A New Look at Hardware Maze Routing

John A. Nestor
ECE Department
Lafayette College
Easton, Pennsylvania 18042
(610) 330-5411
nestorj@lafayette.edu

ABSTRACT

This paper describes a new design for a hardware accelerator to support grid-based Maze Routing. Based on the direct mapped approach of Breuer and Shamsa [3], this work refines their design to substantially reduce the hardware requirements of each processing element while at the same time adding support for mulitilayer routing and fast iterative routing. An RTL implementation has been developed for this design in VHDL, and initial results show promise for its realization using ASIC, custom, or FPGA technology.

1. INTRODUCTION

Wire routing is a key problem in electronic design automation. While routing has been studied extensively over the years and many different algorithms have been explored, the classic grid-based Lee algorithm for Maze Routing [1] remains popular as a basic ingredient of many approaches.

Figure 1 shows the basic operation of the Lee Algorithm, as modified by Akers [2]. The routing surface is represented as a grid, where each gridpoint is connected to adjacent gridpoints to the north, south, east, and west. Given source "S" and target "T" gridpoints, the goal of the Lee Algorithm is to find the shortest existing connection between these points, while avoiding any obstacles. It operates in two phases, commonly known as *expansion* and *traceback*.

During the *expansion* phase, the algorithm searches outward from the source terminal in a breadth-first fashion, labeling each visited node with a *direction* that indicates the direction of the shortest path back to the source. Each time a node is labeled, its unlabeled neighbors are placed at the end of a queue for later processing, so that the search expands outward until the target is reached. The exhaustive nature of this search guarantees that a shortest path will be found if any path exists, but is computationally expensive - O(d^2) for a connection of distance d.

When expansion is complete, each node between the source and target is labeled with the direction of the shortest path from that node to the source. The *traceback* phase simply follows these labeled nodes from the target to the source to find a shortest-path connection. This requires exactly d steps for a connection of distance d.

The ability of the Lee Algorithm to find the shortest path connection makes it very attractive in spite of its computational cost. It remains popular in grid-based routers, often as a "fall-back" that is applied when faster methods fail.

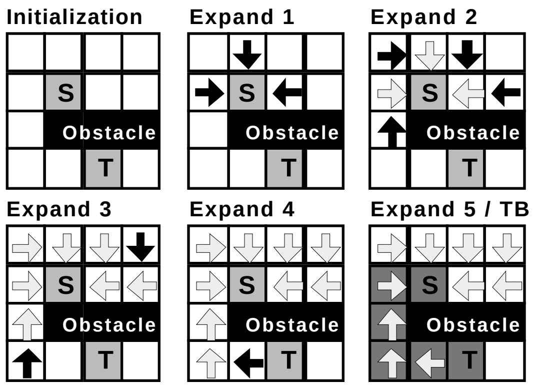


Figure 1 – The Lee Algorithm

Because of the Lee Algorithm's popularity, there have been several attempts to use hardware to reduce its time complexity. In *grid-based* hardware accelerators, an array of addressable processing elements directly represents the gridpoints and performs routing in a single-instruction multiple-data (SIMD) fashion. Grid-based accelerators can be characterized as *Direct Grid* accelerators, in which each processing element is mapped to a single gridpoint, and *Virtual Grid* accelerators, in which each processing element is mapped to several gridpoints.

The first proposal for a Direct Grid accelerator was Breuer and Shamsa's L-Machine [3]. Each processing element in the L-Machine, known as an L-Cell, is a small FSM attached to a labeling register. Neighboring processing elements are connected, allowing expansion and traceback to be performed by the elements in SIMD mode. Decoder hardware attached to the array allows individual processing elements to be addressed, initialized, and queried. Similarly, encoder hardware allows processing elements to identify themselves to an attached control unit by their address when a significant event occurs (for example, when the target is reached). The key advantage of the L-Machine is that it reduces the time complexity of the Lee Algorithm from O(d^2) to O(d). Its obvious disadvantage is its hardware requirement of N^2 processing elements for an N X N grid. Even worse, supporting multiple layers in a straightforward way requires N^2 processing elements for each layer, which quickly becomes impractical.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GLSVLSI'02, April 18-19, 2002, New York, New York, USA.

Copyright 2002 ACM 1-58113-462-2/02/0004…$5.00.

Virtual Grid [4-6] routing accelerators use a smaller number of processing elements by mapping more than one grid point to each processing element. This reduces the size of the array and allows support for multiple layers but substantially increases the complexity of the processing elements, since they must now keep track of multiple gridpoints and the mappings of adjacent gridpoints.

This paper proposes a return to the Direct Grid approach for routing accelerators. A new processing element design has been developed that focuses primarily on the expansion phase, producing a simpler implementation. When combined with recent advances in integrated circuit technology, the construction of large arrays of processing elements becomes much more feasible. An extension to this design supports multiple layers by time-multiplexing a single array of processing elements over each layer. This trades off routing time to save area, but does so in the smallest dimension of the routing problem. Additional extensions support rapid initialization of multiple processing elements in parallel, which can be used to support iterative routing schemes that are popular in many modern routers.

The feasibility of this approach is demonstrated through implementations of the processing element design using a standard-cell ASIC gate library and an FPGA. The initial success demonstrated here favors further research to develop more detailed designs and to evaluate the performance of these designs; this work is ongoing.

This paper is organized as follows. Section 2 describes the general approach of the new accelerator design. Section 3 describes the specific design of a single-layer accelerator, while Section 4 describes the extension of this approach to support multi-layer routing. Section 5 presents the current results of this project, and Section 6 offers conclusions and directions for future research.

2. THE GENERAL APPROACH

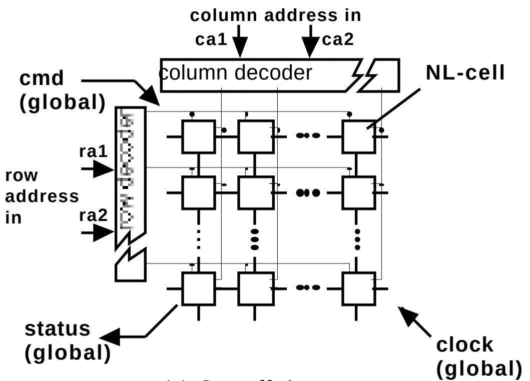
To design an effective hardware accelerator for maze routing, it is important to understand how it is used in routing tools.

First, these routing tools must wire a large number of connections. Since the Lee algorithm can only make one connection at a time, it is used *iteratively* – starting with an empty or partially complete routing surface, the algorithm is applied to successive nets until all nets are routed. This brings up a related issue: while the Lee algorithm is guaranteed to find a shortest path connection for a single net, the connections placed on the routing surface for each net form obstacles for later nets. These obstacles may make the routing of later nets longer than optimal, and sometimes impossible to complete. This requires the use of a "rip-up-and-reroute" [7] strategy in which the connections for some nets are removed and re-routed in a different order in an attempt to improve the routing.

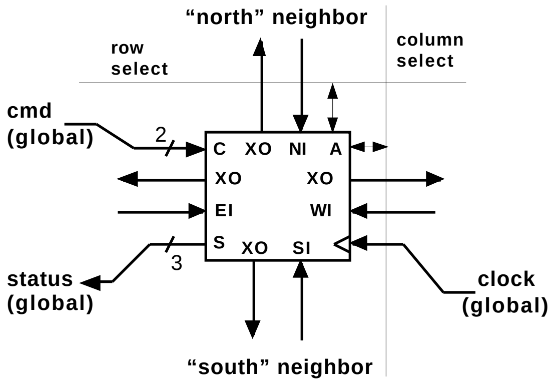
Second, modern design technologies require the use of several layers of interconnect. Current VLSI processes provide up to 6 layers, while multilayer printed circuit boards may use more than 30 layers. Thus to be effective, any attempt to accelerate maze routing must accommodate multiple layers efficiently.

To meet these needs, we propose a new architecture for a hardware routing accelerator. It is based on Breuer and Shamsa's L-Machine, with some key modifications that reduce its implementation cost, support multi-layer routing and speed up rip-up-and-reroute. To acknowledge its legacy, we refer to our design as the "New L-Machine" or NL-Machine.

Figure 2 shows the general organization of the NL-Machine. Like the L-Machine, it consists of an array of processing elements (called "NL-Cells") and an attached control unit (not shown). Each NL-Cell is a finite state machine that represents the status of the gridpoint during routing. Local connections between adjacent cells allow the expansion process to proceed in parallel, reducing the worst-case performance of expansion from O(d^2) to O(d). The control unit communicates with NL-cells using global signals CMD and STATUS. The CMD signal broadcasts commands from the control unit to all cells in parallel. Some commands are SIMD commands that specify an action to be performed by all cells, while others apply only to cells selected using row select and column select lines that are attached to decoders on the periphery of the array. The STATUS signal connects all cells to the control unit via a tristate/wired-NOR bus.



(a) Overall Array



(b) NL-Cell Detail

Figure 2 – NL-Machine Organization

While based on the L-Machine, the NL-Machine has several key differences:

a. *Reduced complexity of Processing Element*. In the original L-Machine, the L-Cell processing element supported both the expansion and traceback phases in SIMD mode. This has great advantage for the expansion phase, since it reduces execution time from O(d^2) to O(d). However, there is no similar advantage for the traceback phase, which requires d steps whether it is done by the processing elements or the attached control unit. Moving this responsibility to the control unit significantly reduces the complexity of the NL-Cell, which is replicated

N^2 times in an N X N array, while increasing the cost of the single control unit. Additional changes further reduce the complexity of the NL-Cell by reducing the number of states in the FSM.

b. *Reduced complexity of supporting hardware*. In the original L-Machine, cell outputs were connected to both row and column *decoders* and row and column *encoders*. The decoders allowed the control unit to address and initialize individual cells, while the encoders allowed individual cells to indicate their address to the control unit during expansion (when the target was reached), and during traceback (to indicate the path of the connection). In contrast, the NL-Machine uses only row and column decoders. The tasks requiring the encoders in the original L-machine are performed instead by addressing and polling cells using the decoders and the STATUS signal.

c. *Support for multiple nets, iterative routing, and "rip-up-and-reroute"*. Because most modern routers repeatedly apply the Lee algorithm for successive nets, our design provides support to rapidly initialize multiple processing elements in parallel. For example, during a "rip-up" step several cells must be initialized to reflect the removal of a net. In the original L-machine, this step would require repeated individual accesses to initialize each cell assigned to that net. The NL-Machine uses a modified decoder design that can select a range of rows or columns, as shown in Figure 3. Each modified decoder accepts an upper and lower address, and selects all rows or columns in this range. Combining row and column decoders allows the selection of an arbitrary rectangular region of cells, as shown in Figure 4. Using this approach, ripping up a net can be reduced to one operation for each horizontal segment in the connection, reducing the time of this common operation.

d. *Cost-effective support for multiple layers*. Multiple layers are essential in modern routers. The straightforward approach to supporting multiple layers in the L-Machine is to extend array cells to connect in the vertical as well as horizontal directions and add an additional array of cells for each layer. Thus supporting L layers of an $N \times N$ grid would require $N \times N \times L$ processing elements, each with connections between adjacent cells. Such an approach is not feasible even with today's technology. Instead, the NL-Machine time-multiplexes each NL-Cell over a number of layers, storing the state of each layer stored in a shift register that circulates the data. While this increases the execution time by a factor of L, we believe that this is a reasonable tradeoff given that the major direction of expansion will be in the horizontal direction. More detail about this feature is given in Section 4.

The next two sections discuss the detailed operation of the NL-Machine for the single-layer and multiple-layer cases, respectively.

3. THE SINGLE-LAYER NL-MACHINE

The processing element of the NL-Machine is called the NL-Cell and is analogous to the L-Cell processing element in the original L-Machine. Like the L-Cell, it performs expansion by communicating with neighboring cells in SIMD mode. However, there are a number of differences in the NL-CELL design which significantly reduce its implementation cost. As mentioned earlier, the NL-Cell does not directly participate in the traceback phase, which reduces the cost significantly. In addition, storage elements are reduced by eliminating states, using a binary encoding when storing the traceback direction, and combining state variable storage with traceback direction storage. Finally, the single bidirectional connection between adjacent cells has been replaced by two connections: an output connection labeled "XO" that is true when the cell is in an expanded state, and an input connection that is true when the neighboring cell is in an expanded state.

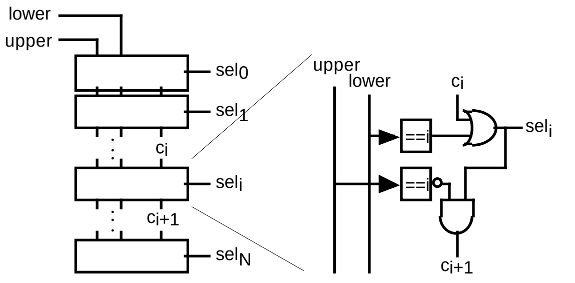


Figure 3 – Extended Decoder Design

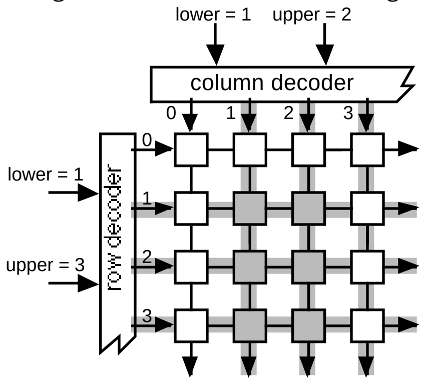


Figure 4 – Selecting Multiple Cells with Extended Decoder

The resulting design is a FSM with 6 states: B (blank), BL (blocked), expanded east (XE), expanded west (XW), expanded north (XN) and expanded south (XS). The function of the NL-Cell is controlled by four commands: CLEAR, SET, EXPAND, and TRACE along with row and column select inputs RSEL and CSEL, as summarized in Table 1.

To initialize the array of NL-cells for routing, the control unit asserts the CLEAR command, which operates in two modes. In SIMD mode, all cells in an expanded state (XE, XW, XN, or XS) are set to the blank state B. This has the effect of clearing the expanded cells while leaving cells in the blocked (BL) state unchanged. This is used to prepare to route a new net while preserving existing obstacles and previously routed connections. In the second mode, the CLEAR command sets all selected cells (i.e., cells where RSEL=CSEL=1) to the blank state B, regardless of their current state. This is used to remove obstacles and rip-up previous connections and can be performed on several cells in parallel using the modified decoder design.

Next, the control unit specifies source terminals for a desired connection using the SET command, which sets selected cells to an arbitrary expand state (currently XE). Note that the XE state is used instead of an explicit "Source" state as in the original L-machine to reduce the cell's implementation cost, since the control unit keeps track of the source location anyway. The SET command has no effect when cells are not selected, allowing it to double as a "no-op" in SIMD mode.

The control unit then starts the expansion process using the EXPAND command in SIMD mode over several successive clock cycles. During each clock cycle a cell in the blank (B) state enters an expansion state if an adjacent cell is asserting inputs EI, WI, NI, or SI. The direction from which expansion occurs is encoded into each of the four states XE, XW,XN, or XS. Each of these states is assigned a 3-bit state code with a 1 in the most significant bit and a 2-bit direction code in the two least significant bits.

When a cell is entering an expansion state, it asserts status bit S1, which is connected to a wired-NOR status bus, as in the original L-Machine. This provides a "watchdog" function since it indicates that expansion has terminated when no nodes "pull down" S1. A cell entering an expansion state will also assert status bit S0 when it is selected by the RSEL and CSEL inputs. This is used by the control unit to address and poll the status of the cell at the target of the connection - when S0 is asserted, this signals the control unit that the target has been found and that the expansion phase has completed successfully.

Table 1 – NL-Cell State Sequencing

CMD	**RSEL * CSEL**	**NI**	**SI**	**WI**	**EI**	**PS**	**NS**	**S1**	**S0**
CLEAR	0	–	–	–	–	XN + XS + XE + XW	B	-	-
CLEAR	1	–	–	–	–	–	B	-	-
SET	1	–	–	–	–	–	XE	-	-
TRACE	1	–	–	–	–	–	BL	D1	D0
EXPD.	0	1	–	–	–	B	XN	1	0
EXPD.	1	1	–	–	–	B	XN	1	1
EXPD.	0	0	1	–	–	B	XS	1	0
EXPD.	1	0	1	–	–	B	XS	1	1
EXPD.	0	0	0	1	–	B	XW	1	0
EXPD.	1	0	0	1	–	B	XW	1	1
EXPD.	0	0	0	0	1	B	XE	1	0
EXPD.	1	0	0	0	1	B	XE	1	1

The traceback phase is performed explicitly by the control unit. It starts by addressing the target cell and applying the TRACE command. This has two effects: (1) it returns the encoded traceback direction to the control unit via status signals S1 and S0, and (2) it sets the target cell to the BL (blocked) state to indicate that it has been used for a connection. The control unit then uses the traceback direction from the status signal to select an adjacent cell that is closer to the source, and again applies the TRACE command to repeat the process. This continues until source node is reached, and a full connection has been specified. While this process requires the explicit cooperation of the control unit, it completes in the same number of steps as the original L-machine. However, it accomplishes this with significantly less hardware since the traceback hardware is not duplicated in each cell.

The TRACE command is also used during the initialization phase to mark cells as obstacles by placing them in the BL (blocked) state. In this case, the direction code asserted on signals S1 and S0 is ignored. This may be applied to arbitrary rectangular regions of cells using the extended decoder described previously, which speeds up the initialization process.

Multiple-point nets are routed using a procedure similar to that in the L-machine, where an initial pair of points is chosen as source and target and connected using the above procedure. Next, all points on the path between these point are set to be the source of a new search using the SET command. This will take multiple clock cycles since each straight-line segment must be initialized separately. Next, another point on the net is selected as a target, and expansion proceeds as before. This process is repeated until all points in the net are connected together.

4. THE MULTI-LAYER NL-MACHINE

Full-grid hardware routers like the L-machine are normally limited to a single layer of interconnect. Since current IC processes use up to 7 layers of interconnect and printed circuit boards use more than 30 layers, this limitation is unacceptable for practical applications.

Extending a full-grid hardware router to multiple layers is not difficult in concept. Each routing cell can be easily modified to expand in vertical as well as horizontal directions. For example, the cell design described above can be extended with two additional expansion inputs for vertical directions (up and down) and additional "layer select" input so that each layer can be selected separately, and two additional expansion states. However, since the N X N array must be duplicated for each layer this approach is impractical.

In this section, we propose a different approach which is based on the observation that although the number of layers supported in modern manufacturing processes is growing, this number remains much smaller than the dimensions of the horizontal routing grid. Moreover, local communication between vertical layers is limited to two directions, rather than four between the horizontal gridpoints. This suggests an approach where the single layer gridpoint processor is extended to process points on more than one layer using time-division multiplexing.

Figure 5 shows the architecture of a single processing element which implements this idea. Each of these NLM-cells implements routing for one horizontal position on multiple layers. It accomplishes this by storing state information for each layer in a 3-bit wide shift register which circulates information to a shared state sequencer. The shift register has a number of stages that is equal to the number of layers supported by the router. The bottom stage of the shift register is attached to a state sequencer which determines the next state of that layer. The states in the state sequencer are identical to those in the NL Cell except that two additional expansion states XU and XD are added to represent the "up" and "down" traceback directions, respectively. During each clock cycle, the new state is loaded into the top stage of the shift register, while the additional layers are shifted down one position, so that during the next clock cycle the layer above the previous layer is processed.

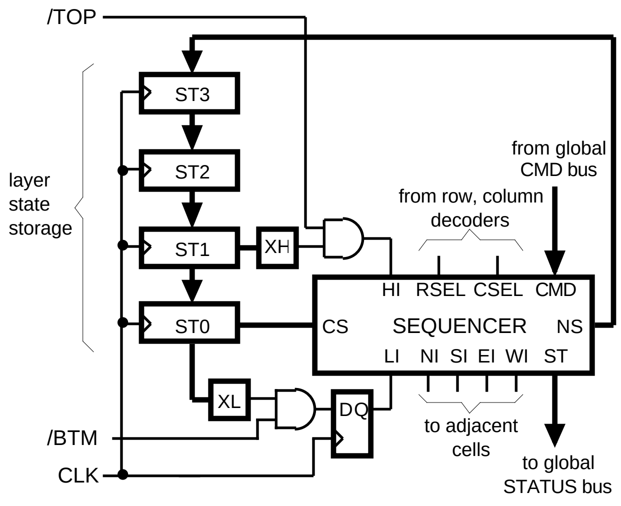


Figure 5 – Multilayer NLM-Cell Design

As in the previous cell design, the next state depends on the value of the CMD input from the control unit, the RSEL and CSEL inputs that specify when a cell is selected, and the NI, SI, EI, and WI inputs from horizontally adjacent cells. In addition, the next state depends on inputs that indicate that vertically adjacent cells are in the expansion state, shown as inputs HI and LI. Input HI is true when the cell above the current is in an expansion state. For the layer above the layer being processed, this condition is detected by the logic block labeled XH, which is true when the state in that layer is an expansion state. The output of the XH block is ANDed with external signal /TOP, an active low signal that indicates that the top layer is being processed when it is asserted low. Thus when /TOP is a logic low, the XH input will always be zero, since there is no layer above the top layer. The output of the XL block determines whether the cell being processed by the state sequencer is in an expansion state. It is ANDed with an external signal /BOT which asserted low when the bottom layer is being processed – this indicates that the bottom layer is being processed and so the LI input should be zero. The result of this AND is stored in a flip-flop for use during the next clock cycle, when processing the layer “above” the current layer.

Layers are processed one at a time during the expansion phase. Extra complexity is added to the control unit, since it must generate the /TOP and /BOTTOM signals. In addition, it must keep track of which layer is currently being processed when it applies commands to change the status of specific cells.

The execution time of the resulting design during the expansion phase will be O(L*d), where L is the number of layers in the router and d is the distance between the source and target, including vertical distance. The traceback phase also requires O(L*d) steps, since the shift register processes one layer at a time. Traceback time can be further reduced by modifying the shift register to “freeze” at a particular level. This would allow traceback to proceed in a horizontal direction without waiting for L clock cycles between steps. Freezing can be accomplished by adding 2-1 multiplexers to the input of each shift register stage, so that data recirculates in a “hold” mode, or by adding a single 2-1 multiplexer to the stage being processed and gating the clock of the other shift register stages.

5. RESULTS

The single-layer (NL) and multilayer (NLM) architectures have been designed as a collection of parameterized VHDL files which can be synthesized to different array sizes and implementation technologies. In this section we discuss our preliminary results implementing the NL-Cell and NLM-Cell designs using both an ASIC standard-cell library and an FPGA.

5.1 Gate-Level Implementation

The gate-level implementation is useful because it provides a comparison to the original L-Machine and also gives insight into the practicality of implementing the NL-Machine as a semi-custom or custom chip. Gate-level implementations were generated for both the NL-Cell and NLM-Cell (with 4 layers) using the Synopsys Design Compiler and the MSU SCMOS Standard Cell Library (chosen because it is non-proprietary). A schematic diagram of the synthesized NLM-Cell is shown in Figure 6.

The first two rows of Table 2 summarize the gate count and flip-flop count of the NL-Cell design and the original L-Cell presented in [3]. Since the original L-Cell was implemented without AND/OR/INVERT (AOI) gates, the rightmost column shows the gate count with AOI gates converted to simple gates. This allows a rough comparison showing that the NL-Cell required 36% fewer gates and 57% fewer flip-flops – a significant reduction in cost.

The second two rows show the implementation costs of the NLM-CELL synthesized for four layers and the general cost for N layers. Note that the gate count stays constant as layers are added, with only the number of flip-flops in the shift register increasing.

Table 2. Gate-Level Implementation

Design	FFs	Gates w/AOI	Smpl. Gates
NL CELL	3	37	48
L Cell [2]	7	-	75
NLM CELL (4 layer)	13	45	53
NLM CELL (N-layer)	3*N + 1	45	53

Designing a full ASIC or custom implementation of a routing accelerator is a major task. To justify whether such an effort would be worthwhile, some simple estimates can be used to predict the size of a large routing array implemented in a modern deep-submicron technology. For example, ASIC vendors currently claim gate densities of 75-100K gates/mm^2 in 0.18µm technology standard cell libraries. Assuming that each flip-flop has an area equivalent to four logic gates and that gate density is 75K gates/mm^2, then the density of a 6-layer NLM-cell array can be predicted at 619 cells/mm^2. Thus a 100 X 100 array of NLM-cells could be implemented in a chip that is approximately 4mm X 4mm. Area could be reduced much further using a custom layout.

5.2 FPGA Implementation

The availability of large FPGA devices and their low nonrecurring engineering cost makes them attractive as a possible implementation technology for the NL-Machine. Moreover, the grid-based structure of the NL-Machine maps well into the structure of many FPGA families.

Table 3 shows results of FPGA implementations for two Xilinx [8] FPGA families – the XC4000 series, and the newer Spartan 2/Virtex 2 families. The XC4000 series is based on Combinational Logic Blocks (CLBs), each of which contains two 4-input lookup tables (LUTs) and two flip-flops. Spartan 2 and Virtex 2 CLBs consist of *slices* that each contain two LUTs and two registers. Spartan 2 devices have two slices in each CLB, while Virtex 2 devices have four slices in each CLB and are available in very large sizes. Since XC4000 series CLBs and Spartan 2/Virtex 2 Slices both contain two LUTs and two flip-flops, they are roughly comparable. However, Spartan 2/Virtex 2 LUTs can also be configured as shift registers of up to 16-bit length. Thus the Spartan 2/Virtex 2 implementations can accommodate a much larger number of layers than the XC4000 style, which would require extra CLBs to accommodate more than 9 layers. An added advantage of the shift-register implementation in the Spartan 2/Virtex 2 is that it places a much lower demand on the routing resources of the FPGA.

Table 3. FPGA Implementation

Design	XC40xx (CLBs)	XC2S/XC2V (Slices)
NL Cell	8	8
NLM Cell	14 (≤9 layers)	19 (≤17 layers)

As in the previous section, these measurements can be used for rough estimates of the capacity of FPGAs to implement large arrays of cells. For example, the largest Virtex 2 part currently announced is the XC2V10000 [8], an array of 128 X 120 CLBs containing 61,440 slices. This part could therefore accommodate approximately 3,200 NLM-cells, or enough for a 56 X 56 array of processing elements, providing that the interconnection resources in the FPGA are sufficient for such a high-density circuit. To test the feasibility of such high-density circuits, a full design of an 8 X 8 single layer NL-Machine was synthesized into a low-end Spartan XC2S100 device. This design could be successfully placed and routed even though it utilized more than 90% of the available cells in the chip.

6. CONCLUSIONS

This paper has explored the feasibility of a new approach for hardware acceleration of maze routing. The new architecture provides an efficient processing element design that supports multiple layers and iterative routing. When combined with advances in integrated circuit technology, we believe that this approach shows much promise.

The next step in this research is to create full implementations of the NL-Machine and evaluate its performance compared to software maze routers. Additional areas for future work include the development of arrays that support nonuniform grids as well as specialized routing architectures like FPGAs.

7. REFERENCES

[1] Lee, C. Y. “An Algorithm for Path Connections and its Applications,” *IRE Transactions on Electronic Computers* vol. EC-10, no. 2, pp. 346-365, 1961.

[2] Akers, S. “A Modification of Lee’s Path Connection Algorithm,” *IEEE Trans. Electronic Computers* vol. EC-16, no. 2, pp. 97-98, 1967.

[3] Breuer, M., and Shamsa, K. “A Hardware Router,” *Journal of Digital Systems*, vol. IV, no. 4, pp. 393-408, 1981.

[4] Suzuki, K., et. al., “A Hardware Maze Router with Application to Interactive Rip-Up and Reroute,” *IEEE Trans. CAD*, vol. CAD-5, no. 4, pp. 466-476, 1986.

[5] Watanabe, T., et. al., “A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor”, *IEEE Trans. CAD*, vol. CAD-6, no. 2, 1987.

[6] Ventkateswaran, R., and Mazumder, P., “Coprocessor Design for Multilayer Surface-Mounted PCB Routing,”, *IEEE Trans. VLSI Systems*, vol. 1, no. 1, 1993.

[7] Dees, W., and Karger, P., “Automated Rip-Up and Reroute Techniques”, *Proceedings 19th Design Automation Conference*, pp. 432-439, 1982.

[8] Xilinx, Inc., *Xilinx Databook*, 2001. Available online at http://www.xilinx.com.

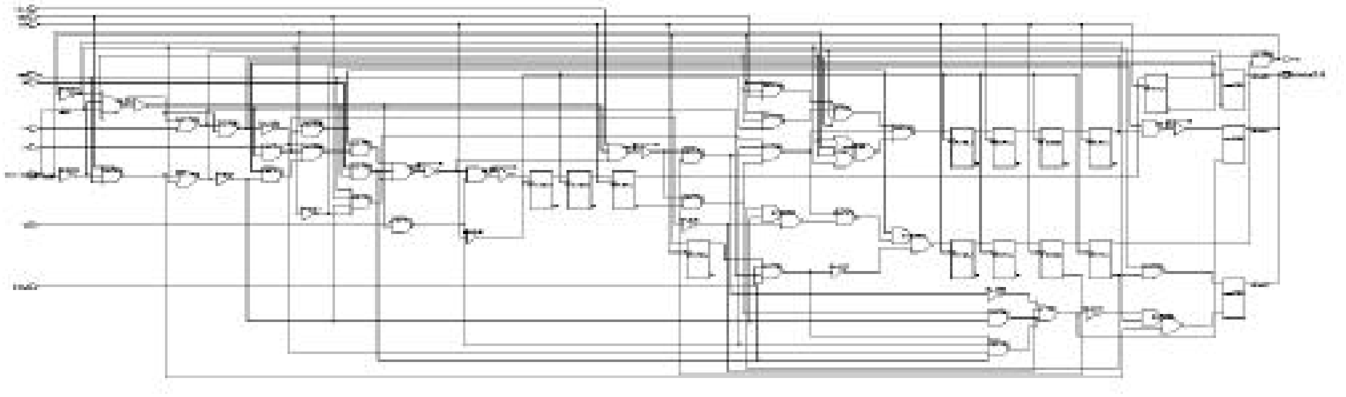


Figure 6 – Gate-Level Implementation of the 4-Layer NLM Cell