FIELD PROGRAMMABLE LOGIC DEVICE WITH DYNAMIC INTERCONNECTIONS TO A DYNAMIC LOGIC CORE

Brief Description of the Invention

This invention relates generally to Field Programmable Logic Devices. More particularly, this invention relates to the architecture, operation and design of a Field Programmable Logic Device which utilizes dynamic interconnections to access a logic core which dynamically changes at staged logic levels.

Background of the Invention

Programmable Logic Devices (PLDs) are widely used to implement logic functions for controlling electronic devices. A mask-programmable logic device is programmed by a manufacturer through the process of fabricating the device. In contrast, a field-programmable logic device is distributed by a manufacturer in an unprogrammed state. The purchaser of the device subsequently programs it in the "field" to execute a desired function. The present invention relates to field programmable logic devices (FPLDs). Field Programmable Gate Arrays (FPGAs) are one form of FPLDs.

The primary benefit of FPLDs is that user-programmability allows for rapid and inexpensive prototype development. Another important benefit associated with FPLDs is that they may be re-programmed to implement different designs.

Specific designs to be implemented within a given architecture are developed with Electronic Design Automation (EDA) techniques, also referred to as Computer Aided Design (CAD) techniques. EDA tools include logic synthesizers, physical design tools, and timing verifiers.

This Sheet Has Been Replaced

REPLACED SHEETS SG
☑ Description : paged ....1.... to paged ....25....
☑ Claim(s) : paged ....21.... to paged ....24....
☑ Drawing(s) : paged ....1/11.... to paged ....11/11....
☐ Abstract : paged ..... to paged .....
DATE OF RECEIPT .....

Logic synthesizers convert a high-level description of a circuit into a netlist which describes circuit components and the connections between the components. Given a netlist, physical design tools determine physical locations of the components and the wire segments required for realizing the connections between the components. This step of the design process is generally computationally expensive. Consequently, refining this process is an ongoing matter.

Timing verifiers analyze the timing performance of the circuit described in the netlist. Based upon the timing performance of the circuit, an engineer may modify the netlist to improve the performance of the circuit.

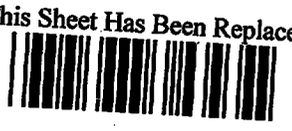
It is desirable to completely automate the implementation of a given circuit into an FPLD architecture. For this to occur, it is important to design the FPLD architecture to insure that the EDA tools can accurately estimate the timing performance and also be able to place and route a given circuit design with ease. The FPLD architecture should have the foregoing features without eliminating the ability to implement complex circuits.

Look-Up Tables (LUTs) are generally used in FPGAs. A LUT is a digital device that provides an output value for a given set of input values. The output value is stored at a memory location which is addressed by the input values.

LUT-based FPGAs utilize a sequence of LUTs to form a multi-level structure. In such a device, the output from a first LUT is combined with new input values at a second LUT to yield a new output value. The second LUT may be viewed as a second logic level. Likewise, the output value from the second LUT may be subsequently combined with additional input values at a third LUT. In such a device, there are three logic levels with a LUT at each logic level.

There are a number of problems associated with traditional LUT-based FPGAs. As described in the

This Sheet Has Been Replaced



preceding paragraph, these devices have a sequential structure in which the LUTs are distributed over the silicon die which implements the logic. This approach can be spatially expensive. Another problem relates to routing the connections between the various LUTs. The connections between the LUTs may introduce unpredictable and undesirable signal propagation delays. The propagation delays may require re-placement and re-routing of the circuit until all timing constraints are met. In some circumstances, it may be impossible to place and route a given circuit design on an FPGA to satisfy reasonable timing constraints.

Summary of the Invention

A novel architecture, operation and design for a Field Programmable Logic Device is disclosed. The device implements a circuit by using a dynamic logic core that executes staged logic corresponding to the logic levels of the implemented circuit. Logic inputs to the dynamic logic core are obtained from a dynamic interconnection array. Appropriate logic inputs for a given logic level are dynamically selected and routed by the dynamic interconnection array. When necessary, signals which are required at subsequent logic levels are buffered by the dynamic interconnection array. The dynamic interconnection array selects logic inputs for a given logic level from circuit input signals, buffered signals and dynamic logic core output signals.

Brief Description of the Drawings

For a better understanding of the nature and objects of the invention, reference should be made to the following detailed description taken in conjunction with the accompanying drawings, in which:

FIGURE 1 is a high-level schematic representation of the FPLD of the invention.

FIGURE 2 is schematic of the FPLD of the invention.

This Sheet Has Been Replaced



FIGURE 3 depicts a level counter which may be used in conjunction with the invention.

FIGURE 4 depicts a dynamic logic module used in accordance with the invention.

FIGURE 5 represents the processing steps associated with implementing a design into an FPLD in accordance with the invention.

FIGURE 6-1 depicts an apparatus which may be used to execute the processing steps of Figure 5.

FIGURE 6-2 depicts the relationship between the downloaded logic generated by the apparatus of Figure 6A and an FPLD which subsequently utilizes the downloaded logic.

FIGURE 7 depicts the control sequence associated with the operation of the FPLD of the present invention.

FIGURE 8 is a schematic of a circuit which, by way of exhibiting the invention, is implemented in the FPLD of the invention.

FIGURE 9 depicts the logic tables or look-up tables associated with the circuit schematic of Figure 8; these tables are implemented in the dynamic logic modules of the invention.

FIGURES 10-13 depict the implementation of the circuit of FIGURE 8 into an FPLD of the invention.

FIGURE 14 is a schematic of a circuit which, by way of exhibiting the invention, is implemented in the FPLD of the invention.

FIGURE 15 depicts the logic tables or look-up tables associated with the circuit schematic of Figure 14; these tables are implemented in the dynamic logic modules of the invention.

FIGURES 16-20 depict the implementation of the circuit of Figure 14 into an FPLD of the invention.

Like reference numerals refer to corresponding parts throughout the several views of the drawings.

Detailed Description of the Invention

This Sheet Has Been Replaced



Figure 1 depicts a Field Programmable Logic Device (FPLD) 20 in accordance with the invention. The FPLD 20 includes a dynamic logic core 22 which includes an array of dynamic logic modules 24. As will be more fully described herein, each dynamic logic module 24 executes a set of logical operations. The set of logical operations includes a number of individual logic stages or levels. Each logic level corresponds to a logic level in a gate-level representation of a circuit which is implemented in the FPLD 20.

The dynamic logic core 22 is coupled to a dynamic interconnection array 26. The dynamic interconnection array 26 dynamically alters the connections to the dynamic logic core 22 so that each logic level of the logic core 22 receives the appropriate input signals. Since the same routing resources are used to implement the entire circuit, the present invention optimizes silicon resources. Another advantage of this technique is that the routing of the circuit is straightforward, and therefore the design automation of the circuit is simplified.

Interface latches 28 are provided between the dynamic logic core 22 and the dynamic interconnection array 26. A buffer array 30 is utilized to bypass selected logic levels of the dynamic logic core 22, as will be more fully described below. Finally, the FPLD 20 of the invention is implemented with the use of input latches 32 and output latches 34.

Figure 2 is a more detailed representation of the FPLD 20 of Figure 1. The dynamic logic modules 24 are fed by a Logic Cross Bar (L-Cross Bar) module 40 which forms a portion of the dynamic interconnection array 26. As will be discussed below, the L-Cross Bar module 40 may be implemented in a number of ways. At the present time the L-Cross Bar module 40 should be viewed as a device for routing values which will eventually be conveyed to the dynamic logic modules 24. The L-Cross Bar module 40 is

This Sheet Has Been Replaced



operated by an L-Cross Bar Dynamic Configuration Controller 42.

One input to the controller 42 is a "level" value. As will be more fully described below, the level value is the logic level of the circuit being implemented. This level value is used to determine appropriate control of the L-Cross Bar module 40.

Figure 3 depicts a level counter control module 44 which may be used in accordance with the invention. The module 44 includes a counter 46 which tracks the circuit level. A controller 48 controls the operation of the internal level counter 50. If a multi-chip mode is selected for implementing the invention, as will be described below, then the controller 48 activates control to the next chip and uses signals from a previous chip. Otherwise, these signals are ignored. The internal level counter 50 is incremented or reset by the controller 48 based on the mode and the word stored in counter 46. The word in counter 50 is then sent to various devices within the FPLD 20, as depicted in Figure 2.

The L-Cross bar 40 routes input values, buffer values, and outputs from the dynamic logic modules 24 into the dynamic logic modules 24. The L-Cross bar 40 dynamically changes the interconnections to the dynamic logic modules 24 for each level of the implemented circuit, as will be illustrated below. As depicted in Figure 2, input latch 32 receives input values. The input values are then conveyed to a multiplexer 54. Multiplexer 54 also receives values from the buffer array 30. Selection of signals from the input or the buffer array 30 is controlled by a Primary Input/Buffer Select Controller 56.

The outputs of the multiplexer 54 are conveyed to a Buffer Cross bar module 58 (B-Cross bar). The B-Cross bar module 58 routes values which are present at a first logic level and which will also be required at a future logic level. Specifically, values which are to be bypassed to

This Sheet Has Been Replaced



another logic level are conveyed from the B-Cross bar module to the buffer array 30. Multiplexers 61 are used to select a value from the B-Cross Bar module 58 or from an output value from the dynamic logic core 22, as will be described below. Latch 34 stores the control signals for multiplexer 61.

The B-Cross bar module 58 is controlled by B-Cross Bar Dynamic Configuration Controller 60. A logic level value forms an input to the B-Cross Bar Dynamic Configuration Controller 60. As indicated earlier, the level value corresponds to the logic level of the circuit being implemented. As depicted in the figure, values may bypass the B-Cross Bar 58 and be connected to the L-Cross Bar module 40. As previously stated, the values from the L-Cross Bar are then conveyed to the dynamic logic modules 24. The outputs of the dynamic logic modules 24 are conveyed to output latches 34 after all logic levels have been executed.

Figure 4 depicts a dynamic logic module 24 in accordance with the invention. Dynamic logic modules may be implemented using look-up tables, multiplexer based modules, or a combination thereof. Look-up based modules are preferred in the present invention because they allow the implementation of all combinational functions for a set of input values. Look-up tables may have single or multiple outputs.

Figure 4 depicts a dynamic logic module 24 in the form of a K-input, 2-output LUT. The K-input value may be viewed as the number of address lines for the LUT. Thus, if K equals 3, then the module 24 is implemented in the form of a LUT with 3 address lines. In addition to input values, the level input is required. As previously stated, the level corresponds to the logic level of a circuit which is being implemented. Once the input and level are set up, multiplexer 66, through its select lines, is used to select an appropriate logic bit(s).

The dynamic logic module 24 requires L x 2^K bits of

This Sheet Has Been Replaced



memory per output. "L" corresponds to the upper limit on the number of levels that can be implemented on a given FPLD device. There are many ways in which the level select lines and the K address lines can be combined together. The particular technique for a given implementation of the architecture depends on area usage and memory read time, which depends on the technology being used.

The overall architecture of the invention has now been described. Attention now turns to the construction and operation of this architecture. The construction and operation of the invention will be initially described at a general level. Thereafter, the construction and operation of the invention will be specifically detailed in reference to several examples.

Figure 5 depicts the processing steps associated with implementing a circuit design onto a FPLD architecture of the invention. The first step associated with the process is to receive a circuit description (block 70). Generally, the circuit description will be in the form of a Hardware Description Language, a Register Transfer Language, or a circuit schematic description, all known to those skilled in the art.

The next step associated with the process is to identify the various logic levels within the circuit (block 72). This step is performed in conjunction with known logic synthesis techniques.

Circuits may be divided into a number of logic levels. That is, a set of input signals are initially processed by a first set of circuit components to generate a first level of output values. The first level output values, and perhaps some of the initial input signals, are then passed to a second set of circuit components. This processes is repeated through a number of logic levels. Specific illustrations of this operation are described below.

The next step associated with the process is to

This Sheet Has Been Replaced



create a logic table for each logic level (block 74). In this step, a logic table is created to define the logical operation performed by the logic modules at each logic level.

The next step in the process is to produce logic for the PI/Buffer Controller (block 76). The logic created results in a set of multiplexer select values which are conveyed to the multiplexer 54 at different processing stages (logic levels). For example, when a value from the buffer array 30 at a first logic level is to be passed for processing at a second logic level, an appropriate select signal would be generated to allow the signal to pass through multiplexer 54.

Generating B-Cross controller logic is the next step (block 78) of the process. In this step, logic is generated which will allow the B-Cross bar 58 to route a value which is subsequently buffered during the execution of a logic stage.

The next step of the process is to define L-Cross controller logic (block 80). In this step, logic is defined to allow the L-Cross bar 40 to route a value which is subsequently latched to the dynamic logic core 22.

After the foregoing logic is defined at steps 74, 76, 78 and 80, it is downloaded (block 82) into a configuration memory 99. Once downloaded, the logic may be executed, as will be illustrated below.

Figure 6A depicts an apparatus 90 which may be used to execute the process of Figure 5. The apparatus 90 includes a Central Processing Unit (CPU) 92 which is coupled through a bus 93 to a user interface 94. The user interface 94 is any combination of known computer input and output devices, such as a keyboard, mouse, scanner, monitor, printer, etc. The user interface 94 receives a description of a circuit 96 which is to be implemented in the FPLD of the invention.

The CPU 92 executes a number of programs stored in memory 98. These programs will generally supplement logic

This Sheet Has Been Replaced



synthesizers of the type known in the art.

The CPU 92 executes a logic level identifier 100 (corresponding to step 72), a logic table creator 102 (corresponding to step 74), a PI/Buffer controller producer (corresponding to step 76), a B-Cross controller generator 106 (corresponding to step 78), and an L-Cross controller definer (corresponding to step 80). These modules respectively generate logic levels, logic tables, PI/Buffer logic, B-Cross controller logic, and L-Cross controller logic.

At the direction of the CPU 92, the generated logic is downloaded into non-volatile configuration memory 99 through an appropriate interface 110. Interfaces for storing values in field programmable logic devices are known in the art.

Figure 6B depicts the relationship between the non-volatile configuration memory 99 and the FPLD 20 of the invention. In particular, after non-volatile configuration memory 99 is loaded with the appropriate logic values, it may be used independent of apparatus 90. Specifically, the non-volatile configuration memory 99 is coupled to the FPLD 20. Standard memory download circuitry 101 is used on the FPLD 20 to coordinate the reception of the logic values from the non-volatile configuration memory 99. As depicted in Figure 6B, FPLD 20 receives logic for the dynamic logic core 22, the L-Cross bar controller 42, the PI/Buffer controller 56, and the B-Cross bar controller 60.

Figure 7 depicts the processing steps associated with the operation of the FPLD 20 of the invention. The first step of the process is to read an input value (block 120). The input values are obtained through multiplexer 54 which is controlled by PI/Buffer select controller 56. Thereafter, buffer values (block 122) and logic values (block 124) are routed. The buffer values are routed by the B-Cross bar controller 60 and the logic values are routed by the L-Cross bar controller 42. These operations

This Sheet Has Been Replaced



are generally performed simultaneously, although they are depicted in Figure 7 as being sequential operations.

The next processing step associated with the FPLD of the invention is to execute the logic (block 126), for a given stage or logic level, which is within the dynamic logic module 24. Simultaneously, the logic level is incremented (block 128) through counter 4 and a decision is made to determine whether all logic levels are processed (decision block 130). Steps 126, 128, and 130 of Figure 7 are preferably performed in parallel, although they are depicted as sequential operations in Figure 7 for the sake of illustration. If all logic levels are processed, an output value is generated. Otherwise, processing once again commences at block 120.

The invention has now been fully described. However, to more fully appreciate the invention, attention is directed to a number of examples. Figure 8 is a gate level schematic of an example circuit 140 which will be implemented in accordance with the invention. Three inputs to the circuit ("a","b","c") are conveyed to an OR gate 142 to generate a "G" output. Two inputs ("d","e") are conveyed to an AND gate 144 to generate an "H" output. OR gate 146 receives the "c" input and the "H" input. XOR (Exclusive-Or) gate 148 receives the "G" input and the "H" input. The circuit 140 may be characterized by the following boolean statements:

$$G = a + b + c \qquad (1)$$

$$H = d * e \qquad (2)$$

$$x = G * \bar{H} + \bar{G} + H \qquad (3)$$

$$y = c + H \qquad (4)$$

Figure 8 constitutes a "circuit description" corresponding to step 70 of Figure 5. Equations (1) through (4) provide an alternate (but equivalent) "circuit description". Based on the implementation of block 70 of Figure 5, a particular form of "circuit description" must be supplied. For the purposes of this illustrative example, a schematic or equation format will suffice.

This Sheet Has Been Replaced



Figure 5 indicates that after a circuit description is received (block 70), the next step associated with implementing a circuit into the FPLD of the invention is to identify logic levels (block 72). In the case of the circuit of Figure 8, there are two logic levels. At the first level is an OR gate 142 and an AND gate 144. At the second level is an OR gate 146 and a XOR gate 148. The next step associated with the processing of Figure 5 is to create a logic table for each logic level (block 74). Figure 9 depicts logic tables corresponding to the circuit of Figure 8. At the first level, a logic table is provided for the "G" output and the "H" output. In reference to the logic table for the "G" output, note that there are eight addressable values (2^3), corresponding to the three input values ("a", "b", "c"). The logic associated with the various input values is provided in column "G". That is, for the OR gate 142, the output will be "1" unless all input values are zero. The logic table for the "H" output only has two input values. Therefore, the most significant input bit may be viewed as a "don't care" condition. Consequently, the logic for the first four values is the same as the logic for the next four values. In any event, the "H" output for the AND gate 144 will be "1" only when "d" and "e" are "1".

The second level of Figure 9 includes logic tables for the "x" output and the "y" output. The logic table for the "x" output describes the logic for the XOR gate 148, while the logic table for the "y" output describes the logic for the OR gate 146. Note once again that each table only has two inputs so there is a "don't care" condition, resulting in a repeated output pattern.

In accordance with step 76 of Figure 5, PI/Buffer controller logic may now be generated. In this example, at the first level of the circuit, the PI/Buffer select controller 56 would generate select signals so that all input values ("a", "b", "c", "d", "e") from input latch 32 are passed through multiplexer 54. At the second level of

This Sheet Has Been Replaced



logic, a select signal would be generated to allow the buffered value "c" to pass through the multiplexer 54.

The B-Cross controller logic associated with step 78 of Figure 5 may also be derived at this juncture. In this example, the B-Cross Dynamic Configuration Controller must provide logic to route the value "c" received at the first logic level. No operation is required at the second logic level.

The L-Cross controller logic associated with step 80 of Figure 5 may be defined as follows. At the first logic level, the L-Cross Bar Dynamic configuration controller 42 must route input values "a", "b", "c", "d", and "e". These values are subsequently latched into the dynamic logic core 22. At the second logic level, input values "G", "H", and "c" are stored.

After the foregoing logic is downloaded in accordance with step 82 of Figure 5, execution of the apparatus of the invention may commence, as characterized in Figure 7.

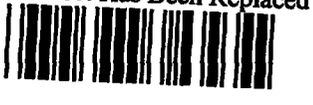
Figures 10 through 13 demonstrate the architecture and operation of the invention in relation to the foregoing example. These figures will be described in the context of the processing steps described in Figure 7.

Figure 10 depicts a FPLD 20A which may be used to implement the circuit of Figure 8. The apparatus 20A generally corresponds with the apparatus of Figure 2. Note that the L-Cross bar 40 and the B-Cross bar 58 are depicted as a grid of crosspoints 150. Figure 10 does not depict a PI/Buffer Select Controller 56. The remaining figures will not depict the L-Cross bar controller 42 or the B-Cross bar controller 60.

Figure 11 depicts the results after the performance of steps 120, 122, and 124 of Figure 7. Recall that some of these steps may be performed in parallel. However, for the purpose of clearly illustrating the invention, the steps will frequently be described as being performed in a sequential manner.

The figure depicts that input values "a", "b", "c",

This Sheet Has Been Replaced



"d", and "e" are latched into input latch 32 (step 120 of Figure 7). The figure also depicts, with a dark circle, that a buffer value "c" is routed through the B-Cross bar 58 (step 122 of Figure 7). The figure also depicts that input values "a", "b", "c", "d" and "e" are routed through L-Cross bar 40 (step 124 of Figure 7).

Figure 12 depicts the results after the performance of steps 126, 128, 130, and 120 of Figure 7. The figure depicts that the values from the L-Cross bar 40 and the B-Cross bar 58 were loaded into latches 28. Thereafter, the L-Cross bar values were conveyed to dynamic logic modules 24A and 24B, corresponding to the "G" logic table and the "H" logic table of level 1. Dynamic logic module 24A yielded value G and dynamic logic module 24B yielded value H (Step 126 of Figure 7). Concurrent with the execution of this logic, the counter 44 (of Figure 3, not shown in Figure 12) increments the logic level (block 128 in Figure 7). In this example, only the first logic level has been processed, therefore one more logic level must be processed.

Since the second logic level is the final logic level, buffer values do not have to be routed (block 122 in Figure 7). On the other hand, logic values must be routed through the L-Cross bar 40 (block 124 in Figure 7). Note that in Figure 12, the "G" output is directly routed from the dynamic logic module 24A through the L-Cross bar at crosspoint 152. Similarly, the "H" output is directly routed from the dynamic logic module 24B through the L-Cross bar at crosspoints 154 and 156. Finally, the buffered value "c" is routed through the L-Cross bar at crosspoint 158.

The L-Cross bar controller 42 enables "writes" to these locations. In defining the L-Cross controller logic (block 80 in Figure 5), it is known, for instance, that at the second logic level, inputs "G" and "H" are required for the "x" logic table which is implemented by dynamic logic module 24A. Therefore, two inputs to the dynamic

This Sheet Has Been Replaced



logic module 24A are enabled to receive the "G" and "H" values. More particularly, rows 160 and 162 are enabled so as to receive data from the output of the dynamic logic modules 24. Implementation options for the cross bar structures of the invention are discussed below.

Figure 13 depicts the processing associated with steps 126, 128 and 130 of Figure 7. Figure 13 shows that the second level input values "G" and "H" are latched in latches 28. These values are then conveyed to logic table "x" which is implemented by dynamic logic module 24A. Second level input values "H" and "c" are similarly latched into logic table "y" which is implemented by dynamic logic module 24B.

The logic modules then execute their respective logic (step 126 in Figure 7) and the counter (not shown) is incremented (step 128 in Figure 7). Since the second level has been processed, the "x" and "y" output values are conveyed to output latch 34.

The apparatus 20A may now process another set of input values (step 120 in Figure 7). Figure 13 depicts a new set of input values at input latch 32. Processing of these values proceeds in the manner previously described.

Figure 14 depicts another circuit 178 which may be implemented in accordance with the invention. Analysis of the circuit 178 reveals that there are four logic levels. The first logic level processes the input signals and does not rely upon output from another level. The first logic level includes XOR gates 180, 182, 184, AND gate 186, and AND gate 188. The second logic level processes the input signals and signals generated at the first logic level. The gates at the second logic level include XOR gate 190, AND gate 192, AND gate 194, OR gate 196 (which is coupled to three AND gates 197A, 197B, 197C), and AND gate 198. The third logic level processes the signals generated at the first logic level and the second logic level; it may also process the input signals, but it does not in this case. The third logic level in this example includes OR

This Sheet Has Been Replaced



gate 200, AND gate 202, and AND gate 204 (which is coupled to OR gate 205). The fourth logic level, the last logic level in this example, produces the output values. The fourth logic level includes AND gate 206, XOR gate 208, and OR gate 210 (which is coupled to AND gate 212). For more complex circuits, Directed Acyclic Graphs, known to those skilled in the art, may be used to simplify the levelization process.

Thus, Figure 14 represents a circuit description (step 70 of Figure 5). The foregoing paragraph identifies the logic levels associated with the circuit (step 72 of Figure 5). Figure 15 depicts logic tables for each level (step 74 of Figure 5). Note that table "HO" at level 1 produces two output values. Output value "O" requires three inputs "b", "c", and "d" (going to XOR gate 184 and AND gate 186). On the other hand, value "H" requires only two inputs "c" and "d", thus a "don't care" state exists for the input value "b". Combinations of this type may be made to form efficient dynamic logic modules 24. Table "KI" in level 2 is another example of an instance in which resources can be shared.

The next processing steps associated with designing an FPLD in accordance with the invention relate to generating the appropriate logic for the various controllers. For the present example, the PI/Buffer controller 56 needs to select all the input values at the first logic level. At the second logic level, the PI/Buffer controller 56 will allow all of the initial input values to pass through the multiplexer(s) 54 for further processing. At the third logic level, the PI/Buffer controller 56 will allow the intermediate signals "F", "H", "O", and the input signal "b" to pass through. At the fourth logic level, the PI/Buffer controller 56 will allow the final stage input values "H", "B", "F", "G", and "K" to pass through for processing.

The B-Cross Bar Dynamic Configuration Controller 60 will utilize four stages of logic to implement the circuit

This Sheet Has Been Replaced



of Figure 14. At the first level of logic the B-Cross controller 60 will enable the B-Cross bar to route all of the input values because they will all be need for subsequent processing. At the second logic level, the B-Cross controller 60 will enable the B-Cross bar to route only the "b" value, which will be needed at the fourth logic level. At the third logic level, the B-Cross controller 60 will enable the B-Cross bar to route the values "F", "H", and "b", which are required for fourth logic level processing. At the final logic level, the B-Cross controller 60 does not have to initiate any routing control.

The L-Cross Bar Dynamic Configuration Controller 42 logic must enable the appropriate input, at various logic levels, for the dynamic logic modules 24. At the first logic level, the L-Cross controller 42 enables the L-Cross Bar 40 to route values "a" and "e" for dynamic logic module 24A; "b", "c", and "d" for dynamic logic module 24B; and "c" and "d" for dynamic logic module 24C. At the second logic level, the L-Cross controller 42 enables the L-Cross Bar 40 to route values "H", "F" and "b" for dynamic logic module 24A; "a" and "e" for dynamic logic module 24B; and "c", "b", and "d" for dynamic logic module 24C. At the third logic level, the L-Cross controller 42 enables the L-Cross Bar 40 to route values "G", "M" and "K" for dynamic logic module 24A; "F", "H" and "I" for dynamic logic module 24B; and "G", "P", and "O" for dynamic logic module 24C. At the fourth logic level, the L-Cross controller 42 enables the L-Cross Bar 40 to route values "J", "N" and "Q" for dynamic logic module 24A; "H", "F" and "b" for dynamic logic module 24B; and "G", "J", and "K" for dynamic logic module 24C.

Returning to Figure 5, appropriate logic for the logic tables, PI/Buffer Controller, B-Cross Controller, and L-Cross Controller has now been described. This logic may now be downloaded to non-volatile memory (block 82).

The processing of input signals associated with the

This Sheet Has Been Replaced



circuit of Figure 14 will now be described with reference to Figure 7 and Figures 16 through 20. Figure 16 depicts processing steps 120, 122 and 124 of Figure 7. In particular, the figure depicts that input values are fed to an input latch 32 (block 120 of Figure 7). The Figure also shows that buffer values are routed through the B-Cross bar 58 at crosspoints 220 through 228 (block 122 of Figure 7). Figure 16 also depicts the logic values routed through the L-Cross bar 40. For instance, note that crosspoint 230 routes value "a" and crosspoint 232 routes value "e". Each of these values is subsequently conveyed to a latch 28 and is then executed by dynamic logic module 24A.

Figure 17 depicts, in a sequential manner, processing steps 126, 120, 122, and 124 associated with Figure 7 (recall that several of these steps may be performed in parallel). The figure depicts latches 28 loading the dynamic logic modules 24 for logic execution (step 126 in Figure 7). The figure also depicts latches 28 loading buffers 31. As shown in Figure 7, the next processing steps are to increment the logic level (block 128) and to query whether all levels have been processed (decision block 130). The logic level is not shown being incremented in Figure 17, but circuitry disclosed for this purpose was disclosed in Figure 3. Since all logic levels have not been processed at this juncture, new input is obtained (block 120 of Figure 7). Thus, at this point, as previously stated, the PI/Buffer Controller 56 would allow all of the buffered values to pass through multiplexers 54. Then, appropriate buffer values would be loaded in the B-Cross Bar 58 (block 122 of Figure 7). Figure 17 depicts that at crosspoint 234 the value "b" is routed, which is the only value at this processing juncture which must be subsequently utilized.

Finally, Figure 17 depicts loaded logic values for the second logic level (block 124 of Figure 7). For example, note that at crosspoint 236 the output value "F"

This Sheet Has Been Replaced



is loaded into the L-Cross bar 40. This value was obtained directly from the dynamic logic module 24A. The crosspoint 238 routes the output value "H", which was also received directly from the dynamic logic module 24A. The crosspoint 238 routes the value "b", which was a previously buffered value.

Figure 18 depicts the continued processing associated with the circuit of Figure 14. The Figure depicts the second level logic values loaded into latches 28. The figure also depicts the outputs from the execution of the logic. The figure also shows how these outputs are reprocessed in accordance with the invention. For example, note that output "G" is conveyed through crosspoints 250 and 252 so that they may subsequently serve as inputs to dynamic logic modules 24A and 24C during execution of the third logic level. Figure 18 also depicts that values "F", "O", and "B" are routed at crosspoints 254, 256, and 258 in B-Cross bar 58 for subsequent processing (values "F" and "O" are used at logic level 3 and value "b" is used at logic level 4). Finally, Figure 18 depicts that previously buffered values are routed through the L-Cross bar 40. For example, crosspoint 260 routes the previously buffered value "O" and crosspoint 262 routes the previously buffered value "P". Both of these values serve as inputs to dynamic logic module 24C during logic level 3.

Figure 19 depicts the third logic level processing associated with the invention. The third logic level input values are shown as being in latches 28. The third logic level output values are shown as output signals of the dynamic logic modules 24. The figure depicts the logic values "N", "J" and "Q" having been conveyed through the L-Cross bar 40. Logic value "N" is conveyed through crosspoint 280, logic value "J" is conveyed through crosspoints 282 and 284, and logic value "Q" is conveyed through crosspoint 286. The remaining values in the L-Cross bar 40 were obtained from buffers 31. Buffer value

This Sheet Has Been Replaced



"F" is conveyed through crossbar 290, value "G" is conveyed through crossbar 292, value "H" is conveyed through crossbar 294, value "K" is conveyed through crossbar 296, and value "b" is conveyed through crossbar 298. Note that there are no values in the B-Cross bar 58 because the final logic level is about to be executed.

Turning now to Figure 20, the fourth level logic inputs are shown in latches 28. Figure 20 also shows dynamic logic module outputs "x", "y", and "z". Since this is the last logic level, the values are conveyed to the output latch 34. The figure also depicts the next stage of input entering the circuit 20B. This configuration is the same as that of Figure 16, which has been previously discussed.

Attention now turns to some implementation considerations. Assume that a given FPLD has L levels and C dynamic logic modules 24, each having m outputs. Assume that the logic network being implemented is a feasible network (i.e., each node in the network has <= K inputs and C <= m outputs). If not, known logic synthesis techniques can be used to transform it into one. Let the combinational part of the feasible network be topologically levelized. Let ρ be the number of levels, and Γ be the maximum number of modules in any level for the circuit. If $\rho <= L$ and $\Gamma <= C$, (e.g. Figure 9, $\rho=2$, $\Gamma=2$, $L=4$, $C=3$; Figure 15, $\rho=4$, $\Gamma=3$, $L=4$, $C=3$) then the mapping is straightforward, with one level of the circuit being evaluated in every internal cycle. If $\rho > L$ or $\Gamma > C$, then it may be possible to implement the circuit by grouping together modules across several levels. Each group can then be realized in more than one internal cycle, before switching over to the next group.

When a design does not fit on a single chip, then the design must be split across several chips. If the partitions are independent (i.e., they do not share any common signals), then the split does not pose any problems.

This Sheet Has Been Replaced



Consider the case where there are ρ ($\rho > L$) levels of combinational logic, with C modules per level. In this case, one requires ρ/L chips, and a mechanism to synchronize their level counting circuitry. Figure 3 shows two signals, "To Next Chip" and "From Previous Chip". These signals are connected in daisy chain fashion, so that chip C_i connected after chip C_{i-1} begins its level counter only after chip C_{i-1} has reached its maximum level. If the design is such that $\Gamma > C$, then splitting across chips could necessitate duplication of logic.

As stated above, the dynamic logic module 24 may be any re-configurable logic structure which can be repetitively changed to implement different functions. Other possible logic module implementations include: (1) a K-input, m-output look-up table, where K and m are integers, (2) interconnection of many look-up tables, each with 1 or more outputs, and having either varying or same number of inputs, (3) multiplexer based logic modules, (4) two level AND-OR logic plane, and variations thereof, with the AND and OR inputs controlled by programmable read/write memory bits, and (5) heterogenous dynamic logic modules employing a combination of logic implementations.

The dynamic interconnection array 26 also has a large number of variations possible, both in terms of the overall architecture, and the actual implementation of the architecture. The following description details alternate implementations.

Each crosspoint of the crossbar may be implemented with a transistor. For example, the source of a transistor may be coupled to an output line from the buffer array 30 or from the logic core 22. The drain of the transistor may be coupled to an appropriate dynamic logic module 24. The L-Cross bar controller 42 or the B-Cross bar controller 60 may then be used to provide a gate signal when a signal is to be conveyed to a given dynamic logic module.

This Sheet Has Been Replaced



Another approach for implementing the cross bar structures of the invention is to use shift registers. Each crosspoint of the crossbar has L bits of memory, a bit being used per level. These bits could be arranged in the manner of a shift register so that the change of level would just mean shifting one bit. The disadvantage in such an implementation would be that the entire L bits will have to shift through before returning to the first level, and hence only circuits with $\frac{1}{L}$ levels, where $\frac{1}{L}$ is a perfect divisor of L can be implemented. A better design is to have L rows of memory, each row consisting of memory on all the crosspoints, and to choose the row in use by means of the level counter.

The cross bar structures may also be implemented using multiplexers. For example, a multiplexer would be used for each row of the crossbar. The inputs to the multiplexer would be the signals from the buffer array 30 and the signals from the dynamic logic core 22. The outputs from each multiplexer would be coupled to a dynamic logic module 22. The L-Cross controller 42 or B-Cross controller 60 would be used to generate appropriate select signals for each multiplexer.

Since there are (MC + B)KC crosspoints in the L-crossbar, (MC + B)KCL memory bits are required. However, each of the K-inputs of a logic module are identical, and it does not matter to which one a signal is assigned. In other words, the (MC + B) signals need to connect to only one of the K inputs per dynamic logic module 24. This means that the K bits can be reduced to $\log_2 K$ bits per signal, reducing the total bits to $\log_2 KCL(MC+B)$. However, this would warrant a decoder at each level. It is preferable to recognize that at each dynamic logic module 24 input a (MC + B)-to-K multiplexer is required, not a (MC + B)-to-K crossbar. A (MC + B)-to-1 multiplexer requires $\log_2(MC + B)$ select lines and K multiplexers. Each select line needs a bit per level, therefore one requires $KCL\log_2(MC + B)$ bits and KC(MC + B)-to-1 decoders.

This Sheet Has Been Replaced



This would be more area efficient if $KCL\log_2(MC + B) + KC(MC + B)\delta < KCL(MC + B)$, where δ is the bit-equivalent of a decoder and gate. Since the dynamic logic modules 24 and interconnections both require decoders, there could be a sharing of decoders. This would require that extra latches be placed at the dynamic logic module outputs.

A further decrease in the number of crosspoints can be achieved through hardwired connections. Each dynamic logic module can have a hard-wired connection to K dynamic logic modules, and each dynamic logic modules will have inputs coming from K dynamic logic modules, preferably those that are adjacent to it. This would imply that the crossbar (or multiplexer, as the case may be) would then be reduced to a $(MC - K + B)KC$ crosspoints, and there would be KC additional 2-input multiplexers (one at each input), with associated KCL select bits. This interconnection scheme would decrease the number of fanout points from the dynamic logic module output to $(MC - K)K + K$, instead of the earlier KC.

A different type of interconnection strategy can be adopted by recognizing that all dynamic logic module inputs are equivalent. Hence, the interconnection switch at the inputs is, by circuit switching terminology, a $(MC + B)$-to-K concentrator. In other words, one must select any K of the $(MC + B)$ signals. This can be achieved by using a two-level sparsely connected crossbar, known in the art as a binomial concentrator. The chief advantage of using such an interconnection strategy is that all the number of crosspoints are reduced and as a result the number of bits used for storage are reduced. The disadvantage is that the signals now have to pass through two crosspoints in series. This could increase the interconnect delay.

An alternate way is to treat the routing as a "copy" function. For example, consider one DLM input. It can be connected to any one of the $(MC + B)$ signals coming from the previous level. That is, its value should be set to

This Sheet Has Been Replaced

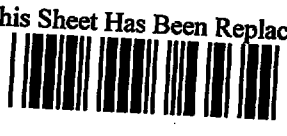


the value of one of the (MC + B) signals. This is equivalent to a memory read operation, where the memory consists of (MC + B) bits, and one of them has to be read onto the switch output. Therefore, the selection can be performed by using precharge circuitry and sense amplifiers. The (MC + B) signals correspond to one column of memory, and one of the cell select transistors is chosen by means of a decoder or local memory bit. There are both area and speed advantages in using this style of implementation. The cell select transistors are of minimum size as opposed to the large pass transistors that would have been required if a direct connection between switch input and output were required. And using the sense amp, the read operation can be performed fast. Further, the precharge time of the memory can be interleaved with the dynamic logic module look-up time, thus shortening the interconnect delay even further. This would mean that the interconnect delay would be of the same magnitude as the logic module delay, and could be even smaller.

In another embodiment, one can assume that there are C dynamic logic modules 24, each with K inputs and 1 output. If the dynamic logic modules are divided into K classes, with C/K modules per class. A module in class j, j = 1 ... K, is connected to the jth input of each dynamic logic module (instead of to every input). This will decrease the number of crosspoints and memory bit by a factor of K. It is possible to combine one or more of the foregoing approaches to develop a hybrid dynamic interconnection array 26.

In sum, it will be appreciated that the an improved architecture for a field programmable logic device has been disclosed. The dynamically changing logic core and interconnection array of the invention facilitates the sharing of silicon resources, while simplifying the circuit implementation process since placement and routing of the circuit is straightforward. Another benefit

This Sheet Has Been Replaced



associated with the invention is the improved timing performance and the associated benefit of being able to accurately predict timing performance.

The foregoing descriptions of specific embodiments of the present invention are presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, obviously many modifications and variations are possible in view of the above teachings. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following Claims and their equivalents.

This Sheet Has Been Replaced



IN THE CLAIMS:

1. A field programmable logic device, comprising:

a dynamic logic core including

means for executing a first logical operation on a set of input signals to generate a plurality of internal signals, and

means for performing a second logical operation on a set of selected internal signals, said first logical operation corresponding to a first logic level of a circuit, said second logical operation corresponding to a second logic level of a circuit; and

a dynamic interconnection array coupled to said dynamic logic core, said dynamic interconnection array including

means for identifying said selected internal signals from said plurality of internal signals, and

means for routing said selected internal signals to said dynamic logic core.

2. The field programmable logic device of claim 1 wherein said dynamic interconnection array further comprises:

means for buffering a set of said input signals to generate a corresponding set of buffered signals, said buffered signals forming a portion of said internal signals.

3. The field programmable logic device of claim 2 wherein said buffering means includes a buffer crossbar interconnect structure for buffering said input signals.

4. The field programmable logic device of claim 3 wherein said buffering means includes a buffer crossbar controller coupled to said buffer crossbar interconnect structure, said buffer crossbar controller executing a set of logical commands to select said buffered signals.

This Sheet Has Been Replaced

5. The field programmable logic device of claim 2 further comprising means for directing said input signals and said buffered signals to said dynamic interconnection array.

6. The field programmable logic device of claim 5 wherein said directing means includes an input/buffer controller for selecting said input signals and said buffered signals.

7. The field programmable logic device of claim 1 wherein said routing means of said dynamic interconnection array includes a logic crossbar structure for routing said selected internal signals.

8. The field programmable logic device of claim 8 wherein said identifying means of said dynamic interconnection array includes a logic crossbar controller which executes a set of logical commands to identify said selected internal signals.

9. The field programmable logic device of claim 1 wherein said dynamic logic core is a look-up table, said first logical operation corresponding to a first set of values in said look-up table and said second logical operation corresponding to a second set of values in said look-up table.

10. The field programmable logic device of claim 1 wherein dynamic logic core includes a counter for selecting said first logical operation and said second logical operation.

11. A method of implementing a circuit on a field programmable logic device, said method comprising the steps of:

This Sheet Has Been Replaced

(A) receiving a set of input signals;

(B) buffering selected signals from said input signals when said selected signals are required at a subsequent logic level, said buffering step generating buffered signals;

(C) executing a primary set of logic on said input signals to generate a set of internal signals;

(D) selecting a set of intermediate signals from said buffered signals and said internal signals;

(E) buffering designated signals from said intermediate signals when said designated signals are required at a subsequent logic level;

(E) performing a preselected set of logic on said intermediate signals to generate a new set of internal signals; and

(F) repeating steps (D) through (E) until a predetermined set of logic is executed, said predetermined set of logic corresponding to a plurality of logic levels in a circuit.

12. The method of claim 11 wherein said preselected set of logic of said performing set is designated by defining a logic level within said plurality of logic levels.

13. The method of claim 12 wherein said logic level of said defining step is incremented after each performing step.

14. A method of designing a field programmable logic device, said method comprising the steps of:

receiving a description of a circuit to be implemented in said field programmable logic device;

identifying logic levels within said circuit;

creating logic tables to execute the logic corresponding to each of said logic levels within said circuit;

producing input/buffer logic to select input values

This Sheet Has Been Replaced

and buffered values to be applied to said logic tables;

generating buffer controller logic to enable the storage of buffered values at a first logic level which must be used at a subsequent logic level; and

defining logic controller logic to enable the storage of input values and buffered values which are subsequently applied to said logic tables.

15. The method of claim 14 further comprising the step of downloading said input/buffer logic, said buffer controller logic, and said logic controller logic to a non-volatile memory device.

16. The method of claim 15 further comprising the step of coupling said non-volatile memory device to a field programmable logic device including an input/buffer controller, a buffer controller, and a logic controller.

This Sheet Has Been Replaced