L3: An FPGA-Based Multilayer Maze Routing Accelerator

John A. Nestor

Department of Electrical and Computer Engineering
Lafayette College
Easton, Pennsylvania 18042 USA
nestorj@lafayette.edu

Abstract

This paper describes a multi-layer maze routing accelerator which uses a two-dimensional array of processing elements (PEs) implemented in an FPGA. Routing for an L-layer N X N grid is performed by an array of N X N PEs that time-multiplex each layer over the array. This accelerates the classic Lee Algorithm from O(L X d^2) in software to O(L X d). Each PE can be implemented in 32 look up tables (LUTs) in a Xilinx Virtex-II FPGA, which makes possible routing arrays that are large enough to support detailed routing for VLSI. Cycle measurements show a speedup of 50-75X over a 2.54GHZ Pentium 4 for a 4-layer 8 X 8 array and 93X for a 4-layer 16 X 16 array.

Keywords: VLSI Routing, Hardware Acceleration, Lee Algorithm, FPGA Design

1. Introduction

Wire routing is a key problem in Electronic Design Automation [1, 2] for VLSI Integrated Circuits. While routing has been studied extensively over the years and many different algorithms have been proposed, the classic grid-based Lee Algorithm [3] for Maze Routing remains popular as a basic ingredient in many approaches.

In Maze Routing the routing surface is represented as a rectangular grid of connected nodes. The Lee Algorithm searches for a connection between a source and target node in three phases, as shown in Fig. 1. During the expansion phase, it searches outward from the source while labeling each encountered node with the direction of the shortest path back to the source. Expansion continues until either the target is reached or no further labeling is possible, in which case no connection exists. During the backtrace phase, the algorithm follows the labels from the target node back to the source node and marks the nodes on the
path as obstacles, indicating that they are part of a connection and unavailable for further routing. During the cleanup phase, it clears the remaining labels so that more connections can be routed using the same approach. The algorithm is easily extended to multiple layers using a three-dimensional grid.

Fig. 1. Maze Routing with the Lee Algorithm.

The Lee Algorithm is popular because it is guaranteed to find a shortest-path connection if one exists. However, it is computationally expensive - the expansion phase worst-case execution time is $O(d^2)$ for a connection of distance $d$. Multilayer routing is even worse - since expansion proceeds on up to $L$ layers simultaneously, execution time grows to $O(L \times d^2)$. Similarly, since the cleanup phase must clear labels in all unoccupied gridpoints its execution time is $O(N^2)$ for an $N \times N$ single-layer grid and $O(L \times N^2)$ for a multi-layer grid.

One approach to deal with this complexity is to partition a large grid into smaller routing regions and break the routing process into two phases: global routing and detailed routing [1-2]. Global routing assigns connections to routing regions but defers decisions about specific connections to the following detailed routing phase. The size of the grid in routing regions is chosen to make detailed routing tractable – grid sizes of "tens of tracks" in each horizontal direction are typical [4]. However, routing remains expensive even with this reduced problem size.

Another approach is to develop hardware accelerators to speed up the routing process by taking advantage of potential parallelism in the Lee Algorithm using an array of processing elements (PEs).
Depending on the structure of the array, these systems are often classified as either *Full Grid* or *Virtual Grid* accelerators.

*Full Grid* accelerators [5-7] use an array of simple processing elements (PEs) that are directly assigned to the nodes of the grid. One of the earliest examples of such an accelerator was Breuer and Shamsa's L-Machine [5]. The L-machine consists of an array of PEs called *L-cells*. Each L-cell represents one gridpoint and is connected to L-cells that represent neighboring gridpoints. An attached *control unit* broadcasts commands which can directed either to an individual L-cell (selected by attached column and row decoders), or to all cells in the array to execute in parallel. During each expansion step, labeling is performed in parallel using connections between neighboring cells used to indicate when a cell can be labeled. Backtrace is also performed in parallel, with results communicated back to the control unit using attached column and row encoders. The main advantage of the direct-grid approach is that it reduces execution time of from $O(d^2)$ to $O(d)$. The main disadvantage is the high cost of the hardware, since $N^2$ PEs are required for an $N \times N$ grid. Supporting multiple layers requires $L \times N^2$ PEs, which is impractical for all except very small grids.

*Virtual Grid* accelerators [8-11] map multiple nodes onto each PE. One of the most sophisticated Virtual Grid approaches is Venkataswaran and Mazumder's Hexagonal Array Machine (HAM) [10], which maps nodes onto an array of PEs connected in a wrapped hexagonal mesh. This results in a much smaller number of PEs – according to [10] a HAM array of $3 \times L \times N$ PEs could route at approximately the same speed as an $L \times N^2$ Full Grid approach. However, each PE in must be significantly more complex since it must keep track of multiple nodes. To properly translate from actual grid positions to Virtual Grid PEs, the physical location of each grid must be broadcast to adjacent PEs, requiring additional communication overhead. This complexity is reflected in a prototype implementation of the HAM PE [11], which was implemented as a custom 1.2µm VLSI chip design that measured 5.1 mm X 5.8 mm.

Other approaches to accelerating the Lee Algorithm have included the use of a special-purpose raster pipeline processor to perform parallel expansion on a small window of cells [12], pipelining of memory accesses and processing without performing expansion in parallel [13], and implementations that run on general-purpose multiprocessor systems [14]. Each of these approaches provides some level of acceleration, but falls short of the potential speedup of the grid-based approaches.
More recently, accelerators have been proposed for routing field-programmable gate arrays (FPGAs), which have a more constrained routing topology. Chan and Schlag [15] proposed acceleration of the well-known PathFinder [16] algorithm for FPGA routing. This approach uses a combination of special-purpose hardware to accelerate the priority queue used by the algorithm and multiple processors to route groups of connections in parallel. A more radical approach proposed by DeHon et. al. [17] is to add hardware to the routing structure of an FPGA to support the search for connections, allocation of connections, and removal of connections using a rip up and reroute algorithm. This provides the potential for dramatic speedup at an estimated area penalty of 50% in the interconnect hardware of the FPGA. In a second paper [18], the same researchers propose the implementation of similar hardware in an FPGA. This is analogous to a single-layer Full Grid routing machine, with extra hardware to support a parallel implementation of path allocation and rip up. While dramatic speedups are achieved, the cost of the routing hardware is high – each PE requires an estimated 155 4-input lookup tables (LUTs) for a single-layer mesh topology.

This paper describes the results of a new approach to multi-layer routing acceleration called the \textit{L3 Architecture} [19, 20]. L3 extends the Full Grid approach to efficiently support multiple layers by \textit{time-multiplexing} a two-dimensional array of PEs over multiple layers. Each PE stores the state of all layers for a specific gridpoint in a shift register. It processes one layer during each clock cycle as state values circulate through the shift register. This tradeoff reduces the number of processing elements for an L-layer N X N grid from L X N^2 to N^2 but at the same time increases the execution time of expansion from O(d) to O(L X d). However, since the number of layers typically much smaller than the connection distance d, this can result in substantial acceleration. Even when d is small, L3 still provides significant acceleration over software since the expansion step in software requires multiple memory accesses which can consume thousands of processor clock cycles.

The L3 PE can be efficiently implemented using a Xilinx Spartan or Virtex FPGA [21], in part because these devices allow shift registers of 16-bits or less to be realized in a single look-up table (LUT). The multi-layer PE design can be implemented using only 32 LUTs. This makes feasible the construction of accelerator array that can easily support the size of grid needed for detailed routing.

When compared to traditional Full Grid accelerators, L3 has two main advantages. First, it provides efficient support for up to 16 layers using a single array of N X N PEs. In contrast, prior Full Grid
approaches require an additional N X N PEs for each additional layer, making the approach practical for only one or two layers. Second, careful design of the PE has reduced the hardware demands over prior Full Grid accelerators; it was shown in [19] that for the single-layer case, the L3 processing element required 36% less logic (in equivalent 2-input gates) than the L-machine PE design described in [5].

L3 also has advantages over the Virtual Grid approach. First, although a Virtual Grid array can support routing with a much smaller number of PEs, each PE must be more complex to support expansion between multiple unrelated gridpoints. Second, each Virtual Grid PE requires more time to perform expansion, backtrace, and other tasks. For example, the HAM PE [11] requires 14 clock cycles to perform one expansion step compared to L clock cycles in L3.

When compared to FPGA routing approaches such as [17, 18], the L3 approach has an advantage in implementation efficiency. While the FPGA approach supports path search, back trace, and rip up directly in hardware, the cost of these features is substantial – a PE for a single-layer mesh requires 155 LUTs compared to the 32 PEs required for a multi-layer L3 PE that can support up to 16 layers.

Additional features of L3 include support for rapid removal of obstacles and connections as required by routers that use a “rip up and reroute” strategy, and support for preferential routing on a layer-by-layer basis. The grid-based nature of the L3 architecture maps naturally to the structure of common FPGAs, while the local nature of most connections allows high utilization of logic blocks. As larger FPGAs come to market, larger arrays can be implemented with only minor modifications to the architecture.

The initial ideas for the L3 Architecture were described in [19] along with design results for a preliminary PE design using either standard cells or FPGA for both single-layer and multi-layer accelerators. A refined PE design and full array implementation of the L3 Architecture was described in [20], which focused mainly on the single-layer version of the array and reported implementation results using a small FPGA. This paper builds on these ideas while focusing on an FPGA-based implementation of the multi-layer version of the accelerator. It provides expanded details about the design of the array and attached control unit, provides new implementation results for multi-layer accelerator arrays, and presents performance measurements that evaluate the speedup of the routing array over software on a Pentium 4 PC.

The remainder of this paper is organized as follows: Section 2 reviews the general organization of the L3 architecture. Section 3 describes how the architecture is used to route single-layer connections, while
Section 4 describes how the architecture is extended to perform multi-layer routing using time-multiplexing. Section 5 describes the design of the control unit that sequences the operation of the processing elements. Section 6 describes implementation results. Finally, Section 7 concludes the paper and discusses possible future work.

2. The General Approach

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

First, these routing tools must wire a large number of connections or *nets*. Since the Lee algorithm can only find one connection at a time, it is used *iteratively*. While the Lee algorithm is guaranteed to find a shortest path for a single net, each routed net forms obstacles for later nets. These obstacles may make the routing of later nets longer than optimal, and sometimes impossible to complete. This requires a “rip up and reroute” 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 8 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. Some of these layers may have preferred directions for routing (e.g. all horizontal or all vertical).

The L3 architecture was designed with these needs in mind. Fig. 2 shows the basic organization, in which a Host Processor (a conventional computer system) broadcasts commands that request routing between a given source and target. The accelerator performs the routing and communicates the results of routing back to the Host Processor as a list of endpoints of horizontal and vertical wire segments. FIFO (First In First Out) buffers between the Host Processor and the accelerator reduce communication overhead, since communication and routing take place concurrently.

Like other Full Grid accelerators, the L3 accelerator consists of an array of processing elements called *cells* and an attached control unit (not shown). Each cell is a finite state machine that represents the status of the gridpoint during routing.
Each cell is connected locally to neighboring cells in the grid using output XO. XO is true when a cell is labeled during expansion. It is connected to the WI, EI, NI, and SI inputs of the cell’s west, east, north, and south neighbors, respectively.
3. Single-Layer Operation

In single-layer routing, each cell is a finite state machine with 6 states: E (empty), BL (blocked), XE (expanded east), XW (expanded west), XN (expanded north) and XS (expanded south). Each “expanded” state encodes the direction of the shortest path back to the source node.

Table 1 describes the function of each cell as it responds to one of four commands: CLEAR, SET, EXPAND, and TRACE.

<table>
<thead>
<tr>
<th>CMD</th>
<th>SEL</th>
<th>EW * PF1</th>
<th>WI * PF1</th>
<th>NI * PF0</th>
<th>S1 * PF0</th>
<th>PS</th>
<th>NS</th>
<th>S1</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any X</td>
<td>E</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any</td>
<td>E</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SET</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any</td>
<td>XE</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>TRACE</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any</td>
<td>BL</td>
<td>D1</td>
<td>D0</td>
</tr>
<tr>
<td>EXPAND</td>
<td>–</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>E</td>
<td>XE</td>
<td>0</td>
<td>SEL’</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>0</td>
<td>1</td>
<td>–</td>
<td>–</td>
<td>E</td>
<td>XW</td>
<td>0</td>
<td>SEL’</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>–</td>
<td>E</td>
<td>XN</td>
<td>0</td>
<td>SEL’</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>E</td>
<td>XS</td>
<td>0</td>
<td>SEL’</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any X</td>
<td>PS</td>
<td>1</td>
<td>SEL’</td>
</tr>
<tr>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>Any X</td>
<td>PS</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Single-layer routing starts when the control unit selects all cells and broadcasts a CLEAR command to set all cells to the E (empty) state. It then selects the source node and uses the SET command to set the source cell to the XE (expanded east) state. It next applies the EXPAND command while selecting the location of the target cell. During each successive clock cycle, a cell will enter an expanded state if one of its neighbors is in an expanded state and the corresponding preference input (PF1 for east/west, PF0 for north/south) is high. This allows preferential expansion to be performed by the control unit, e.g., to bias expansion (and connections) in either the east-west or north-south direction.

Expansion is terminated depending on the value of two status bits, labeled S1 and S0. Status bit S0 indicates whether the selected target cell has been reached during expansion – it is pulled low by the target cell when it enters an expansion state to indicate that expansion has successfully completed. Status bit S1 is a “watchdog” bit that is pulled low by any cell in the array that is currently entering an expansion state. A high value on S1 indicates that expansion has failed because obstacles block all possible connections between the source and target.
If expansion is successful, the control unit starts the backtrace phase by selecting the target cell and broadcasting the TRACE command. In response, the target cell asserts its state code on the STATUS bus and enters the obstacle state (BL). The control unit uses the direction information encoded in the state code to determine the address of the next cell in the path and repeats this process until the source node is reached and the entire path is marked as an obstacle. Each time the backtrace direction changes, the location of the cell represents the endpoint of a straight wire segment. The coordinate of this point is enqueued in the "routing results" FIFO so that by the end of the backtrace phase all wire segment endpoints have been enqueued for transmission back to the host processor.

Finally, during the cleanup phase the CLEAR command is applied with no cells selected. This clears expansion labels (but not obstacles) so that additional nets can be routed using the same process.

The entire process takes \( d \) clock cycles for expansion, \( d \) clock cycles for backtrace, and 1 clock cycle for cleanup. This is a substantial improvement over the \( O(d^2) \) worst-case performance of expansion and the \( O(N^2) \) performance of cleanup in a conventional software approach.

4. Multi-layer Routing

L3M is the multi-layer version of the L3 architecture. It uses the same array structure but time-multiplexes cell hardware over multiple layers. Fig. 3 shows the internal organization of a L3M cell, which uses a shift register to store and recirculate the states of multiple layers. It processes states from bottom to top on each successive clock cycle. An attached layer counter (not shown) keeps track of the current layer being processed by all cells in the array and asserts the /TOP signal low when the top layer is being processed.

During each clock cycle, the state of the layer that is currently being processed is stored in register ST0. State information for the remaining layers is stored in the shift register. At the end of the clock cycle, the new state of the cell is shifted into the top stage of the shift register and the state in the cell “above” the current cell is loaded into ST0 to be processed on the next clock cycle.

The state sequencer is similar to that of the single-layer cell except that it has two additional states XU (expanded up) and XD (expanded down). State values remain 3 bits wide, so that a total of 3L bits of storage are required for the L-1 stage shift register and register ST0.
Like the single-layer cell, the multi-layer cell uses the EI, WI, NI, and SI inputs to determine the expansion status of its horizontal neighbors on the current layer. Additional logic is needed to determine the expansion status of the vertical neighbors i.e. the nodes immediately above and below the node being processed. This is accomplished using the logic blocks marked XH and XL in Fig. 3.

The XH logic block determines the expansion status of the node above the current node. Its inputs are tied to the bottom stage of the shift register, which (unless the current node is on the top layer) contains the state of the node immediately above the current node. This block asserts its output true when the node above the current node is in an expansion state. The AND gate connected to the XH block output suppresses this value when the /TOP signal is low; this indicates that the top layer is being processed and there is no layer above the top layer. This prevents expansion from “wrapping around” from the bottom layer to the top layer.

Similarly, the XL logic block determines the expansion status of the node below the current node. However, since layers are processed from bottom to top, this status must reflect the state of the lower layer.
before the current expansion step. Thus XL is calculated when the previous layer is being processed and stored in a flip-flop for use during the next clock cycle, when it reflects the expansion status of the layer below the current layer. As in the XH logic an AND gate connected to /TOP suppresses the XL logic when the top layer is being processed to keep expansion wrapping around from the top layer to the bottom layer.

The preferential routing input PFV allows vertical expansion to be suspended while allowing horizontal expansion to proceed. Specifically, when the PFV input is zero the shift register is bypassed and the sequencer’s next state is fed directly back into the ST0 register. This has the effect of performing horizontal expansion on a single layer in isolation from all other layers. Holding PFV zero for a number of clock cycles before asserting it high biases expansion in the horizontal directions, which is often desirable due to the additional cost of vertical connections ("vias") in manufacturing processes.

The PFV input can also be used to reduce the execution time of the backtrace phase – when the TRACE command is applied to a cell and the indicated backtrace direction is horizontal, the control unit can assert PFV to zero so that the current layer is processed again on the next clock cycle rather than waiting L clock cycles for the layer to recirculate through the shift register. Thus successive horizontal backtrace steps take one clock cycle each. A vertical backtrace step in the “up” can also be executed in one clock cycle using the normal action of the shift register.

On the other hand, a vertical backtrace step in the “down” direction will take L-1 cycles since all the other layers must be recirculated through the shift register before the new layer is reached. This exposes a potential problem with the design: in the worst case, where a connection is being made from the top routing layer to the bottom routing layer, \((L-1)^2\) clock cycles will be required during the backtrace phase. This penalty is acceptable when the number of layers is fairly small, say, less than 10, and top-to-bottom routing connections are rare. When this is not the case, the hardware can be modified to use a bidirectional shift register. This would allow backtrace in the “down” direction to proceed in just one clock cycle but would add considerable cost to the hardware for each cell.

5. Control Unit Design

The control unit sequences the operation of the cells in the array. It accepts commands and operands from an external host such as a personal computer and returns status and data. Commands and status are communicated one byte at a time. The current version uses an open source RS-232 serial transceiver [23]
to provide a simple interface to the PC host processor. However, this interface can easily be replaced with a higher performance interface such as a PCI bus interface. Table 2 summarizes the commands implemented by the control unit.

Table 2
Commands executed by the control unit.

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTE</td>
<td>sx, sy, sz, tx, ty, tz</td>
</tr>
<tr>
<td>SELECT</td>
<td>x1, y1, z1, x2, y2, z2</td>
</tr>
<tr>
<td>CLEAR</td>
<td>(selected region)</td>
</tr>
<tr>
<td>EXPAND</td>
<td>(selected region)</td>
</tr>
<tr>
<td>SET</td>
<td>(selected region)</td>
</tr>
<tr>
<td>TRACE</td>
<td>(selected point)</td>
</tr>
<tr>
<td>CLEARX</td>
<td>(all cells)</td>
</tr>
<tr>
<td>GET_XCOUNT</td>
<td>(none)</td>
</tr>
<tr>
<td>GET_TCOUNT</td>
<td>(none)</td>
</tr>
</tbody>
</table>

The ROUTE command accepts a sequence of coordinates for the source and target of a desired connection. The control unit responds to this command by performing cleanup, expansion and backtrace. If routing is successful, it returns the coordinates of the endpoints of the wire segments used to make the connection followed by a "SUCCESS" status code. If routing is not successful, separate status codes indicate that the failure occurred either in the expansion ("XFAIL") or backtrace ("TFAIL") phases.

The SELECT command selects a (possibly multi-layer) rectangular region of cells to which a primitive cell command will later be applied. The CLEAR command sets all selected cells to the E (Empty) state. The EXPAND, SET and TRACE commands apply the corresponding low-level cell commands to the selected cells for debugging purposes. The CLEARX is applied to all cells and resets all cells in an expanded state (i.e.,XE, XW, XN, XS, XU, XD) to the E state while leaving cells in the obstacle state (BL) unchanged. It is used to independently test the cleanup phase.

The GET_XCOUNT and GET_TCOUNT commands return the values of cycle counters for the expansion and backtrace phases, respectively. These counters, which are cleared at the beginning of each ROUTE command, are used in performance measurements.

Fig. 4 shows the control unit and its connections to the processing array. Datapath elements in the control unit include three up/down counters labeled x1, y1, z1 and three registers labeled x2, y2, and z2. During routing, the (x1,y1,z1) counters are loaded with the target coordinate of the desired connection while (x2,y2,z2) registers are loaded with the source coordinate. The control FSM applies the SET
command to set the source to the XE state and then applies the EXPAND command over multiple clock cycles while selecting the target coordinate and monitoring the status bus. If expansion completes successfully, the cell representing the target will pull status signal S1 low when it is reached. The control unit then performs the backtrace phase using the (x1,y1,z1) up/down counters. More specifically, it applies the TRACE command to the cell selected by (x1,y1,z1), reads the backtrace direction of this cell from the status bus, and increments and/or decrements the appropriate up/down counters to select an adjacent cell in the direction of the shortest path. This process continues until the (x1,y1,z1) coordinate matches the source coordinate stored in the (x2,y2,z2) registers, as detected by the three comparators in the datapath.

The layer selection logic is used to determine when a command should be applied to a specific layer or range of layers. This is used to apply commands to a rectangular region of cells over a range of layers, for example, when clearing a range of cells to rip up a net.

Fig. 4. Connections between Control Unit and PE Array.

The multiplexer at the bottom of Fig. 4 is used to select which information will be passed back to the host processor. This includes the endpoints of segments that are routed in the backtrace phase (from the x1,
y, and z1 registers), the status output of the cell array (used for debugging), status codes that indicate error conditions and/or successful completion, and cycle counters that are used for performance measurements.

The datapath elements of the Control Unit are sequenced by a Finite State Machine with 22 states that reads commands from the host interface, sequences the various control signals, and reads status signals from the datapath and the processing array.

6. Results

6.1 Implementation

A design of the L3 accelerator has been coded in the Verilog hardware description language and synthesized into Xilinx SpartanII and Virtex-II FPGAs [21] using the Xilinx ISE 4.2i and Synopsys FPGA Express tools. When implemented alone, the synthesized multilayer cell requires between 27 4-input look-up tables (LUTs) when preferential routing is disabled up to 32 LUTs when preferential routing is included. Three of these LUTs implement shift registers using the Xilinx SRL16 feature; each cell also requires three flip-flops and three tristate buffers for the STATUS outputs. The LUT requirements of the cells can be used to predict the maximum size of routing array that can be accommodated by a larger FPGA. For example, the Virtex-II Pro XC2VP125 device contains 111,232 LUTs and could accommodate a 58 X 58 array of cells.

Table 3 shows the results of synthesizing complete routing arrays. 4-layer 4 X 4 and 4-layer 8 X 8 arrays have been synthesized and tested with a Xilinx XC2S300E-6 FPGA on a Memec Development Board [22]. This board includes a 24Mhz clock so synthesis was performed with a 41ns timing constraint. A larger 4-layer 16 X 16 array has also been synthesized targeting a larger XC2V6000-4 FPGA and uses about 13% of that part's capacity. While this design has not yet been tested, timing results from the synthesis tools can be combined with cycle measurement results from Verilog simulation to predict this design's performance. Two versions of the 4-layer 8 X 8 array were synthesized – a basic design in which no preference inputs are used, and a modified version in which the vertical preference input is used to speed up the backtrace phase as described at the end of Section 4. The 4-layer 4 X 4 and basic 4-layer 8 X 8 arrays both meet the 41 ns clock cycle constraint; the larger arrays require longer clock cycle times. Table
3 gives the number of look-up tables (LUTs) and flip-flops for each array, not including the serial interface, which requires an additional 173 LUTs and 141 flip-flops in each design.

Table 3
Implementation Costs and Predicted Clock Period

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Control LUTs/FFs</th>
<th>Array LUTs/FFs</th>
<th>Total LUTs/FFs</th>
<th>CLK (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4L X 4 X 4</td>
<td>238 / 23</td>
<td>425 / 50</td>
<td>663 / 73</td>
<td>37</td>
</tr>
<tr>
<td>4L X 8 X 8</td>
<td>244 / 27</td>
<td>1819 / 194</td>
<td>2063 / 221</td>
<td>38</td>
</tr>
<tr>
<td>4L X 8 X 8 VP</td>
<td>386 / 43</td>
<td>1841 / 259</td>
<td>2227 / 302</td>
<td>45</td>
</tr>
<tr>
<td>4L X 16 X 16</td>
<td>290 / 47</td>
<td>7483 / 817</td>
<td>7773 / 864</td>
<td>54</td>
</tr>
</tbody>
</table>

Comparing L3 to previous routing accelerators is difficult since they have been proposed using a wide range of technologies. Also, in many cases a full implementation was not completely constructed but only simulated. However, the Verilog design of the L3 processor can be used with synthesis tools to compare the design of L3 to these approaches.

For example, in [19] a preliminary version of the L3 cell design was synthesized into a public-domain standard cell library for comparison to the L-Machine cell design, which was shown in [5] as a schematic containing 75 2-input logic gates and 7 flip-flops. The PE design in a similar Full Grid approach required approximately 71 gates and 6 flip-flops [7]. In contrast, the single-layer L3 cell required 48 2-input logic gates and 3 flip-flops. Thus the L3 design is shown to be more efficient than previous Full Grid approaches.

Comparison of L3 with Virtual Grid accelerators is more difficult since the number of PEs in the accelerator array is quite different. For example, the HAM accelerator[10] requires 3 X L X N PEs to perform routing at approximately the same speed of a Full Grid machine. In contrast, L3 requires N^2 L-layer PEs to perform the same task. However, it must the implementation cost of a Virtual Grid PE is much larger than that of an L3 PE. To quantify this difference in area, an 8-layer L3 multi-layer PE was synthesized, placed, and routed using standard cells with 1.2µm design rules similar to those used to create the HAM PE prototype reported in [11]. While a modern implementation would use a much more aggressive technology, this allows a direct comparison between the two approaches. Table 4 shows the area of the HAM PE (adjusted to remove the area of the chip pad frame) and 8-layer L3 PEs, the number of PEs required to implement an 8-layer 32 X 32 array, and the total area consumed by a 32 X 32 array. While this can only be considered a rough comparison, the advantage of the L3 approach is clear.
A rough comparison can also be made with the FPGA routing accelerator of [17, 18]. This work attacks a different and more ambitious problem that integrates path search, path allocation, and rip up and promises speedups of 2-3 orders of magnitude over software FPGA routers. However, the hardware costs of this approach are substantial, with a mesh-style switchpoint requiring 155 4-input LUTs [18]. L3 compares favorably here since each cell (roughly comparable to a switchpoint) requires only 32 LUTs.

Table 4. Area requirements for HAM vs. L3 to implement an 8-layer 32 X 32 Array.

<table>
<thead>
<tr>
<th></th>
<th>PE Area</th>
<th>PEs Required</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAM [11]</td>
<td>24.13 mm²</td>
<td>3 X 8 X 32 = 768</td>
<td>18,531 mm²</td>
</tr>
<tr>
<td>L3 (L=8)</td>
<td>0.303 mm²</td>
<td>32 X 32 = 1024</td>
<td>310 mm²</td>
</tr>
</tbody>
</table>

6.2. Performance

To compare the implementation of the hardware accelerator with software, a reference software implementation of the Lee Algorithm was written in ANSI C. In this program the grid is represented by a three dimensional array of bytes, with each byte representing the state of one gridpoint. This current implementation supports only two-terminal routing and no preferential routing. The resulting code consists of about 1,000 lines of C and was compiled using gcc version 2.96 with the "-O2" optimization option.

Performance measurements were made on a 2.53GHz Pentium 4 system with 1 GByte of RAM running Redhat Linux version 7.3.

Software performance was measured with the Pentium 4 cycle counter using code provided in [24]. The cycle counter was accessed at the beginning and end of each expansion, backtrace, and cleanup step and a running tally was kept of the cycles consumed by each phase. Similar measurements of the hardware performance were made using the cycle counters designed into the control unit, as described in the previous section. These measurements were compared for the two versions of the 8 X 8 X 4 array and the 16 X 16 X 4 array. Measurements for each hardware version were made using a Verilog simulator, with cycle counts multiplied by the minimum clock period from the Xilinx ISE tool to get the execution time of each phase.

Measurements for the software and two hardware versions of the 8 X 8 X 4 array were made by routing a sequence of ten randomly selected source/target pairs on an initially empty grid. Fig. 5 shows the
locations of the source (S) and target (T) terminals of each pair along with the route found by the hardware router.

Table 5 summarizes the total clock cycle measurements, the equivalent execution time (i.e. clock cycles * clock period), and the speedup of the two hardware implementations over the hardware implementation. Not surprisingly, the biggest source of speedup comes from the expansion phase. As expected, the hardware array can perform multiple expansion steps in parallel. In addition, each software expansion step is relatively slow, consuming on the average 4,000-5,000 clock cycles as multiple memory accesses are performed. Therefore the hardware implementation pays off even for short routes with relatively little parallelism. In the first version of the 8 X 8 X 4 hardware array, most backtrace steps require four clock cycles to complete, so that the backtrace phase actually consumes more clock cycles than the expansion phase. This is addressed by the second version, which uses the vertical preference signal to “freeze” the array on a given layer while backtrace is moving in a horizontal direction. This reduces the number of cycles required during backtrace phase at a penalty of a slightly higher clock period, with the overall result an improvement in the speedup from 49.94 to 76.41.

Table 6 shows the results of similar measurements for the 16 X 16 X 4 hardware array compared to software routing using thirty randomly selected source/terminal pairs. This larger array runs at a predicted clock period of 54ns. This somewhat slower clock is more than offset by the increased parallelism of the expansion stage, with an overall speedup of 93.62.

Fig. 5 – Result of routing ten random connections an 8 X 8 X 4 accelerator.
Table 5
Performance Summary – 10 Random Connections on an 8 X 8 X 4 Array.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Software (2.53 GHz)</th>
<th>L3M (24MHz)</th>
<th>L3M w/ mod. Backtrace (22.2Mhz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles</td>
<td>Time (µs)</td>
<td>Cycles</td>
</tr>
<tr>
<td>Cleanup</td>
<td>97,512</td>
<td>38.54</td>
<td>40</td>
</tr>
<tr>
<td>Expansion</td>
<td>2,860,280</td>
<td>1130.55</td>
<td>227</td>
</tr>
<tr>
<td>Backtrace</td>
<td>78,084</td>
<td>30.86</td>
<td>319</td>
</tr>
<tr>
<td>Total</td>
<td>3,035,876</td>
<td>1199.95</td>
<td>586</td>
</tr>
</tbody>
</table>

Table 6
Performance Summary – 30 Random Connections on a 16 X 16 X 4 Array.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Software (2.53 GHz)</th>
<th>L3M (18.5 MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles</td>
<td>Time (µs)</td>
</tr>
<tr>
<td>Cleanup</td>
<td>2,119,164</td>
<td>837.61</td>
</tr>
<tr>
<td>Expansion</td>
<td>36,847,352</td>
<td>14,564.17</td>
</tr>
<tr>
<td>Backtrace</td>
<td>428,820</td>
<td>169.49</td>
</tr>
<tr>
<td>Total</td>
<td>39,395,336</td>
<td>15,571.28</td>
</tr>
</tbody>
</table>

Performance comparisons of the L3 architecture with other routing accelerators are difficult for two reasons. First, most of the proposed architectures have not been fully implemented, and in many efforts few comparable performance results are provided. Second, given the dramatic improvements in technology, a fair comparison of hardware developed over the span of several years is difficult. Therefore, results can only be compared to get an idea of the general trend. For example, in [7] an implementation of a 2-layer 16 X 16 Full Grid array is assumed to have a clock frequency of 10 MHz; at this clock rate the time to route a single net is estimated to be 3.8µs. This can be compared to the results in table 6 for a 4-layer 16 X 16 array, which shows an average routing time of 5.5µs for a single net. This compares favorably given the difference in number of layers. In [11], it is predicted that the design of the HAM PE can perform a single expansion step in 0.84µs assuming a 16MHz clock. In contrast, in L3 a single expansion step takes four clock cycles, or 0.21µs for the four-layer 16 X 16 array. These comparisons show that L3 is competitive with previous approaches on performance while supporting multi-level routing with lower implementation costs.

7. Conclusion

This paper has described a new architecture for an FPGA-based multi-layer maze routing accelerator. The efficient design of the accelerator makes it possible to implement large arrays on a single FPGA;
multiple-FPGA solutions could be used to support even larger arrays. Performance measurements show the promise of this approach, with speedups over software on modern processors ranging from 50-94 depending on the size of the array and the design used. Larger versions of the accelerator should offer even larger speedups because of increased parallelism. Special features of the architecture support preferential routing on a layer-by-layer basis and fast rip up and reroute.

There are many directions for future work with this architecture. First of all, the development of larger routing arrays is currently underway using a Xilinx XC2V6000 FPGA on development board with a PCI interface. This should allow the design of arrays at least 32 X 32 in size, which is more appropriate for VLSI routing using a global routing/detailed routing paradigm.

Larger arrays using multiple FPGAs are also being considered, although this adds the overhead of off-chip communication since cells on the edges of the array must be connected to cells on neighboring arrays. For example, consider connecting an FPGA containing an L-layer N X N array to neighboring arrays four adjacent FPGAs. This would require 4N-4 XO outputs, 4N input connections (N each for XE, XW, XN, and XS), 4log₂(N) input connections for the row and column address, 7 global input connections for CLK, CMD, /TOP and PREF, and 3 output connections for STATUS. This gives a total of 8N + 4log₂(N) + 6 I/O connections for an N by N array. If N=32, this would require 282 I/O connections. This can easily be accommodated by a large FPGA such as the Virtex-II XC2V6000, which offers a total of 824 I/O connections in the FF1152 package [21].

The implementation of the architecture can be redesigned so that it can run at faster clock speeds. This can be done by converting the Mealy-style control unit to a Moore machine, by inserting registers between the array and the control logic, and by pipelining the processing to overlap the processing of successive layers.

A more advanced interface between the array and the attached host processor should be developed and host software should be written that can be used to explore different routing strategies for multiple nets and nets with more than two terminals. For example, the control unit could be modified to work with an entire netlist and perform a rip up and reroute algorithm without the intervention of the Host Processor. This would allow the interface to transfer both input and output information using block transfers that would maximize the bandwidth of communication and reduce overhead.
Finally, the architecture should be extended to support variable width and spacing requirements. This could be accomplished by modifying the control unit to “bloat” the wire segments that it marks as obstacles during backtrace. This would have the effect of reserving additional space around the connections to enforce spacing restrictions.

Acknowledgments

This author acknowledges the assistance of C. Briand, T. Ihimoyan, and M. Marbell in implementing early versions of the L3 hardware and software and helpful discussions with M. Abramovici and J. Greco.

References


