EVE: A CAD Tool for Manual Placement and Pipelining Assistance of FPGA Circuits

William Chow
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada M5S 3G4
choww@eecg.toronto.edu

Jonathan Rose
Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto
Toronto, Ontario, Canada M5S 3G4
jayar@eecg.toronto.edu

ABSTRACT

As FPGAs push ever deeper into mainstream digital design, there is an increasing desire for high-performance circuits. This paper describes a manual editor, called EVE, which can assist a designer to perform manual packing, placement and pipelining of commercial FPGA circuits to achieve a meaningful increase in performance. This effort is inspired by Von Herzen's paper [15] [16], which proposed the notion of an "Event Horizon" – a high-speed circuit design approach in which complete knowledge of the timing effect of every synthesis change is used. It is very laborious to implement circuits using this approach; therefore we try to augment manual design tools in order to make this Event Horizon methodology easier to perform. This paper describes a first step in that direction, which focuses on placement, packing and pipelining. EVE provides an interactive environment that immediately reroutes and timing analyzes after each user circuit modification, giving an exact value for critical path delay. It can also suggest good placement positions and provide flip-flop insertion assist during pipelining. Compared to a state-of-the-art Synthesis and place and route flow, we used EVE to achieve an average of 12.7% higher operating frequency on a set of eight Xilinx Virtex-E circuits of 250 or fewer LUTs.

Keywords

FPGA, programmable logic, manual placement and pipelining, event horizon

1. INTRODUCTION

Most FPGA circuits are designed using a traditional "push-button" CAD flow, which involves design entry, logic optimization, technology mapping, floorplanning, placement and routing. When a high circuit speed that pushes the limits of the silicon's capability is desired, this approach often fails to achieve the required performance. Designers will typically repeatedly floorplan, place and route the circuit until the design goal is met. This iterative process is very time consuming because the resulting design speed is not known until after timing analysis is performed, and the result may seem to be decoupled from the changes applied. There is a clear need for a different high-speed circuit design methodology. In [15] [16], Von Herzen described the design of a signal processing circuit in FPGA running at 250MHz in 1997 using 0.6μm CMOS technology. This remarkable achievement stands in stark contrast to the struggles that designers face to achieve speeds on the order of 150MHz in today's 0.18 μm CMOS technology.

Von Herzen demonstrated a high-speed circuit design methodology using the notion of an "Event Horizon", which refers to the boundary that a circuit element can be placed within in order to satisfy a timing budget. This methodology demands that the designer create each microscopic piece of the circuit with the timing budget in mind. During this process, the complete routing delays are included in the time accounting. Von Herzen used low-level manual design tools to select routing resources carefully, and to avoid the placement of logic elements outside of the horizon. However, it is very laborious to implement circuits using the low-level manual design tools. We therefore became interested to augment such tools to facilitate circuit design employing the Event Horizon methodology. This paper describes the features and implementation of the editor (called EVE, for EVent horizon Editor) as well as quantitative results achieved using it. This initial work focuses on the packing, placement, routing and timing analysis phase of circuit implementation, and uses a push-button flow result as the starting point somewhat different from Von Herzen's design-from-scratch approach. This work relates somewhat to the full-custom VLSI editors Magic [10] and Electric [12] developed in the early eighties to enhance design capabilities with assistance.

In the following section we review Von Herzen's work and present the objectives and context for our work. Section 3 describes the basic move generation mode of the editor while Section 4 describes its pipelining-assist features. Section 5 presents experimental results and Section 6 concludes.

2. BACKGROUND, GOALS and CONTEXT

Von Herzen achieved a circuit speed far beyond the otherwise typical capability of silicon. He did this by employing low-level manual design tools to configure each logic block of the circuit and to select the routing resources manually. He carefully chose where to place each circuit element physically on the chip, because routing delays made up a significant portion of the critical delay of the circuit. He proposed the concept of an "Event Horizon", which could be used to quickly estimate where circuit elements could be placed without violating a very tight timing budget.

FPGA'02, February 24-26, 2002, Monterey, California, USA.

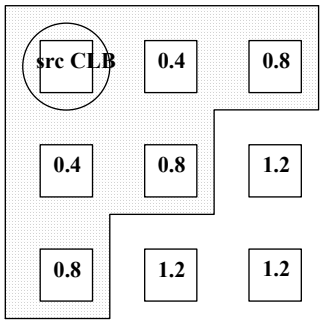

Figure 1: Event Horizon

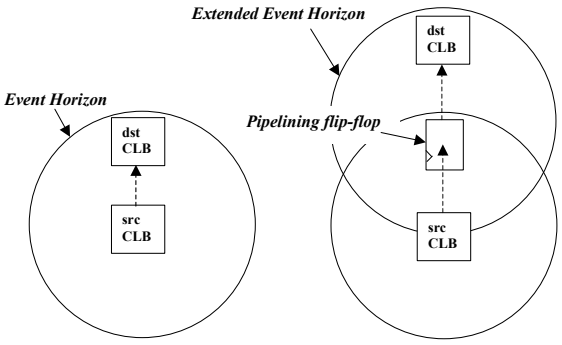

Figure 2: Extending the Event Horizon using Pipelining

The "Event Horizon" concept is illustrated in Figure 1. Assume that a circuit needs to run at 250MHz, so the critical path time budget is 4.0ns. A flip-flop (FF) in a source logic block (CLB) (marked with a circle in the figure) drives a LUT-to-FF combination in another logic block. To determine how far away the target logic block can be placed, we first need to obtain some timing characteristics about the chip such as the maximum clock skew, the LUT delay, the routing delays, as well as the clock-to-output delay and FF setup time. Assume that the maximum clock skew is 0.1ns, the clock-to-output delay is 1.3ns, and that LUT delay + FF setup time through is 1.5ns. In order for the time budget to be met, the routing delay can then take at most 4.0-0.1-1.3-1.5 = 1.1ns. Suppose, for simplicity, that the FPGA has a routing architecture that requires 0.4ns to travel the distance of one logic block (in Manhattan distance) in all directions. For the example given in Figure 1, the target logic block can only be placed at locations that can support a routing delay of at most 0.8ns, as indicated by the shaded box in Figure 1. Von Herzen called such a boundary box the Event Horizon of the source logic block in the context of a circuit required to run at 250MHz. The "Event Horizon" is then defined as the boundary within which the target logic block can lie, such that the routing delay to reach the target logic block from the source logic block is small enough to satisfy the timing budget. With a timing goal in mind, Von Herzen then calculated the Event Horizon for each circuit element, and tried to place the connecting elements in its Event Horizon. In cases when this was not possible (for example, all locations in the Event Horizon were already occupied), extra flip-flops could be introduced to pipeline the circuit, permitting the target logic block to move outside of the current Event Horizon, as illustrated in Figure 2. By first calculating the Event Horizon of each circuit element at a given target speed, Von Herzen incrementally built each part of the circuit, knowing that the timing budget would be met throughout the design process.

2.1 Objectives

Our work has two objectives. Firstly, we would like to construct a manual editor that augments the current low-level floorplanning and circuit editing tools, by employing elements of the Event Horizon methodology. Secondly, we would like to gain more insights to better placement and routing techniques by extensively using the tool to augment the speed of real designs. EVE has the following design objectives:

1. **Target Real FPGA Architectures**. Traditional FPGA research tools tend to work on simplified models of real FPGAs [2]. These tools, for example, rarely represent carry chains correctly. Our goal in this work is to apply the Event Horizon concept and its implications to real devices so that we can deal with all of the realities that designs possess, and try to achieve usable improvements. We chose the Xilinx Virtex-E [20] family as our target.
2. **Give Full Low-Level Control**. The Event Horizon notion requires careful design of each microscopic piece of the circuit. Our editor must permit the user to easily control placement and packing of each LUT, carry element and flip-flop, and to precisely control where flip-flops are inserted when pipelining.
3. **Give Instant Performance Feedback**. After each user circuit modification, the editor should immediately reroute and perform a full timing analysis to report the real circuit performance. This is usually not possible in automated placement of large circuits, but it is feasible for interactively editing small designs. In this work we focus on designs with 250 or fewer 4-input logic cells.
4. **Be Timing Budget Aware**. EVE should be timing-budget aware. It should highlight circuit elements that violate the timing budget and quickly and accurately estimate the effect of a change to the circuit before it is applied. It should also provide a visual aid to illustrate the Event Horizon itself (see Figure 5).
5. **Assist Pipelining**. EVE should assist the user to pipeline the circuit by maintaining correct functionality of the circuit throughout the pipelining process. It should also select good physical placement for pipelining flip-flops to minimize the critical path delay.

2.2 The Xilinx Virtex-E Architecture

In this section we describe the salient features of the target Xilinx Virtex-E [20] architecture. The Virtex-E is fabricated in a *0.18µm* CMOS technology. It is an island-style [2] FPGA architecture in which routing resources surround Configurable Logic Blocks (CLBs). Each CLB has two slices and each slice has two 4-input look up tables (4-LUTs) and two flip-flops (FFs). The two 4-LUTs in each slice can be combined to form a 5-LUT and two such 5-LUTs in the same CLB can be combined to form a 6-LUT. There are carry chains for high-speed arithmetic that run vertically upwards in each slice. Each slice also contains dedicated AND gates and XOR gates to support fast addition and multiplication. A 4-LUT can also be configured as RAM, ROM,

or a 1-bit Shift Register LUT (SRL) of variable depth (*1-16*). Figure 3, taken from [20], shows a Virtex-E CLB.

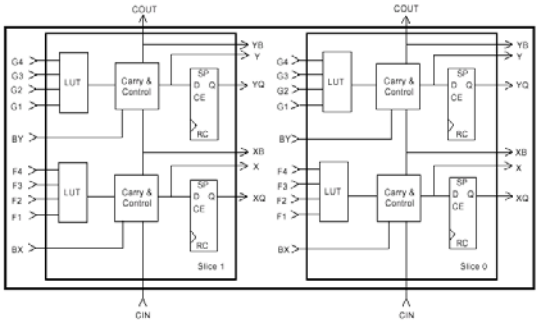

Figure 3: A Virtex-E CLB (taken from [20])

3. TIMING EXACT MICROSCOPIC PLACEMENT (TEMP) MODE

In this section we describe the features and implementation of the basic packing and placement editor assistant. We call this the "Timing Exact Microscopic Placement" (TEMP) mode of EVE. It permits microscopic placement and packing/unpacking of circuit elements while giving instant exact timing feedback. The graphical user interface of EVE is built using EasyGL for Windows [3]. Figure 4 illustrates the concept of a "Timing Horizon". It is based on the Event Horizon concept described in Section 2. When a circuit element (such as a LUT) is to be moved EVE will calculate the change in critical path delay that would occur if the element is placed in a series of target locations (like those marked in gray in Figure 4). In Figure 4, a Timing Horizon of radius one (CLB) is displayed. A negative number means that the critical path delay improves. When a target position is not feasible (it may be occupied or the target slice configuration is not compatible), it does not appear in the Timing Horizon. In EVE, we will call this Timing Horizon simply the "Horizon". It is an important feature of EVE that the designer can use to evaluate the effect of moving a circuit element in the chip.

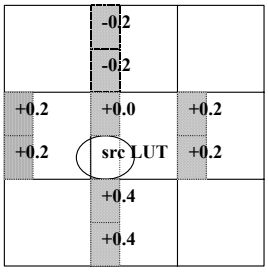

Figure 4:Timing Horizon

In the Timing Exact Microscopic Placement (TEMP) mode, the circuit is represented in a grid format, with each grid cell representing a CLB. Figure 5 shows a circuit on a Xilinx XCV100E [20] chip. Each CLB has two slices, and each slice is divided into six components: two LUTs, two carry cells, and two FFs. In this mode, the placement of all circuit elements is shown, and logic packing and placement operations can be easily modified using a drag-and-drop paradigm. EVE recognizes structural grouping of circuit elements such as carry chains, 5-LUTs, and 6-LUTs. On start up, the critical path of the circuit is highlighted. The current maximum operating frequency of the circuit is also calculated and displayed in the status window. The user can then perform the following operations:

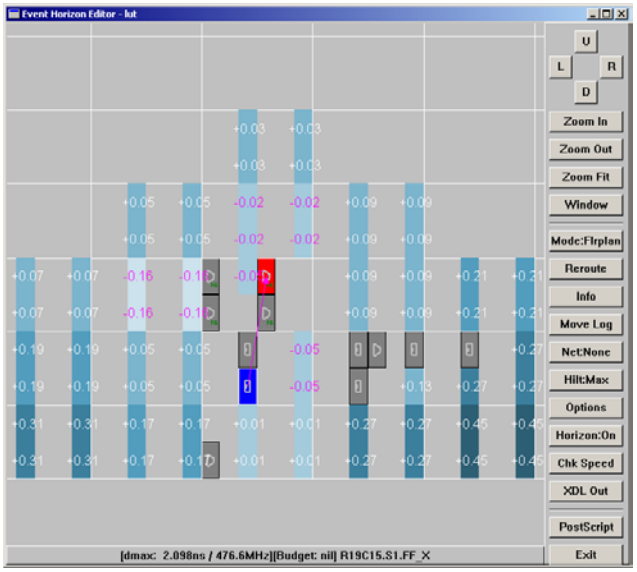

Figure 5: Screen capture of TEMP mode showing a "Horizon"

1. **Change Placement of Components.** Select the components and drag them to the destination location. Eve does this better than the native Xilinx Floorplanner because it immediately reroutes the circuit and reports the real circuit timing. EVE also performs immediate legality checking of move and informs the user of illegal moves by displaying "X" markers on invalid target positions.

2. **Packing/Unpacking of Slices.** When a LUT is moved from one slice to another, the packing of the source and destination slices may be altered. The native Xilinx Floorplanner can only pass this packing/unpacking directive to the mapper as a slow batch task; and such packing/unpacking operation may not be applied successfully. In contrast, EVE can determine the packing feasibility instantly.

3. **Change/Set Timing Budget.** When the timing budget is set or changed, the design is timing-analyzed and the components and nets that violate the budget are highlighted. A typical methodology is to slowly decrease the timing budget, and focus on a more timing critical part of the circuit, until the desired timing goal is met.

4. **Invoke Horizon.** Here the user selects a component and press the "Horizon" button. A Horizon is displayed with a gradient of colors indicating the "goodness" of placing the selected components at the indicated positions. A number is displayed in each valid target position, indicating the change in critical path delay should the component be moved there. The user can control a "Horizon Radius" parameter that controls in Manhattan distance within how many CLBs the horizon calculations should be performed. (A Large radius will take a long time to compute, but reasonable ones are

quick) Figure 5 illustrates a Horizon of radius three for a selected flip-flop component.

5. **Nets Reroute.** We have found that, after a series of microscopic packing and placement changes, that the critical path can be improved by rerouting all of the timing-critical elements. In this option, the user selects some components and invokes the reroute. All nets connected to the selected components will be re-routed leaving other nets in the circuit intact.

6. **Display Dynamic Delay Distribution.** After each user move, EVE analyzes the timing of the circuit and outputs a delay distribution, which summarizes the number of delay paths having delays within different delay ranges. This gives the user an overall picture on the current state of the circuit.

After each move, EVE modifies the netlist as needed, incrementally re-routes nets, and performs timing analysis to report the real timing of the modified circuit. Using instant timing feedback and various budget-aware features, a user can produce superior performance circuits. The following sections describe how the above features of the editor are implemented.

3.1 Interactivity

EVE has to provide high interactivity, which requires very quick partial placement, routing, and timing analysis of the circuit after a user move. One approach would be to use the set of command-line based backend tools from Xilinx including MAP (technology mapping) PAR (placement and routing), TRACE (timing analyzer), and XDL (Xilinx proprietary circuit format to ASCII conversion utility). Since these are relatively slow batch-based tools, each user move could still take minutes to process. Clearly we need to bypass using these tools yet still perform the necessary tasks in a much shorter time. EVE achieves this by interfacing with the Xilinx manual editor directly. The Xilinx FPGA Editor [18] has a full-featured set of textual commands for controlling various operations including slice configuration, placement, and routing. On start up, EVE spawns two copies of FPGA Editor. One copy serves as the "backend" where the real circuit changes are applied. The other copy serves as a "net delay reporter" which calculates net delays on the fly. EVE instructs both the "backend" and "net delay reporter" by sending commands to them using named pipes supported by the Windows NT based platform. The execution result in the backend is obtained for further analysis, by capturing text from the FPGA Editor window using Windows messaging API calls.

EVE determines the timing of the entire circuit at all times. It obtains the initial timing information from the Xilinx timing analyzer (TRACE). It then builds a timing graph of the circuit and performs subsequent timing analysis internally. When the user makes a move, EVE will calculate the effect of the move by using delay values stored in a delay database (described in Section 3.2), and estimate the resulting circuit critical path delay. The change is then communicated to the backend using named pipes. The backend makes the corresponding change, including logic configuration modification, netlist modification, and placement. Then it unroutes all nets affected by the change, and re-routes them using the critical path delay estimated by EVE as a timing constraint for timing-driven routing. This routing is very quick since it is only a partial re-route, and the unaffected nets are left untouched. Finally, the net delays of the modified nets are queried by EVE through the backend, and it will once again have complete knowledge of the updated circuit timing information.

3.2 Delay Modeling

Two types of delays are needed for timing analysis: the logic delay within a slice, and routing delay. Logic delays are usually modeled as constants since the number of configurable paths that exist within each slice is limited. They are usually pre-calculated and stored in look-up tables for fast future retrieval. Routing delays however, vary according to the routing resources taken up by a route. Each routing delay value is governed by an RC model, such as Elmore [4] and Penfield-Rubinstein [13] models. These models take into account the length of the routing wires, and the number and type of switches (buffered or non-buffered) that the routes pass through. For EVE, obtaining such delay information is hard because it has no knowledge of the proprietary RC characteristics of the commercial device. Without this knowledge, EVE has to obtain both logic and routing delay values by querying the Xilinx backend tools one delay at a time and then storing the delays in data files, which we refer to as the delay database.

Logic delays are calculated and stored automatically for each chip with a given speed grade the first time EVE encounters the chip. It does so by enumerating all the possible configurable pathways present within a slice or in-between slices (as in the case of a delay involving 6-LUT). It then writes out a Xilinx Description Language (XDL) description of the circuit containing all the paths of interest, with each path using different CLBs. XDL is a text-based language describing the internals of Xilinx circuits, including slice configuration and routing information. It can be translated into the Xilinx native NCD circuit format using the *XDL* utility. The design is then timing analyzed using a command-based timing analyzer program called *TRACE*. The logic delay values for each path are then extracted from the report file to form a delay-matching table. Such a table will provide mapping from various slice configurations to logic delay values. For Virtex-E, there are *230* such logic delay values. The extraction of routing delays is much more difficult because of the large number of CLB pin to CLB pin delay values present in the Virtex-E devices. For example, an XCV100E chip has *20* rows and *30* columns of CLBs and each CLB has two slices. For any given pin-to-pin route, it can originate from one out of six output pins in either slice *S0* or *S1*. It can also terminate in one out of twelve input pins in either slice *S0* or *S1*. The total number of possible routes of Manhattan distance of length five or less is $(2*5*5 + 2*5 + 1)*20*30*2*2*6*12 \approx 10.5$ million delay values. We estimate that a *450MHz* Pentium-III processor can process four delay values per second, and each delay can be stored using *10* bytes. We would thus need about *30* days to generate the database and the data will take up *100MB* of disk space. To make the delay search space smaller, we devise a compression scheme making use of the symmetric nature of the Virtex-E routing architecture.

3.2.1 Delay Database Compression

To describe the compression scheme better, we have to introduce some notation. We group related pin-to-pin routing delay values together in a group identified by the following format: *G=(S1,P1,S2,P2,X,Y)*. *S1* and *P1* specify the source slice and pin, while *S2* and *P2* specify the target slice and pin. *X* and *Y*

are integers that represent the relative position of the target pin to the source pin. G is used to refer to this group of delays. Figure 6 shows a 3-D plot of the real routing delay values for the delay group *G=(0,XQ,0,G1,-1,-1)* of the XCV100ECS144-8 device, which refers to the group of delay values with source pin located on *XQ* pin of *S0*, and target pin located on *G1* pin of *S0*, and target slice is one CLB west and one CLB north of the source slice. The pin-to-pin routing delay values in group G will then be represented using the notation *(G,R,C)* where *R* and *C* represents the row and column coordinate of the source pin. Each delay value (in ns) is plotted in 3-D space against the *(R,C)* coordinate. We can observe from Figure 6 that the routing delays are indeed fairly symmetrical across identical rows and columns. Now we will discuss the compression scheme we use:

1. **Using Two One-Dimensional Functions.** A two-dimensional grid, with notation *D(r,c)*, where *r* and *c* corresponds to the row and column coordinate, is used to represent all the delay values in the group. It requires *r*c* data points. The following procedure is used:
   a. All *D(r,c)* values are converted from floating point numbers into integers using a scaling factor of *0.02ns* (*0.35ns* will become *0.35/0.02 = 17*). We call the scaled values *D′(r,c)*.
   b. We locate an "intersect" point on the 2-D grid at the "base" of the 3-D plot, and record its delay. We refer to it as the "base delay" (*b*). All *D′ (r,c)* values are normalized by subtracting from the base delay to form *D″(r,c)*. The resulting data points will then contain mostly zeroes.
   c. From the same "intersect" point, we can form two one-dimensional functions, $D_r(r)$ and $D_c(c)$, using the column and row vectors at the intersect, such that $D''(r,c) = D_r(r) + D_c(c)$ (illustrated in Figure 6). With these two functions, the number of data points needed to represent all *D(r,c)* values becomes *r+c*.
2. **Eliminating Zeroes.** $D_r(r)$ and $D_c(c)$ are found to contain mostly zeroes. For example, $D_r(r)$ may be a vector like [*0 0 0 0 2 3 0 0 0 ...*]. Instead of processing the columns with *0* entries in $D_r(r)$, these column numbers are recorded, and are skipped for all subsequent rows. The same rule applies to $D_c(c)$.
3. **Eliminating Duplicates.** $D_r(r)$ and $D_c(c)$ frequently contain entries with the same value. Instead of processing all the entries with the same delay value, only one entry is processed. For example, for the vector [*0 0 2 3 0 0 2 3 0 0 ...*], columns *3-4* and *7-8* are the same, so columns *7-8* are not processed. This rule can be applied to both $D_r(r)$ and $D_c(c)$.
4. **Using Symmetry of Pins.** Delays with *P1 = X* and *P1 = Y* are the same. Only one *P1* value needs to be processed. The same also applies for *P1 = XQ* and *P1 = YQ*.
5. **Record Extra Data Points.** Data points, which cannot be calculated accurately using the above compression scheme, are recorded individually.

Using the heuristics given above, the search space is compressed by about *100* times. All delay values as well as other information including: base delay, intersect point coordinate, zero matching columns/rows, duplicate matching columns/rows and extra data points, are generated and recorded in data files using a set of PERL scripts. In EVE, the data files are loaded into a group of efficient data structures, which we refer to as the delay database. Delay retrieval from the database is quick, and the whole database consumes about *20MB* of physical memory.

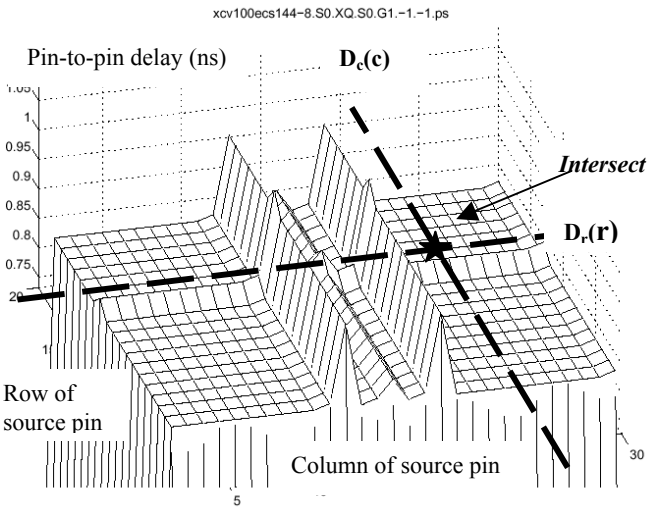

Figure 6: Routing Delay Profile for group G

3.3 Instant Timing Feedback

To provide instant timing feedback, full timing analysis is performed internally within EVE. It is based on a forward and backward sweep approach described in [6]. The "horizon" demonstrates instant timing feedback. For each target position, EVE first determines if the move is valid. Then, it builds a temporary circuit resulting by moving the target to each valid location and performs a full-timing analysis on it. The change in critical path delay timing is displayed in the target position. A horizon of radius three takes about two seconds to calculate on a 1GHZ Pentium-III machine.

4. PIPELINING MODE

Pipelining traditionally occurs during logic design, when the designer introduces pipeline stages to enable parallel execution of multiple circuits to achieve a higher throughput. Pipelining in the Event Horizon methodology context, however, refers to the need to register logic elements when the physical placement becomes an obstacle to satisfy a high-speed design goal as illustrated in Figure 2. We believe research in pipelining at the physical level will become more important as circuit speed pushes towards the limit of the silicon. We need a pipelining assistant that allows the designer to fully control where pipelining flip-flops are inserted, yet helping the designer retain correct functionality of the circuit. The TEMP mode displays the physical locations of each circuit element, so it is ideal for performing packing/unpacking and placement operations. For pipelining, however, such a circuit representation cannot present clearly to the user where flip-flops can be inserted because the graphical display will be very cluttered. We thus propose the pipelining mode in EVE as a way to present the circuit in a better form to assist pipelining. When the user insert or move a flip-flop in the circuit, EVE will automatically determine where in the circuit to insert additional flip-flops to maintain correct functionality of the circuit.

In the pipelining mode, the circuit is displayed as a directed acyclic graph (DAG). Each graph node represents an input or output pin of a logic slice, or the input and output ports of sequential elements, including flip-flops and Shift Register LUTs (SRLs). Each edge represents a logical connection between the graph nodes, which usually has an associated delay value, corresponding to an internal logic delay, or an external routing delay. Primary inputs are displayed at the top and primary outputs at the bottom of the DAG (See Figure 8). If there exists a sequential loop in the circuit, we need to detect and collapse it down into a single graph node. Graph edges are colored differently to indicate their status: critical nets are marked red while edges that are flip-flop insertable are marked green. A square appears when a flip-flop is inserted on an edge. A number appeared next to an edge indicates the number of edges connecting the nodes. Information that is not useful such as the sub-graphs within loops is eliminated from the graph to make it less cluttered. With a detail representation of connected graph nodes and edges, flip-flop insertion and flip-flop motion can be done intuitively as if the circuit is a combinational circuit.

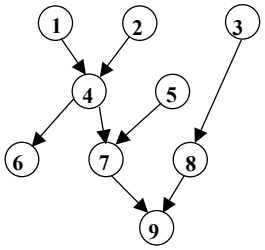

Figure 7: A Timing Graph

During flip-flop insertion, the user selects an edge and clicks the "Insert FF" button. A flip-flop is inserted in the specified location. Then EVE will then insert additional flip-flops in the DAG to maintain correct circuit behavior. For example, for the timing graph in Figure 7, if a flip-flop is inserted at edge *4→6*, additional flip-flops must be inserted at edges *4→7, 5→7, 8→9*. The resulting circuit still functions properly, with one additional cycle of latency across all paths. (Note that other flip-flop to edge assignments are also possible.) The additional flip-flop positions are determined based on a continuous forward and backward sweeping algorithm. The algorithm first inserts a flip-flop on the supplied edge, then it marks all its transitive fanin and fanout edges as processed. For back edges encountered during a forward traversal or forward edges encountered during a backward traversal, FFs are inserted if they are not marked as processed. This process continues until all edges are visited.

After flip-flop insertion, the user is able to move the newly inserted flip-flops forward or backward using the "up" and "down" arrow keys. When a flip-flop is moved forward or backward across a node, EVE will make sure that the circuit is still functioning properly, by moving other flip-flops affected by the move. For example, for the timing graph in Figure 7 , assume that flip-flops are inserted at edges *4→6, 4→7, 5→7, 8→9*. Now if the user moves the flip-flop from *5→7* to *7→9*, flip-flops at edge *4→7 and 5→7* will be removed, and a flip-flop is added to edge *7→9*.

The new circuit speed is calculated on the fly as the user changes the flip-flop positions. The actual placement of the inserted flip-flops is selected one by one in the order of the flip-flop's criticality. Each flip-flop position is then selected by an exhaustive search over a limited set of promising flip-flop positions to minimize critical path delay. When the user is satisfied with the flip-flop positions, the "Synthesize" button is pressed, and the inserted flip-flops are synthesized into the netlist and placed.

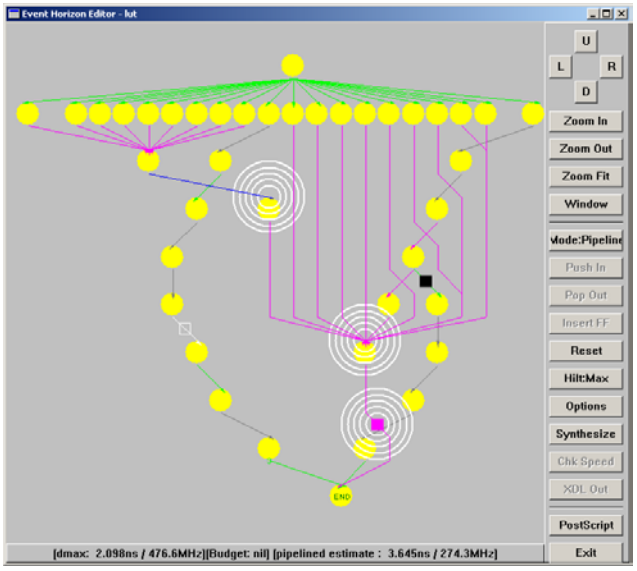

Figure 8: Screen Capture of the Pipelining Mode

5. EXPERIMENTAL RESULTS

In this chapter, we evaluate the quality of results EVE produced for both the Timing Exact Microscopic Placement (TEMP) and pipelining mode using eight circuits. Each circuit has approximately *250* or fewer LUTs. They are:

1. **Vision.** The Vision circuit is an FIR filter circuit used in a vision application presented in [8]. The circuit is highly pipelined using a pyramid structure of shifters and adders. It uses 142 LUTs and 241 FFs.
2. **Batcher**. The Batcher circuit is an ATM packet-sorting network that sorts incoming packets by serially comparing the bits of two packets. It is a component of the StarBurst ATM chip [1] project developed at the University of Toronto. It uses 252 LUTs and 455 FFs.
3. **Banyan.** The Banyan circuit is also a component of the StarBurst ATM chip [1] described above. It is a packet routing network that is responsible for delivering ATM packets to specific destination ports based on the address field stored in the ATM packets. It uses 165 LUTs and 311 FFs.
4. **Trap.** The Trap circuit is also a component of the StarBurst ATM chip [1]. It is a comparator circuit used to detect duplicated packets. It uses 187 LUTs and 470 FFs.
5. **Miim.** The Miim circuit [9] is an MII Management module of an Ethernet IP core obtained from OpenCores.org. The complete Ethernet IP core is designed for implementation of CSMA/CD LAN in accordance with the IEEE 802.3 standards. It uses 122 LUTs and 112 FFs.

6. **Div.** The Div circuit is an IP Core circuit generated by the Xilinx LogiCORE Pipelined Divider for Virtex Version 2.0 generator [17]. It has unsigned 8-bit dividend and divisor with integer remainder. It has throughput of one division per clock cycle with a latency of eight clock cycles. It uses 87 LUTs and 255 FFs.

7. **Dotproduct.** The Dotproduct circuit computes the dot product of two 8-bit 3D vectors. It is a part of a 3D ray-tracing application under development at the University of Toronto [5]. It uses 243 LUTs and 178 FFs.

8. **Crossproduct.** The Crossproduct circuit computes the cross product of two 4-bit 3D vectors. It is also a part of the 3D ray-tracing application [5]. It uses 129 LUTs and 126 FFs.

5.1 Baseline Circuits Generation

To evaluate EVE, we obtain a full implementation of a set of baseline circuits from an automatic push-button flow. These form the starting points for the manual editor, and the basis for comparison. We use the following state-of-the-art synthesis and placement and routing tools: Synplify Pro 6.20 [14] (one of the preeminent synthesis tools for FPGAs) for logic synthesis, and Xilinx Foundation 3.1i [19] for mapping, placement and routing. As an exception, the Div circuit does not require logic synthesis because it is directly generated from an IP Core netlist generator from Xilinx [17]. It is placed and routed in the usual way using the Xilinx backend tools. The baseline results are obtained following the steps below:

The input is VHDL or Verilog code obtained as described in Section 5:

1. Synthesize the HDL code using Synplify Pro 6.2, set to perform automated pipelining.
2. Place and route using Xilinx Foundation 3.3i Service Pack 7 tools.
3. Obtain final circuit frequency from P&R reports.
4. Repeat step (1) to (3), increasing frequency *10%* each time until the best frequency is obtained.
5. Use the frequency obtained in (4), place and route again using the Multi-Pass Place&Route (MPPR) option for ten runs, and pick the best resulting design.

The options used to generate the baseline circuits are recorded in Table 1. It is worth noting that these settings get the best results we could achieve in a push-button flow.

5.2 Results: Using TEMP Mode Only

We spent approximately two hours using the EVE editor to improve timing on each circuit. The machine used is a 1GHz Pentium-III PC with 512MB ram running Windows2000 and Xilinx Foundations 3.3i SP7. When we used the Timing Exact Microscopic Placement (TEMP) mode, we limited the area within which circuit elements can be placed. This ensures that we do not improve circuit speed at the expense of increase in occupied chip area. The results are summarized in Table 2 below. The first column of the table gives the circuit name, then the number of LUTs and flip flops, the original clock period and frequency, and then the new frequency after editing with EVE.

On average over the eight circuits, the circuit speed improved by 12.7% over the baseline. Below we discuss the properties of each circuit and the nature of the operations we performed using EVE to improve circuit performance.

1. **Vision.** By using the initial delay profile, we focused on improving the placement of circuit element on the k-most (k is about 1 to 5) critical paths and achieved good speed improvement. This is done by setting a slightly tighter timing budget, exposing more nets that are in timing violations. Also, when the critical path is in a carry chain, the reroute operation is observed to be able to relieve routing congestions effectively.

Table 1: Options used in Synthesis & P&R tools for Baseline Circuit generation

Options used for Synplify Pro:	Options used for Xilinx backend tools:	Frequency setting (for Synplify Pro & Xilinx backend):
Max Fanout = 100	P&R effort = 4	Vision = 200MHz
Disable I/O = on	Trim unconnected logic = no	Batcher = 330MHz
Pipelining = on	Replicate logic = yes	Banyan = 335MHz
FSM Compiler = on	MPPR initial placement seed = 1	Trap = 400MHz
Resource Sharing = on	MPPR P&R passes =- 10	Miim = 165MHz
	MPPR save N Best = 1	Div = 220MHz (for Xilinx backend only)
		Dotproduct = 150MHz
		Crossproduct = 220MHz

Table 2: Results for Using the TEMP Mode

Circuit	# LUTs	# FFs	Period (ns)	Freq (MHz)	New Freq (MHz)	% Change
Vision	142	241	4.92	203.3	224.8	10.6%
Batcher	252	455	3.06	326.8	380.1	16.3%
Banyan	165	311	2.94	340.7	395.3	16.0%
Trap	187	470	2.45	408.3	460.4	12.8%
Miim	122	112	6.16	162.4	168.5	3.8%
Div	87	255	4.65	215.1	229.6	6.7%
Dotproduct	243	178	6.74	148.4	173.3	16.8%
Crossproduct	129	126	4.54	220.1	261.4	18.8%
Average	166	269	4.43	238.7	268.8	12.7%

2. **Batcher.** The circuit is highly pipelined by design with a starting speed over 300MHz. It is interesting to note that even for circuits operating at such a high speed, their placement and packing can be improved further over the result generated with an automatic approach.
3. **Banyan.** The Banyan circuit has a high baseline circuit speed of 340MHz. To achieve speeds close to 400MHz, which approaches the physical speed limit of the FPGA, we need to place circuit elements no more than one CLB apart horizontally on the chip. This circuit orientation guides routing to use extremely fast nearest neighbor connections [11] which are present across neighboring horizontal CLBs.
4. **Trap.** The Trap circuit has the highest speed among all experimental circuits. However, we found out that many of the circuit elements are actually not optimally packed together in the same logic slice, and we were able to improve the circuit speed further to over 460MHz by doing packing/unpacking operations.
5. **Miim.** Although we tried very hard to improve the speed of the Miim circuit, we could only improve it by 3.8%. The critical path of the Miim circuit is in a single carry chain which loops back to itself tightly.
6. **Div.** Editing the placement of the design can only improve by 6.7%. The critical path has a carry chain feeding into another carry chain.
7. **Dotproduct.** The Dotproduct circuit is dominated by a large number of carry chains employed for multiplication. The initial placement was not very good, because carry chains were not aligned correctly for signal to flow through naturally. We rearranged the carry chains manually by simply examining the signal flow of the nets connecting the carry chains. A floorplanning tool may well have achieved similar gains.
8. **Crossproduct.** The circuit contains 4-bit multipliers synthesized into short carry chains. Again, as observed in the Dotproduct circuit above, the signal flow of the carry chains is poor. The initial circuit placement has a critical path spanning nine CLBs horizontally. Subsequent rearrangement of carry chains order greatly improved circuit speed.

In this section, we have shown the effectiveness of EVE's Timing Exact Microscopic Placement mode to further improve on high circuit speeds. From this experience, we make the following observations in order of effectiveness:

1. The ability to pack and unpack logic slices during placement and routing is essential.
2. An automatic floorplanning algorithm based on signal flow analysis should help timing. This observation has been made by FPGA design experts [7].
3. Focusing on improving delay on the critical path or the k-most critical paths is effective (as described above).
4. Floorplanning or placement editing tools should inform the user of any high speed routing resources available in the chip, so the user can make better micro-placement decisions.
5. Partial re-routing of timing-critical regions of the circuit is effective because routing resources in the surrounding area of critical paths can be freed up, and more critical nets can be reassigned faster routing resources.
6. The presence of un-occupied space near the critical path made the manual-editing task much easier.
7. The delay distribution (described in Section 3) helps the user identify improvement opportunities.
8. The more pipeline stages a circuit has, the easier the placement-editing task will be.

5.3 Results: Using Both TEMP and Pipelining Modes

In this section we present results obtained by using both the TEMP and pipelining modes of EVE to improve circuit speed. We only successfully obtained results for two circuits: D*iv* and M*ult*. While Div was used in the previous section, the Mult circuit is a new circuit designed to test EVE's pipelining ability. It is a non-pipelined 4x4bit multiplier built using full and half adder blocks. The circuit is synthesized using the procedure described in Section 5.1, except that we does not turn on the pipelining and retiming features of Synplify Pro. The resulting circuit does not contain any carry chains, and so it is highly pipeline-able by design. Results for the Vision and Miim circuits are not available because the critical paths are inside loops, which cannot be pipelined. Results for the Batcher, Banyan and Trap circuits are not gathered because the circuits are already sufficiently pipelined. Results for the Dotproduct and Crossproduct circuits are not available due to software instability.

Table 3 shows the summary of results.

These results are gathered after one stage of pipeline insertion. For the already well-pipelined Div circuit, minimal performance increase after the pipelining operation is expected. For the Mult circuit, however, we achieve a performance increase of 42.2%. It proves that the pipelining feature is functional. However, pipelining at the logic synthesis level could probably have increased the speed of the Mult circuit to about 220MHz. Pipelining at the logic synthesis level is still the preferred choice over pipelining at the physical level. But for extremely high-speed circuits, pipelining at the physical level may be the only way to obtain accurate post-placement/routing delay information for performing optimal pipelining operation. The current user interface of EVE's pipelining mode is very limited. It can demonstrate basic ideas for pipelining at the physical level, but the actual pipelining operation is not easy to do. Future research that looks at the Synthesis stage in the Event Horizon methodology may make better use of the pipelining feature that EVE currently offers.

Table 3: Results for Using both TEMP and Pipelining Modes

Circuit	# LUTs	# FFs	# FFs added	Freq (MHz)	New Freq (MHz)	% Change
Vision	142	241		224.8	N/A : critical path in loop	
Batcher	252	455		380.1	N/A : already well pipelined	
Banyan	165	311		395.3	N/A : already well pipelined	
Trap	187	470		460.4	N/A : already well pipelined	
Miim	122	112		168.5	N/A : critical path in loop	
Div	87	255	66	229.6	237.7	3.5%
Dotproduct	243	178		173.3	N/A : due to tool instability	
Crossproduct	129	126		261.4	N/A : due to tool instability	
Mult	39	23	38	123.1	175.1	42.2%

6. CONCLUSION AND FUTURE WORK

In this paper, we present a tool for manual packing, placement and pipelining, with the goal of aiding designers seeking very high-speed implementation of circuits. We implemented the method in a manual editor called EVE. EVE provides an intuitive GUI interface, which can perform powerful operations such as packing/unpacking, placement, and routing operations. It integrates tightly with the Xilinx backend tools to allow editing of real commercial FPGA circuits based on the Xilinx Virtex-E architecture. It gives the user full low-level control of the circuit, and it provides instant real timing feedback while during placement editing and pipelining operations. It is timing-budget aware and it provides useful features to help designers meet the timing goal. Experimental results show that EVE is capable of improving the maximum operating frequency of real circuits by up to 19%, and we show that it improves a group of eight circuits on average by 12.7%. The pipelining mode in EVE demonstrates important ideas involved in pipelining at the physical level. EVE will serve as a good reference CAD tool for further research into the area of high-speed manual-assisted design tools. In the future, we will explore the use of this framework together with logic synthesis to perform the ground-up design creation as articulated by Von Herzen. We may also extend the tool to support newer device architectures.

7. ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from NSERC, MICRONET and Xilinx. We would also like to thank Dr. Kevin Chung of Xilinx for his timely and helpful advice on the use of Xilinx backend tools.

8. REFERENCES

[1] P. Bade, W. Chow, P. Kundarewich, N. Saniei, A. Wang, "Starburst ATM Chip project at University of Toronto", October 2000. (Available from http://www.eecg.utoronto.ca/~wangk/report.ps)

[2] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer Academic Publishers, February 1999.

[3] W. Chow, "EasyGL For Windows," 2001. (Available from http://www.eecg.utoronto.ca/~choww/easygl.html)

[4] W. Elmore, "The Transient Response of Damped Linear Networks," *Journal of Applied Physics*, Vol. 19, pp. 55 - 63, Jan 1948.

[5] J. Fender, University of Toronto, Bachelor's Thesis in progress, working title: "A 3D Ray Tracing Engine on TM-3", April 2002.

[6] R. Hitchcock, G. Smith and D. Cheng, "Timing Analysis of Computer-Hardware," *IBM Journal of Research and Development*, Jan. 1983, pp. 100-105.

[7] T. Maniwa, "FPGA 2000 Panel," *ISD Magazine*, February 2000. (Available from http://www.isdmag.com/articles/fpga0002.html).

[8] R. McCready, J. Rose, "Real-Time Face Detection on a Configurable Hardware System," *FPL 2000*, pp 157-162, August 2000.

[9] OpenCores.org, "Ethernet MAC 10/100 Mbps project," March 2001. (Available from http://www.opencores.org/cores/ethmac/).

[10] J. Ousterhout, G. Hamachi, R. Mayo, W. Scott, G. Taylor, "Magic: A VLSI layout system," in Proc. of 21st Design Automation Conf., pp. 152-159, 1984

[11] A. Roopchansingh, University of Toronto, Master's Thesis in progress, working title: "Research on Nearest Neighbor Connections", 2002.

[12] S. Rubin, "An Integrated Aid for Top-Down Electrical Design," VLSI '83 (Anceau and Aas, eds), North Holland, Amsterdam, pp.63-72, August 1983

[13] J. Rubinstein, P. Penfield and M. Horowitz, "Signal Delay in RC Tree Networks," *IEEE Trans. On CAD*, 1983, pp. 202-211

[14] Synplicity, Inc, "Synplify Pro 6.20," 2000. (Available from http://www.synplicity.com/literature/pdf/SynPro_datasheet.pdf).

[15] B. Von Herzen. Signal processing at 250 MHz using high-performance FPGA's. In *Proc. ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays (FPGA'97)*, pages 62-68.

[16] B. Von Herzen, "Signal Processing at 250 MHz Using High-Performance FPGA's," in *IEEE Trans. on VLSI Systems*, Vol 6, No.2, pp. 238-246, June 1998.

[17] Xilinx Corporation, "Pipelined Divider Core", May 1999. (Available from http://www.xilinx.com/dsp/docs/pipediv.pdf).

[18] Xilinx Corporation, “FPGA Editor Guide, V3.1i,” 2000 (Available from http://toolbox.xilinx.com/docsan/3_1i/pdf/docs/fpg/fpg.pdf.)

[19] Xilinx Corporation, The Xilinx Foundation Series 3.1, 2000. (Available from http:// www.xilinx.com).

[20] Xilinx Corporation, “Virtex-E 1.8V FPGA Family: Detailed Functional Description,” 2001 (Available from http://www.xilinx.com/partinfo/ds022-2.pdf.)