DIGITAL LOGIC OPTIMIZATION USING SELECTION OPERATORS

Inventors: Sterling R. Whitaker, Albuquerque, NM (US); Lowell H. Miles, Albuquerque, NM (US); Eric G. Cameron, Lake Forest, CA (US); Jody W. Gambles, Albuquerque, NM (US)

Assignee: Science & Technology Corporation @ UNM, Albuquerque, NM (US)

Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 28 days.

Appl. No.: 10/172,743
Filed: Jun. 14, 2002

Prior Publication Data

Related U.S. Application Data
Provisional application No. 60/298,832, filed on Jun. 15, 2001.

Int. Cl.7 ............................................. G06F 17/50
U.S. Cl. ...................... 716/3; 716/1; 716/2; 716/18
Field of Search ....................... 716/3, 1-2, 4-18

References Cited
U.S. PATENT DOCUMENTS
4,792,909 A 12/1988 Serlet
5,040,139 A 8/1991 Tran

FOREIGN PATENT DOCUMENTS
WO WO 02/103757 12/2002

OTHER PUBLICATIONS


Primary Examiner—Matthew Smith
Assistant Examiner—Magid Y. Dimyan
Attorney, Agent, or Firm—Townsend and Townsend and Crew LLP

ABSTRACT

According to the invention, a digital design method for manipulating a digital circuit netlist is disclosed. In one step, a first netlist is loaded. The first netlist is comprised of first basic cells that are comprised of first kernel cells. The first netlist is manipulated to create a second netlist. The second netlist is comprised of second basic cells that are comprised of second kernel cells. A percentage of the first and second kernel cells are selection circuits. There is less chip area consumed in the second basic cells than in the first basic cells. The second netlist is stored. In various embodiments, the percentage could be 2% or more, 5% or more, 10% or more, 20% or more, 30% or more, or 40% or more.

23 Claims, 18 Drawing Sheets
### U.S. Patent Documents

<table>
<thead>
<tr>
<th>Patent Number</th>
<th>Issue Date</th>
<th>Inventor(s)</th>
<th>Additional Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,128,871 A</td>
<td>7/1992</td>
<td>Schnitz</td>
<td></td>
</tr>
<tr>
<td>5,162,666 A</td>
<td>11/1992</td>
<td>Tran</td>
<td></td>
</tr>
<tr>
<td>5,200,907 A</td>
<td>4/1993</td>
<td>Tran</td>
<td></td>
</tr>
<tr>
<td>5,228,991 A</td>
<td>7/1993</td>
<td>Dougherty</td>
<td></td>
</tr>
<tr>
<td>5,299,204 A</td>
<td>3/1994</td>
<td>Daniels</td>
<td></td>
</tr>
<tr>
<td>5,432,707 A</td>
<td>7/1995</td>
<td>Leung</td>
<td></td>
</tr>
<tr>
<td>5,526,276 A</td>
<td>6/1996</td>
<td>Cox et al.</td>
<td></td>
</tr>
<tr>
<td>5,548,231 A</td>
<td>8/1996</td>
<td>Tran</td>
<td></td>
</tr>
<tr>
<td>5,596,742 A</td>
<td>1/1997</td>
<td>Agarwal et al.</td>
<td></td>
</tr>
<tr>
<td>5,724,251 A</td>
<td>3/1998</td>
<td>Heavlin</td>
<td></td>
</tr>
<tr>
<td>5,780,883 A</td>
<td>7/1998</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td>5,796,128 A</td>
<td>8/1998</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td>5,801,551 A</td>
<td>9/1998</td>
<td>Lin</td>
<td></td>
</tr>
<tr>
<td>5,859,547 A</td>
<td>1/1999</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td>5,867,396 A</td>
<td>* 2/1999</td>
<td>Parfure</td>
<td>.................... 716/18</td>
</tr>
<tr>
<td>5,894,227 A</td>
<td>4/1999</td>
<td>Acuff</td>
<td></td>
</tr>
<tr>
<td>5,953,519 A</td>
<td>9/1999</td>
<td>Fura</td>
<td></td>
</tr>
<tr>
<td>5,987,086 A</td>
<td>11/1999</td>
<td>Raman et al.</td>
<td></td>
</tr>
<tr>
<td>6,051,031 A</td>
<td>4/2000</td>
<td>Subbat et al.</td>
<td></td>
</tr>
<tr>
<td>6,173,435 B1</td>
<td>1/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,184,718 B1</td>
<td>2/2001</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6,185,719 B1</td>
<td>2/2001</td>
<td>Sako</td>
<td></td>
</tr>
<tr>
<td>6,205,572 B1</td>
<td>3/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,263,483 B1</td>
<td>7/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,275,973 B1</td>
<td>8/2001</td>
<td>Wein</td>
<td></td>
</tr>
<tr>
<td>6,286,128 B1</td>
<td>9/2001</td>
<td>Pileggi et al.</td>
<td></td>
</tr>
<tr>
<td>6,288,593 B1</td>
<td>9/2001</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td>6,289,491 B1</td>
<td>9/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,289,496 B1</td>
<td>9/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,292,931 B1</td>
<td>9/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,295,636 B1</td>
<td>9/2001</td>
<td>Dupenloup</td>
<td></td>
</tr>
<tr>
<td>6,313,666 B1</td>
<td>11/2001</td>
<td>Yamashita et al.</td>
<td></td>
</tr>
<tr>
<td>6,356,112 B1</td>
<td>3/2002</td>
<td>Tran et al.</td>
<td></td>
</tr>
<tr>
<td>6,359,468 B1</td>
<td>3/2002</td>
<td>Park et al.</td>
<td></td>
</tr>
<tr>
<td>6,467,074 B1</td>
<td>10/2002</td>
<td>Katsoulas et al.</td>
<td></td>
</tr>
<tr>
<td>6,526,543 B1</td>
<td>2/2003</td>
<td>Rosser</td>
<td></td>
</tr>
<tr>
<td>6,536,028 B1</td>
<td>* 3/2003</td>
<td>Katsoulas et al.</td>
<td>................ 716/17</td>
</tr>
<tr>
<td>6,564,361 B1</td>
<td>* 5/2003</td>
<td>Zolotykh et al.</td>
<td>................ 716/8</td>
</tr>
<tr>
<td>6,588,006 B1</td>
<td>7/2003</td>
<td>Watkins</td>
<td></td>
</tr>
</tbody>
</table>

* cited by examiner
Fig. 1A

Fig. 1B

Fig. 1C
BEGIN WITH SYNTACTIC EXPRESSION FOR CELL

REMOVE EXISTING REDUNDANCIES

PERFORM APPROPRIATE PERMUTATIONS

ENTER CONTROL VARIABLES INTO TRUTH TABLE

ANY CHANGES TO SYNTACTIC EXPRESSION?

GO TO NEXT TRANSFORMATION

FIG. 9B
FROM PREVIOUS TRANSFORMATION

BEGIN WITH SYNTACTIC EXPRESSIONS FOR MULTIPLE CELLS

NO

ANY EXPRESSION OUTPUT A CONTROL VARIABLE?

YES

CONVERT CORRESPONDING CONTROL VARIABLES INTO MAP-ENTERED VARIABLES

COMBINE CONTROL

DEFINE STATES IN MERGED EXPRESSION

GO TO NEXT TRANSFORMATION

FIG. 9C
BEGIN WITH SYNTACTIC EXPRESSION FOR CELL

PERFORM PERMUTATIONS TO ALIGN LOW STATES

ARE HALF OF TRUTH-TABLE STATES LOW?

CHANGE CONTROL VARIABLE TO RESET INPUT TO A FLIP FLOP

PERFORM PERMUTATIONS TO ALIGN HIGH STATES

ARE HALF OF TRUTH-TABLE STATES HIGH?

CHANGE CONTROL VARIABLE TO SET INPUT TO A FLIP FLOP

GO TO NEXT TRANSFORMATION

FIG. 9D
TRANSLATE NETLIST TO ULG NETLIST

IDENTIFY DATA AND CONTROL

REMOVE CONNECTION CELLS TO VDD/VSS

REMOVE NON-INVERTING BUFFERS

REMOVE INVERTERS

REMOVE REDUNDANT NODES

REMOVE INVERSIONS

FANOUT NODES?

NO

YES

PERMUTE NODES AND REDUCE

SYNCHRONOUS CELL WITH HALF VSS?

NO

YES

MORE VARIABLE TO RESET INPUT

MORE VARIABLE TO SET INPUT

SYNCHRONOUS CELLS WITH HALF VDD?

GROUP BY COMMON INPUTS

IDENTIFY SUB-FUNCTIONS

MAKE ALL BUFFERS INVERTING

TRANSLATE INTO STANDARD CELLS

FIG. 10
Determine Layout Rules for Target Fabrication Process

Layout Kernel Cells with Manual Customization

Specify Basic Cell Configurations

Automatically Generate Basic Cells

Perform Verification and Any Hand Optimization of Basic Cells

Layout Any Specialized Cells

Fig. 12
Edit Design with HDL Entry Tool

Perform Dynamic Timing Analysis

Synthesize & Optimize HDL Behavioral Netlist to Produce Structural Netlist

Perform Static Timing Analysis

Perform Dynamic Timing Analysis

Place & Route Design

Perform Static Timing Analysis

Perform Dynamic Timing Analysis

Fabricate Design

Any Problems?

Yes = Rework

No = Continue

Fig. 19
DIGITAL LOGIC OPTIMIZATION USING SELECTION OPERATORS

CROSS-REFERENCES TO RELATED APPLICATIONS

This application is a nonprovisional of and claims priority to U.S. Prov. Pat. Appl. No. 60/298,832, entitled “MULTIPLEXOR-BASED DIGITAL DESIGN,” filed Jun. 15, 2001 by Sterling R. Whitaker et al., the entire disclosure of which is herein incorporated by reference for all purposes.


STATEMENT AS TO RIGHTS TO INVENTIONS MADE UNDER FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

The U.S. Government has a paid-up license in this invention and the right in limited circumstances to require the patent owner to license others on reasonable terms as provided for by the terms of Grant No. NAGS-9152 awarded by NASA.

BACKGROUND

This invention relates in general to digital circuits and, more specifically, to design of digital circuits that are laid out with cells.

Mathematics is one attempt for humankind to understand the universe around them. As technological advancement occurs, mathematical concepts and algorithms grow to enable and/or support those advancements. Within the context of digital design, Boolean logic is the mathematical construct used to manipulate and optimize digital circuits. Nearly every electronic device today relies upon some type of Boolean logic for any embedded digital circuits. Other mathematical constructs, however, are possible that allow further optimization of digital designs. Changes to the processing of digital design are necessary when avoiding Boolean logic elements.

Today application specific integrated circuit (ASIC) are specified using netlists of library cells for a particular process of a foundry or fabrication facility. These netlists are used to fabricate integrated circuits made up of the library cells. A few hundred library cells are typically available for a particular process that include AND gates, OR gates, flip-flops (F/F), and buffers. When a new fabrication process is developed, engineers custom layout each of the library cells to get the most optimal performance from each cell.
FIG. 17 is a block diagram of still another embodiment of a design flow that uses a verification tool throughout the design flow;
FIG. 18 is a block diagram of still another embodiment of a design flow that uses a verification tool throughout the design flow and after fabrication; and
FIG. 19 is a flow diagram of an embodiment of a design process.

In the appended figures, similar components and/or features may have the same reference label. Further, various components of the same type may be distinguished by following the reference label by a dash and a second label that distinguishes among the similar components. If only the first reference label is used in the specification, the description is applicable to any one of the similar components having the same first reference label irrespective of the second reference label.

DETAILED DESCRIPTION

The ensuing description provides preferred exemplary embodiment(s) only, and is not intended to limit the scope, applicability or configuration of the invention. Rather, the ensuing description of the preferred exemplary embodiment(s) will provide those skilled in the art with an enabling description for implementing a preferred exemplary embodiment of the invention. It is to be understood that various changes may be made in the function and arrangement of elements without departing from the spirit and scope of the invention as set forth in the appended claims.

In one embodiment, the present invention provides a digital design method for manipulating a digital circuit netlist. In one step, a first netlist is loaded. The first netlist is comprised of first basic cells that are comprised of first kernel cells. The first netlist is manipulated to create a second netlist. The second netlist is comprised of second basic cells that are comprised of second kernel cells. A percentage of the first and second kernel cells are selection circuits. There is less chip area consumed in the second basic cells than in the first basic cells. The second netlist is converted to a third netlist by, at least, substituting some of the second basic cells with Boolean cells. The third netlist also uses the language format. The third netlist is stored.

I. Cells

In one embodiment, a basic cell is a construct that includes one or more of a universal logic gate (ULG), a memory element or flip-flop (F/F), and/or a buffer. In this embodiment, the ULG is a multiplexor or select circuit. The ULG, memory element, and buffer are implemented with relatively-small number of kernel cells, which typically have layouts that are individually optimized, and often, by hand. The memory cells are arranged into the higher-level basic cells having at least one of the ULG, memory element, and buffer, but the basic cells do not have more than one of any type of kernel cell in this embodiment. For a given semiconductor process, there is a ULG ASIC cell library which is composed of the basic cells and specialized cells. These specialized cells may differ from the basic cell construct and could include, for example, clock dividers, memory arrays, analog circuits, phase-locked loops, oscillators, analog circuits, etc.

Referring first to FIG. 1A, an embodiment of a basic cell is shown in block diagram form. This embodiment includes all three of a ULG or selection circuit, a memory cell, and a buffer. Some of the kernel cell components of the basic cell are shown in a generalized manner. The ULG is shown having any number of data and selection control inputs, however the relationship between the maximum data inputs for a number of selection control inputs follows the following relationship 2^n. The memory kernel cell is shown having a select data input. A buffer kernel cell 112 shown has both an inverting and non-inverting output, although, other buffer implementations will have either an inverting or non-inverting output.

The ULG in this embodiment is implemented with a multiplexor. Multiplexors can be used to implement any Boolean function, but are not Boolean operators. Combinatorial logic in conventional designs is not implemented with selection functions, but uses Boolean logic gates. Further, multiplexors in conventional circuits are converted to Boolean equivalents during synthesis.

The below Table I shows the fourteen kernel cells used in one embodiment:

<table>
<thead>
<tr>
<th>ULG</th>
<th>Symbol Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>8 to 1 (U8), 4 to 1 (U4) or 2 to 1 (U2) Multiplexors</td>
</tr>
<tr>
<td>D</td>
<td>D FF (D1)</td>
</tr>
<tr>
<td>DR</td>
<td>Latchable D FF/Synchronous (DR1), Clock Edge Synchronization (DR2) or Asynchronous (DR3)</td>
</tr>
<tr>
<td>DS</td>
<td>Selectable D FF/Synchronous (DS1), Clock Edge Synchronization (DS2) or Asynchronous (DS3)</td>
</tr>
<tr>
<td>B</td>
<td>Non-inverting buffer (B1) or Hybrid Inverting and Non-inverting buffer (B2)</td>
</tr>
<tr>
<td>BN</td>
<td>Inverting buffer (BN1)</td>
</tr>
<tr>
<td>CB</td>
<td>High-drive buffer (CB1)</td>
</tr>
<tr>
<td>ZB</td>
<td>Tristatable buffer (ZB1)</td>
</tr>
</tbody>
</table>

The embodiment of the kernel cells in the above Table I could be augmented in other embodiments to include other cells. The ULGs could include multiplexors of any size, for example, 16 to 1, 32 to 1, 64 to 1, etc. Larger multiplexors could be formed with a number of smaller multiplexors if a larger multiplexor is not supported in the kernel cells.
Various other types of memory cells could also be supported such as EEPROM, EPROM, PROM, DRAM, SRAM, NVRAM, magnetic core memory, J-K F/Fs, setable and resetable F/Fs, various F/F with scan ATPG capability, etc. The J-K, setable, or resetable functionality of a F/F can be implemented by a D F/F and logic that can be embedded in the mux before or after the D F/F. The F/Fs could also be falling edge triggered in some embodiments. Also the buffers could be of various strengths and sizes. Some buffers could support input and output pins of the chip with various thresholds, voltages, etc.

Table II lists the various configurations in which kernel cells are used to create basic cells 100-1 that use all of a ULG 104, a memory cell 108 and a buffer cell 112 such as the example in FIG. 1A. These basic cells 100-2 are the variations found in one embodiment of the ULG ASIC cell library.

With reference to FIG. 1B, a block diagram of another embodiment of a basic cell 100-3 composed of ULG and memory kernel cells 104, 112 is shown. There are other possible configurations of this type of basic cell 100-4. The variations of this basic cell 100-4 for one embodiment of theASIC library are listed in Table V. From Tables II-V, around 80% of the 142 available basic cells include ULG circuits.

### TABLE II

<table>
<thead>
<tr>
<th>Basic Cell Type</th>
<th>Configuration</th>
<th>Various Basic Cell Layout Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDB U → D → B</td>
<td>U2D1B1, U4D1B1, U8D1B1, U2D1B2, U4D1B2, U8D1B2</td>
<td></td>
</tr>
<tr>
<td>UDBN U → D → BN</td>
<td>U2D1BN1, U4D1BN1, U8D1BN1</td>
<td></td>
</tr>
<tr>
<td>UDZB U → D → ZB</td>
<td>U2D1ZB1, U4D1ZB1, U8D1ZB1</td>
<td></td>
</tr>
<tr>
<td>UDRB U → DR → B</td>
<td>U2DR1B1, U4DR1B1, U8DR1B1, U2DR1B2, U4DR1B2, U8DR1B2</td>
<td></td>
</tr>
<tr>
<td>UDRBN U → DR → BN</td>
<td>U2DR1BN1, U4DR1BN1, U8DR1BN1, U2DR1BN2, U4DR1BN2, U8DR1BN2</td>
<td></td>
</tr>
<tr>
<td>UDRZB U → DR → ZB</td>
<td>U2DR1ZB1, U4DR1ZB1, U8DR1ZB1, U2DR1ZB2, U4DR1ZB2, U8DR1ZB2</td>
<td></td>
</tr>
<tr>
<td>UDSB U → DS → B</td>
<td>U2DS1B1, U4DS1B1, U8DS1B1, U2DS1B2, U4DS1B2, U8DS1B2</td>
<td></td>
</tr>
<tr>
<td>UDBN U → DS → BN</td>
<td>U2DS1BN1, U4DS1BN1, U8DS1BN1, U2DS1BN2, U4DS1BN2, U8DS1BN2</td>
<td></td>
</tr>
<tr>
<td>UDSZB U → DS → ZB</td>
<td>U2DS1ZB1, U4DS1ZB1, U8DS1ZB1, U2DS1ZB2, U4DS1ZB2, U8DS1ZB2</td>
<td></td>
</tr>
</tbody>
</table>

With reference to FIG. 1C, a block diagram of yet another embodiment of a basic cell 100-3 is shown that is composed of ULG and memory kernel cells 104, 108. This is just one example of the various similar basic cells 100-3 that might form an embodiment of the ULG ASIC cell library. Other possible configurations are enumerated in Table IV below.

### TABLE IV

<table>
<thead>
<tr>
<th>Basic Cell Type</th>
<th>Configuration</th>
<th>Various Basic Cell Layout Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>UD U → D</td>
<td>U2D1U, U4D1U, U8D1U</td>
<td></td>
</tr>
<tr>
<td>UDR U → DR</td>
<td>U2DR1U, U4DR1U, U8DR1U, U2DR1U2, U4DR1U2, U8DR1U2</td>
<td></td>
</tr>
<tr>
<td>UDS U → DS</td>
<td>U2DS1U, U4DS1U, U8DS1U, U2DS1U2, U4DS1U2, U8DS1U2</td>
<td></td>
</tr>
</tbody>
</table>

With reference to FIG. 1D, a block diagram of still another embodiment of a basic cell 100-4 composed of ULG and buffer kernel cells 104, 112 is shown. There are other possible configurations of this type of basic cell 100-4. The variations of this basic cell 100-4 for one embodiment of the ASIC library are listed in Table V. From Tables II-V, around 80% of the 142 available basic cells include ULG circuits. The 142 basic cells are based upon the 14 kernel cells of Table I.

Although the embodiment in Tables II-V show some possible basic cells, other embodiments could include additional basic cells. These additional basic cells could be optimized for output power, power consumption, layout area, response time, leakage, etc. such that there are multiple cells with the same logical properties, but that are optimized for particular circumstances. For example, there may be three non-inverting buffers of having different drives to support larger fanout and/or higher speeds. In various embodiments, there could be less than, for example, 100, 75, 50, 40, 30, 20, or 10 kernel cells. At the lower limit, there is three kernel cells in one embodiment with just one of each of the types of kernel cells.

### TABLE V

<table>
<thead>
<tr>
<th>Basic Cell Type</th>
<th>Configuration</th>
<th>Various Basic Cell Layout Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>UB U → B</td>
<td>U2B1U, U4B1U, U8B1U</td>
<td></td>
</tr>
<tr>
<td>UBN U → BN</td>
<td>U2BN1, U4BN1, U8BN1</td>
<td></td>
</tr>
</tbody>
</table>

The building blocks of a digital circuit could be abstracted beyond the ULG ASIC cell library. In some embodiments, the ULG ASIC cell library components could be combined in higher-level macro cells such as adders, multipliers, registers, barrel shifters, ALUs, comparators, decoders, state machines, counters, etc. There could be thousands of possible macro cells. Further, designs can be abstracted to a level higher than the macro cells by using cores that implement higher level functions such as microprocessors, graphics processors, interface busses or ports, digital signal processing processors, digital signal processors, etc.
ASIC cell library is typically manually laid out to optimize process. Each of the hundreds of cells in the conventional kernel cells used in this embodiment. Table VI shows a 2B.

In this embodiment, however, a small circuit.

Truth table for the enable buffer to represent the layout of the circuit to implement the 2 to 1 a metal formed over an oxide such as SiO,. For such an

II. Layout of Cells

Each fabrication process at a fab or foundry generally has a conventional ASIC cell library that is customized for that process. Each of the hundreds of cells in the conventional ASIC cell library is typically manually laid out to optimize its configuration. In this embodiment, however, a small number of customized kernel cells are used to automatically or manually compile the basic cells 100. For a target fabrication process, care is taken to optimize the layout of the kernel cells 104, 108, 112 for factors such as power consumption, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, glitches, 110 placement, etc. In some cases, the fabrication processes are similar enough to other fabrication processes that only minor tweaking to kernel cells is done.

With reference to FIG. 3, a block diagram of an embodiment of a ULG layout 300 is shown. A cloud graphic is used to represent the layout of the circuit to implement the 2 to 1 ULG 104. Input ports 316, 320 and I/O 324 are detailed within the circuit cloud, more specifically, ports for the inputs \((I_0, I_1)\) 316 and the select \((Y_0)\) 320 and a trace is shown for the Q output 324. For the ULG kernel cell, the input and select signals are ports 316, 320 within the ULG layout 300. The ULG circuit 304 is attached by traces to latitudinal power traces 308, 312.

Abutment is used to link certain signals by coupling adjacent kernel cells. Adjacent placement of the cells may join the abutted I/O or a small conductive trace may join the abutted I/O. The kernel cells have a uniform height and differing depths such that the power traces 308, 312 for each kernel cell align with the next kernel cell. Also, certain I/O signals use a uniform latitude. For example the Q output 324 of the ULG layout 304 would align latitudinally with an input for an adjacent memory or buffer kernel cell.

Referring next to FIG. 4, a block diagram of an embodiment of a memory kernel cell layout 400 is shown. This memory circuit 404 implements a D F/F with a D input 412, a clock input 408 and a Q output 416. Coupled to the memory circuit 404 are a VDD and VSS power busses 308, 312. The height of the memory kernel cell layout 400 is the same as the ULG cell layout 300 such that the power busses for both kernel cells align latitudinally.

With reference to FIG. 5, a block diagram of an embodiment of a buffer kernel cell layout 500 is shown. As with the other kernel cell circuits 304, 404, a non-inverting buffer kernel cell circuit 504 is coupled to power busses 308, 312 with a height uniform to the other kernel circuits 304, 404. The buffer circuit includes a D input 508 and a Q output 512, where the D input 508 is latitudinally aligned with the outputs from either a ULG circuit 304 or a memory circuit 404. In this embodiment, the Q output 512 is offset from the latitude of the D input 508.

Referring next to FIG. 6, a block diagram of an embodiment of a basic cell 600 abutted together from three kernel cells 300, 400, 500. A U2 ULG, D1 F/F and B1 non-inverting buffer kernel cells 300, 400, 500 are connected in serial to form the basic cell 600. The power busses 308, 312 are for each kernel cell 300, 400, 500 align to form a larger whole. The Q output 324 from the U2 ULG circuit 304 aligns with the D input 412 to the D1 memory circuit 404, and the Q output 416 from the D1 memory circuit 404 aligns with the D input 508 to the B1 buffer circuit 504. Other embodiments could have additional power busses, for example, a substrate bus connection.

With reference to FIG. 7, a block diagram of an embodiment of two basic cells 600, 704 laid out together in a row 700 is shown. During layout of a chip, all the ULG ASIC cells are arranged. The basic cells 600, 704 are aligned in horizontal rows. In some cases (not depicted), there is routing of one or more signals between the basic cells 600, 704. In the depicted embodiment, an output from a first basic cell 704 is coupled with a trace 712 to an input of a second basic cell 600. The clock inputs for both basic cells 600, 704 are latitudinally aligned such that a clock bus can pass strait across a row 700 of basic cells.

In some embodiments, additional size reductions are realized in the ULGs by having them comprise one or more depletion-mode transistors. Schematic diagrams are provided in FIGS. 8A and 8B that compare enhancement-mode and depletion-mode transistors. FIG. 8A shows the structure of an n-type enhancement-mode transistor 802 that comprises a source 804, a drain 806, and a gate 810. Connections are made with the source 804 and drain 806 respectively through pads 812 and 814. The gate 810 usually comprises a metal formed over an oxide such as SiO2. For such an
n-type transistor 802, both the source 804 and drain 806 comprise n-doped regions in a p-doped substrate. The transistor operates so that when at least a threshold voltage is applied to the gate 810, current flows between the source 804 and drain 806 through an intermediate channel region. In circuits, the enhancement-mode transistor 802 is denoted with symbol 820.

The depletion-mode transistor 842 illustrated in FIG. 8B also comprises a source 844, a drain 846, and a gate 850 formed over an oxide 848, with connections to the source 844 and drain 846 provided respectively by pads 852 and 854. For the depletion-mode transistor, however, the channel region 856 between the source 844 and drain 846 is also n-doped, allowing the flow of current even without a gate voltage. The current can be stopped by applying at least a negative cutoff voltage to the gate 850. In circuits, the depletion-mode transistor 842 is denoted with symbol 860.

FIG. 8C provides an example of a circuit for a ULG element that exploits the different properties of enhancement- and depletion-mode transistors to allow a smaller circuit area than a ULG design that uses only enhancement-mode transistors. The illustration is provided for the U8 cell, which acts as an 8:1 multiplexer. The U8 cell 872 comprises eight inputs 874 labeled I,1, . . . , 8; three selection controls 876 labeled Y[0] , . . . , 2], and one output 878 labeled Q. The multiplexing functions of the cell are governed by the action of 48 transistors, of which half are provided as depletion-mode transistors. Each of the controls 876 and its inversion is provided to a transistor along the path of one of the inputs 874, with the inversions being effected by inverters 880. To effect the multiplexing functions, the depletion-mode transistors are distributed according to the level of the control. For the least significant control, the depletion-mode transistors are positioned alternately for the next significant control, they are positioned alternately in pairs; for the next significant control, they are positioned alternately in quads; etc. Thus, for the U8 cell 872, depletion-mode transistors for Y[0] are provided for inputs I,1, I,2, I,3, and I,4; and for the inverted control YN[0], depletion-mode transistors are provided for inputs I,5, I,6, and I,7. For Y[1], depletion-mode transistors are provided for inputs I,8, I,9, I,10, and I,11, and for the inverted control YN[1], they are provided for inputs I,12, I,13, I,14, and I,15. Similarly, depletion-mode transistors are provided for Y[2] for inputs I,16, I,17, I,18, and I,19, and for YN[2] for inputs I,20, I,21, I,22, and I,23.

For certain embodiments of the ULGs that comprise depletion-mode transistors, this pattern may be used for a cell of any size. For a U2s+1 ULG that has 2s+1 inputs and n+1 controls, depletion-mode transistors may be provided for each control/input combination as follows:

(i) For control Y[0 . . . n], depletion-mode transistors are provided for inputs I,y where k<2 mod 2s+1;

(ii) For control YN[0 . . . n], depletion-mode transistors are provided for inputs I,y where k≥2 mod 2s+1.

In other embodiments, a different distribution of depletion-mode transistors may be used to implement the multiplexing functions of the ULG.

III. Logical Structures

In addition to the structural characteristics described above, there are a number of logical properties and features that may be used both to characterize individual cells and to characterize libraries of such cells. An example of a formalism that may be used in one embodiment to describe the functionality of the basic cells and from which at least some such logical characterizations may be extracted is now described.

As discussed above, the selection circuits embodied by ULGs used for forming basic cells may be implemented using 2:1, 4:1, 8:1, or perhaps even larger, multiplexors. The inputs and control of each multiplexor are programmed to achieve the desired logical characteristics of the cell. The following syntax has been developed to describe the programming of such inputs and selection controls in a general fashion:

Q < QN > . . . < E > Y[n - 1:0] I,0,1,2, . . . , I,n <

R/S > < CLK > (Parameter = Option);

In addition to describing the programming of the ULG kernel cells, the syntax may be used to describe any of the basic cells, including both those that comprise ULG kernel cells and those that do not comprise ULG kernel cells. Optional parameters in the syntax are denoted with angular brackets and the components of the syntax are summarized in Table I.

<table>
<thead>
<tr>
<th>Component</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>Non-inverting output</td>
</tr>
<tr>
<td>&lt;QN&gt;</td>
<td>Optional inverting output</td>
</tr>
<tr>
<td>. . .</td>
<td>Cell name</td>
</tr>
<tr>
<td>&lt;E&gt;</td>
<td>Optional tri-state enable</td>
</tr>
<tr>
<td>Y[n - 1:0]</td>
<td>Control variables</td>
</tr>
<tr>
<td>. . .</td>
<td>State variables</td>
</tr>
<tr>
<td>I,y</td>
<td>Truth-table state</td>
</tr>
<tr>
<td>. . .</td>
<td>Next state</td>
</tr>
<tr>
<td>&lt;R/S&gt;</td>
<td>Optional reset or set input</td>
</tr>
<tr>
<td>&lt;CLK&gt;</td>
<td>Optional clock input</td>
</tr>
<tr>
<td>(Parameter = Option)</td>
<td>Selects an option</td>
</tr>
<tr>
<td>; Terminator</td>
<td></td>
</tr>
</tbody>
</table>

There are a number of features of the syntax worthy of note. The first component of the syntax indicates the output of the cell, using either Q or QN respectively to denote the output Q or Q. This is followed by the name of the cell, which is generally constructed by concatenating the names of the kernel cells comprised by the cell. The names of the kernel cells have been set forth above in Table VIII. Thus, for example, a basic cell that comprises a D F/F and a non-inverting buffer would be named .DB (D→B); a basic cell that comprises a setable D F/F and an inverting buffer would be named .DRBN (DR→BN); a basic cell that comprises a ULG multiplexor and a setable D F/F would be named .UDS (U→DS); and a basic cell that comprises a ULG multiplexor, a D F/F, and a tristatable buffer would be named .UDZBS (U→D→ZB). It is noted that some of these examples of basic cells include a ULG multiplexor kernel cell while others do not, but all of these may be described with the syntax.

In those instances where the syntax is used to describe a basic cell comprising a ULG, the number of selection control inputs provided to the ULG is n. In a specific embodiment, the control inputs are ordered by significance, with the most significant control on the left and the least significant control on the right. While the syntax is equally robust for describing basic cells for any value of n, for purposes of explanation the examples provided herein generally correspond to cases in which n≤3. The states of the n control inputs Y dictate which of 2n inputs are passed to the output of the ULG. While in some instances, the control inputs may be identified individually, in other instances a range of control inputs is identified by using a colon in the argument of Y. Specifically, "Y[a;b]" is intended to refer to the full expression "Y[a] Y[a-1] Y[a-2] . . . Y[b+2] Y[b+1] Y[b]". The set of parameters I,0, I,1, I,2, . . . , I,n represents the
logical function to be applied by the cell, and as discussed in greater detail below may comprise a truth table for implementing a combinational logic device or may comprise an identification of the next state of a sequential circuit. In some embodiments, these logical states \( I_{12}^{\text{...}} I_{2} \ldots I_{1} \) may be assigned to logical 1's or 0's (sometimes referred to herein as "base Boolean values"), but may more generally include map-entered variables as well. For the basic cell, this corresponds to a connection to VDD for a logic 1, to a connection to VSS for a logic 0, and to a connection to a signal for a mapped entered variable. Parameters such as the type of reset, i.e. asynchronous, synchronous, or clock-edge, are assigned and enclosed in parentheses at the end of the statement.

This formalism permits the expression of a number of manipulations that are possible with embodiments of the invention and which are discussed in detail in order to enable one of skill in the art to perform such manipulations. The nature of such manipulations may be clarified with a simple example for the combinational logic function \( C = A + B \). This logic function may be expressed in a concise hardware description language ("CHDL") formalism as follows:

\[
C \_UB \ A \ B \ VDD \ VDD \ VSS \ VDD 
\]

That this is a correct implementation of the logic function in which \( C \) is equal to "A or not B" is evident by comparing the entries in the expression to the syntax discussed above. The name of the cell \_UB indicates that the function is implemented with a cell that comprises a universal logic gate U and a non-inverting buffer B. On either side of the name, the parameters involved in the function are denoted, with the left-most component of the expression \( C \) indicating the output, and the variables to the right of the name A and B indicating the inputs. The following four entries before the terminator define the following truth table \( I_{2}^{\text{...}} I_{2} \ldots I_{1} \) for the combinational function, with VDD being equivalent to a logic 1 and VSS being equivalent to a logic 0. The individual truth-table states \( I \) are noted:

**TABLE IX**

<table>
<thead>
<tr>
<th>( I_i )</th>
<th>( I_2 )</th>
<th>( I_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_i )</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( I_i )</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( I_i )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( I_i )</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When logical operations are performed on expressions in this formalism, they indicate directly how the resulting expression may be implemented with basic blocks in accordance with an embodiment of the invention. For example, a simple logical operation is inversion of the output, which may be implemented by using an inverting buffer:

\[
C \_UB \ A \ B \ VDD \ VDD \ VSS \ VDD 
\]

As can be seen, the same truth table as that defined in Table IX is used for implementing \( C \), but the implementation is with a basic cell comprising a universal logic gate U and an inverting buffer BN. An alternative implementation of \( C \) uses the same \_UB basic cell, but instead uses a different truth table by inverting all of the input states:

\[
C \_UB \ A \ VDD \ BN 
\]

In other instances, alternative implementations of the same logical function may be achieved by performing operations on the control inputs. For example, the control inputs A and B may be permuted. Permuting the control for the function acts to rearrange the truth table. In an embodiment that includes this example, the truth-table states \( I_i \) and \( I_j \) remain in the same position because they represent states where both controls are high or both are low. States \( I_2 \) and \( I_1 \), which represent states where one control is high and the other is low, are interchanged:

\[
C \_UB \ B \ A \ VDD \ VDD \ VDD \ VDD \ VDD \ VDD 
\]

A permutation of the truth table may also result from inversion of one or more of the control inputs. In this example, inverting the least significant control B interchanges neighboring states in the truth table:

\[
C \_UB \ A \ BN \ VDD \ VDD \ VDD \ VDD \ VDD \ VSS 
\]

This alternative expression for may be viewed as defining an implementation for \( C \) that uses the general truth table for \( X+Y \), but with control inputs defined so that \( X=A \) and \( Y=B \). If the next significant control A is inverted in the original expression instead, neighboring pairs of states in the truth table are interchanged:

\[
C \_UB \ AN \ B \ VDD \ VDD \ VDD \ VDD \ VDD \ VDD 
\]

It is evident that this expression implements the general truth table for \( X+Y \), but with control inputs defined so that \( X=A \) and \( Y=B \). If both control inputs are inverted,

\[
C \_UB \ AN \ BN \ VDD \ VDD \ VDD \ VDD \ VDD \ VDD 
\]

The truth table in this expression implements the general function \( X+Y \), but with control inputs defined so that \( X=A \) and \( Y=B \).

The CHDL syntax also permits control variables to be entered as elements in the truth-table states. For example, the syntax makes it easy to recognize that the result \( C \) is high whenever \( A \) is high and that \( C \) takes the value of \( BN \) when \( A \) is low. This may be expressed in this CHDL syntax as:

\[
C \_UB \ A \ VDD \ BN 
\]

and corresponding to the truth table shown in Table X:
but this same ability is manifested in the implementations when B is low. This may be expressed in this CHDL syntax as

\[ \text{When B is low, C is high whenever B is low and that C takes the value of A when B is low.} \]

This syntax makes it easy to recognize that the result C is high whenever B is low and that C takes the value of A when B is low. This may be expressed in this CHDL syntax as

\[ C \leftarrow \begin{cases} 1 & \text{if } B = 0 \\ 0 & \text{otherwise} \end{cases} \]

and corresponding to the truth table shown in Table XI:

**TABLE XI**

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>VDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
values among its inputs and/or output(s). In still other embodiments of the invention, even greater orders of Boolean functions are used, such orders admitting all lower orders of Boolean functions among their inputs and/or output(s) in addition to admitting the base Boolean values used in conventional design. All orders of Boolean functions other than zero-order Boolean functions are sometimes referred to herein collectively as "higher-order" Boolean functions.

This generalization may be illustrated with an example based on the $C=A+B$ example discussed earlier:

<table>
<thead>
<tr>
<th>J</th>
<th>UB</th>
<th>G H</th>
<th>VDD VSS F C</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>UB</td>
<td>D E</td>
<td>VSS C VDD C</td>
</tr>
<tr>
<td>C</td>
<td>UB</td>
<td>A B</td>
<td>VDD VDD VSS VDD</td>
</tr>
</tbody>
</table>

In this example, the third expression corresponds to the zero-order function $C=A+B$, which admits only base Boolean values 0 and 1 among its arguments. Such a function uses a combinational mapping and may be implemented using gates, such as with an OR gate and a NOT gate. The second expression corresponds to a first-order function, or functional, that admits the zero-order function $C$ as one of its arguments, in addition to admitting the base Boolean values. The first expression corresponds to a second-order function that admits the first-order function $F$, the zero-order function $C$, and the base Boolean values among its arguments. The first and second expressions thus each correspond to expressions for higher-order functions. All three of the expressions may be implemented in embodiments of the invention using the cells as described above.

One effect of the ability to use higher-order functions may be seen with a comparison to the exclusive use of Boolean operations. Such Boolean operators operate only on the base Boolean values 0 or 1, i.e. that have already had a value of 0 or 1 bound to it. Boolean minimization or optimization techniques are based on decomposing the expressions being minimized to consider the meaningful possible combinations of assignment of 0 or 1 to each variable (with the possible existence of "don't care" states for some variables under some circumstances reducing the meaningful possible combinations downward from the set of all possible combinations). Higher-order functions allow one to optimize, or minimize a circuit, without the requirement to decompose the function result to each possible value and considering each separately. In other words, when a circuit with the algorithms described below, one need not know what the value of the functions or variables are; optimization is performed regardless. In conventional methods limited to the use of Boolean operators, each variable and function is decomposed into all possible values for the functions and variables, i.e. to define a complete truth table, before any optimization can be performed; in such conventional methods one must exhaustively assign a value to all variables and functions.

IV. Optimization

The expanded availability of logical operations provided by embodiments that use cells based on the ULGs and as represented by the formalism described permits increased optimization. In many instances, these logical operations may be used to determine optimized methods of implementing a given function. A number of such logical operations are illustrated, and it will be understood by those of skill in the art that similar logical operations may derive from the formalism in other embodiments of the invention. Moreover, while the formalism is used as a matter of convenience to illustrate the nature of the optimizations, it will be understood that all the expressions that follow may be implemented using the previously described cells in the manner explained. This is true even in instances where the expressions correspond to functions not accessible by standard Boolean logic. In some cases, use of the formalism shows how multiple manipulations may be performed to achieve an optimization, it being necessary only to implement the final result with the previously described cells to achieve the optimized function.

A number of the operations that may be performed with logical functions as expressed using the formalism described herein are summarized in FIGS. 9A-9D, which provide flow diagrams to explain how such operations may be performed. In different embodiments, various combinations of one or more such operations may be performed and the invention is not limited to any particular order or number of such operations. Accordingly, each of FIGS. 9A-9D indicates that it may be entered as part of a greater flow of operations from a previous transformation. It is not necessary that a previous transformation necessarily have been performed in any case, although the indication is included to note that some embodiments of the invention contemplate that previous transformations may have been performed. Also, while the flow diagrams in each of FIGS. 9A-9D shows an exemplary order in which operations may be performed, such an ordering is not necessary and alternative embodiments permit alternative orderings. Moreover, in some embodiments, it is possible that some operations may be performed simultaneously, such as when different parts of a large structure are optimized at the same time.

FIG. 9A summarizes a number of operations that may collectively be considered to correspond to the removal of inversions. Accordingly, the method shown in FIG. 9A begins at block 902 with a syntactic expression for a cell, perhaps, but not necessarily, after certain previous logical transformations have been effected. At block 904, an identification is made whether there are any inversions in the syntactic expression for removal. If not, the method proceeds to a potentially subsequent transformation at block 906. In the event that it is desirable to remove an inversion, the method may proceed along one of at least three branches depending on the type of inversion. Branch 903 corresponds to inversions in the control or state variables $Y$; branch 905 corresponds to inversions in the truth table; and branch 907 corresponds to inversions in the buffer.

In one embodiment, inversions of the control or state variable may proceed at block 908 by interchanging adjacent groups in the truth table. The size of the groups to be interchanged depends on the significance of the control or state variable to be inverted. Thus, if a control $Y[k]$ is to be inverted, groups of size $2^k$ are inverted. This may be illustrated by considering a cell comprising a ULG and a buffer:

$$Q \quad UB \quad Y[2] \quad Y[1] \quad YN[5] \quad ABCDEFGH$$

In this instance, the least significant control $Y[0]$, defined by $k=0$, is to be inverted so that adjacent states are interchanged:
performing the control inversions and respective truth-table accomplished by entering the control as a map-entered variable appropriate level.

When \( k = 2 \) for the control to be inverted, adjacent quads of states are to be interchanged:

The flow diagram in Fig. 9A provides a loop back to block 904 after a particular control has been inverted by interchanging states. This contemplates the possibility of performing inversions on multiple controls, which are therefore effected by performing the relevant interchanges in succession. The interchanges are commutative so that the resulting syntactic expression is independent of the order in which they are performed:

The resulting expression, which may be implemented using the cells as described above, follows from any order of performing the control inversions and respective truth-table interchanges.

The flow diagram of Fig. 9A notes at block 910 that in some instances the removal of inversions may be accomplished by entering the control as a map-entered variable instead of performing interchanges. In such cases, the ability of embodiments of the invention to accommodate non-Boolean selection operations is exploited to achieve greater levels of optimization. Within the syntax used to illustrate the principles described herein, the identification of an inverted control \( \text{YN} \) with a sequence \( \text{VSS VDD} \) permits removal of the inversion by entering the control into the truth table:

As indicated, entry of the control in the truth table will usually also require a repetition of a state \( A \) at the appropriate level.

In some cases, it may be desirable to permute the truth table to achieve such \( \text{VSS VDD} \) sequences by permuting the control:

In the above progression, optimization of the cell is achieved by noting that the sequence \( \text{VSS VDD} \) may be achieved through a permutation of the truth table and by noting the repetition of state \( A \). Permuting the controls results in a truth table identical to that of the preceding example, and therefore the least significant control may become a map-entered variable.

The same principles apply with more significant levels of control, for which optimization may remove an inversion by entering the more significant control as a map-entered variable under some circumstances. In one embodiment, for example, this is achieved when the less significant controls are redundant:

The optimization has been achieved by recognizing the existence of an analogous pattern, namely the repetition of \( A \) at a higher significance level and the existence of the sequence \( \text{VSS VSS VDD VDD} \). It will now be evident to those of skill in the art that permutation of control may be used to restructure the truth table to identify such sequences and thereby optimize the function by removing the inversion. In addition, it will also be evident that these principles may be applied to any significance level for the control. For example, an eightfold repetition of \( A \) coupled with the sequence \( \text{VSS VSS VSS VSS VDD VDD VDD VDD} \) will permit removal of a \( \text{YN} [2] \) inversion, perhaps after permuting the controls to achieve such a sequence in the truth table.

At block 912 of Fig. 9A, it is noted that inversion of the truth-table states may be achieved by inverting the buffer in the cell:

By inverting the buffer, all entries in the truth table are inverted. Block 914 notes the converse function in which the buffer is inverted by inverting all elements of the truth table. While the functional effect of blocks 912 and 914 is identical, they are conceptually converse because in one instance the goal of inverting the truth table is achieved by inverting the buffer and in the other instance the goal of inverting the buffer is achieved by inverting the truth table.

Every operation in the Fig. 9A cycles back to block 904.

As noted with respect to block 908, this aspect emphasizes that multiple of these transformations may be used in effecting optimizations and that they may be performed in different orders. For example, for some cells, optimization might be achieved by: (1) first, inverting a buffer to invert a truth table; (2) second, permuting the control so that the resulting truth table includes sequences that permit the entry of control variables; and (3) finally, performing interchanges within the truth table to remove other control inversions.

The entry of a control variable into a truth table as a map-entered variable, such as discussed with respect to block 910 in Fig. 9A not only has the effect of removing an inversion, but also reduces the number of nodes in the cell. There are other truth-table sequences that permit optimization by accepting the entry of control variables and thereby reducing the number of nodes. The flow diagram in Fig. 9B provides a general explanation of how such sequences may be achieved. Essentially, the same procedures are followed as discussed with respect to block 910 for inversions: a
truth-table having elements of certain sequences is identified and permuted to realize those sequences, which are then optimized by entering the corresponding control variable. A simple example corresponds to the example discussed with respect to block 910, but without the inversion:

<table>
<thead>
<tr>
<th>Q, UB</th>
<th>Y[1]</th>
<th>Y[0]</th>
<th>VDD VSS AA</th>
</tr>
</thead>
</table>

In this example, a repetition of A with the sequence VDD VSS (instead of the sequence VSS VDD) is sufficient to enter the least-significant control variable into the truth table. This is done with the control variable directly, instead of with its inversion as was done in block 910.

Thus, the general procedure illustrated in FIG. 9B begins at block 916 with a syntactic expression for a cell, with the figure noting that it is possible (but not required) in some embodiments for certain other transformations to have been performed previously with the syntactic expression. At block 918, existing redundancies in the control are removed. Such redundancies are manifested by a repetition in the truth table at the level of the redundant control, i.e. in groups of size $2^k$ for control $Y[k]$. A trivial case occurs for the lowest level of control:

<table>
<thead>
<tr>
<th>Q, UB</th>
<th>Y[0]</th>
<th>AA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q, UB</td>
<td>Y[0]</td>
<td>A</td>
</tr>
</tbody>
</table>

This example is trivial because the cell does nothing other than produce the result A for every input; control is unnecessary and may be removed entirely. The same principle applies, however, for higher levels of control. For example, when $k=1$, the repetition of pairs of states may permit the removal of $Y[1]$:  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q, UB</td>
<td>Y[0]</td>
<td>A B</td>
<td></td>
</tr>
</tbody>
</table>

This result simply uses the fact that the higher control level has no effect, with the output of the cell depending solely on the least significant control. Similarly, when $k=2$, the repetition of quads of states may permit the removal of $Y[2]$:  

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Q, UB</td>
<td>Y[1]</td>
<td>Y[0]</td>
<td>A B C D</td>
<td></td>
</tr>
</tbody>
</table>

This result expresses the fact that the output of the cell is dependent only on the two lowest control levels and that the highest control has no effect. These principles may be extended to still larger repeated blocks and the consequent removal of still more significant control levels.

At block 920, permutations may be performed in the control to rearrange the truth table to identify sequences that permit the entry of the control variables. For the entry of a lowest control level, sequences of VDD VSS, coupled with a pair of repeated variables AA, is sought. For the entry of the next level control, sequences of VDD VDD VDD VSS, coupled with four repeated variables AAAA, is sought. For the next level control, sequences of VDD VDD VDD VDD VSS VSS VSS VSS, coupled with eight repeated variables AAAAAA, is sought. Similar sequences for still higher control levels follow the same pattern. At block 922, the control variable(s) are entered into the truth table to account for these patterns.

Thus, one example of applying blocks 920 and 922 is as follows:

|-------|------|------|------|------------------------|

In the initial syntactic expression, the sequence VDD VDD VSS VSS appears, but it is not possible to remove the $k=1$ control because there is no corresponding sequence of four repeated variables. The existence of duplicates of both A and B, however, suggests that the controls may be permuted to achieve sequences of VDD VSS coupled with pairs of repeated variables. This is achieved in the second line by permuting $Y[1]$ and $Y[0]$. Accordingly, it is possible in the third line to enter two occurrences of the $Y[1]$, which is now the lowest level of control, into the truth table. The corresponding cell is therefore optimized by reducing the number of nodes and entering the original $Y[1]$ control variable into the truth table.

Block 924 notes that the process of identifying and removing redundancies and permuting control variables to permit their entry into the truth table may be repeated to achieve further optimizations. The method is thus looped until these procedures have optimized the syntactic expression in this way as much as desired. At block 926, the method thus proceeds to another type of transformation, if desired, to effect further optimizations.

In addition to reducing nodes for a single cell, it is possible in embodiments of the invention to provide optimizations by combining nodes from multiple cells. The flow diagram shown in FIG. 9C provides a method corresponding to one embodiment for combining nodes. The method begins at block 928 with syntactic expressions for multiple cells. As for the other optimization procedures, FIG. 9C notes explicitly that prior transformations may have taken place on these syntactic expressions, although this is not required. In considering whether nodes can be combined, a check is made at block 930 whether any of the syntactic expressions outputs a control variable present in another of the expressions. If so, the control variables are converted into map-entered variables at block 932. Examples of combining nodes in which such conversions are used are provided below, but the principles of combining nodes are initially illustrated for cases where there is no such conversion.

One method for combining nodes is thus summarized by blocks 934 and 936 in which the control for the multiple syntactic expressions is combined and then states in the merged expression are defined. For example, consider the following two syntactic CHDL expressions in which the output of the second expression, A, is one of the inputs to the first expression:

<table>
<thead>
<tr>
<th>Q, UB</th>
<th>YQ</th>
<th>A B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, UB</td>
<td>YA</td>
<td>A B</td>
</tr>
</tbody>
</table>

The expressions are merged, and the nodes thereby combined, in the following way. First, the control is combined at block 934 by adding the control for the second expression to the first expression—YQ then functions as a $k=1$ level control and YA functions as a $k=0$ level control:
As can be seen, increasing the level of the YQ control by a single level to $k=1$ acts to duplicate each of the truth-table entries. The order in which the controls were combined was determined by the relationship between the inputs and outputs of the expressions. Specifically, since the expression for Q has the output of the expression for A as an input, YQ was made the higher-level control and YA the lower-level control. After combining the controls, the states are defined in the merged expression in accordance with the expressions at block 936:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

In this instance, the sequence AA is substituted with CD in accordance with the syntactic expression for A. The final expression achieves the optimization by permitting implementation of the resulting expression with a single cell as described above.

While this example showed how two expressions could be merged, it may be applied more generally to any number of expressions. For example, the following three expressions may be merged in a similar fashion with a sequential process:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

The second and third expressions both have outputs that correspond to inputs of the first expression. Accordingly, in combining control pursuant to block 934, the control of the first expression is used as the highest level control. First, the first and second expressions are merged by combining their control and defining the states in the merged expression in the same way as for the two-expression example:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

Subsequently, the third expression is merged into this combination. First, the additional control causes YQ to become a $k=2$ level control and YA to become a $k=1$ level control, with YB remaining as a $k=0$ level control:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

As seen in this expression, the additional level of control causes a duplication of each of the truth-table elements. The states in this expression are now defined according to block 936 in terms of the original third expression by substituting pairs of B's with the sequence EF:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

This result thus corresponds to an expression that combines the original three expressions and may be implemented as a cell in the manner described above.

Both of these examples have begun with expressions that correspond to ULG cells that may be implemented with multiplexors of the same size. There is, however, no limitation on embodiments of the invention that requires that they be the same size. It is possible to perform optimizations for combining nodes that correspond to merging a smaller multiplexor into a larger multiplexor or to merging a larger multiplexor into a smaller multiplexor. This may be seen in the following examples in which each of the initial expressions corresponds to a different-sized multiplexor when such an implementation is used. For example, in the set

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

the second expression has an output that is used as an input in the first expression, and corresponds to a smaller-sized multiplexor than does the first expression. The nodes are combined in the same fashion already described. First, control is combined in accordance with block 934:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

Subsequently, states are defined in the merged expression, in this instance by substituting pairs of A's with EF:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

This final expression may thus be implemented as a cell and achieves optimizations resulting from merging the smaller multiplexor into the larger multiplexor.

It is similarly possible to combine nodes in a fashion that corresponds to merging a larger multiplexor into a smaller multiplexor in embodiments that use multiplexors:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]

In this example the output of the expression corresponding to the larger multiplexor is an input to the expression corresponding to the smaller multiplexor. The nodes are combined in the same way, by first combining the control of the two expressions in accordance with block 934. Since two levels of control from the second expression are to be combined with the first expression, YQ becomes a $k=2$ level control:

\[
\begin{array}{ccc}
Q & UB & YQ \\
A & UB & YA \\
B & UB & YB \\
\end{array}
\]
Subsequently, the states are defined in accordance with block 936 by substituting quads of A’s with C D E F as dictated by the second original expression:

\[
\begin{array}{cccc}
Q & \text{UB} & YQ & YA[1] \\
\end{array}
\]

Each of these examples illustrates how to combine nodes in different circumstances where the output of one of the expressions is one of the inputs to another of the expressions. In some cases, however, the output of one of the expressions may be one of the controls of another expression, a condition checked for a block 930. In such instances, the control variable is converted into a map-entered variable at block 932 before combining control and defining states. This may be illustrated with the following two examples, the first of which corresponds to an AND sub-function and the second of which corresponds to an OR sub-function.

Thus, consider merging nodes for the following two syntactic expressions:

\[
\]

Finally, defining states at block 936 so that pairs of YQ[1] are substituted with B C as required by the original second expression results in the merged expression

\[
\]

The same procedure may also be used for the following two syntactic expressions:

\[
\]

This conversion effectively recognizes the equivalence of truth tables XIIIa and XIIIb:

\[
\begin{array}{cccc}
Q & YQ[0] & YQ[1] & A \\
\end{array}
\]

Combining control with the second expression at block 934 results in

\[
\]

Finally, defining states at block 938 so that pairs of YQ[1] are substituted with B C as required by the original second expression results in the merged expression

\[
\]
It is noted that in certain instances, the method outlined in FIG. 9C may be combined with moving flip flops forward to facilitate reductions. For example, consider application of the method to the following set of expressions:

\[ Y[2:1] \rightarrow A B C D ; \]
\[ Y[0] \rightarrow E F CLK ; \]
\[ Y[0] \rightarrow G H CLK ; \]
\[ Y[0] \rightarrow I J CLK ; \]
\[ Y[0] \rightarrow K L CLK ; \]

As previously mentioned, the notation \( Y[2:1] \) is equivalent to the expression \( Y[2] Y[1] \). This set of expressions could be implemented using five cells, one that comprises a ULG and a buffer and four that comprise a ULG and a F/F. The result of moving the flip flops forward:

\[ Y[2:1] \rightarrow A B C D CLK ; \]
\[ Y[0] \rightarrow E F ; \]
\[ Y[0] \rightarrow G H ; \]
\[ Y[0] \rightarrow I J ; \]
\[ Y[0] \rightarrow K L ; \]

Accordingly, applying the method of FIG. 9C to combine the nodes results in:

\[ Y[2:0] \rightarrow E F G H I J K L CLK ; \]

This expression may be implemented with a cell comprising a ULG and a D F/F in certain embodiments of the invention.

Thus, the method outlined in FIG. 9C permits nodes to be reduced from multiple syntactic expressions. This includes a variety of different circumstances, including cases where there are arbitrarily many syntactic expressions, where some of the expressions correspond to implementations of different sizes, and cases where some of the expressions have outputs that correspond to either inputs or controls of other expressions. After nodes have been merged, the method may proceed to another transformation at block 938, although this is not a requirement.

Additional optimization functions may be realized by using the set and reset facilities that are provided in the formalism and which may be implemented by using those facilities in cells according to embodiments of the invention. The use of set and reset facilities are summarized in FIG. 9D and arise primarily when half the truth table states are either high or low. Thus, a method for optimizing begins with a syntactic expression for a cell at block 940, with FIG. 9D noting explicitly that previous transformations may also have been performed on the cell expression, although this is not a requirement. A check is made at block 944 whether half the truth table states are low. If so, the control variable may be changed to a reset to a F/F in accordance with block 946. For example, consider the syntactic function

\[ Y[1:0] \rightarrow VDD A VDD B CLK ; \]

In this example, a cell that implements this function comprises a ULG and a D F/F. As previously mentioned, the notation \( Y[2:0] \) is equivalent to \( Y[2] Y[1] Y[0] \). Half of the truth table states in this expression are low, i.e. VSS, so that the highest level control may be entered into the map as a reset input to the F/F:

\[ Y[2:0] \rightarrow VSS VSS VSS VSS A B C D CLK ; \]

The name of the cell explicitly notes that a resetable D F/F (DR) is used and the reset has been noted. This function may then be implemented using the cells described above.

FIG. 9D notes at block 942 that in some instances it may be useful to perform permutations of the control variables to put the truth table into a form that allows using the reset input for optimization. If half the truth table states are low, but do not appear as a group, they may be aligned with the permutations. For example, the function:

\[ Y[1:0] \rightarrow VDD VSS VSS VSS VSS A B C D CLK (RST = 'C') ; \]

has half of its truth table states low, but they are not aligned. Interchanging the control variables aligns them

\[ Y[1:0] \rightarrow VSS A VSS B CLK ; \]

so that the most significant control variable may be entered into the map at block 946 as a reset input to the F/F:

\[ Y[1:0] \rightarrow A B Y[0] CLK (RST = 'C') ; \]

Note that in this example, the lack of alignment among the low states has resulted in \( Y[0] \) being entered into the map as part of the optimization rather than \( Y[1] \).

Similar optimizations may be achieved half the truth table states are high, as checked at block 948. If so, permutations of the control variables may be performed at block 950 to align the high states and the control variable entered into the map as a set input to a F/F at block 952. For example, consider the function:

\[ Y[1:0] \rightarrow VDD A VDD B CLK ; \]

Half of the states in the truth table are high, i.e. VDD, so that optimization with a set input may be achieved. Permuting the control variables to align the high states in accordance with block 950 results in
permutation may be used to achieve the conditions for using

begins with the syntactic expression

based on pairs:

variable as a set input to the F/F results in

set or reset for optimization.

It is now apparent that since there are only two states in the

truth table and one of them is low, that half the states are low

interchanging based on single entries and a second interchange

interchange based on pairs:

Entering the most significant control as a map-entered

variable results in

VDD VDD A B CLK :

This function may then be implemented using the cells
described above in an embodiment.

Still other combinations of expansion, inversion, and/or

permutation may be used to achieve the conditions for using

set or reset for optimization. An example that illustrates

several of the optimization manipulations discussed above

begins with the syntactic expression

Q .UDS γ[1] A B γ[0] CLK (SET = "C") :

Optimization of this function may proceed by first expand-
ing the NOT_RESET as a control variable:

Q .UD AN NOT_RESET VSS VSS VDD VSS CLK :

Inversion of the two control variables in accordance with

FIG. 9A results in two interchanges of the truth table, a first

interchange based on single entries and a second interchange

based on pairs:

Q .UD A RESET VSS VDD VSS VSS CLK :

Interchanging the controls to prepare for entering A as a

map-entered variable results in

Q .UD RESET A VSS VSS VDD VSS CLK :

The presence of the VDD VSS sequence and the repeated

VSS element permits the A control variable to be entered

into the map in accordance with FIG. 9B:

Q .UDR A RESET CLK (RST = "C") :

Since all of the control variables have been entered into

the map, there is no need for a ULG element in implement-
ing this optimized expression. Instead, it may be imple-

dented in one embodiment using only a resetable D F/F,
even though the original expression corresponded to an

implementation comprising both a ULG and a D F/F.

Each syntactical expression of the formalism may be

viewed as an entry for a ULG netlist that defines a digital
circuit. The ULG netlist uses basic cells and follows the

CHDL syntax. Such a ULG netlist may be at least partially

optimized by successively performing some of the indi-

vidual manipulations described above. In some

embodiments, the resulting optimized ULG netlist may be

implemented directly using the cells described above. In

other embodiments, however, the manipulation of the ULG

netlist may be viewed as intermediate step in optimizing a
digital design that has been expressed in another netlist

format. In such instances, the other netlist format using

another cell library and/or another syntax (e.g., VHDL or

Verilog) is initially translated to the ULG netlist format.

After performing some optimization steps, the optimized

ULG netlist may be translated back into the original format

for implementation. In this way, an embodiment is provided

that achieves optimization of digital designs within preex-

isting netlist formats.

There are a variety of ways in which the individual

manipulations of the syntactic expressions comprised by a

ULG netlist may be performed and the degree to which the

corresponding digital design is optimized may depend on

how those manipulations are executed. While in some

instances it is possible for the manipulations to be performed

by hand by a digital designer, it is expected that at least some

level of automation may be used. In one embodiment, for

example, a computer program may be provided that allows

a digital designer to select the types of manipulations to be

performed and sections of the ULG netlist on which to

perform them, with the execution of the manipulations being

performed automatically. In another embodiment, a prede-

termined algorithm is used in a completely automated way
to perform the manipulations or optimizations.

One example of such a predetermined algorithm is illus-

trated with the flow diagram provided in FIG. 10, although

it will be appreciated by those of skill in the art that

numerous other algorithms may alternatively be used. In

some specific instances, alternative algorithms use the same

manipulations but perform them in a different order. In some

other specific instances, alternative algorithms use a differ-

ent set of manipulations. The algorithm shown in FIG. 10

cannot at block 1004 by translating an existing netlist to

a ULG netlist if the design to be optimized was not initially

created using the ULG formalism. In one embodiment,

translating from the existing netlist to the ULG netlist is

performed as a one-to-one translation between syntactic

expressions. For one embodiment, this translation may be

viewed conceptually in terms of the elements used in that

embodiment to implement the original and ULG netlists—

every logic gate, such as NAND, OR, etc. is converted into

a multiplexor-based implementation amenable to optimiza-

tions provided by the syntactic manipulations discussed above.

At block 1008, data and control elements are discerned in

the resulting ULG netlist. The distinction between such data

and selection control elements was previously discussed

with respect to Table VIII. In one embodiment, such dis-

cernment may be performed with a high-level design lan-

guage. In certain embodiments, there are additional advan-

tages in the subsequent optimization where the data and

control are identified from the original behavioral netlist.

Once the data and control elements have been identified, the

operations identified in blocks 1016–1068 may be per-
formed by using the syntactic manipulations described above. Thus, at block 1016, connection cells to base Boolean values are removed by incorporating the corresponding functions into the syntactic expressions. Similarly, non-inverting buffers are removed at block 1020 and also incorporated directly into the syntactic expressions. At block 1024, inverters are removed by syntactically inverting the relevant data elements in individual syntactic expressions. The removal of redundant nodes at block 1028 may proceed by reducing, combining, and permuting nodes in the syntactic expressions. Methods for such reductions, combinations, and permutations for certain embodiments were described above in connection with FIGS. 9B and 9C. At block 1032, inversions are removed. The inversions removed may include control/state variable inversions, truth table inversions and buffer inversions, for which methods of removing were described above in connection with FIG. 9A.

A loop comprising blocks 1036 and 1040 may be executed to identify and merge fanout nodes. A “fanout node” describes a configuration in which a single output is directed to multiple parts of truth tables and, in some instances, optimizations may be realized by merging such fanout nodes. A check is first performed at block 1036 to identify whether the ULG netlist includes any fanout nodes, in which case they are reduced at block 1040 by performing permutations and reductions as described in connection with FIG. 9C. The loop between blocks 1036 and 1040 is included because the reductions performed at block 1040 may produce some new fanout nodes that may be identified and merged through additional permutations and reductions. The method thus loops until no fanout nodes are identified at block 1036.

After the fanout nodes have thus been merged, a check is made at block 1044 to identify syntactic expressions corresponding to synchronous cells in which half the truth-table states are low. If such an expression is identified, the corresponding control variable is moved to be a reset input at block 1048. A method for doing so is described in connection with FIG. 9D and may include performing permutations to align the low truth-table states. A similar check is made at block 1052 to identify syntactic expressions corresponding to synchronous cells in which half the truth-table states are high. If such an expression is identified, the corresponding control variable is moved to be a set input at block 1056. A method for doing so is also described in connection with FIG. 9D and may include performing permutations to align the high truth-table states.

At block 1060, the syntactic expressions are grouped by common inputs. Such grouping permits identification of subfunctions at block 1064. The common subfunctions have shared characteristics that may be extracted before reducing other nodes. At block 1068, buffers are made to be inverting buffers.

If the original ULG netlist produced at block 1004 was nonoptimal, performing the above manipulations may produce a different ULG netlist that is amenable to implementation with smaller area, greater speed, and/or lower power requirements. In some embodiments, it is possible that not all of the manipulations will be performed, that some may be performed multiple times, and that they may be performed in a different order than described. Once the method has produced a new ULG netlist, it may be implemented at block 1072 by translating the syntactic expressions in the ULG netlist with the cells described above. In such embodiments, the method functions not only to optimize the digital design but also to provide a multiplexor-based implementation of it. In some alternative embodiments, the resulting ULG netlist may be amenable to translation back into the original netlist syntax for implementation using Boolean logic gates. In these embodiments, the method may be viewed as providing an optimization of a digital design while retaining its underlying structural characteristics.

In some embodiments, it is desirable for the techniques used for the syntactic manipulations to be embedded in an optimization tool or synthesizer. Accordingly, the methods of the invention for converting a netlist into a ULG netlist and for optimizing the ULG netlist may be performed by a computer, one example of a suitable configuration for which is shown in FIG. 1. This figure broadly illustrates how individual system elements may be implemented in a separated or more integrated manner. The computer 1100 is shown comprised of hardware elements that are electrically coupled via bus 1112, including a processor 1102, an input device 1104, an output device 1106, a storage device 1108, a computer-readable storage medium reader 1110a, a communications system 1114, a processing acceleration unit 1116 such as a DSP or special-purpose processor, and a memory system 1118. The computer-readable storage media reader 1110a is further connected to a computer-readable storage medium 1110b, the combination comprehensively representing remote, local, fixed, and/or removable storage devices plus storage media for temporarily and/or more permanently containing computer-readable information. A communications system 1114 may comprise a wired, wireless, modem, and/or other type of interfacing connection.

The computer 1100 also comprises software elements, shown as being currently located within working memory 1120, including an operating system 1124 and other code 1122, such as a program designed to implement optimization methods of the invention. It will be apparent to those skilled in the art that substantial variations may be used in accordance with specific requirements. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, software (including portable software, such as applets), or both. Further, connection to other computing devices such as network input/output devices may be employed.

EXAMPLE

The method described with respect to FIG. 10 was used to optimize a netlist for a microcontroller. A synthesized netlist of commercial library cells was translated to the ULG netlist formalism and optimized by performing the described syntactic manipulations. The size of implementing the optimized ULG netlist with the cells described herein was then compared with the size of the original implementation. The overall size was reduced by about 37%, a significant reduction.

V. Digital Design with Syntactic Manipulation

Referring next to FIG. 12, a flow diagram of an embodiment of a process for preparing a ULG ASIC cell library is shown. In this embodiment, a new fabrication process is being adapted to use the kernel cell/basic cell topology. The depicted portion of the process begins in step 1204 where a layout engineer or technician analyzes the layout rules for the target fabrication process. The fabrication process could have different amounts of metalization layers, different semiconductor compositions, different transistor types, and different topologies such as SOI, etc. that are considered during layout. The kernel cells are laid out, at least partially, by hand to optimize the circuitry in step 1208. Some embodiments could start the kernel cell layout with a computer routed design that is hand-customized. Care is taken to have kernel cells of a consistent height and to
adhere to abutment guidelines such that clock signals and some of the I/O are latitudinally aligned. Other embodiments could rely upon autorouting entirely to build the kernel cells.

The kernel cells can be assembled in a number of ways to potentially create hundreds of basic cells in the ULG ASIC cell library. The basic cell configurations are specified in step 1216. This could be done by editing a script fed to the tool that combines the kernel cells into basic cells. In step 1216, the kernel cells are automatically assembled into basic cells 100 in this embodiment. Some embodiments could assemble the kernel cells manually or in-art manually. Once the basic cells are generated, they are verified in step 1220. This verification validates the digital and analog performance of the basic cells. Any problems uncovered in verification can be fixed in the kernel and/or basic cells. To complete the ULG ASIC cell library, any specialized cells are laid-out in step 1224.

In one embodiment, selection logic is used in digital circuits to replace some or all combinatorial logic. A significant proportion of the basic cells use a ULG 104. In contrast, conventional semiconductor circuits do not use ASIC cell libraries that include ULG circuits. In one embodiment, the ULG circuits come in various sizes, that have between two and eight inputs and between one and three select lines.

The extensive use of ULGs or selection circuits in this invention could be characterized in a number of ways in the various embodiments, those characterizations include:

1. A digital IC design using an ASIC cell library that includes a proportion of cells with selection circuits. In various embodiments, the proportion of the cells with selection circuits in the ASIC cell library could include, for example, 5% or more, 10% or more, 25% or more, 50% or more, 75% or more, 80% or more, 90% or more.

2. A digital IC design using an ASIC cell library that includes a percentage of basic cells that each include two or more kernel cells. For example, that percentage could be more than 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, or 90%.

3. A digital design that includes a proportion of selection circuits. Where the proportion is defined in terms of circuit area, power consumption or number of kernel cells. In various embodiments, the proportion of a digital design that includes selection circuits could include 1% or more, 2% or more, 5% or more, 10% or more, 20% or more, 30% or more, 40% or more, or 50% or more.

4. A digital design implemented in a semiconductor circuit where the digital building blocks include memory cells, one input Boolean operators and selection circuits. The selection circuits are non-Boolean operators and have three or more inputs. In some cases, there could be a small proportion of the digital building blocks could be Boolean operators, such as 1% or less, 2% or less, 5% or less, 10% or less, 20% or less.

Those digital designs could be embodied in the form of, for example, a structural netlist, a behavioral netlist, a HDL entry tool 1304, an optimization tool, a static & dynamic timing analysis tool 1316, and a place & route tool 1320. In this embodiment, the digital designer uses all the tools of their normal design flow, but includes the optimization tool 1312 after the synthesis tool 1308. The optimization tool 1312 performs the syntactic manipulation in this design flow.

The HDL entry tool 1304 is a software edit tool that allows the digital designer to enter HDL as a behavioral netlist. The HDL could be VHDL, Verilog or Concise Hardware Description Language (CHDL). CHDL is a HDL that is tuned for the design constructs beneficial for designs with ULG cells. The HDL entry tool 1304 could receive feedback from the other tools to identify portions of the code that have problems found by those other tools. Other design capture tools could be used instead of a HDL entry tool, for example, state machine tools, RTI tools, schematic capture tools, etc. Dynamic timing analysis could be performed on the behavioral netlist to confirm proper functionality.

A behavioral netlist is converted by the synthesis tool 1308 into a structural netlist using the conventional cell library 1324 in this embodiment. The behavioral constructs are converted to Boolean constructs and optimized. From the conventional cell library 1324, cells are used for the optimized Boolean constructs. In some cases, typical timing values are used by the synthesis tool 1308 to identify potential problems in the conversion process using static timing analysis. Dynamic timing analysis could also be performed on the structural netlist to confirm functionality wasn’t compromised during the conversion.
The optimization tool 1312 uses syntactic manipulation to improve the design in at least one of the following areas: power consumption, leakage current, fanout, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, glitches, etc. In this embodiment, variables can be fed to the optimization tool 1312 in order to set the priorities among these design factors. These variables could be set on a scale of one-to-ten to indicate relative value along a sliding scale.

Some embodiments could optimize for various factors by use of alternative cells in the ULG library. Certain cells could be optimized for various design factors such as power consumption, leakage current, fanout, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, glitches, etc. During the optimization, the alternative cells could be used based upon how the digital designer set the priority variables.

In this embodiment, the optimization tool optimizes the structural netlist from the synthesis tool to produce an optimized structural netlist that uses the same conventional cell library 1324. The synthesized structural netlist is read and converted into an intermediate netlist that uses an embodiment of the ULG ASIC cell library. This embodiment of the ULG cell library can be somewhat simplified as the variation used in an ASIC could be unnecessary when only optimizing. Syntactic manipulation is performed upon the intermediate netlist according to the optimization priorities, if specified. Some embodiments could perform a default optimization that may or may not be modifiable by the digital designer. Once the intermediate netlist is optimized, it is converted to an optimized structural netlist that uses the conventional cell library 1324.

Static & dynamic timing analysis is run on the optimized structural netlist. The static timing analysis takes into account timing relationships for the optimized structural netlist and identifies portions of the circuit that may fail to meet the timing requirements of the circuit. Parameters can be entered into the synthesis and/or optimization tools 1308, 1312 to prioritize certain portions of the circuit to make meeting static timing requirements easier.

Dynamic timing analysis can come in a few forms. Input waveforms can be designed to stimulate the design, whereafter the digital designer checks for correct output waveforms. To automate this process, test vectors can be developed and applied to the inputs of the circuit whereafter output test vectors are tested against the actual output. Discrepancies are noted as errors and fixed by tweaking the behavioral code and synthesis/optimization tools 1308, 1312 such that errors are not introduced into the process.

Once the digital designer is happy with the structural netlist, the place & route tool 1320 performs a physical layout of the circuit. A location for each cell in the optimized structural netlist is chosen and traces are laid-out to interconnect those cells according to the netlist. These types of tools 1320 are automated or semi-automated. More accurate timing values are available after place & route because the trace lengths interconnecting the cells are known. Further static/dynamic analysis 1316 can be performed to assure that new errors aren't introduced during the place and route process. Once a suitable layout is achieved, masks can be produced and fabrication can start. Dynamic testing on the resulting chip may be performed to test functionality after fabrication.

Referring next to FIG. 14, a block diagram of another embodiment of a design flow 1400 is shown that uses syntactic manipulation and the ULG ASIC cell library. In this embodiment, a ULG cell library 1404 is used during synthesis 1308. By targeting the ULG cell library, the synthesized structural netlist is in a format readily understood by the optimization tool 1312 such that a conversion to an intermediate netlist is unnecessary. The optimization tool 1312 performs the syntactic manipulation before converting from the intermediate format to the conventional cell library 1324. The optimized structural netlist is used in the conventional cell library 1324, which is understood by the place & route tool 1320 and the fab or foundry.

With reference to FIG. 15, a block diagram of another embodiment of a design flow 1500 is shown that uses the ULG ASIC cell library 1404 for the final netlist used by the fab or foundry. In this embodiment, the ULG ASIC cell library 1404 has been produced for the target process at the fab or foundry. The synthesis tool 1308 converts the behavioral netlist into a synthesized structural netlist that uses the ULG ASIC cell library 1404. The optimization tool can process the netlist without any conversion between cell libraries such that the resulting optimized structural netlist also uses the ULG ASIC cell library.

Referring next to FIG. 16, a block diagram of yet another embodiment of a design flow 1600 is shown that combines synthesis and syntactic manipulation into a single tool 1604. The synthesis & optimization tool 1604 takes the behavioral netlist from the HDL entry tool 1304 and converts it to an intermediate structural netlist using the ULG ASIC cell library 1404. The intermediate structural netlist is optimized using syntactic manipulation with the tool 1604 to produce an optimized structural netlist that uses the ULG cell library 1404. In this embodiment, the HDL entry tool 1304 uses CHDL.

In some embodiments, the synthesis & optimization tool 1604 may not have a ULG cell library 1404 for the target ASIC process. In that case, the synthesis & optimization tool 1604 would convert the intermediate netlist to an optimized netlist using the conventional cell library 1324.

With reference to FIG. 17, a block diagram of still another embodiment of a design flow 1700 is shown that uses an interactive direct verification tool 1704 throughout the design flow 1700. Verification allows debugging a digital design throughout the design flow. In this embodiment, the interactive direct verification tool 1704 tracks the evolution of the design from a behavioral netlist through to a structural netlist that has been placed and routed. The HDL entry tool uses CHDL and the design flow uses the ULG cell library 1304 for the target fabrication process. The various forms of the design embodied in the evolving netlist are accessible to the interactive direct verification tool 1704.

Any identified portion of the circuit can be traced through the various steps of the design flow 1700 using the interactive direct verification tool. For example, a case statement in the CHDL behavioral code can be followed through to the various structural netlists to see how that case statement was implemented. Conversely, a portion of a structural netlist can be followed back to preceding structural netlists or even the behavioral netlist.

Referring next to FIG. 18, a block diagram of still another embodiment of a design flow 1800 is shown that uses an interactive direct verification tool 1804 throughout the design flow and after fabrication. This embodiment shows the fabrication 1808 and final test 1812 in the design flow. The fabrication 1808 is performed after the design is thoroughly tested. After production, the chips can be tested again. Test vectors are applied to the chip inputs and scan ports in the dynamic analysis in final test 1812. Where an error can be isolated to a pin or node in final test 1812, the interactive direct verification tool 1804 can show the engineer the progression of the design that relates to that failure.
With reference to FIG. 19, a flow diagram of an embodiment of a design process 1900 is shown. In the depicted portion of the design process 1900, the sequential steps are shown along with the test and rework steps. In step 1904, the digital design is entered using an entry tool. A HDL such as Verilog, VHDL or CHDL is used to enter the behavioral netlist for the circuit being designed. Throughout the design entry phase, dynamic timing analysis is performed in step 1916 to verify that the behavioral netlist is probably being prepared correctly. At this stage, the dynamic timing analysis is probably done in a waveform simulation tool. Various scenarios are designed with the simulation tool to test various conditions of the digital design. During the timing analysis of step 1916, problems could be found in step 1906. Rework of the behavioral code could be performed by returning to step 1904 where problems are found. Processing continues from step 1916 to step 1908 where no problems are found.

Synthesis and optimization is performed in step 1908 to convert the behavioral netlist to an optimized structural netlist. Conversion between cell libraries can also be performed in this step. Some embodiments may perform the synthesis and optimization as separate steps. Static and dynamic timing analysis are performed in steps 1912 and 1916. If problems are found, rework may be done in step 1904 or step 1908. Rework in step 1908 could include changing parameters fed to the synthesis & optimization tool 1604 or correcting problems with libraries.

Once generally satisfied with the optimized structural netlist, processing continues to step 1920 for place & route of the design. In this step, the trace lengths and drive requirements are more accurately analyzed to improve the verification that can be performed on the digital design. Once again static and dynamic timing analysis is performed in step 1912 and 1916 to verify the laid-out design still behaves properly. Where there are problems as detected in step 1906, rework could be performed in steps 1904, 1908 or 1920. The interactive direct verification tool 1804 can assist the digital designer find where flaws were likely introduced into the netlist. Where there are no problems after layout, the design is fabricated in step 1924.

Having described several embodiments, it will be recognized by those of skill in the art that various modifications, alternative constructions, and equivalents may be used without departing from the spirit of the invention. Accordingly, the above description should not be taken as limiting the scope of the invention, which is defined in the following claims.

What is claimed is:

1. A digital design method for manipulating a digital circuit netlist, the digital design method comprising:
   - loading a first netlist, wherein the first netlist is comprised of a first plurality of basic cells that are comprised of a first plurality of kernel cells;
   - manipulating the first netlist to create a second netlist, wherein the second netlist is comprised of a second plurality of basic cells that are comprised of a second plurality of kernel cells, wherein:
     - a percentage of the first and second plurality of kernel cells are selection circuits, and
     - there is less chip area consumed in the second plurality of basic cells than in the first plurality of basic cells;
   - and
   - storing the second netlist.

2. The digital design method for manipulating the digital circuit netlist as recited in claim 1, further comprising steps of:
   - converting a netlist using a language format into the first netlist by, at least, substituting some Boolean cells for the first plurality of basic cells;
   - converting the second netlist to a third netlist by, at least, substituting some of the second plurality of basic cells for Boolean cells, wherein the third netlist also uses the language format.

3. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein another percentage of the first and second plurality of kernel cells are memory circuits.

4. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein another percentage of the first and second plurality of kernel cells are buffer circuits.

5. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein each selection circuit includes a multiplexor.

6. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein a total number of different kernel cells in the first and second plurality of kernel cells is less than one of 10, 20, 30, 40, 50, 75, and 100.

7. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein the percentage is chosen from a group consisting of 2% or more, 5% or more, 10% or more, 20% or more, 30% or more, and 40% or more.

8. The digital design method for manipulating the digital circuit netlist as recited in claim 1, further comprising a step of receiving one or more optimization parameters that influence the manipulation step according to the optimization parameters.

9. The digital design method for manipulating the digital circuit netlist as recited in claim 8, wherein the optimization parameters influence choosing a subset of possible kernel cells to use in the manipulation step.

10. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein:
   - the second netlist is comprised of a plurality of syntactic expressions,
   - a percentage of the plurality of syntactic expressions identify a higher-order function of base Boolean values,
   - the percentage is chosen from a group consisting of 100% or more, 20% or more, 30% or more, 40% or more, 50% or more, and 60% or more.

11. The digital design method for manipulating the digital circuit netlist as recited in claim 1, wherein the manipulation step optimizes at least one of power consumption, leakage current, fanout, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, and glitches.

12. A digital design method for tuning a digital circuit netlist, the digital design method comprising:
   - loading a first netlist, wherein the first netlist is comprised of a first plurality of basic cells that are comprised of a first plurality of kernel cells;
   - manipulating the first netlist to create a second netlist, wherein the second netlist is comprised of a second plurality of basic cells that are comprised of a second plurality of kernel cells, wherein:
     - a percentage of the first and second plurality of kernel cells are selection circuits, and
     - there is less chip area consumed in the second plurality of basic cells than in the first plurality of basic cells;
   - and
   - each of the first and second plurality of kernel cells is chosen from a group consisting of a selection circuit, a memory circuit and a buffer circuit, and
there are less kernel cells in the second plurality of kernel cells than the first plurality of kernel cells; and storing the second netlist.

13. The digital design method for tuning the digital circuit netlist as recited in claim 12, further comprising steps of:
   converting a netlist using a language format into the first netlist by, at least, substituting some Boolean cells for the first plurality of basic cells;
   converting the second netlist to a third netlist by, at least, substituting some of the second plurality of basic cells for Boolean cells, wherein the third netlist also uses the language format.

14. The digital design method for tuning the digital circuit netlist as recited in claim 12, wherein each selection circuit includes a multiplexor.

15. The digital design method for tuning the digital circuit netlist as recited in claim 12, further comprising a step of receiving one or more optimization parameters that influence the manipulation step according to the optimization parameters.

16. The digital design method for tuning the digital circuit netlist as recited in claim 15, wherein the optimization parameters influence choosing a subset of possible kernel cells to use in the manipulation step.

17. The digital design method for tuning the digital circuit netlist as recited in claim 12, wherein the manipulation step optimizes at least one of power consumption, leakage current, fanout, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, and glitches.

18. A digital design method for manipulating a digital circuit netlist, the digital design method comprising:
   loading the digital circuit netlist;
   converting the digital circuit netlist having a language format into a first netlist by, at least, substituting some Boolean cells with a first plurality of basic cells, wherein the first netlist is comprised of a first plurality of basic cells that are comprised of the first plurality of kernel cells;
   manipulating the first netlist to create a second netlist, wherein the second netlist is comprised of a second plurality of basic cells that are comprised of a second plurality of kernel cells, wherein:
   a percentage of the first and second plurality of kernel cells are selection circuits, and
   there is less chip area consumed in the second plurality of basic cells than in the first plurality of basic cells;
   converting the second netlist to a third netlist by, at least, substituting some of the second plurality of basic cells with Boolean cells, wherein the third netlist also uses the language format; and
   storing the third netlist.

19. The digital design method for manipulating the digital circuit netlist as recited in claim 18, further comprising a step of synthesizing behavioral constructs into Boolean cells.

20. The digital design method for manipulating the digital circuit netlist as recited in claim 18, wherein each selection circuit includes a multiplexor.

21. The digital design method for manipulating the digital circuit netlist as recited in claim 18, further comprising a step of receiving one or more optimization parameters that influence the manipulation step according to the optimization parameters.

22. The digital design method for manipulating the digital circuit netlist as recited in claim 21, wherein the optimization parameters influence choosing a subset of possible kernel cells to use in the manipulation step.

23. The digital design method for manipulating the digital circuit netlist as recited in claim 18, wherein the manipulation step optimizes at least one of power consumption, leakage current, fanout, chip area, number of masks, number of process steps, yield, capacitance, inductance, resistance, and glitches.

* * * * *