TIERS: Topology IndependEnt Pipelined Routing and Scheduling for VirtualWire™ Compilation

Charles Selvidge, Anant Agarwal, Matt Dahl, Jonathan Babb
Virtual Machine Works, Inc.
1 Kendall Sq. Building 300
Cambridge, MA 02139
email: selvidge@vmw.com

Abstract

TIERS is a new pipelined routing and scheduling algorithm implemented in a completeVirtualWire™ compilation and synthesis system. TIERS is described and compared to prior work both analytically and quantitatively. TIERS improves system speed by as much as a factor of 2.5 over prior work. TIERS routing results for both Altera and Xilinx based FPGA systems are provided.

1 Introduction

Reprogrammable, multiple-FPGA systems offer a means to produce application-specific digital hardware without incurring the effort and expense associated with a typical custom hardware design project. Nonetheless, the process of implementing the behavior of a large, monolithic system via a collection of FPGAs is challenging for reasons related below:

- **Pin and Interconnect Limitations:** Satisfaction of constraints due to fixed interconnect limitations, ranging from FPGA pincounts to backplane or cable capacities, makes mapping and partitioning a large design onto multiple FPGAs difficult. These constraints tend both to make the partitioning process unpredictable and unstable to design changes, and to limit the utilization of FPGA logic resources, leading to large hardware requirements.
- **Unpredictable Timing:** The inherent sensitivity of FPGA timing to final placement and routing is magnified in systems where signals may propagate combinatorially across many FPGAs. Systems which are free of timing violations must be constructed despite this timing variability.
- **Methodology and Infrastructure:** Methodology and Infrastructure specific to the use of multiple-FPGA systems is neither well-developed nor well-integrated.

VirtualWire™ technology [3,5,7], a methodology and supporting software infrastructure for the construction and use of multiple-FPGA systems, addresses these challenges. Given a monolithic input description, it synthesizes a behaviorally equivalent multi-FPGA system. The system is fully retimed to a single high-speed clock. This retiming allows interconnect to be multiplexed, removing any fixed interconnect limitations and related partitioning constraints. Registers are inserted at each FPGA crossing, producing combinatorial timing paths which involve at most a pair of FPGAs. Retiming also provides a means of controlling the internal or IO timing behavior of the system at a fine granularity, if required. This process is embodied in a compiler which is functional today.

Central to the compilation process is the requirement to route information which must travel between FPGAs across the multiplexed system interconnect. TIERS is a new algorithm for routing this information and scheduling the interconnect resources. In comparison to the algorithm described in the original work [3,5,7], referred to as PhaseRoute

Proceedings of the Third International ACM Symposium on Field-Programmable Gate Arrays (FPGA'95)
0-7695-2550-4/95 $ 10.00 ACM

within this paper, TIERS demonstrates significant performance improvements, as much as a factor of 2-2.5 on estimated and measured cases.

The remainder of this paper is structured as follows: Section 2 presents a detailed description of the TIERS algorithm and the routing problem it solves. Section 3 makes an analytical comparison between TIERS and PhaseRoute, deriving and comparing performance bounds for the systems produced by these algorithms. Section 4 presents performance results from TIERS and compares these results to actual and estimated results from PhaseRoute. Section 5 offers conclusions.

2 Pipelined Routing and Scheduling

2.1 Specification of Scheduling Problem

TIERS is a network scheduling algorithm. It accepts two graphs as input: a design graph specifying the connectivity of an input design, and a target graph, specifying the interconnect topology of a particular multiple-FPGA target system. It schedules the time-multiplexed reuse of the target interconnect network to achieve the connectivity of the input design.
The design graph is derived from a partitioned logic netlist for an input design. The design graph has one node for each FPGA in the target system, each node representing all the design logic to be assigned to the FPGA. The graph has an arc, referred to as a **link**, for each bit of information to be transmitted from one FPGA to another. Thus, in general, there are many arcs between any pair of nodes in the design graph.

The target graph has one node for each FPGA in the target system and an arc between each directly connected pair of FPGAs. These arcs, referred to as **channels**, are annotated with a width. The width specifies the number of signals which can be carried by a channel on each multiplexing timeslice and is simply the number of wires connecting the associated pair of FPGAs in the target system.

In addition to the design and target graphs, TIERS uses two other forms of information. There is a defined one-to-one correspondence between each node in the design and target graphs representing the placement of design logic into particular target FPGAs. There is a dependence relationship between input and output links of each node in the design graph, indicating a combinatorial path between the input and output within the corresponding logic. We will refer to the set of output links which depend on an input link j as *Dep(j)*, and the inverse of this relationship, inputs depended upon by output j, as *IDep(j)*.

The task of TIERS is to route and schedule each link by identifying a sequence of one or more channels and associated timeslices on these channels which can be used to transmit the value represented by the link from its source to its destination FPGA. This process will be referred to as routing, each channel/timeslice sequence as a route and the aggregate of all routes as a schedule.

Routing is subject to three constraints:

- **Dependence:** The final timeslice in the route for a link j must precede the initial timeslice in the route for any link in *Dep(j)*.
- **Causality:** A link traversing multiple channels to reach its destination must do so at increasing timeslices.
- **Capacity:** A channel can carry no more links than its width in any timeslice.

Figure 1 illustrates a design graph, target graph, node correspondence information and dependence information.

2.2 TIERS Algorithm

The TIERS algorithm is a greedy algorithm which proceeds in two phases. It orders the links and then routes each link in the order specified, reserving channel resources.

2.2.1 Critical-Path Sensitive Link Ordering

Link ordering achieves two goals, it aids in satisfying dependence constraints and incorporates a heuristic which serves to produce high quality schedules. Dependence constraints require the departure of outputs to occur after the arrival of any inputs upon which they depend. This in turn is most readily satisfied if the ordering of link scheduling satisfies the partial ordering that any link *j* precedes all links in *Dep(j)*. In the absence of combinatorial cycles, such a partial order exists.

In order to produce a schedule which uses few timeslices, links on critical paths are given priority in the routing order. Each link is assigned a depth. The

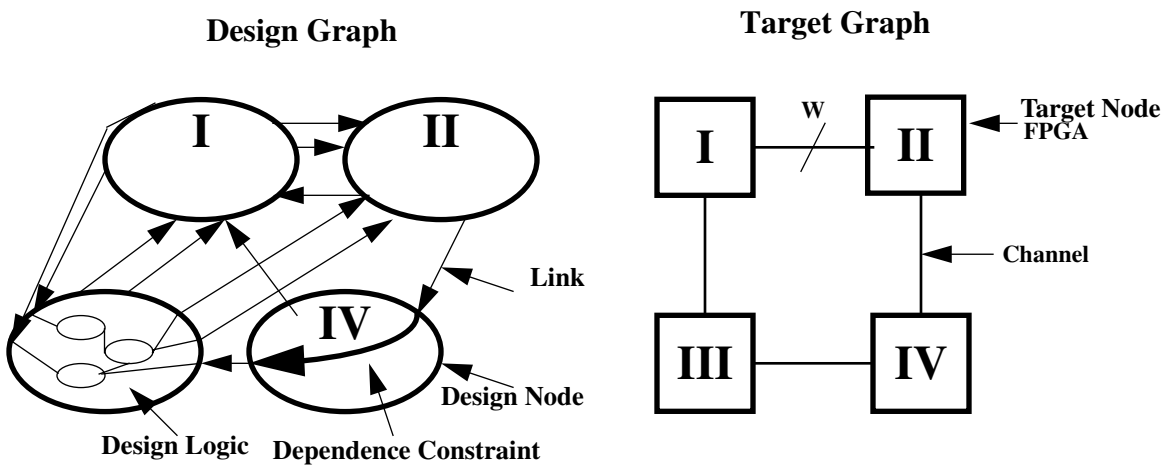

FIGURE 1 Design and Target Graphs with Correspondence and Dependence Information

depth of a link j is computed as one greater than the maximum of the depth of any link in *Dep(j)*, as indicated in Equation I.

$$\text{I} \qquad \begin{aligned} \text{Depth}(j) &= 0 \quad \text{if} \quad \text{Dep}(j) = \varnothing \\ \text{Depth}(j) &= 1 + \max_{k \in \text{Dep}(j)} \{\text{Depth}(k)\} \end{aligned}$$

Links are sorted according to depth with larger depths occurring earlier. Notice that depth order is consistent with the dependence partial order since the depth of a link is necessarily greater than the depth of all links in *Dep (j)*.

2.2.2 Link Routing/Scheduling

Several tasks are associated with the routing of an individual link, including: determination of earliest legitimate routing time, identification of a route which starts at or after this time, and reservation of channel capacity consumed by this route.

Departure Time Computation

The earliest legitimate departure time of link *j*, *Depart(j)*, is one unit later than the latest arrival time, *Arrive(k)*, of any link *k* in *IDep(j),* as indicated in Equation II.

$$\text{II} \qquad \begin{aligned} \text{Depart}(j) &= 0 \quad \text{if} \quad \text{IDep}(j) = \varnothing \\ \text{Depart}(j) &= 1 + \max_{k \in \text{IDep}(j)} \{\text{Arrive}(k)\} \end{aligned}$$

Requiring a route which starts on or after *Depart(j)* in time satisfies dependence constraints.

2.2.3 Link Routing

Many packet routing algorithms are based on weighted shortest path algorithms on network graphs where weights are based on channel utilization levels [4]. If a simple weighting scheme is used with wieghts of 1 for channels with remaining capacity and – for full channels, the shortest path algorithm can be reduced to a breadth first search which excludes full channels from the network graph. Our algorithm identifies routes by performing such a breadth first search on the target graph, starting at the node representing the source FPGA and stopping once the target FPGA is reached. The search has been augmented to satisfy channel capacity constraints. Each channel maintains a record of its utilization within each timeslice and this information can be used to determine whether it is available for use in any particular timeslice. The breadth-first search associates a time parameter with each node discovered on the search. This time is one greater than the time for the node from which this node was reached in the search. Channels are only traversed in the search if they have capacity at the time associated with their source node.

Due to capacity constraints, it is possible that a search starting at some time may not find a route. It is also possible even if a route is discovered that an alternative route, starting at a later time, might result in an earlier arrival. To accommodate this, the search process described above is embedded in a loop that tries searches at successive times until it finds the route which arrives earliest. The length of the route is compared to the length of the shortest path from source to destination and the search is continued for additional timeslices equal in duration to the difference in lengths, thus guaranteeing that the earliest arriving route is found.

2.2.4 Reservation

Once a route has been identified, the channel utilization for each channel on the route must be

updated at the relevant time. This is achieved by traversing the channels in the final route and incrementing the utilization information for the timeslice during which the channel is used.

3 Comparison of TIERS and PhaseRoute

This section derives analytic lower bounds on the performance of systems resulting both from TIERS and from PhaseRoute, the original virtual wire routing algorithm. Routing performance is effected both by critical path length of the design graph and channel capacities of the most heavily utilized channels. Each of these factors can be used to produce a performance bound. In the designs examined, critical path length appears to be the dominant factor so we present only this bound.

Define L as the maximum depth of any node in the design graph, where depth is as computed in Equation I.

Define P as the maximum pathlength weighted depth of any node, where pathlength weighted depth is computed by the modified form of Equation I below, in which $Dist(j,k)$ is the length of the shortest path in the target graph between the FPGAs associated with design nodes j and k.

$$\text{III} \qquad \begin{aligned} PDepth(j) &= 0 \quad \text{if} \quad Dep(j) = \varnothing \\ PDepth(j) &= \max_{k \in Dep(j)} \{PDepth(k) + Dist(j,k)\} \end{aligned}$$

The pathlength weighted depth of any particular node is a minimum bound on the routing time of the longest path emanating from the node.

Define t_l as the maximum combinatorial latency of a design graph node and t_h as the time required for a routing timeslice.

3.1 TIERS

A lower bound on system clock period for systems produced by TIERS is simply:

$$\text{IV} \qquad T_{clk} \geq L \times t_l + P \times t_h$$

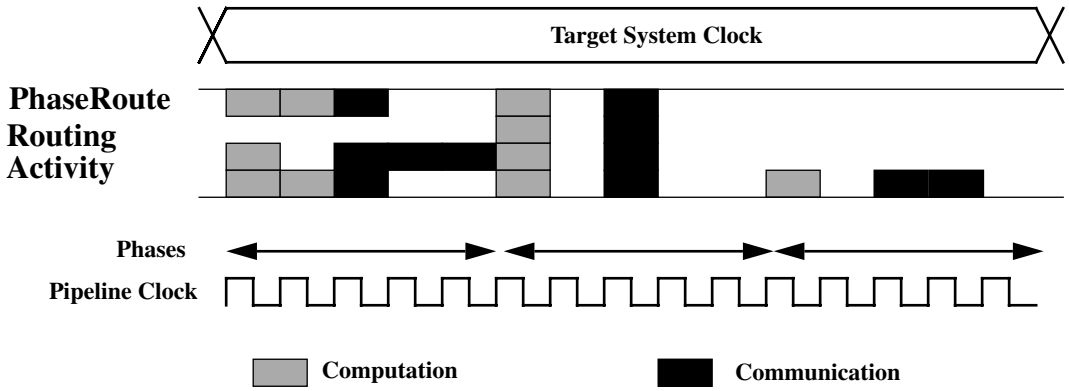

FIGURE 2 **PhaseRoute Routing Activity Profile**

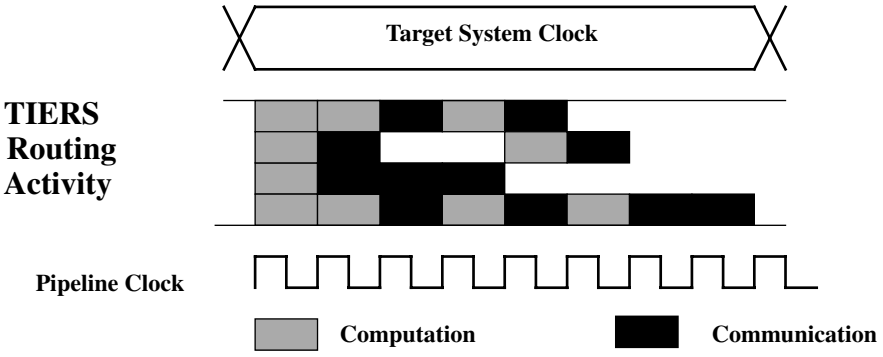

FIGURE 3 **TIERS Routing Activity Profile**

The value of *P* is typically close to the product of *L* and the average pathlength over all links in the schedule. The bound can be recast into the following form:

$$T_{clk} \geq L \times \left(t_l + p_{av} \times t_h \right) \quad \text{V}$$

3.2 PhaseRoute

PhaseRoute employs routing phases consisting of two non-overlapping activities, computation and communication. The communication phase is broken into individual high-speed cycles, just as in TIERS. Dependence constraints are satisfied in PhaseRoute by routing dependent links in distinct phases, thus a minimum of *L* phases is required by PhaseRoute. All phases are required to have the same length and they must accommodate the worst case pathlength of any individual link, which in general is the diameter of the topology graph. Defining *D* as the diameter of the graph and worst case pathlength, one arrives at the performance bound below:

$$T_{clk} \geq L \times \left(t_l + D \times t_h \right) \quad \text{VI}$$

Equations V and VI differ only in the use of *D* and p_{av}. P_{av} is bounded above by *D* and will be less than *D* in general.

3.3 Comparison of TIERS and PhaseRoute

TIERS has several advantages in comparison to PhaseRoute. Figures 2 and 3 illustrate routing activity during a system clock period for systems produced by PhaseRoute and TIERS, respectively, These figures show the following differences between TIERS and PhaseRoute.

TIERS can initiate computation and subsequent routing as soon as a signal arrives at a destination, whereas PhaseRoute must wait for a new phase.

TIERS supports simultaneous, overlapped computation and communication in different parts of the system, whereas PhaseRoute performs exclusively computation or communication at any given time within the system.

TIERS can readily be adapted to allow distinct combinatorial paths through user logic different amounts of time for propagation, whereas PhaseRoute requires a worst-case time for all paths.

An additional difference, not evident from the bounds analysis or figures, is that TIERS can change the purpose of any wire within the system at the granularity of a single timeslice, whereas in PhaseRoute it is dedicated to a particular FPGA-FPGA path for the duration of a phase. This flexibility has benefits when bandwidth limits performance.

4 Results

TIERS is implemented within a functional VirtualWire™ compilation system which can target arbitrary multiple-FPGA targets containing either Altera Flex8000 or Xilinx 4000 series parts.

4.1 Routing Results

This section presents results for several netlists compiled onto a variety of target system configurations. In all cases, data produced by TIERS is compared to both the TIERS lower bound, the PhaseRoute lower bound, and where possible, to analogous PhaseRoute data.

Three design netlists are considered:

- **Sparcle** is an augmented Sparc microprocessor consisting of approximately 18K logic gates plus a three-ported register file [2].
- **Sparcle_Nomem** is Sparcle with the register file removed.
- **Alewife** is a multiprocessor cache-controller consisting of approximately 110K gates of logic and 11 memories of varying configurations[6].

Five target system configurations are evaluated:

- **RIPP:** a RIPP10 card, produced by Altera, consisting of 8 Altera 81188 FPGAs connected by a Clos network [1].
- **XClos5:** an 8 node Clos network of Xilinx 4013 FPGAs [8], with three of the nodes removed.
- **MIT20:** a 5x4 mesh of 20 Xilinx 4005 parts with 8 bit data paths. This configuration is described in [7].
- **X64**: an 8x8 two-dimensional torus with additional connections between FPGAs separated by two and four in each torus dimension.
- **X48:**an X64 with 16 nodes removed.

Sparcle on the Altera RIPP10 card has been run using the card's bus interface to provide and consume vectors. Other topologies have been verified in simulation.

Figures 4 and 5 illustrate the performance of selected combinations of the test netlists and topologies.

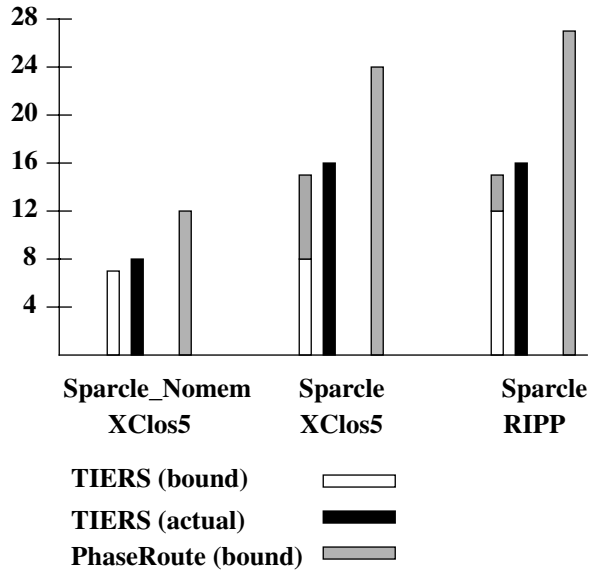

Figure 4: Sparcle Routing Results

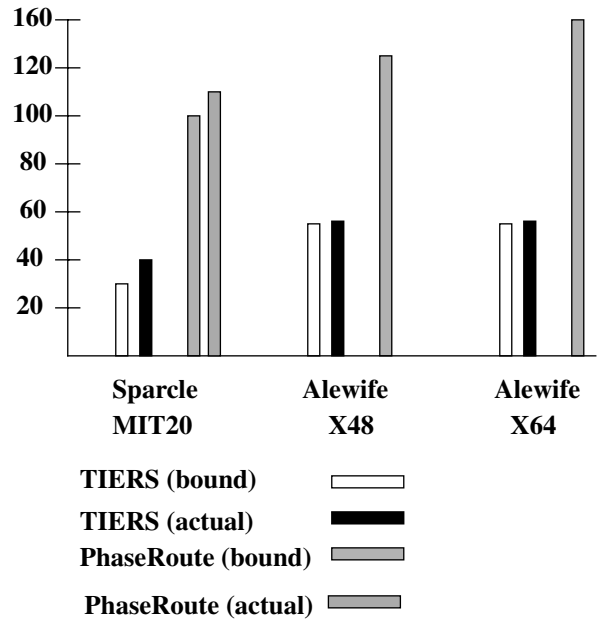

Figure 5: Sparcle and Alewife Routing Results

4.2 Analysis

Three of the test results, Sparcle_Nomem and the two Alewife tests, are critical path limited. In these cases, TIERS nearly achieves the performance bound based on critical path length. The bound is exceeded by 1 timeslice in the first two of these cases and by 2 in the third.

In two of the examples, Sparcle on XClos5 and RIPP, there is a readily computed bandwidth bound which exceeds the critical path related bound. The memory from Sparcle is mapped onto a single SRAM which is twelve-way multiplexed. The combination of twelve memory access timeslices, a timeslice for memory bus turn-around, and the critical path leading up to the first memory access produces a lower bound on performance of 15 timeslices. (The lengthening of the bound due to memory access is illustrated by hash-marks within the bound in Figure IV). For these tests, as well, TIERS achieves within 1 cycle of a derived performance bound.

In the case of Sparcle on the MIT20 configuration, the schedule produced by TIERS exceeds the critical path length bound by about 40%. MIT20 has far fewer wires per FPGA than the other target topologies, and the level of multiplexing is much higher. As with the Sparcle examples above, there are bandwidth limitations which cause the result of even an optimal algorithm to exceed the critical path length bound. Unlike these examples, a better bound on optimal performance cannot be easily computed.

Across the full test suite, TIERS exhibits nearly optimal performance on five of six examples. It performs particularly well when designs are highly critical path limited since it is optimized for this case. In the bandwidth limited cases, the results are less consistent. For the two memory bandwidth limited cases, the critical path to the memories was on the design critical path and was optimized. In the final case, bandwidth limitations were present throughout the system. In this situation, the algorithm leaves some room for improvement.

In comparison to PhaseRoute, the results of TIERS are consistently better,as is expected. Performance varied from 50% better on the small designs to better than a factor of 2 on the large designs and these are comparisons of actual TIERS performance to a PhaseRoute lower bound. In the one place where a direct comparison could be made, the performance difference was above a factor of 2.5.

5 Conclusions

This paper presents TIERS, a new routing algorithm for VirtualWire™ compilation. TIERS is designed with heuristics to optimize the routing of design critical paths. This technique results in near-optimal performance across a set of designs and target topologies, with one exception. The exception is a highly bandwidth-constrained routing problem resulting from a topology with unnecessarily narrow routing channels.

TIERS exhibits routing performance superior to that of PhaseRoute, the prior work, by between a factor of 2 and 2.5. Further improvements in VirtualWire™ system performance may be achieved but they will not be achieved through routing advancements, since TIERS nearly achieves performance bounds inherent in its inputs already. Further performance improvements must be achieved through better partitioning and placement technology which produce the design graph input and design node/ target node correspondence used by TIERS.

6 Bibliography

[1] Altera, Inc., RIP User Manual. December 18, 1993

[2] A. Agarwal, et al. Sparcle: An Evolutionary Processor Design for Multiprocessors. *IEEE Micro*, 13(3):48-61, June 1993.

[3] J. Babb et al. Virtual Wires: Overcoming Pin Limitation in FPGA-based Logic Emulators. In *Proceedings, IEEE Workshop on FPGA-based Custom Computing MAchines*, pages 142-151, Napa, CA April 1993. IEEE. Also as MIT/LCS TM-491, January 1993.

[4] Bertsekas,D. and Gallager, R., Data Networks. Prentice Hall, Inc., pages 308-340, 1987.

[5] M. Dahl et al. Emulation of the Sparcle Microprocessor with the MIT Virtual Wires Emulation System. In *Proceedings, IEEE Workshop on FPGAs For Custom Computing Machines*, pages14 -22, Napa Valley California, April 1994. IEEE Computer Society, IEEE Computer Society Press.

[6] J. Kubiatowicz. User's Manual for the A-1000 communications and Memory Management Unit. ALEWIFE Memo No. 19, Laboaratory for Computer Science, Massachusetts Institute of Technology, January 1991.

[7] R. Tessier et al. The Virtual Wires Emulation System: A Gate-Efficient ASIC Prototyping Environment. Submitted to *1994 ACM International Workshop on Field-Programmable Gate Arrays*, Berkeley, CA, February 1994, ACM.

[8] Xilinx, Inc., The Programmable Logic Data Book. 1994.