) computation $i$ **do**
5: **for** each feasible clock cycle j it can be assigned to **do**

6: calculate the self-force of assigning node $i$ to cycle $j$
7: add predecessor and successor forces to self-forces to get the total force for node $i$ in cycle $j$
8: **end for**
9: select the cycle with the lowest total force for node $i$
10: **end for**
11: Pick the node with the lowest total force and schedule it in the selected cycle
12: **end for**

**Temporal Clustering**

**[00109]** After scheduling, a network of LUTs is assigned to each folding stage. For each folding stage, we use a constructive algorithm to assign LUTs to LEs and pack LEs into MBs and SMBs. To construct each SMB, an unpacked LUT with the maximal number of inputs is first selected as an initial seed. Then, new LUTs with high attractions to the seed LUT are chosen and assigned to the SMB. The attraction between a LUT $i$ and the seed LUT, $Attraction_{i,seed}$, depends on timing criticality and input pin sharing [17], as follows:

$$Attraction_{i,seed} = \alpha * Criticality_i + (1-\alpha) * \frac{Nets_{i,seed}}{G} \qquad (15)$$

and as described in the following, which is incorporated herein by reference for its useful background information:

- A.S. Marquardt, V. Betz, and J. Rose, "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density," in *Proc. Int. Symp. FPGA*, Feb. 1999, pp. 37-46.

In Equation 15, $Criticality_i$ models the timing criticality of LUT $i$ (e.g., the number of critical paths that this LUT is on), $Nets_{i,seed}$ is the number of shared I/Os between these two LUTs, and $G$ is the number of I/Os per LE. $\alpha$ is a parameter that allows a tradeoff between timing criticality and interconnect demand.

**[00110]** To support temporal logic folding, inter-folding stage resource sharing needs to be considered during clustering. Since due to logic folding, several folding stages may be mapped to a set of LEs, some of the LEs may be used to store the internal results and transfer them to another folding cycle. Such LEs may perform this job over several cycles and feed other LEs in each folding cycle. As illustrated in FIG. 24(a), in an earlier folding cycle, two LUTs may have very few attractions between them (C and D in FIG. 24(a)), but may have a large number of attractions in a later cycle. When performing temporal clustering, the attractions of two LUTs

over all the cycles need to be accounted for. Thus, the attraction of such a LUT is set to the maximum of its attractions over all the cycles.

**Placement and Routing**

[00111] In the present invention, placement and routing is performed by a modified VPR. VPR refers to the techniques of the following, which is incorporated herein by reference for its useful background information:

- V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," *Proc. Int. Wkshp. FPGA,* Aug. 1997, pp. 213-222.

Placement uses a two-step simulated annealing approach. Placement starts with a fast low-precision placement. Routability analysis and delay estimation are then used to evaluate the quality of the initial placement. For routability analysis, a highly-efficient empirical estimation technique is used, as described in the following, which is incorporated herein by reference for its useful background information:

- C.L.E. Chang, "VRISA: Accurate and Efficient Placement Routability Modeling," in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1994, pp. 690-695.

The routing demand for the interconnect resources for horizontal and vertical channels, $D_k^{i,horisontal}$ and $D_k^{i,vertical}$, of each net $k$ is estimated as:

$$D_k^{i,horisontal} = q * \frac{1}{Y}; \qquad D_k^{i,vertical} = q * \frac{1}{X} \qquad (16)$$

where $(X, Y)$ are the dimensions of the net bounding box for net $k$, and $q$ is a pin-count dependent net-weight, as detailed in C.L.E. Chang. The sum of the demands of all the nets is then compared to the per-channel routing resources of NATURE to make sure the resources are adequate. Delay estimation is based on the timing analysis step of VPR. Routability analysis and delay estimation results are then used to evaluate the feasibility of the initial placement, which determines whether a high-precision placement or another round of logic folding should be invoked.

[00112] VPR placer was modified in the present invention to support temporal logic folding. Such temporal folding introduces inter-folding stage dependencies. Consider the example in FIG. 24(b). In folding cycle 1, since there are few connections between C and D, they may be placed far apart. However, such a placement would not be good for folding cycle 2 in which C and D communicate a lot. The Manhattan distance is computed between each pair of SMBs

belonging to different temporal folding stages. The net bounding box in other unplaced cycles are estimated using this Manhattan distance and added to the cost function for the current cycle to guide placement. Routing is conducted in a hierarchical fashion, first using length-1, then length-4 and finally global interconnects (i.e., the three types of interconnects in NATURE, above). Note that a length-$i$ interconnect spans $i$ SMBs.

*Experimental Results-An instance of NATURE using NanoMap*

[00113] Presented here are experimental results for the mapping of seven RTL/gate-level benchmarks to an instance of NATURE using NanoMap to illustrate the benefits of run-time reconfiguration and logic folding. NATURE is a family of architectures, which may vary in the number of inputs and registers in an LE, number of LEs in an MB, number of MBs in an SMB, etc. In this experimental instance, an architecture having one four-input LUT in an LE, four LEs in an MB, and four MBs in an SMB, are selected to obtain good area-delay trade-offs. Observations show that temporal logic folding greatly reduces the area for implementing logic, so much so that the number of registers in the design becomes the bottleneck for area reduction. Thus, as opposed to traditional LEs that include only one register, the present invention, in this example, includes two registers per LE, which increases an SMB's area to $1.5X$ (all experiments are based on a 100nm technology). However, the LE area increase is more than offset by the significant reduction in overall area. To fully explore the potential of logic folding, we assume that a varying number of reconfiguration sets, $k$, is available in carbon nanotube RAMs depending on the application. We also show the tradeoffs when the size of carbon nanotube RAM is instead fixed to 16.

[00114] Among the seven benchmarks targeted, ex1 is the circuit shown in Fig. 18, but with a bit-width of 16. Paulin is a differential-equation solver, and FIR and Biquad are two types of digital filters. ASPP4 is an application-specific programmable processor. c5315 is a gate-level ALU implementation from the ISCAS'85 benchmark suite. NanoMap was run on a 2GHz PC with 1GB of DRAM under RedHat Linux 9. The mapping CPU times were less than a minute for all the benchmarks.

[00115] First, all benchmarks were mapped under the area-time (AT) product minimization objective to show the logic density benefits of temporal logic folding against the traditional no-folding case. Table II of FIG. 25 shows the mapping results. The first five columns describe the benchmark name and structure. Columns 6 and 7 of Table II of FIG. 25 show the number of LEs required and circuit delay for the no-folding case. Columns 8 and 12 show the best folding level

established by the present invention, NanoMap, for AT product optimization without and with limitations on $k$. AT product optimization is achieved with folding level-1 in all the cases when there is no restriction on $k$, since an increase in circuit delay is more than overcome by the dramatic reduction in area when using level-1 folding.

[00116] The corresponding area (where the number of LEs is used as a proxy for area due to the regular architecture), circuit delay and AT product improvement with respect to the no-folding case for examples without and with limitations on $k$ are shown in Table II - Columns 9-11 and 13-15, respectively. The average reduction in the number of LEs is 14.8$X$ (9.2$X$) and in the AT product 11.0$X$ (7.8$X$), at the price of a 31.8% (19.4%) increase in circuit delay for large enough $k$ (with $k$ limited to 16).

[00117] Accordingly, the present invention can target many different optimization objectives: (i) minimization of circuit delay with or without an area constraint; (ii) minimization of area with or without a delay constraint; (iii) minimization of the AT product; and (iv) finding a feasible implementation under both area and delay constraints.

[00118] Different optimization objectives for different benchmarks are selected, with results presented in Table III. Objectives are noted in Column 2 of Table III of FIG. 25, and the constraint (i.e., area or delay) is noted in Columns 3 and 4 of Table III. Table III of FIG. 25 illustrates the versatility of NATURE and NanoMap. Further, a significant side-benefit of area reductions made possible by logic folding is the associated reduction for a deep interconnection hierarchy in NATURE. Since cycle-by-cycle reconfiguration makes LE utilization very high, the need for global communication greatly reduces. Global interconnect use was reduced by more than 50% when using level-1 folding as opposed to no-folding, due to trading interconnect area for increased carbon nanotube RAM area in NATURE.

***Conclusion***

[00119] The present invention presents a hybrid nanotube/CMOS dynamically reconfigurable architecture, NATURE, and an RTL/gate-level automatic design optimization flow, NanoMap, for the NATURE reconfigurable architecture. NATURE supports run-time fine-grain reconfiguration and, hence, enables temporal logic folding. Through logic folding, significant logic density improvement and flexibility in performing area-delay tradeoffs are possible.

[00120] NanoMap incorporates temporal logic folding during the logic mapping, temporal clustering and placement steps. NanoMap provides for automatic selection of a best folding level, and uses force-direct scheduling to balance resources across the different folding stages.

Mapping, as provided by the present invention, can target various optimization objectives and user constraints. With NanoMap, the potential of NATURE can be effectively realized.

[00121] These and other advantages of the present invention will be apparent to those skilled in the art from the foregoing specification. Accordingly, it will be recognized by those skilled in the art that changes or modifications may be made to the above-described embodiments without departing from the broad inventive concepts of the invention. It should therefore be understood that this invention is not limited to the particular embodiments described herein, but is intended to include all changes and modifications that are within the scope and spirit of the invention.

What is Claimed:

1. A reconfigurable computer architecture, comprising:
a plurality of logic elements; and
an equal number of random access memories, one random access memory being associated with each logic element.

2. The architecture of claim 1, further comprising a crossbar with each logic element and respective random access memory to provide communication between components.

3. The architecture of claim 1, wherein four logic elements and four respective random access memories comprise a macroblock (MB).

4. The architecture of claim 3, wherein inputs for each logic element arrive from outputs of logic elements in the macroblock or the inputs to the macroblock.

5. The architecture of claim 3, wherein four macroblocks are arranged to comprise a super-macroblock (SMB), wherein each of the four macroblocks is associated with a respective random access memory.

6. The architecture of claim 5, further comprising a crossbar with each macroblock and respective random access memory to provide communication between components of the super-macroblock.

7. The architecture of claim 5, wherein inputs of a macroblock arrive from outputs of other macroblocks or inputs to the super-macroblock.

8. The architecture of claim 5, wherein outputs from a logic element are used within the respective macroblock, are transmitted to the super-macroblock, or are transmitted to other super-macroblocks through a routing network.

9. The architecture of claim 1, wherein the random access memory is a carbon nanotube random access memory.

10. The architecture of claim 1, wherein the random access memory is selected from the group consisting of phase change random access memory, magnetoresistive random access memory, and ferroelectric random access memory.

11. The architecture of claim 1, wherein the random access memory stores run-time reconfiguration bits of the respective logic element.

12. The architecture of claim 1, wherein each logic element includes two flip-flops.

13. The architecture of claim 12, wherein different computation values are stored in each of the two flip-flops at any point in time.

14. A field programmable gate array, comprising:
a plurality of reconfigurable blocks; and
an equal number of nanotube random access memories, one nanotube random access memory being associated each reconfigurable block to store run-time reconfiguration bits of the respective block.

15. The gate array of claim 14, wherein the reconfigurable blocks are logic elements or switch blocks.

16. A reconfigurable computer architecture, comprising:
a plurality of programmable elements; and
a random access memory physically adjacent and connected to each programmable element.

17. The architecture of claim 16, wherein the random access memory stores run-time reconfiguration bits of the respective programmable element.

18. The architecture of claim 16, wherein n- programmable elements and n- random access memories comprise a macro-block (MB), and wherein m- macro-blocks and m-random access memories comprise a super macro-block (SMB), and wherein one SMB and one local switch matrix comprise a logic block (LB), and wherein a plurality of LBs are included in the architecture.

19. The architecture of claim 18, wherein the value of m and n is four.

20. The architecture of claim 18, wherein inputs/outputs of the SMB are connected to neighboring logic blocks through an interconnection network through the local switch matrix and

SMBs of the neighboring logic blocks are also direct link connected to the inputs/outputs of the SMB.

21. The architecture of claim 18, further comprising wire routing segments of length-1, length-2 and long wires, wherein length-1 wire segments span one logic block before connecting to a switch block, length-2 wire segments span four logic blocks before connecting to a switch block, and long wire segments traverse and are connected to all logic blocks in a respective horizontal and vertical path.

22. The architecture of claim 21, further comprising direct link connections from outputs of each logic block to each of four immediately neighboring logic blocks.

23. The architecture of claim 18, wherein the value of m and n is four, wherein each SMB has 64 inputs and 32 outputs, and wherein 128 horizontal and vertical wires are included in each routing channel.

24. The architecture of claim 23, wherein a 25%, 50%, and 25% distribution of length-1, length-2, and long wires, respectively, exists among the 128 horizontal and vertical wires.

25. The architecture of claim 23, wherein 32 wires are provided for direct link connection between adjacent SMBs.

26. The architecture of claim 18, wherein $F_c$ represents a number of adjacent tracks a pin of a logic block connects to, wherein $F_s$ represents a number of tracks to which each track entering a switch block connects to, wherein $F_c = 1/N$ for a cluster of N look-up tables, and wherein $F_s = 6$.

27. The architecture of claim 18, further comprising a connection block and a switch block associated with each LB, wherein each connection block is depopulated and each switch block is populated.

28. The architecture of claim 18, wherein each MB and SMB further comprises a crossbar to provide communication between components of the respective MB and SMB, wherein the crossbar includes a pass transistor.

29. The architecture of claim 18, wherein a multiplexer is used in each switch matrix to connect to inputs of each SMB.

30. The architecture of claim 18, wherein outputs of each SMB are connected through long interconnects through tri-state buffers in the respective switch matrix.

31. The architecture of claim 16, wherein each programmable element includes two flip-flops.

32. The architecture of claim 31, wherein different computation values are stored in each of the two flip-flops at any point in time.

33. A method of run-time reconfiguration, comprising the steps of:

writing reconfiguration bits into a first random access memory at a time of initial configuration from off-chip storage; and

placing reconfiguration bits into a second random access memory during run-time reconfiguration to configure one or more logic elements or switches to implement different logic functionality or interconnections.

34. The method of claim 33, wherein reconfiguration commences at one edge of clock signal, followed by computation at another edge of the clock signal.

35. The method of claim 33, wherein the first random access memory is nanotube random access memory.

36. The method of claim 33, wherein the second random access memory is static random access memory (SRAM).

37. A method of run-time reconfiguration, comprising the steps of:

mapping a series of n- serially connected look-up tables (LUT1, LUT2, ..., LUTn) to a logic element (LE); and

configuring the LE to implement LUT1 in a first cycle, to implement LUT2 in a second cycle, and continuing until configuring the LE to implement LUTn in nth cycle, wherein n cycles are needed for execution.

38. The method of claim 37, wherein all communications between the LUTs are local.

39. A method of run-time reconfiguration, comprising the steps of:

mapping a series of n- serially connected look-up tables (LUT1, LUT2, ..., LUTn) to one logic element (LE);

configuring the LE to implement LUT1 in a first cycle;

executing LUT1 in the first cycle;

configuring the LE to implement LUT2 in a second cycle;

executing LUT2 in the second cycle; and continuing until:

configuring the LE to implement LUTn in nth cycle; and

executing LUTn in the nth cycle.

40. A method of run-time reconfiguration, comprising the steps of:

mapping one or more of a series of look-up tables (LUTs) to one or more logic elements (LEs);

configuring the LE to implement a first LUT in a first cycle;

executing the first LUT by the LE in the first cycle;

configuring the LE to implement a second LUT in a second cycle; and

executing the second LUT in the second cycle.

41. The method of claim 40, wherein a second LE executes a LUT in the first cycle using output from the execution of the first LUT by the LE in the first cycle.

42. A method of run-time reconfiguration, comprising the steps of:

mapping one or more of a series of look-up tables (LUTs) to one or more logic elements (LEs);

configuring each LE to implement a LUT in a first cycle;

reconfiguring each LE, after the implementation of two sequential LUT computations, to implement a LUT in a second cycle.

43. A computer architecture, comprising:

a plurality of logic blocks, including:

a) a super macroblock (SMB) including:

1) at least one macroblock (MB) including:

i) at least one logic element; and

ii) at least one random access memory, wherein one random access memory is associated with each logic element, and at least as many random access memories as logic elements are included in the MB; and

2) at least one random access memory, wherein one random access memory is associated with each MB, and at least as many random access memories as MBs are included in the SMB; and

b) a switch matrix;

a plurality of switch blocks; and

a plurality of connection blocks.

44. The architecture of claim 43, wherein each connection block is depopulated and each switch block is populated.

45. The architecture of claim 43, wherein each MB and SMB further comprises a crossbar to provide communication between components of the respective MB and SMB, wherein the crossbar includes a pass transistor.

46. The architecture of claim 43, wherein a multiplexer is used in each switch matrix to connect to inputs of each SMB.

47. The architecture of claim 43, wherein outputs of each SMB are connected through long interconnects through tri-state buffers in the respective switch matrix.

48. The architecture of claim 43, further comprising wire routing segments of length-1, length-2 and long wires, wherein length-1 wire segments span one logic block before connecting to a switch block, length-2 wire segments span four logic blocks before connecting to a switch block, and long wire segments traverse and are connected to all logic blocks in a respective horizontal and vertical path.

49. The architecture of claim 43, further comprising direct link connections from outputs of each logic block to each of four immediately neighboring logic blocks.

50. A reconfigurable computer architecture, comprising:

a plurality of logic elements, each logic element including at least two flip-flops; and

at least an equal number of random access memories, wherein one random access memory is associated with each logic element to store run-time reconfiguration bits for the logic element.

51. A method implemented at least in part by machine for determining an efficient logic folding configuration for a reconfigurable architecture, and for balancing resource use across the logic folding configuration, the method comprising the steps of:

determining a folding level based upon user specified design constraints, optimization objectives, and input circuit structure, wherein force-directed scheduling is used to assign LUTs and LUT clusters to folding stages, to balance inter-folding stage resource usage, and to produce a LUT network of each temporal folding stage;

assigning the LUTs and LUT clusters to each folding stage using the LUT network as input, wherein the LUTs are assigned to LEs, and the LEs are assigned to MBs and SMBs, all to minimize global interconnects and to simplify placement and routing;

performing LUT placement, wherein inter-folding stage dependencies are considered;

generating intra-SMB and inter-SMB routing; and

generating a layout for each folding stage and a configuration bitmap for each folding cycle of the reconfigurable architecture.

52. A method implemented at least in part by machine for determining an efficient logic folding configuration for a reconfigurable architecture, and for balancing resource use across the logic folding configuration, the method comprising the steps of:

entering an input circuit design specified in register-transfer level or gate-level VHDL;

determining a folding level by identifying each plane of the input circuit design, obtaining circuit parameters within each plane, and obtaining a user optimization objective;

partitioning the register-transfer level or gate-level VHDL module into LUTs and LUT clusters;

scheduling each LUT and LUT cluster to a folding stage;

mapping each LUT and LUT cluster to a super-macroblock (SMB);

performing temporal placement;

generating intra-SMB and inter-SMB routing; and

generating a layout for each folding stage and a configuration bitmap for each folding cycle of the reconfigurable architecture.

53. The method of claim 52, wherein the user optimization objective includes circuit delay minimization under area constraint or area minimization under delay constraint.

54. The method of claim 53, wherein if an area constraint is obtained, the area constraint is satisfied first, then best possible delay determined.

55. The method of claim 52, wherein the partitioning into LUT clusters considers each of intra-folding stage and inter-folding stage data dependencies.

56. The method of claim 52, wherein a folding stage is not limited to one plane.

57. The method of claim 52, wherein the scheduling of each LUT and LUT cluster to a folding stage is implemented by force-directed scheduling, wherein force-directed scheduling comprises the steps of:

constructing a distribution graph describing resource use of LUT computations;

constructing a distribution graph describing register storage use; and

calculating a force based on both distribution graphs, wherein the force represents a probability of resource use concurrency, and a higher force calculated implies higher concurrency of run-time operations, thereby requiring more resources in parallel, wherein a LUT computation is scheduled into a folding cycle having a lowest force.

58. The method of claim 52, wherein the circuit parameters obtained are selected from the group consisting of number of planes in the input circuit design, number of LUTs in a plane, logic depth of a plane, maximum number of LUTs among all of the planes, maximum logic depth among all of the planes, area constraint and number of reconfiguration copies in each random access memory.

59. The method of claim 58, wherein the area constraint is defined as an available number of LEs.

60. The method of claim 52, wherein performing temporal placement is directed to minimizing an average length of inter-SMB interconnects.

61. The method of claim 52, wherein performing temporal placement comprises the steps of :

performing a fast placement to derive an initial placement; and

performing a low-precision routability and delay analysis; wherein, if pre-determined constraints of the delay analysis are satisfied, a detailed placement is invoked to derive a final placement, and if the pre-determined constraints of the delay analysis are not satisfied, the initial placement is refined, the pre-determined constraints of the delay analysis are verified for satisfaction, or the method step of determining a folding level is revisited.

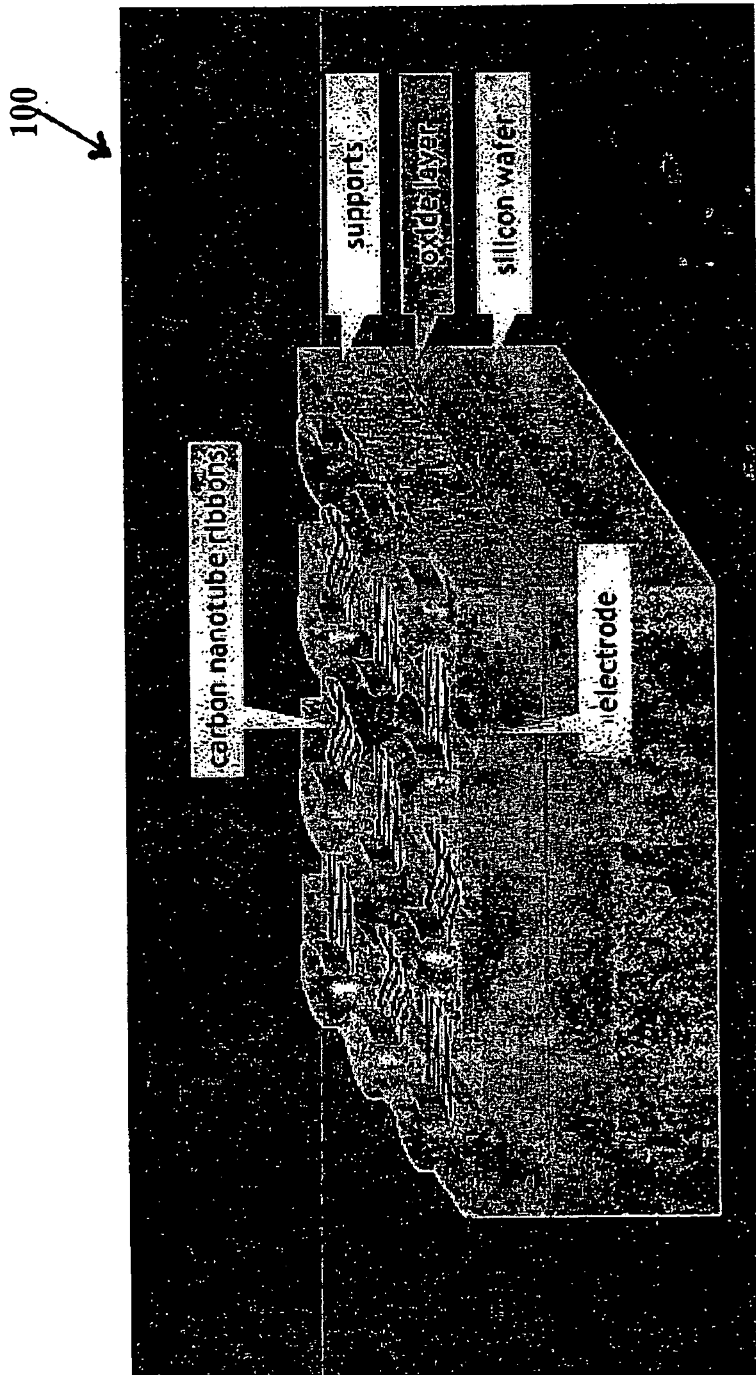



FIG. 1

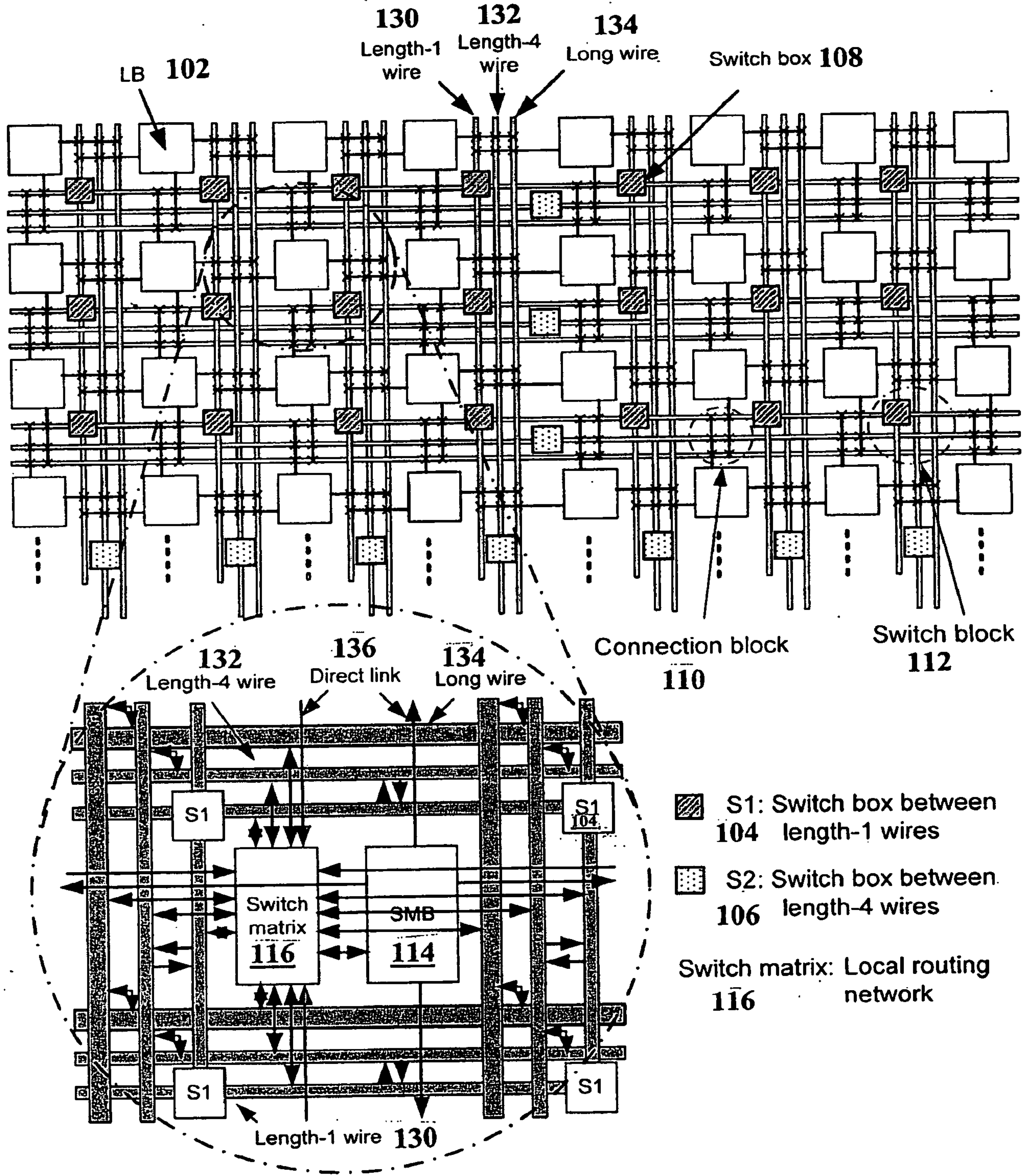



FIG. 2

8 Outputs Of MB

NRAM

48 SRAM bits

Inputs to MB

12 to 4 crossbar

LE

CLK and Global signals

Reconfiguration bits

120

122

124

100

118

FIG. 3

114

NRAM 100

144

SRAM bits 124

48 to 16 crossbar 122

MB 118

16

From Switch matrix

8

MB

48 to 16 crossbar

SRAM bits

NRAM

CLK and Global signals

Reconfiguration bits

32 Outputs Of SMB

FIG. 4

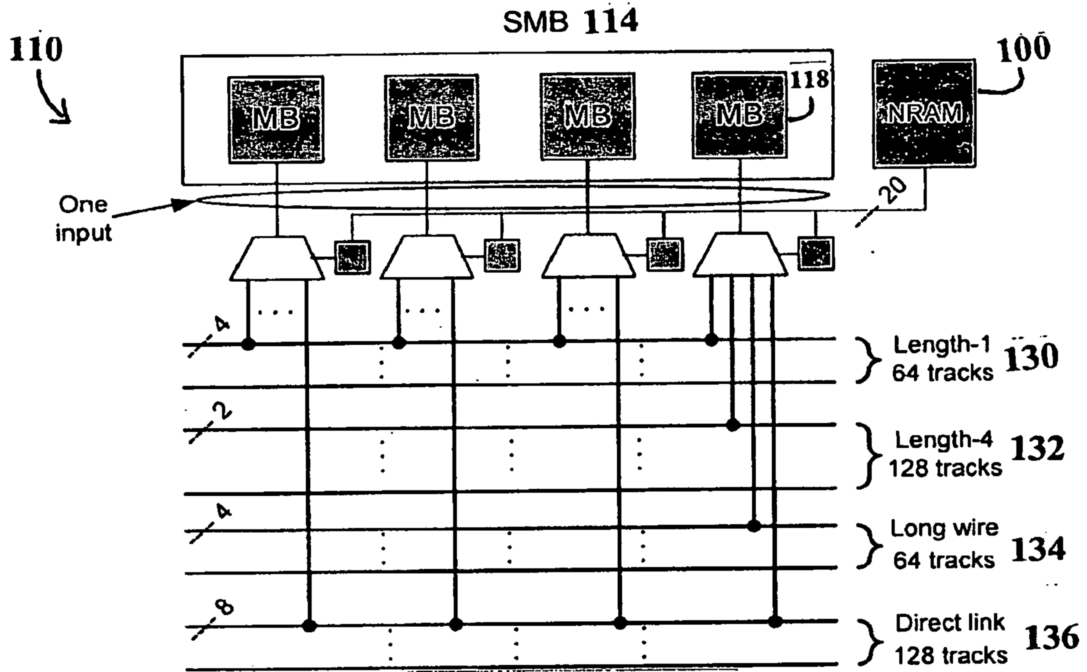



FIG. 5A

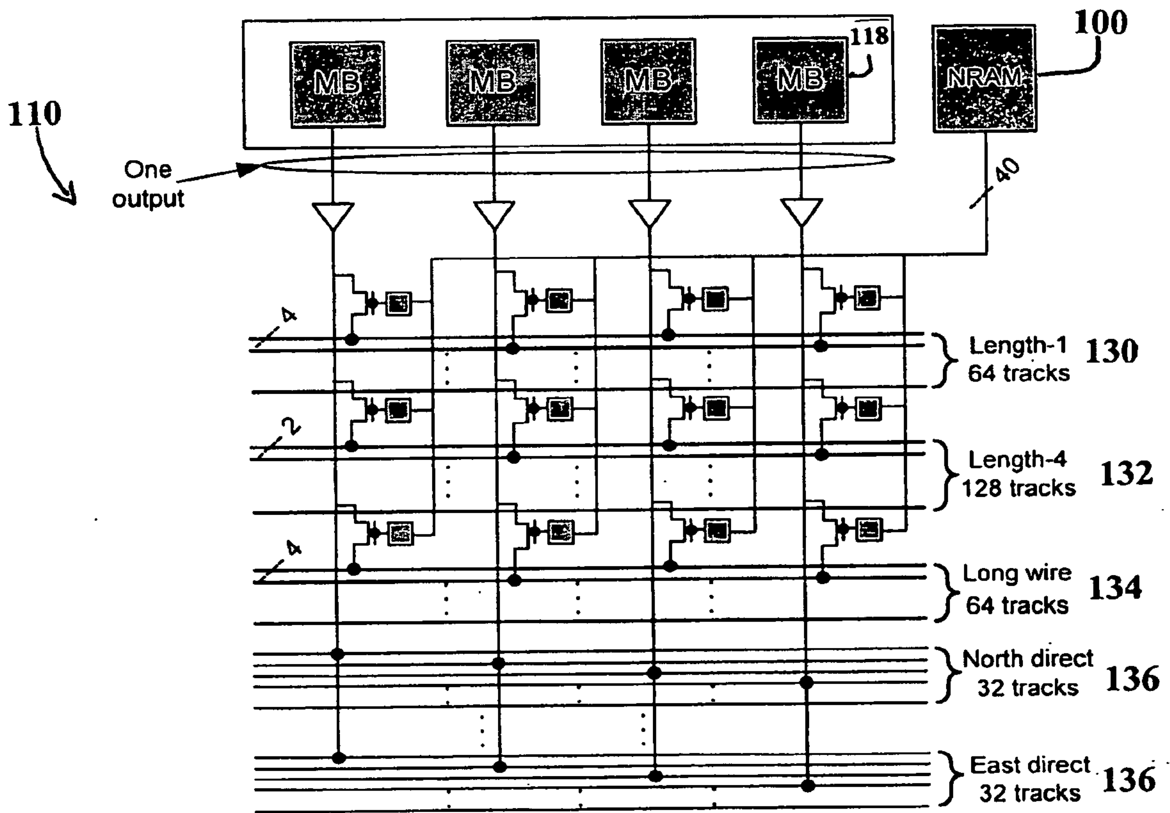



FIG. 5B

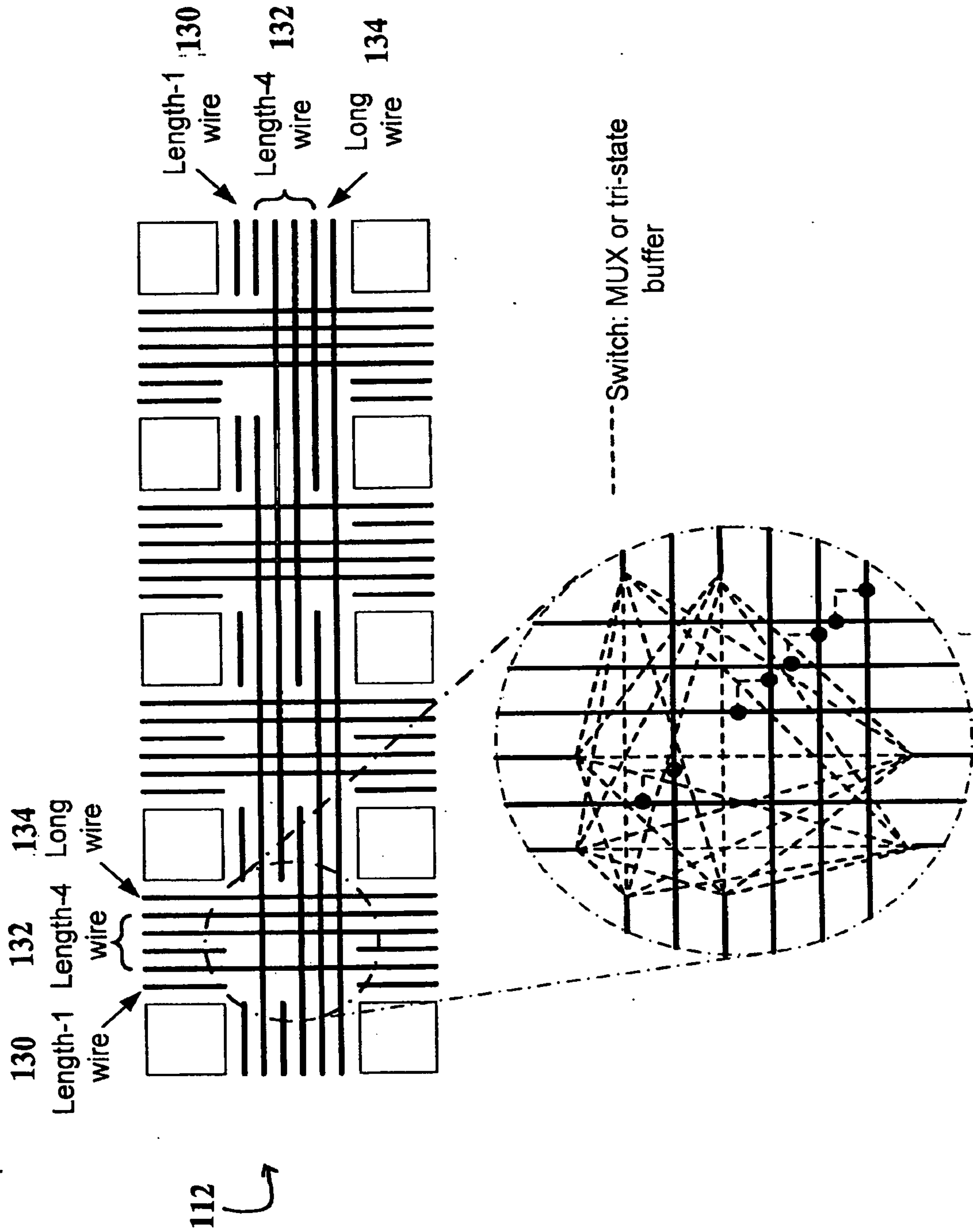



FIG. 6

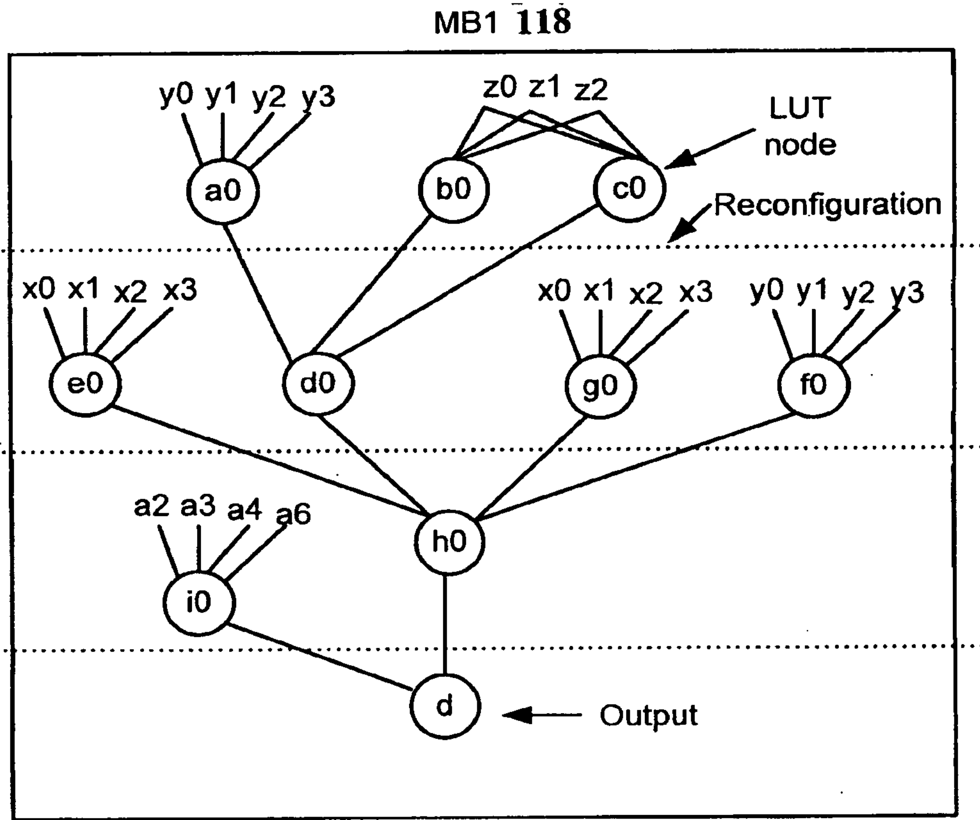



FIG. 7A

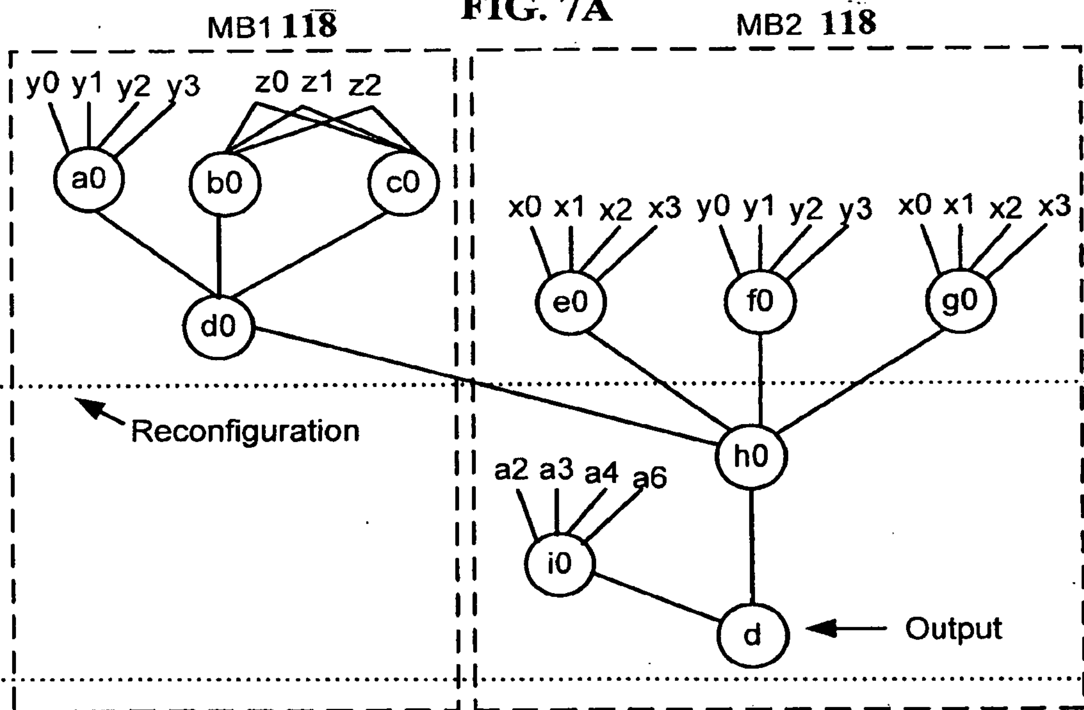



FIG. 7B

## TABLE I

## CIRCUIT MAPPING RESULTS

| Circuit | LUT graph depth | Level-1 folding | | | | Level-2 folding | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LEs | Delay (*ns*) | LEs *x* Delay | Freq. (*GHz*) | LEs | Delay (*ns*) | LEs *x* Delay | Freq. (*GHz*) |
| pml | 3 | 9 | 1.08 | 9.72 | 2.78 | 11 | 1.00 | 11.00 | 2.00 |
| sct | 4 | 9 | 1.64 | 14.76 | 2.86 | 16 | 1.10 | 17.60 | 1.82 |
| cm163a | 4 | 4 | 1.40 | 5.60 | 2.86 | 8 | 1.10 | 8.80 | 1.82 |
| z4ml | 5 | 2 | 1.75 | 3.5 | 2.86 | 3 | 1.47 | 4.41 | 2.04 |
| cc | 5 | 13 | 2.05 | 26.65 | 2.44 | 14 | 2.03 | 28.35 | 1.48 |
| poler8 | 7 | 9 | 2.87 | 25.83 | 2.44 | 10 | 2.20 | 22.0 | 1.82 |
| cordic | 7 | 7 | 2.87 | 20.09 | 2.44 | 12 | 2.20 | 26.40 | 1.82 |
| lal | 7 | 24 | 2.52 | 60.48 | 2.78 | 32 | 2.00 | 64.00 | 2.00 |
| ldd | 7 | 10 | 2.94 | 29.40 | 2.38 | 17 | 2.48 | 42.16 | 1.61 |
| 9symml | 18 | 10 | 10.36 | 103.60 | 1.72 | 30 | 7.45 | 223.50 | 1.20 |
| alu2 | 31 | 12 | 12.71 | 152.52 | 2.68 | 24 | 11.76 | 282.24 | 1.36 |
| 16-bit ripple-carry adder | 16 | 2 | 4.80 | 9.60 | 3.33 | 4 | 3.52 | 14.08 | 2.27 |
| 32-bit ripple-carry adder | 32 | 2 | 9.60 | 19.20 | 3.33 | 4 | 7.04 | 28.16 | 2.27 |
| 64-bit ripple-carry adder | 64 | 2 | 19.20 | 38.40 | 3.33 | 4 | 14.08 | 56.32 | 2.27 |
| 16-bit carry-lookahead adder | 10 | 13 | 3.60 | 46.80 | 2.78 | 26 | 3.43 | 89.13 | 1.46 |
| 32-bit carry-lookahead adder | 18 | 13 | 6.48 | 84.24 | 2.78 | 26 | 6.17 | 160.42 | 1.46 |
| 64-bit carry-lookahead adder | 34 | 13 | 12.24 | 159.12 | 2.78 | 26 | 11.65 | 302.90 | 1.46 |
| 16-bit carry-select adder | 8 | 30 | 2.88 | 86.4 | 2.78 | 55 | 2.76 | 151.80 | 1.46 |
| 32-bit carry-select adder | 14 | 30 | 5.04 | 150.12 | 2.78 | 55 | 4.80 | 264.00 | 1.46 |
| 64-bit carry-select adder | 26 | 30 | 9.36 | 280.80 | 2.78 | 55 | 8.91 | 490.05 | 1.46 |
| 8-bit multiplier | 8 | 16 | 3.88 | 62.08 | 2.06 | 32 | 3.24 | 103.68 | 1.23 |
| 16-bit multiplier | 16 | 32 | 7.76 | 248.32 | 2.06 | 64 | 6.48 | 414.72 | 1.23 |
| 32-bit multiplier | 32 | 64 | 15.52 | 993.28 | 2.06 | 128 | 12.96 | 1658.88 | 1.23 |

| Circuit | LUT graph depth | Level-4 folding | | | | No folding | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | LEs | Delay (*ns*) | LEs *x* Delay | Freq. (*GHz*) | LEs | Delay (*ns*) | LEs *x* Delay | Freq. (*GHz*) |
| pml | 3 | 22 | 0.61 | 13.42 | 1.64 | 22 | 0.61 | 13.42 | 1.64 |
| sct | 4 | 32 | 0.62 | 19.84 | 1.61 | 32 | 0.62 | 19.84 | 1.61 |
| cm163a | 4 | 14 | 0.59 | 8.26 | 1.69 | 14 | 0.59 | 8.26 | 1.69 |
| z4ml | 5 | 5 | 1.25 | 6.25 | 1.60 | 9 | 0.73 | 6.57 | 1.37 |
| cc | 5 | 24 | 1.91 | 45.84 | 1.05 | 28 | 1.14 | 31.78 | 0.88 |
| poler8 | 7 | 16 | 1.54 | 24.64 | 1.30 | 35 | 1.01 | 35.35 | 0.99 |
| cordic | 7 | 14 | 1.60 | 22.40 | 1.25 | 28 | 1.07 | 29.96 | 0.93 |
| lal | 7 | 56 | 1.56 | 87.36 | 2.56 | 73 | 1.07 | 78.11 | 0.93 |
| ldd | 7 | 31 | 2.05 | 63.55 | 0.98 | 44 | 1.38 | 60.72 | 0.72 |
| 9symml | 18 | 54 | 6.85 | 369.90 | 0.73 | 124 | 3.83 | 474.92 | 0.26 |
| alu2 | 31 | 44 | 9.36 | 411.84 | 0.85 | 211 | 6.14 | 1295.54 | 0.16 |
| 16-bit ripple-carry adder | 16 | 8 | 2.88 | 23.04 | 1.39 | 32 | 2.39 | 76.48 | 0.42 |
| 32-bit ripple-carry adder | 32 | 8 | 5.76 | 46.08 | 1.39 | 64 | 5.00 | 320.00 | 0.20 |
| 64-bit ripple-carry adder | 64 | 8 | 11.52 | 92.16 | 1.39 | 128 | 10.21 | 1306.88 | 0.10 |
| 16-bit carry-lookahead adder | 10 | 52 | 3.63 | 163.35 | 0.83 | 104 | 2.35 | 244.40 | 0.43 |
| 32-bit carry-lookahead adder | 18 | 52 | 6.05 | 272.25 | 0.83 | 208 | 4.45 | 925.60 | 0.23 |
| 64-bit carry-lookahead adder | 34 | 52 | 10.89 | 490.05 | 0.83 | 416 | 8.65 | 3598.4 | 0.12 |
| 16-bit carry-select adder | 8 | 93 | 2.42 | 225.06 | 0.83 | 134 | 2.21 | 296.14 | 0.45 |
| 32-bit carry-select adder | 14 | 93 | 4.84 | 450.12 | 0.83 | 268 | 4.31 | 1155.08 | 0.23 |
| 64-bit carry-select adder | 26 | 93 | 8.47 | 787.71 | 0.83 | 536 | 8.51 | 4561.36 | 0.12 |
| 8-bit multiplier | 8 | 64 | 2.92 | 186.88 | 0.68 | 128 | 2.39 | 305.92 | 0.42 |
| 16-bit multiplier | 16 | 128 | 5.84 | 747.52 | 0.68 | 512 | 4.98 | 2549.76 | 0.20 |
| 32-bit multiplier | 32 | 256 | 11.68 | 2990.08 | 0.68 | 2048 | 9.54 | 19537.92 | 0.10 |

**FIG. 8**

8 Outputs of MB
LE
LE
LE
LE
120
13 to 5 crossbar 122
13 to 5 crossbar
13 to 5 crossbar
13 to 5 crossbar
65 SRAM bits 124
65 SRAM bits
65 SRAM bits
65 SRAM bits
NRAM
NRAM
NRAM
NRAM
100
Inputs to MB
Inputs to MB
Inputs to MB
CLK and Global signals
CLK and Global signals
Reconfiguration bits
Reconfiguration bits
118

FIG. 9

114

NRAM

100

MB

118

120

SRAM bits 124

20 44X1 MUX 138

8

20

From Switch matrix 116

Output to Interconnect

CLK and Global signals

Reconfiguration bits

SRAM bits

FIG. 10

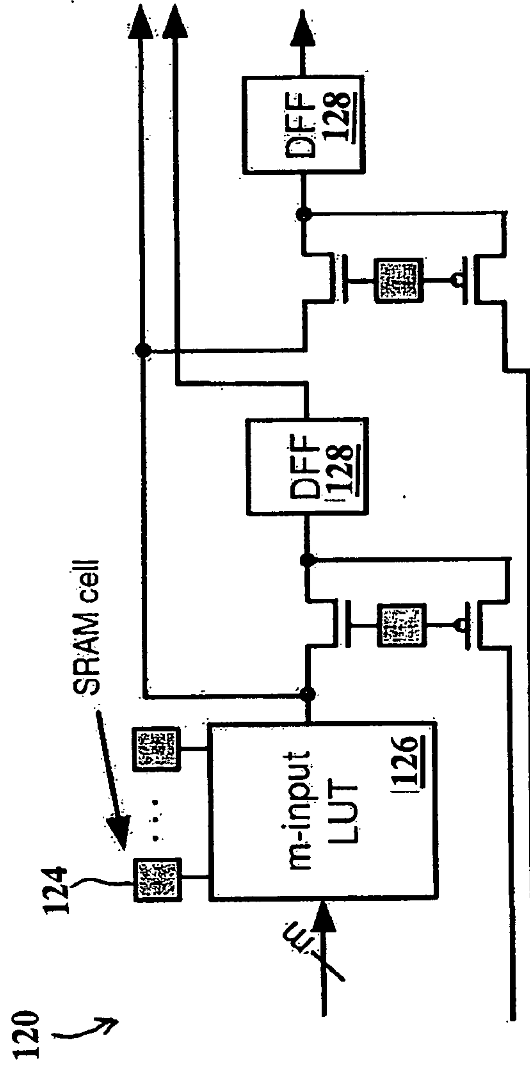



FIG. 11B

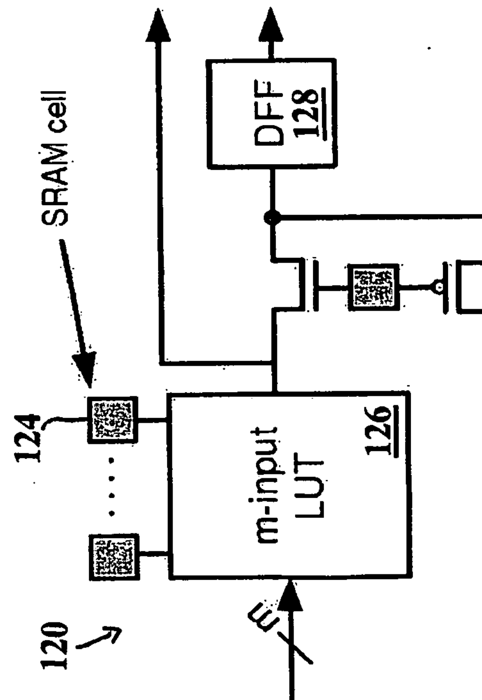



FIG. 11A

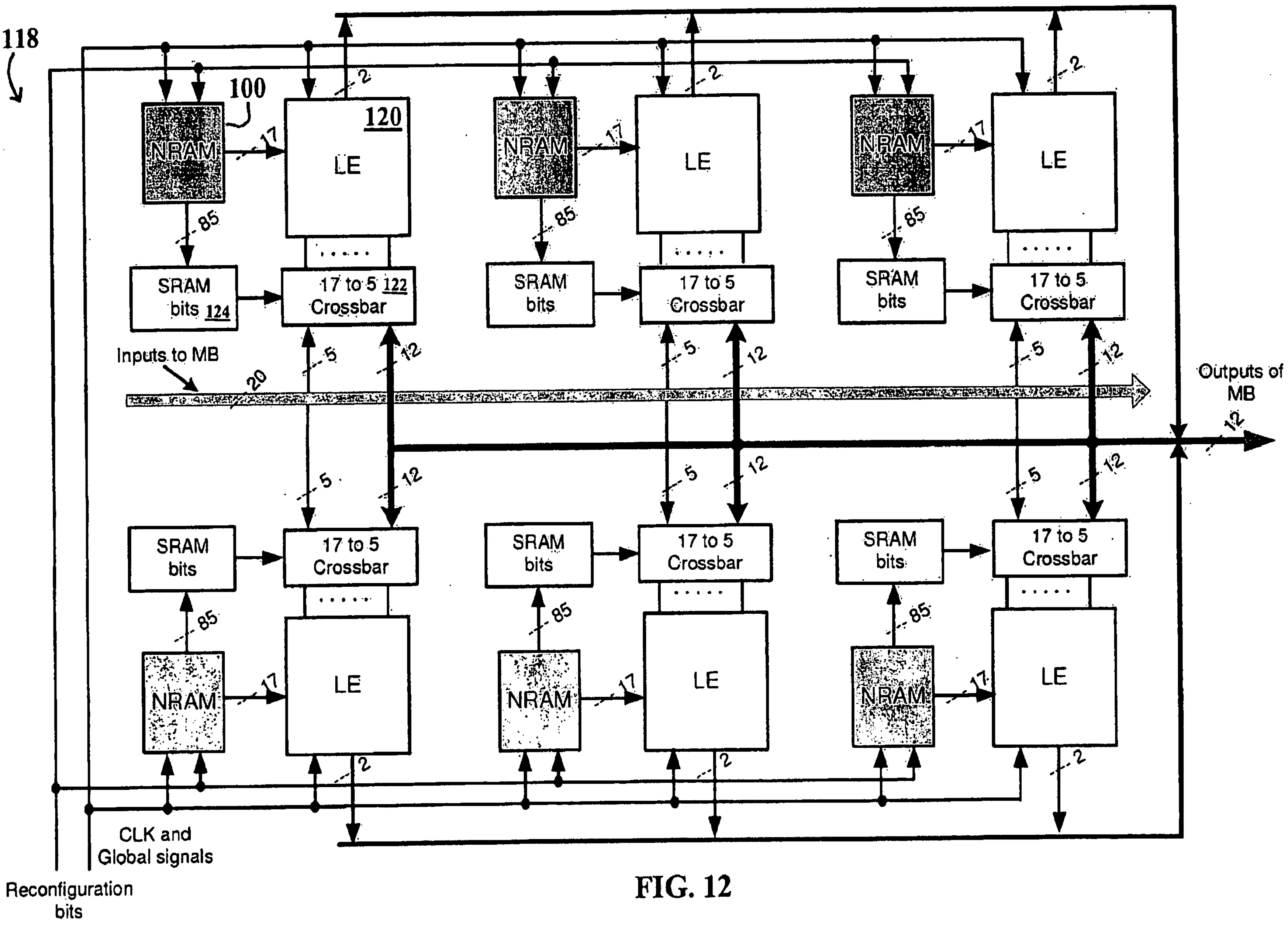



FIG. 12

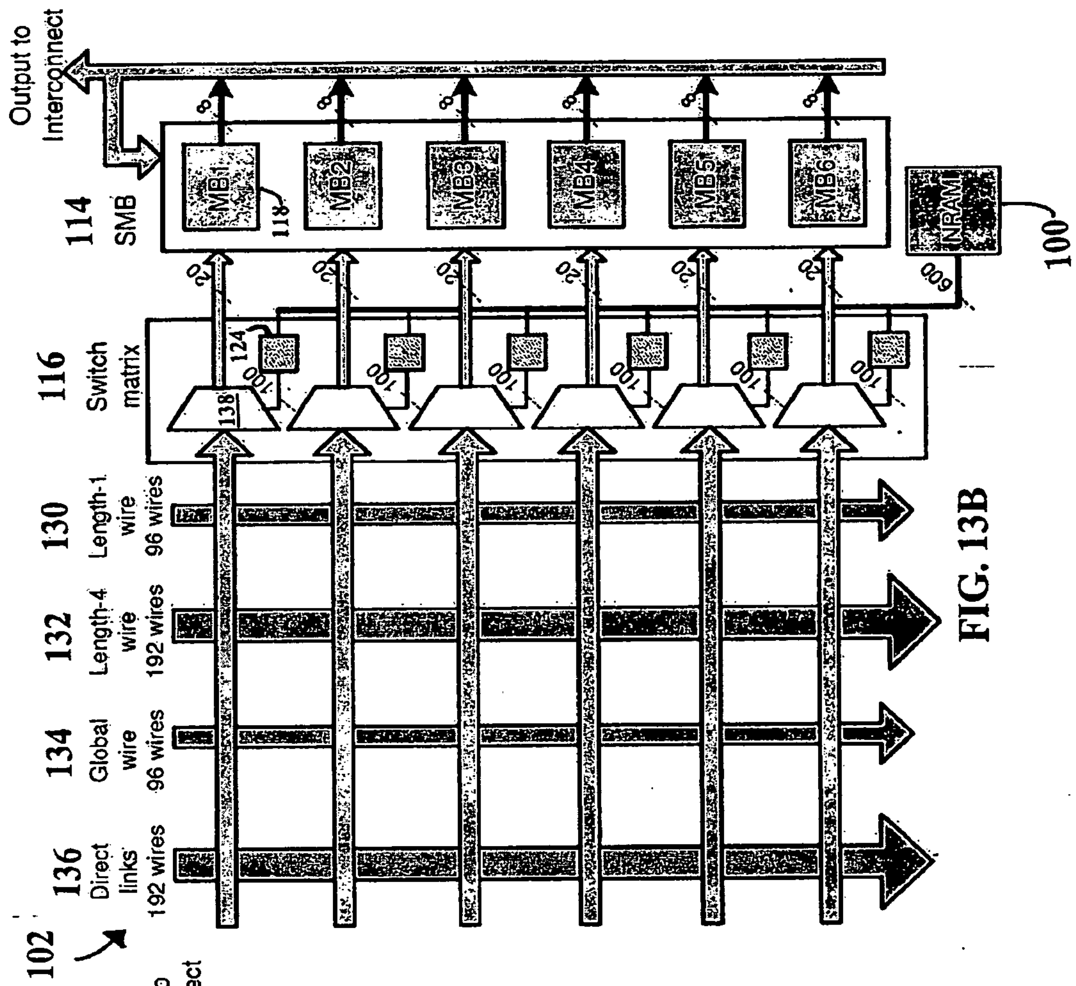



FIG. 13B

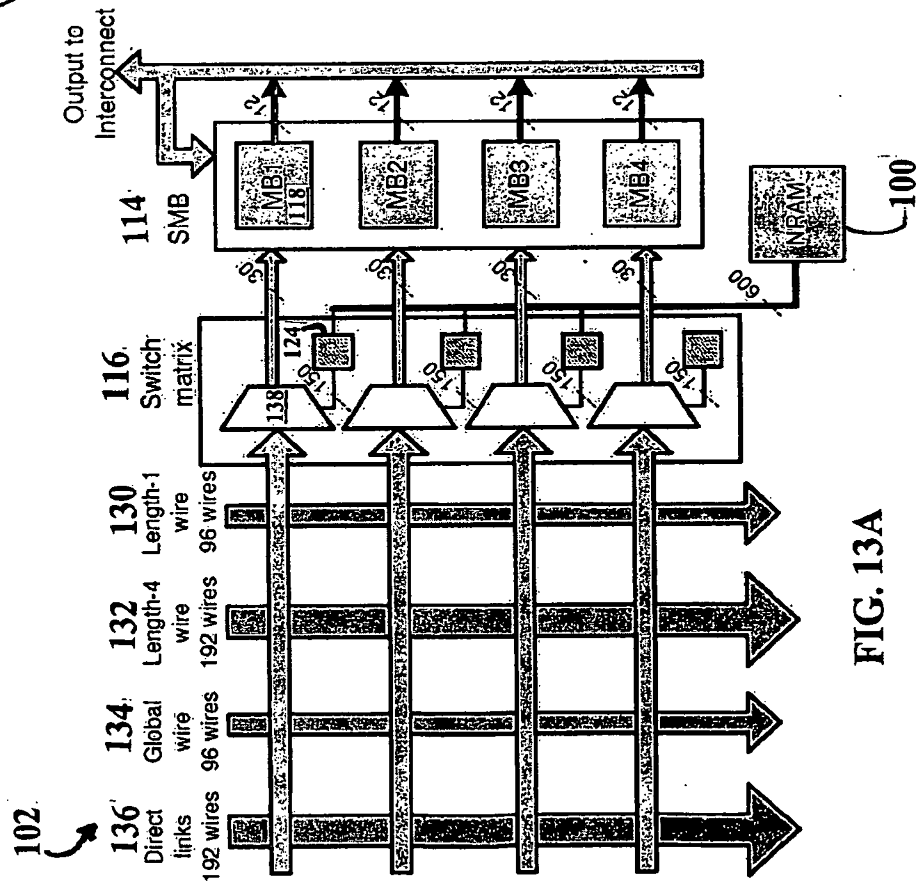



FIG. 13A

FIG. 14

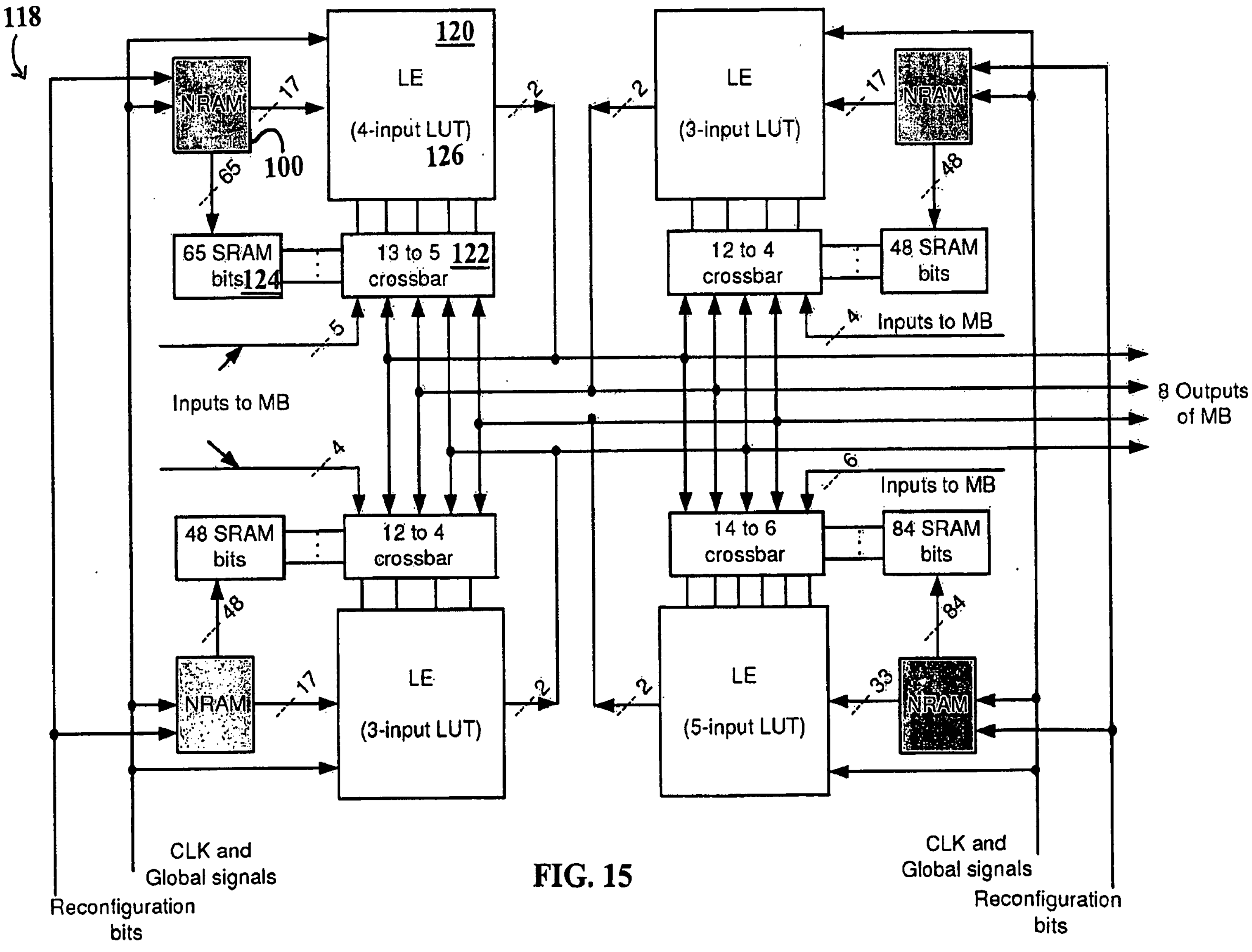



FIG. 15

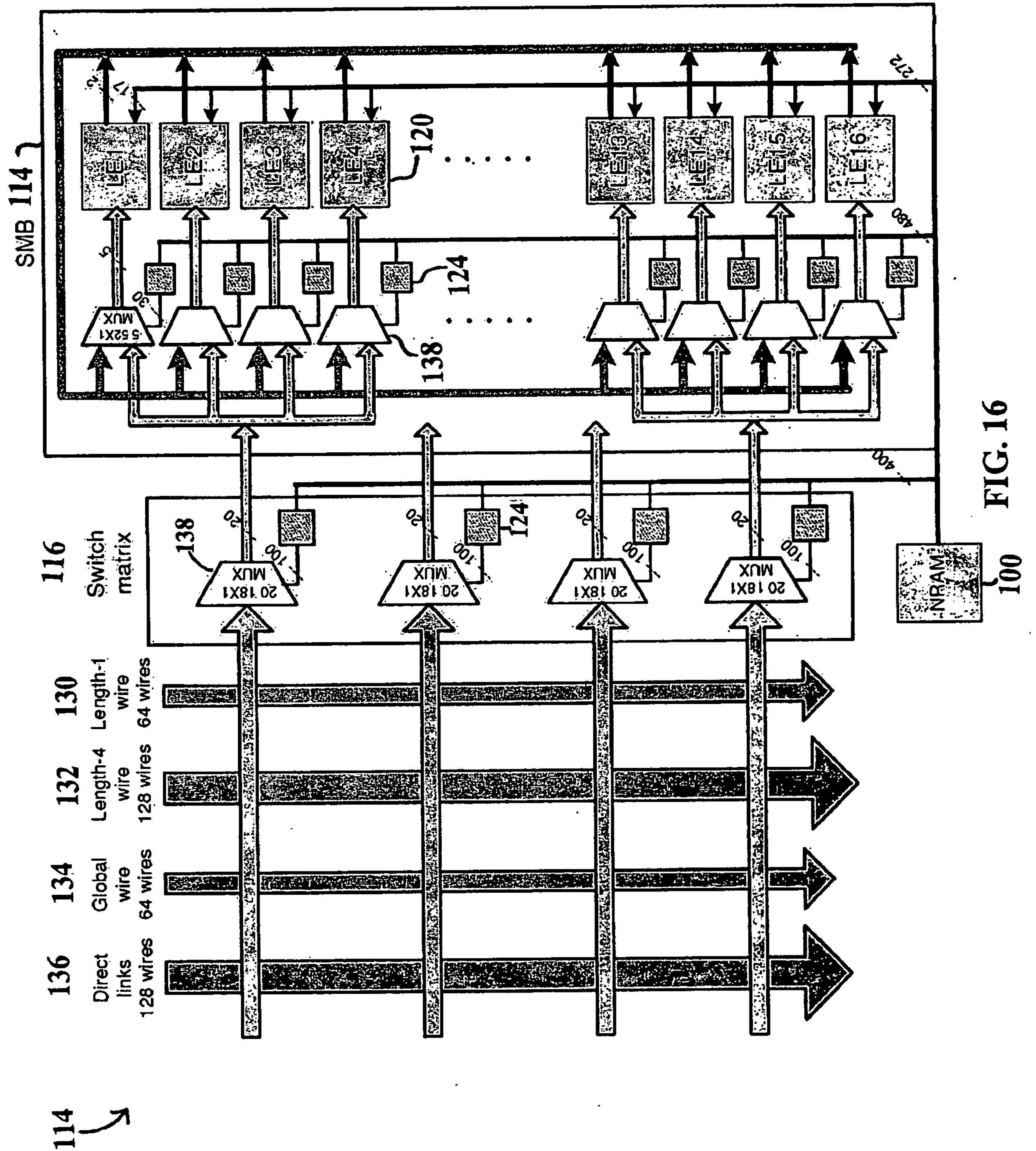



FIG. 16

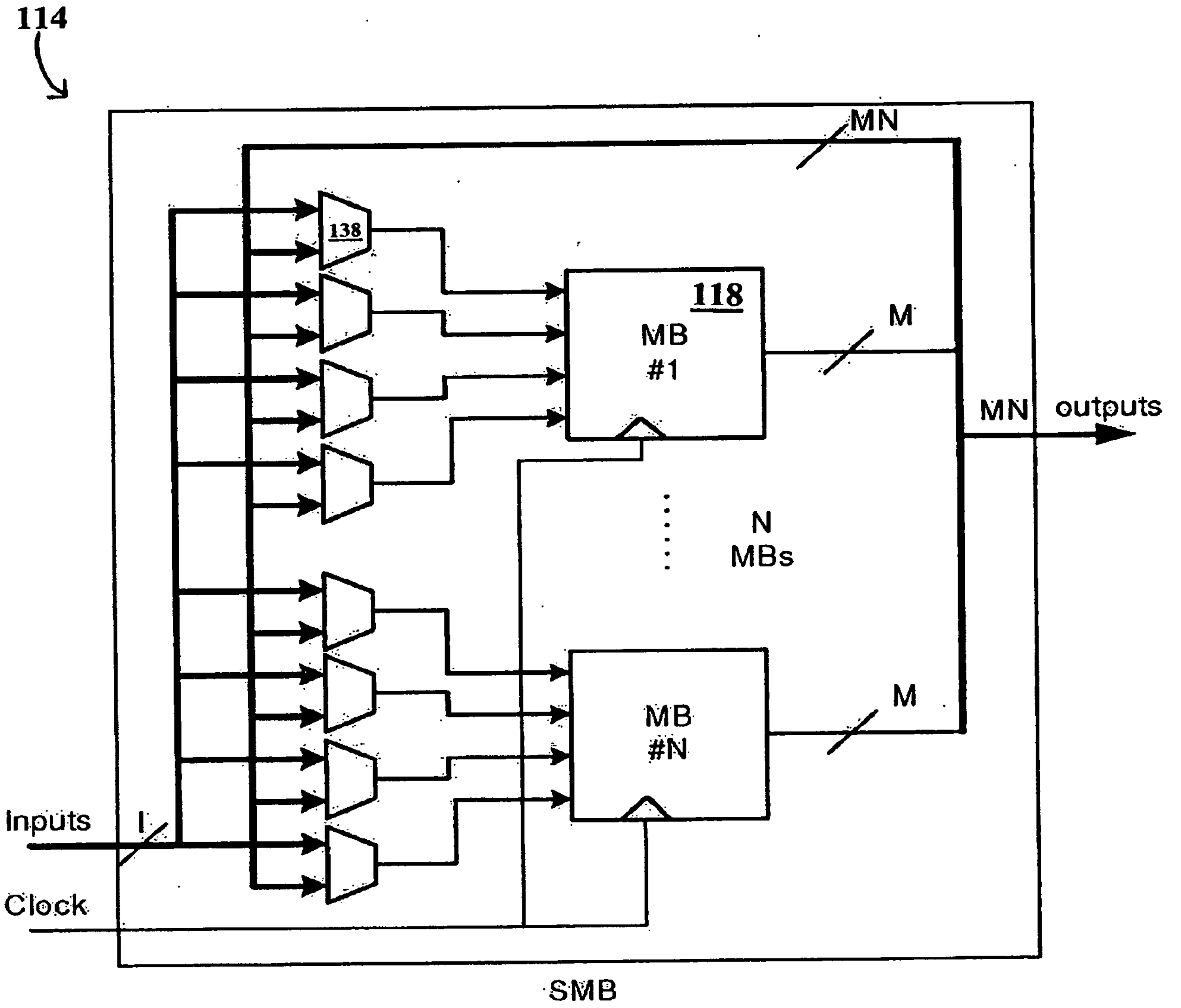



FIG. 17

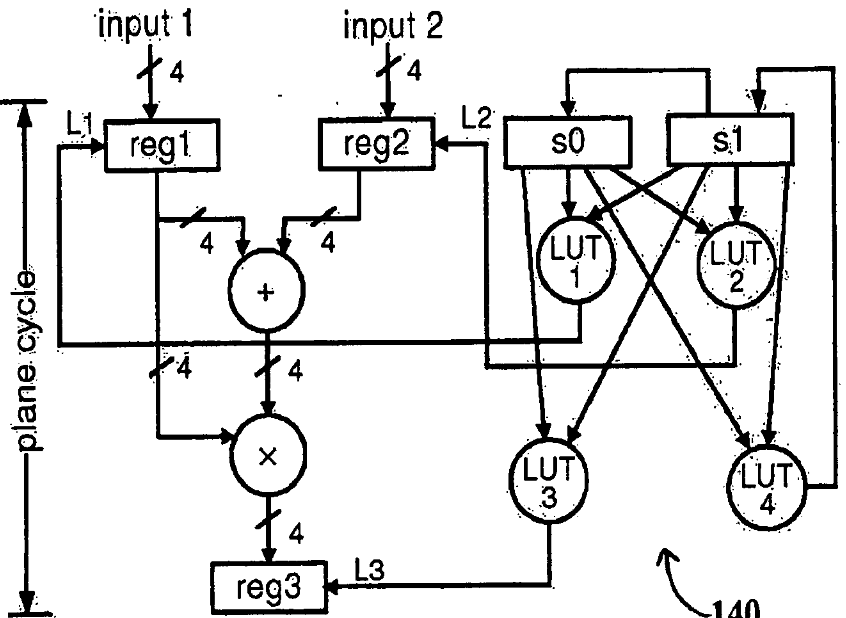


FIG. 18A

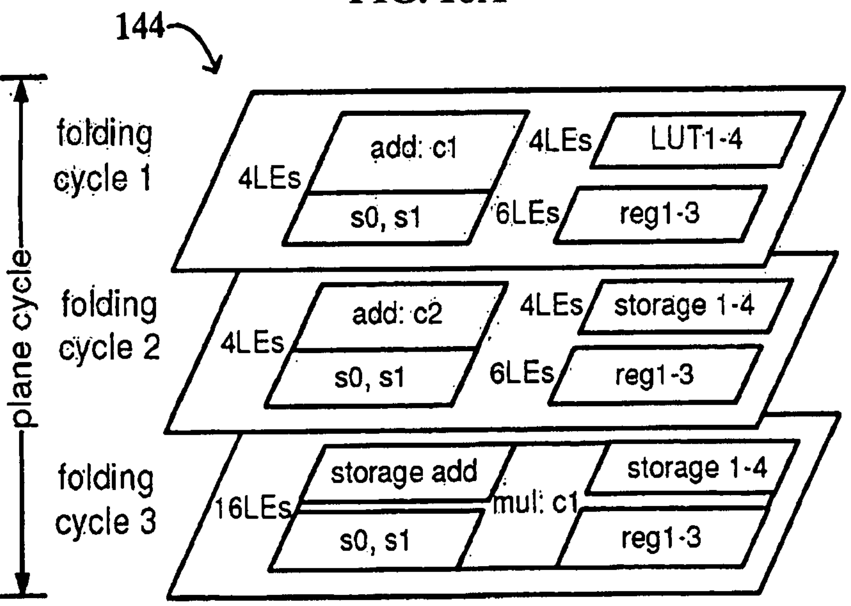


FIG. 18C

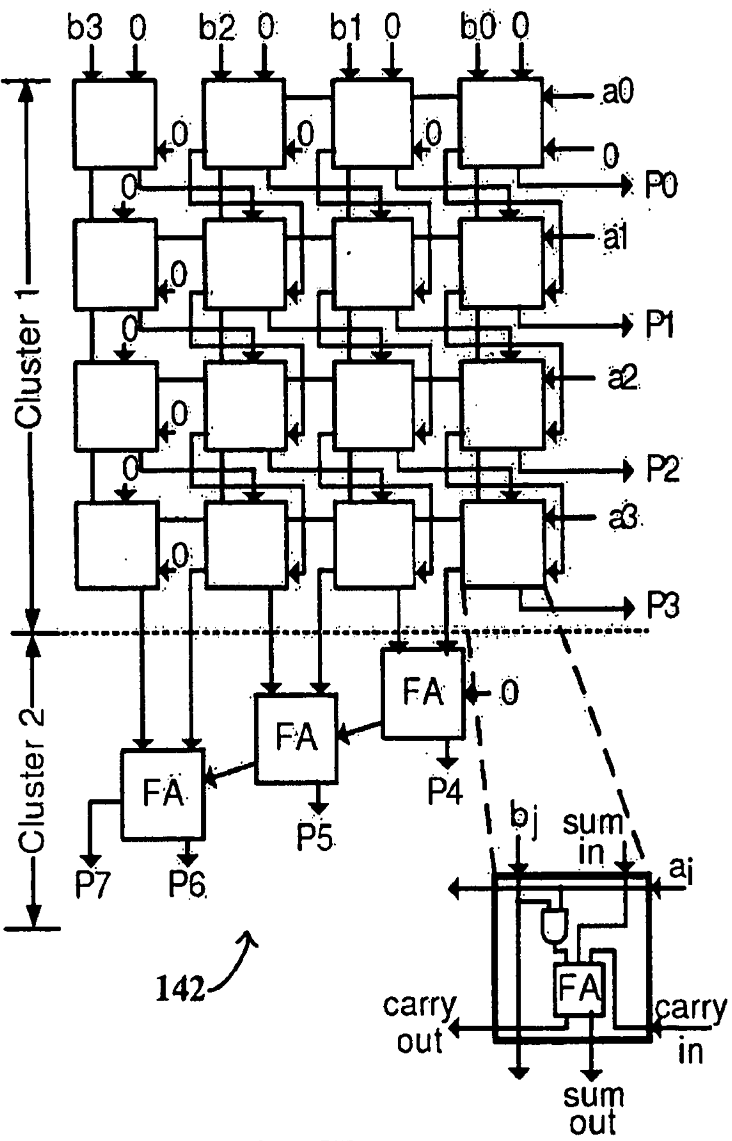


FIG. 18B

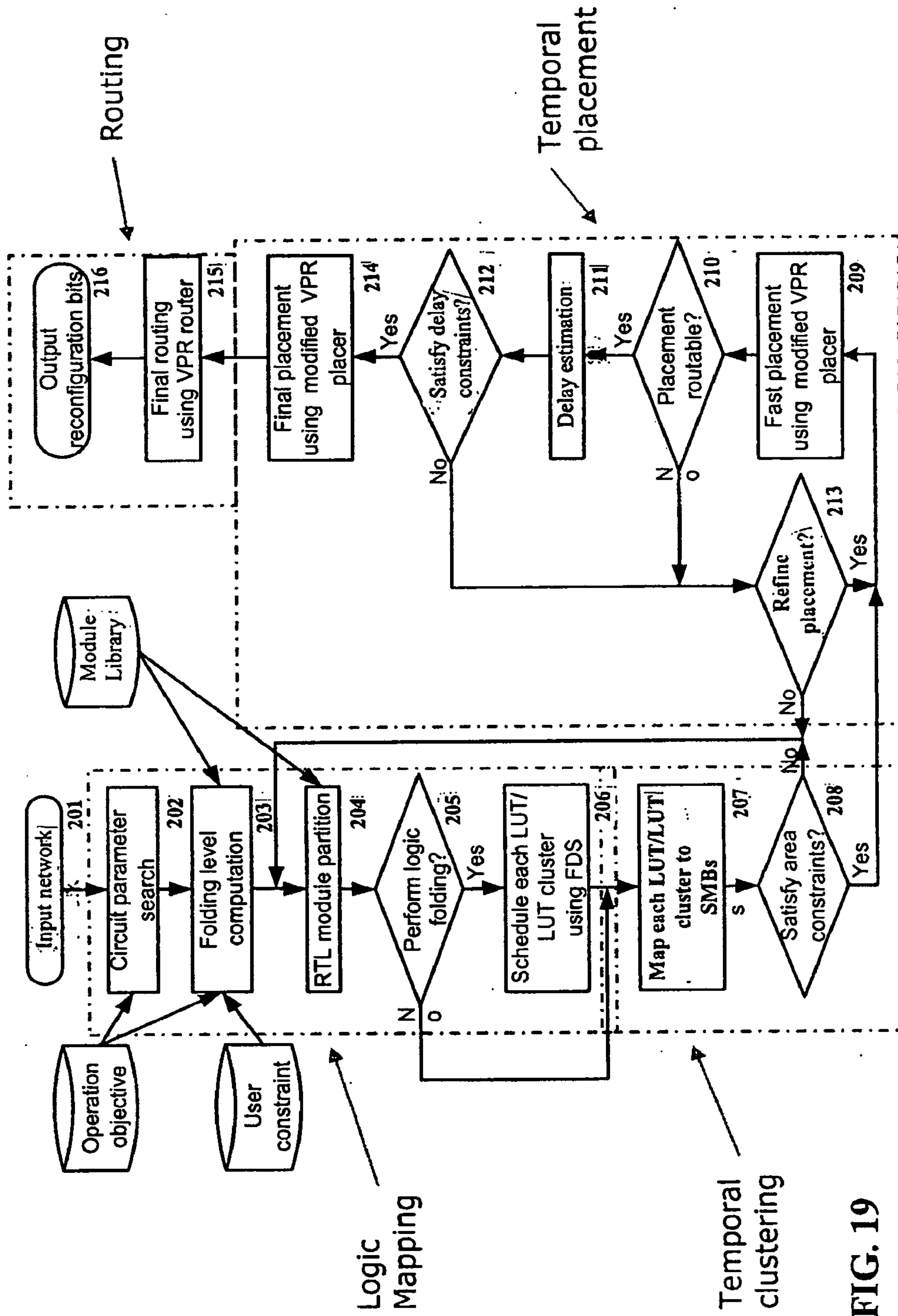



FIG. 19

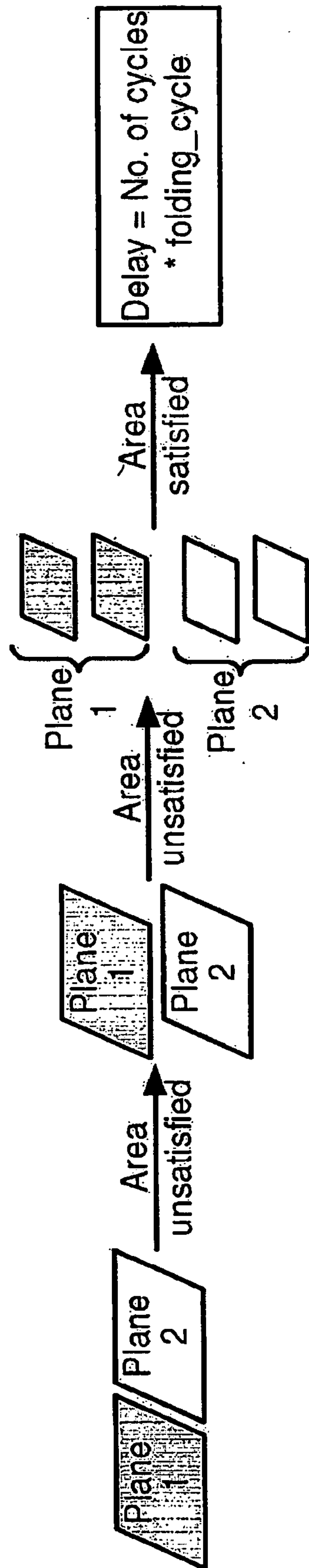



FIG. 20

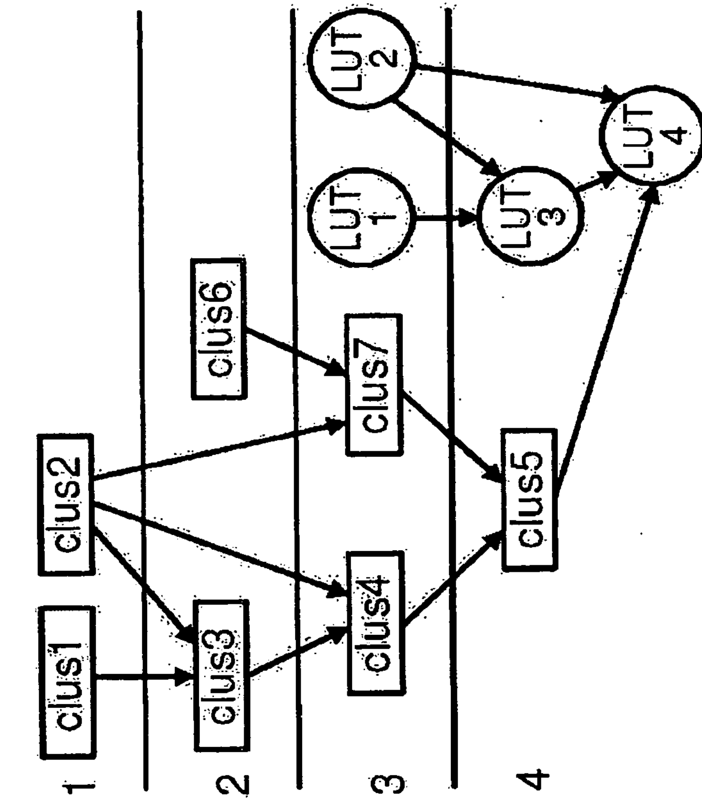



FIG. 21B

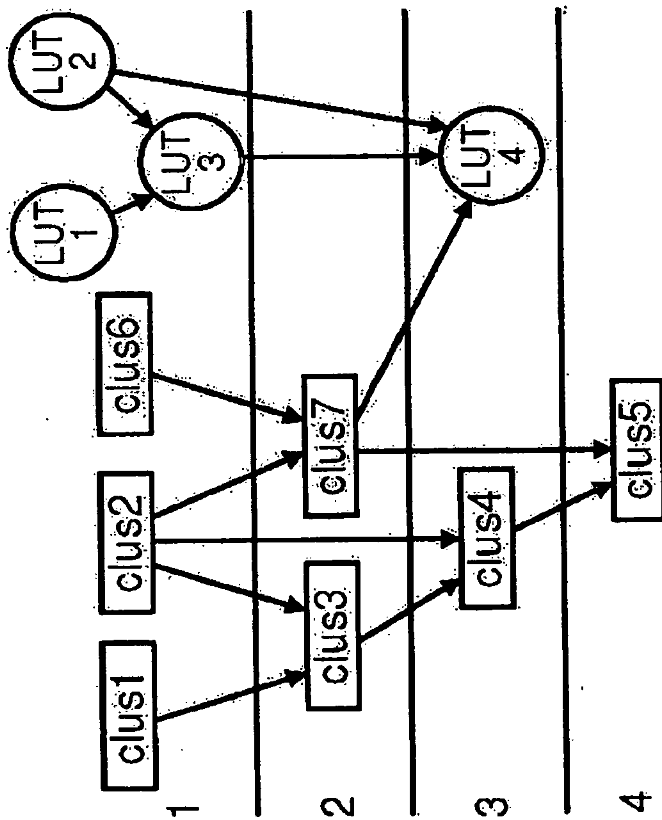



FIG. 21A

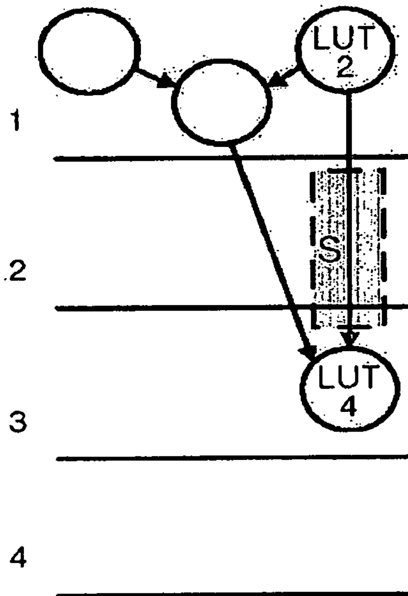



FIG. 22A

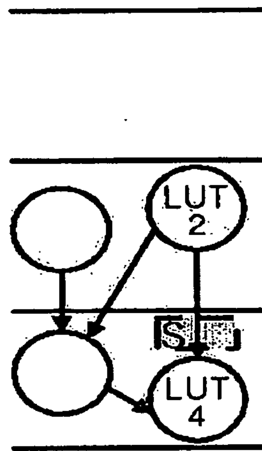



FIG. 22B

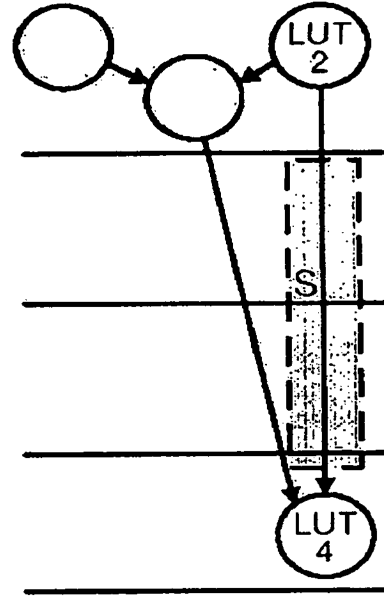



FIG. 22C

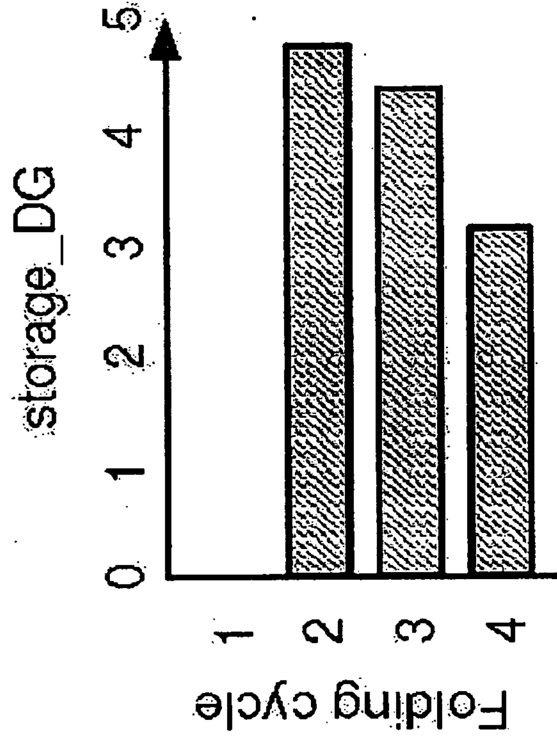



FIG. 23B

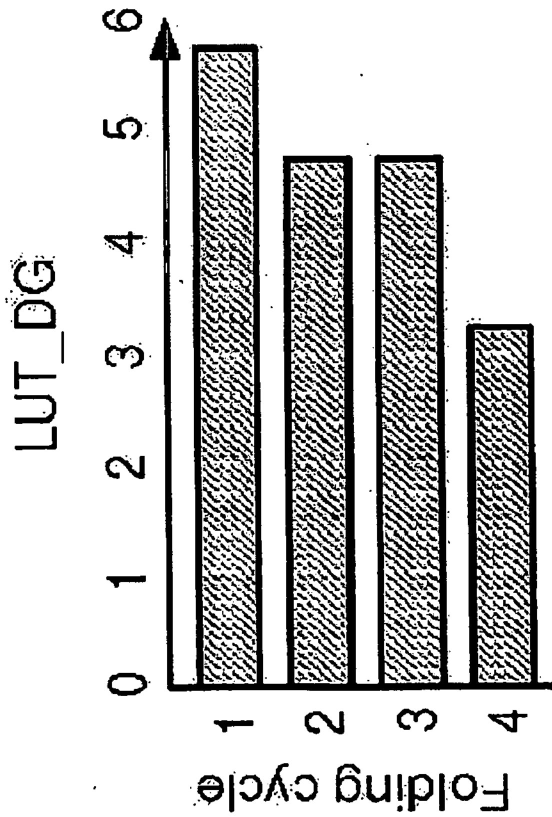



FIG. 23A

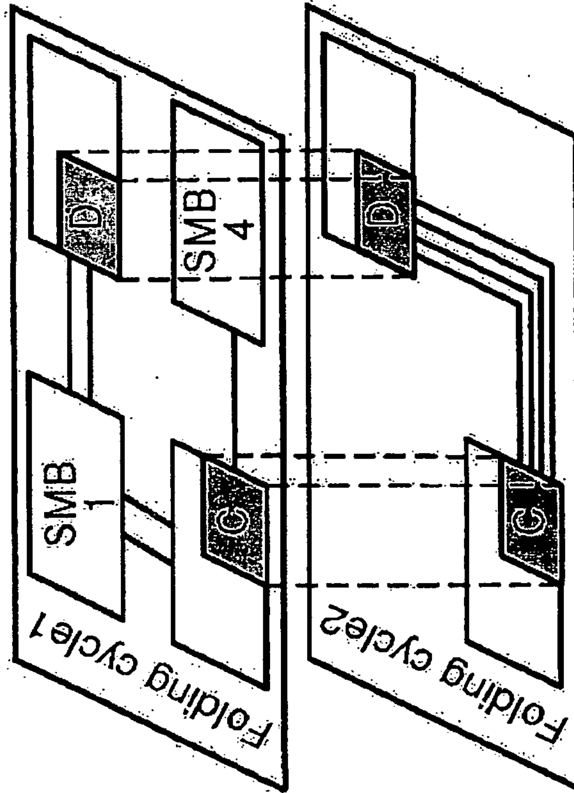



FIG. 24B

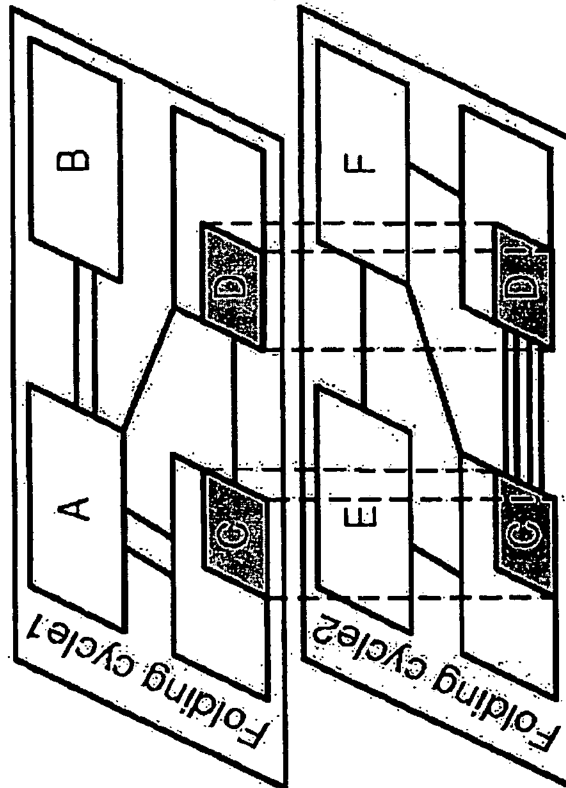



FIG. 24A

## TABLE II

## CIRCUIT MAPPING RESULTS FOR AT PRODUCT OPTIMIZATION

| Circuit | # Planes | Max plane depth | #LUTs | #Flip flops | No folding | | AT optimization ($k$ enough) | | | | AT optimization ($k$ = 16) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | #LEs | Delay ($ns$) | Folding level | #LEs | Delay ($ns$) | AT Improv. | Folding level | #LEs | Delay ($ns$) | AT Improv. |
| ex1 | 1 | 24 | 644 | 50 | 644 | 12.90 | 1 | 34 | 17.02 | 14.36X | 2 | 68 | 15.60 | 7.83X |
| FIR | 1 | 25 | 678 | 112 | 678 | 14.20 | 1 | 56 | 18.50 | 9.29X | 2 | 72 | 16.90 | 7.91X |
| ex2 | 3 | 22 | 694 | 130 | 694 | 38.76 | 1 | 67 | 48.84 | 8.22X | 2 | 88 | 42.90 | 7.13X |
| c5315 | 1 | 14 | 792 | 0 | 792 | 7.86 | 1 | 144 | 10.36 | 4.17X | 1 | 144 | 10.36 | 4.17X |
| Biquad | 1 | 22 | 1376 | 64 | 1376 | 12.34 | 1 | 68 | 16.28 | 15.34X | 2 | 136 | 14.30 | 8.73X |
| Paulin | 2 | 24 | 1468 | 147 | 1468 | 26.74 | 1 | 106 | 35.52 | 10.43X | 2 | 136 | 31.20 | 9.25X |
| ASPP4 | 2 | 24 | 2240 | 160 | 2240 | 26.80 | 1 | 100 | 36.96 | 16.24X | 2 | 200 | 32.40 | 9.26X |

## TABLE III

## CIRCUIT MAPPING RESULTS FOR TYPICAL OPTIMIZATION

| Circuit | Optimization | Area const. (#LEs) | Delay const. ($ns$) | Folding level | #LEs | Delay ($ns$) |
|---|---|---|---|---|---|---|
| ex1 | Delay | - | - | 1 | 34 | 17.02 |
| FIR | Delay | 110 | - | 3 | 108 | 16.74 |
| ex2 | Area | - | 40 | 11 | 352 | 38.04 |
| c5315 | Area | - | - | 1 | 144 | 10.36 |
| Biquad | Delay | 100 | - | 1 | 68 | 16.28 |
| Paulin | - | 210 | 30 | 3 | 204 | 29.76 |
| ASPP4 | Area | - | 28.5 | 6 | 600 | 28.32 |

**FIG. 25**