Interconnect Enhancements for a High-Speed PLD Architecture

Michael Hutton, Vinson Chan, Peter Kazarian, Victor Maruri, Tony Ngai, Jim Park, Rakesh Patel, Bruce Pedersen, Jay Schleicher and Sergey Shumarayev

Altera Corporation, 101 Innovation Drive, San Jose, CA 95134 (mhutton@altera.com)

ABSTRACT
As programmable logic grows more viable for implementing full design systems, performance has become a primary issue for programmable logic device architectures. This paper presents the high-level design of Dali, a PLD architecture specifically aimed at performance-driven applications. We will present significant portions of the background research that contributed to our architectural decisions, an overview of the core routing architecture and benchmarking experiments used to evaluate the prototype device.

Keywords
Programmable logic, FPGA, architecture, interconnect.

1. INTRODUCTION
The design of a PLD architecture involves many conflicting goals, one of which is the tradeoff between performance and efficiency. In fact, PLDs themselves are in the tradeoff space between truly flexible but slow microprocessors and high-performance but inflexible ASICs.

This paper discusses research experiments aimed at designing a PLD architecture that is as fast as possible without unreasonably increasing die-size or non-routability (i.e. cost). The Dali architecture discussed in this paper forms the basis for the recently announced Mercury PLD family from Altera, a commercial product. However, we will continue to use the name Dali to emphasize that all of the results reported on here are based upon the prototype architecture and software. This paper concentrates on the core routing architecture of Dali, and we will not try to cover modifications to the I/O structure in this discussion.

In a PLD the majority of the electrical delay occurs in interconnect and interconnect switching, so our primary concentration will be on routing enhancements relative to previous-generation PLD families. The basic interconnect (wire) delay incurred by a critical path is common between PLD and ASIC devices. The time spent in interconnect switching is a PLD-specific design tradeoff. Flexibility introduced by interconnect switching is paid for with the additional loading delay which it puts on the routing. The key item in the design of a PLD routing architecture is to always provide enough flexibility to implement a high proportion of designs, but not so much flexibility that performance or area suffers.

The target density for our architecture family is approximately 2000 to 15000 LEs. This is an important design parameter because decisions related to the degree of hierarchy, accessibility to re-buffering signals and the length of interconnect wires can be quite different at high vs. low density spaces. Since current high-density PLDs reach close to 75,000 LEs [2,22] our target range would be considered a mainstream rather than high-density PLD family.

There are a number of ways to make PLD interconnect faster. One approach is to make the device less programmable by replacing switches with less flexible or even dedicated connections, or by restricting availability of certain types of routing. A second approach is to optimize wires for speed with width, space and shielding. Both result in an increase in overall cost: the first manifests itself as loss of routability, the second as an increase in die-size. The amount to which we are willing to do this and the ways in which we can accomplish it comprise the major topics of this paper, and can be categorized as follows:

- Hierarchy: choosing the correct level of hierarchy for the density range of the family, in our case a semi-hierarchical organization rather than flat or fully hierarchical.
- Heterogeneous routing: some wires are fast, but more expensive. Others are slower, but more efficient/dense. The place and route software is responsible for utilizing the wires appropriately by placing critical paths on the fast wires and non-critical paths on the regular wires.

- Dedicated wiring: dedicated short connections between adjacent LEs and dedicated carry-chain and multiplier circuitry allow special purpose critical functions to be implemented with faster interconnect.

- Greater variety of wiring resources: Different netlist connections are of different lengths and fanout. Our goal is to provide the best possible mix of routing resources for the requirements of most designs.

The organization of the paper is as follows: Sections 2 through 5 cover the topics just mentioned which related to the core routing architecture, and Section 6 then summarizes the core architecture of the device and gives experimental results from in-house evaluation of the prototype device. We make some closing remarks in Section 7.

2. HIERARCHY / BASE ARCHITECTURE

Before beginning any detailed experimentation, it is necessary to describe a base architecture. For our purposes this really means deciding on the amount of hierarchy and a general theme to the organization of the interconnect. There is a spectrum of known architectures from which to choose our starting point.

The APEX architecture [2,11] is a fundamentally hierarchical device. Groups of LEs are clustered into fully-connected LABs, which are grouped into MegaLabs. Global or high-level interconnect can only enter a MegaLab by switching onto MegaLab interconnect and then into individual LE inputs. Horizontal and vertical wires in this type of architecture are full-length, half-length, or quarter-length rather than staggered.

At the opposite extreme from APEX is a flat or island-style architecture, such as the XC4000 presented by Trimberger *et.al.* [20]. Though provision is made for longer lines in this architecture, the primary communication is through short wires which stitch together through switching blocks to go longer distances. The part is symmetric, in that vertical and horizontal connectivity is identical.

The FLEX 10K architecture (similar to the FLEX 6K presented in [21]) is semi-hierarchical. Though LEs are still grouped into LABS with full connectivity, the interconnect is row-biased. Unlike APEX, top level full or half-length horizontal lines can communicate directly with LEs in a LAB, but similar to APEX vertical lines must switch onto a horizontal line before entering a LAB. See Figure 1. In FLEX 6K adjacent LABs can also communicate by interleaving their local connections to accomplish unit-length horizontal wires.

An alternative definition of semi-hierarchical was given by Kaptanoglu *et. al.* [12]. Their architecture is hierarchical at the top-level, but flat in 16x16 blocks at the lowest level.

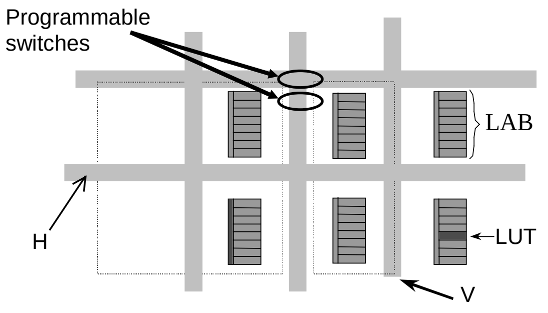

Figure 1. Semi-hierarchical architecture similar to FLEX10K or FLEX6K, and our base architecture for Dali.

We experimented with a range of architectures throughout the available space. Some comments on the various tradeoffs are the following. A short-line architecture will always be the fastest and most dense for unit connections. However, when more connections are required a connection must go through multiple switches to go even a relatively short distance. A long-line hierarchical architecture is very good for medium to long connections, but pays the extra price in both density and area for the short connections. At a very high level, one could argue that unit-length connections are always the most efficient for density alone. However, since modern processes have 8 or more layers of metal, this makes some naïve assumptions about the tradeoff of metal for switching.

In actuality, the choice of which base architecture to start with will always be filled with caveats, experimental bias due to assumptions and, in particular, software bias towards already existing parts. In this case, after numerous experiments with HSPICE and software place and route, we chose a semi-hierarchical architecture roughly modeled on the FLEX 10K/6K architecture of Figure 1. To fill in the details, our basic logic element (LE) is a 4-input LUT and DFF pair as is common in most PLDs. These are grouped into clusters, called LABs, of size 10 with full programmable connectivity (local lines). Connections outside of a LAB are accomplished with global wires: half-horizontal or HH lines, global (full-length) horizontal or GH lines, and global vertical or GV lines. GV lines can turn onto the horizontal resources with programmable switches at their intersection points.

Though the FLEX model serves as our starting point, we will make a significant number of modifications and improvements upon the architecture.

3. HETEROGENEOUS WIRES

It is well known in VLSI that one can optimize certain wiring connections by spending greater area in their implementation. Some common techniques are to increase the width and spacing of long interconnect wires, introduce optimally sized and spaced buffering for the required loading on the wire, or laying out more critical connections on faster metal layers.

The difficulty with these approaches on a PLD is that routing is intended to be generic. We don't know which wires are going to be critical until after a user's design has gone through place and route, and different user-designs will have different critical paths. It is infeasible (i.e. too expensive) to optimize all wires in the part for delay.

The major innovation in the Dali architecture is to provide wires in the routing architecture that are functionally identical, but have different delay characteristics. We can then rely on our place and route tools to properly balance the use of the fast and slow wires. The keys to this approach are in deciding how many wires to make faster, and how to organize the connectivity so that both fast and slow wires are reachable from all areas of the chip.

For a number of designs targeting this device and for a fixed placement based on FLEX 10K, we performed experiments as summarized by the following algorithm:

```
For varying p (percentage fast wires per row or column):
    Loop
        Do timing analysis and determine most critical path.
        For each non-fast wire W on the critical path
            If W's row or col has fast wires remaining,
                "promote" W to fast by halving its delay.
    Until critical path cannot be sped up.
    Record new critical path length
End For
```

The results of this experiment are shown in Figure 2. Each test design generates one data point for each value on the X axis. When we allow maximum of 5% of the global wires to speed up, we see the normalized critical path delay decrease by between 0% and 30% of the original delay, depending on the design, with an average of about 11%. The trend-line shows the average decrease in critical path length for increasing fast wires. The dotted lines show the curve for the two designs which eventually showed the most and least benefit.

The two main observations from this data are that we can gain an approximately 30% decrease in critical path delay, on average, by making only 20% of the wires fast, and that speeding up more than 20% of the wires will have only a marginal effect on the critical path. Figure 3 emphasizes these diminishing marginal returns – the first 5% fast wires affect all 33 designs in the experiment, whereas increasing from 5% to 10% affects only 32 designs. Increasing from 45% to 50% affects only one design. Figure 4 views marginal returns from the perspective of overall effectiveness. Though Figure 3 shows diminishing effects by number of designs it underestimates the acceleration of the decreased returns arising from less benefit even on those designs that are affected.

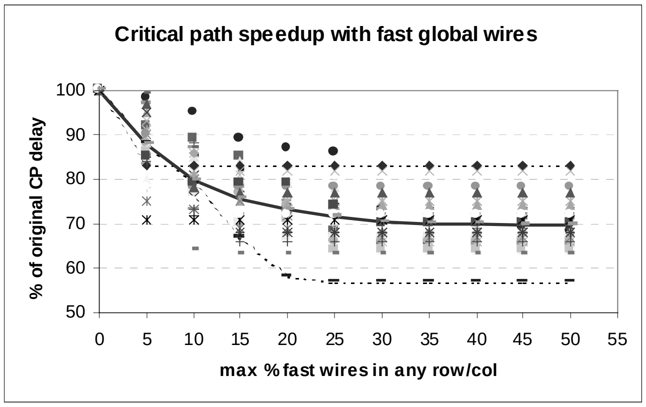

Figure 2. Effect on test designs of speeding up given proportions of row and column global interconnect.

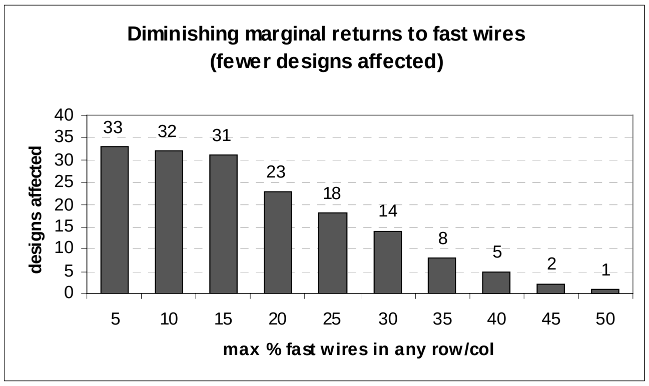

Figure 3. Diminishing marginal returns to fast wires.

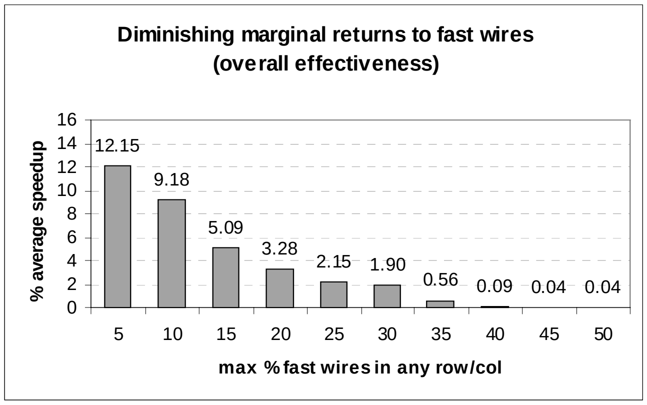

Figure 4. Diminishing average delay returns to fast wires.

These statistics have important ramifications on the design of our routing architecture: even if the 20% of wires made fast cost twice the die-area of the standard wires, we increase the overall chip size by only 20%. And were we to speed up all wires on the chip, we would be wasting a significant amount of die-area to do so.

This simplified experiment, however, makes an assumption of perfect routability which is not realistic in practice – just making the wires fast is not sufficient to make them accessible to the subset of LEs and turning connections that require them. Routing flexibility could either shift or skew the curves to the right. Doubling the number of wire types greatly increases the number of possible ways to route from A to B in the part, and hence complicates the software significantly.

We remark that Dali is the first and only PLD architecture to utilize this heterogeneous wiring scheme to achieve greater performance. In Dali, we implement roughly 20% to 25% of wires as "fast". The exact number for each wire-type is dependent on aspect ratio of the part, numerology of the switching network, output muxing structures and detailed architectural experiments, with the general experiment used as a guideline or starting point. To derive the correct output flexibility for LEs driving the fast and standard interconnect we performed incremental tuning experiments throughout the design of the architecture.

Similar qualitative results to these were quoted by Betz and Rose in independent work [3]. They found that by modifying 20% of the tracks they were able to obtain a 13% circuit speedup, whereas speeding up all tracks gained only a 15% speedup. Though their results show less quantitative improvement than ours, they agree on both on the location of the "elbow" of the curve and the decreasing marginal returns. Among the reasons that we gain more benefit is that we optimized more than simply width and spacing for the fast wires. Also, we use significantly larger designs and devices in which speedup with respect to fanout can be greater.

In an accounting sense, we claim after this modification to have decreased critical path delay by approximately 30% with a roughly 20% die-size cost.

4. DEDICATED CONNECTIONS

A second way to speed up a PLD is to introduce dedicated connections for special purpose logic. There are three primary structures with which Dali achieves this goal. We introduce a new form of cascaded LUT connection, dedicated carry look-ahead structures important for arithmetic functions, and a dedicated multiplier.

4.1 Cascaded 4-LUT Connection.

One issue that must be re-visited in every new architecture is LUT-size: do we want to continue with a 4-LUT, or perhaps use 3 or 5 LUTs? The use of 5-input LUTs decreases unit delay in a technology mapped circuit by roughly 25%, 6-LUTs by an additional 15%, and so on. (One can easily generate this data using RASP [5] and the MCNC benchmarks.) However, the size of the LE layout doubles with each unit increase in LUT-size, which affects both area and propagation delay through the logic cell, and also the die area required for the LAB input region.

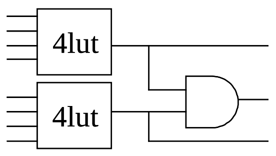

Figure 5(a) FLEX / APEX style AND-cascade.

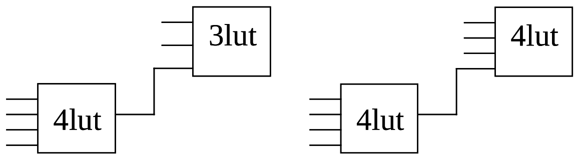

Figure 5(b) Cascade LUT suggested by Cong and Hwang, and cascaded LUT implemented in Dali.

Recent work by Ahmed and Rose [1] confirms previous studies by Rose *et. al.* (e.g. [18]) that a 4-LUT continues to be delay-area efficient, especially when paired with clusters (LABs) of size 4-10. Similar experiments which we conducted found a greater die-size penalty than Rose *et. al.* in using larger LUT-sizes, and a greater benefit to larger cluster sizes when all layout effects are taken into account, hence we continue to use 4-LUT LEs in clusters (LABs) of size 10.

However, there are alternatives to implementing larger LUTs in all cases, which can allow us to maintain both area efficiency and achieve better delay properties. For example, all FLEX and APEX devices from Altera contain special circuitry to allow adjacent LEs to cascade (AND) their outputs (see Figure 5(a)). This facilitates the implementation of wide and/or functions and multiplexors. Cong and Hwang [5] showed that a majority (96-98%) of all "practical" 5-LUTs and 85% of all practical 6-LUTs can be implemented by the 4-3 structure shown to the left of Figure 5(b). Though they did not directly study the 4-4 structure, we find it empirically even better (>98,98,73 for 5,6,7-LUTs), and significantly more layout and software friendly than heterogeneous LUT sizes. Note that though the AND-cascade of FLEX devices is comparable we found for the types of designs we looked at the 4-4 was more flexible.

Dali uses this 4-4 cascade in place of the previous and-cascade of FLEX 10K. The D input of each 4-LUT can be either a standard LUT input or the cascaded output of its adjacent LE.

4.2 Carry Look-Ahead.

Carry-chain delays, though significantly smaller than LE propagation delays, still require a signal to traverse the entire length of the carry chain after arriving at the first LE in the chain. To speed up this process, we added new carry lookahead circuitry which reduces the delay for an length-N carry chain to (N div 5) + 1 + 5. Thus for longer carry chains more likely to be on the critical path, a significant delay savings can be achieved.

Adding carry lookahead to speed up addition is not a new concept. However, typical carry lookahead circuitry is not common to PLDs because of the irregularity of the layout. Also, the fact that only arithmetic functions benefit while all logic increases in cost is prohibitive.

The carry lookahead scheme in Dali uses a regular layout, rather than a tree-like structure, and is designed to speed up all carry logic, not just arithmetic functions. The basic mechanism is to compute both the "if 0" and "if 1" branches of the carry chain, and then choose the correct result with a 2:1 mux at the end of 5 carry computations to achieve the cost savings quoted above.

4.3 Dedicated Multiplier.

As DSP and other communication-specific logic increasingly dominate the PLD landscape arithmetic logic is becoming more and more important. We have already discussed methods to speed up addition, which does help multiplication also, but we chose to add further specific circuitry to further attack the long propagation delays associated with multipliers.

The biggest drawback in implementing a multiplier in programmable logic is that the routing architecture is not amenable to optimal placement. Artificial boundaries may cause signals that should be local to use un-naturally longer wires (or an un-balanced number of short wires). The regularity of carry chain circuitry might also cause layout of the multiplier to be unnatural, following a zig-zag chain along a row (as in the case of an Altera FLEX 10K architecture).

To combat these issues we added two new routing features to Dali. First, we added vertical carry chain lines to allow logic in one level of the adder tree to align itself vertically. Then we added short dedicated lines to directly connect the output of one level of the adder tree to the next. To speed up signed multiplication, we added special circuitry in each LE so that the AND functions can be absorbed into the first level of the adder tree.

The use of these dedicated routing resources introduces a number of restrictions on the placement of a multiplier in the part. It is not possible to fully explain the restrictions here, but they are documented in the Mercury data-sheet.

There is, of course, the option of adding dedicated hard multiplier circuitry to the part. However, this introduces the typical PLD architecture problem of "How many?" Since some customers want no multipliers at all, they don't want to pay for them, and others want 23 when we provide 20 (or want 18 bit rather than a generic 16 bit), it is impossible to choose a single architecture to solve the problem. As density increases, dedicated multipliers will no doubt become mainstream, but for Dali we found the placement friendly circuitry for synthesized generic logic to be the best solution. The flexible (rather than hard-wired) approach also allows for the user to choose their latency strategy. In the Dali architecture a 16x16 multiplier can run over 130 MHz without pipelining, and over 300 MHz with pipelining.

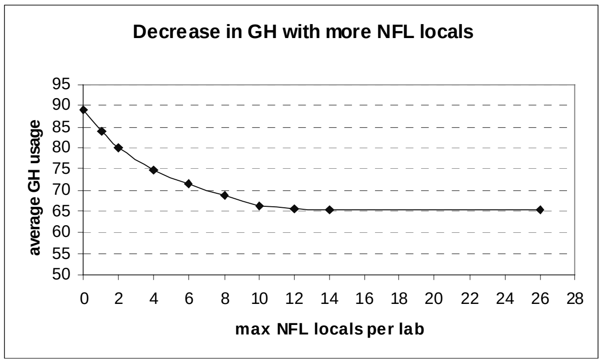

Figure 6. Density benefit of mixing staggered and full-length lines.

5. NEW WIRE TYPES

Some connections in a placed netlist will naturally be closer together, and others further apart. In an attempt to accommodate as many different types of connections as possible, we introduce a new type of interconnect that is staggered and is dedicated to the source LAB at which it is driven, supplementing the FLEX 10K base architecture. These are called NFL lines (short for network of fast lines). See Figure 8. NFL lines originate at a LAB and drive 5 LABs horizontally in either direction.

Because NFL lines have dedicated drivers, they are faster than typical GH or HH lines. We also achieve a moderate density benefit, because the staggering of the lines allows for a 20% better packing than regular GH lines. Figure 6 shows decreasing requirements for GH lines as the number of NFL lines approaches 10 per LAB. NFL lines are not stitched at their endpoints, in order to keep their delay reasonable. Note that the benefit of these lines is over-stated in the chart shown, because raw line-counts do not directly translate to layout area, due to different switching and sizing requirements for the different wire types.

A second new wire-type is the interleaved vertical (V) line, or "short V", which allows for nearest-neighbor row connections and saves the use of a longer global V line. Just as we found it advantageous to have unit-length connections between adjacent LABs [21], short-cut paths

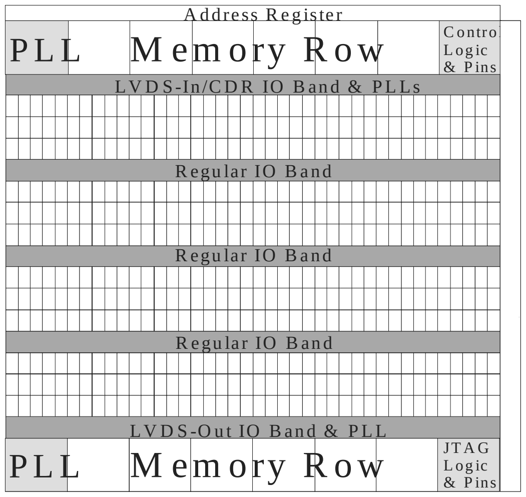

Figure 7. Top-level Floorplan of 4800 LE Mercury.

between rows are useful, particularly for special functions such as the multiplier circuitry.

6. DALI CORE ARCHITECTURE.

In this section we tie together the discussion of the previous several sections by presenting the overall routing architecture of the Dali family. This is best done by a top-level floorplan, shown in Figure 7. The prototype device shown has 10 LEs per LAB, 40 LABs per row and 12 logic rows for a total of 4800 LEs. Other family members in the family contain, for example, 14,400 LEs in different row/column configurations.

In addition to programmable logic rows, the device has two rows devoted to Embedded System Blocks (ESBs) which are 4Kbit user-configurable RAMs. PLLs or phase-lock loops (in varying numbers depending on the family member) allow fine control of externally generated clocks.

An idealized "die-shot" of the part is shown in Figure 8.

The operation of the RAM blocks is shown in Figure 9. Many earlier-generation PLDs implemented partial dual-port RAM (i.e. FIFO-RAM), but Dali implements not only a full dual-port, but also quad-port access and a number of new configuration options which are beyond the scope of this paper. The ESB continues to be configurable as a Content-Addressable Memory [7,8] as in the APEX family.

It is interesting to note that I/O locations in Dali are in rows directly in the core, as opposed to a pad-ring surrounding the chip. This allows us to take advantage of modern flip-chip technology in order to provide higher pin-counts, and also allows us to give I/Os very fast access to core registers.

6.1 Benchmarking Results.

Our judge of success for the Dali routing architecture is to compare it to an equivalently sized FLEX 10K part. In addition to being the closest point for comparison, the

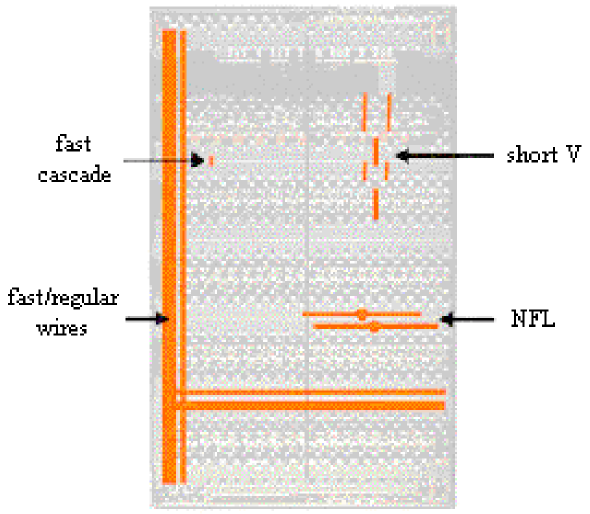

Figure 8. Mercury "Die-shot" showing different wire types.

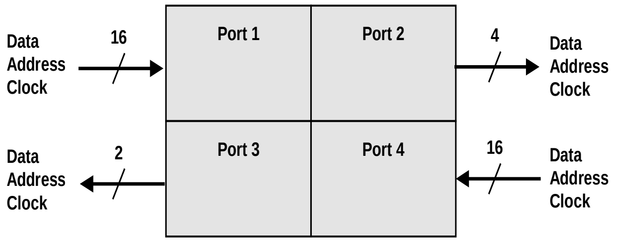

Figure 9. Quad-port ESB RAM.

APEX architecture was not yet available in production at this time. Our evaluation uses same-process, simulation-to-simulation timing and area models. A direct comparison of actual Mercury device to 10K would overly favor it simply because of the process advantage, whereas these comparisons fairly evaluate our architectural decisions.

To remove the effects of software from the benchmarking as much as possible we created a fake FLEX 10K family member with the same number of rows and columns as our base part. We then start with a back-annotated placement. By beginning with the same placement, we can make sure that we do not give any advantages due to new/better algorithms, or disadvantages due to immature software to the new part. However, it would be too unfair to make no modifications for use of the interleaved V-lines and staggered short lines, so we add pre-processing and post-processing steps which locally modify the input placement to better take advantage of these new lines – i.e. LABs are re-ordered within their row, and rows are shifted around as blocks to opportunistically use the new wire-types. We consider this to be a fair yet conservative implementation, because it allows for improvements that legitimately exist in the new part, without adding the experimental noise of using two completely different place and route tools.

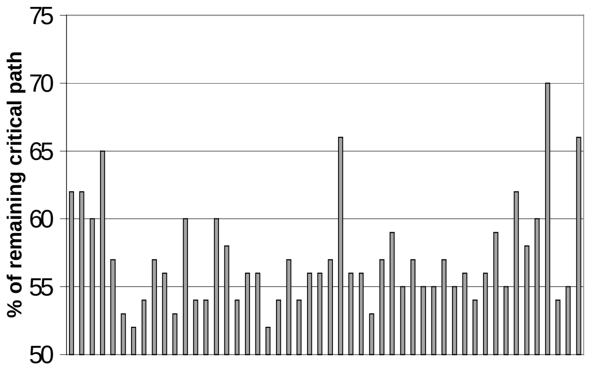

Figure 10. Percentage of remaining critical path after software modification of 10K to Dali architecture.

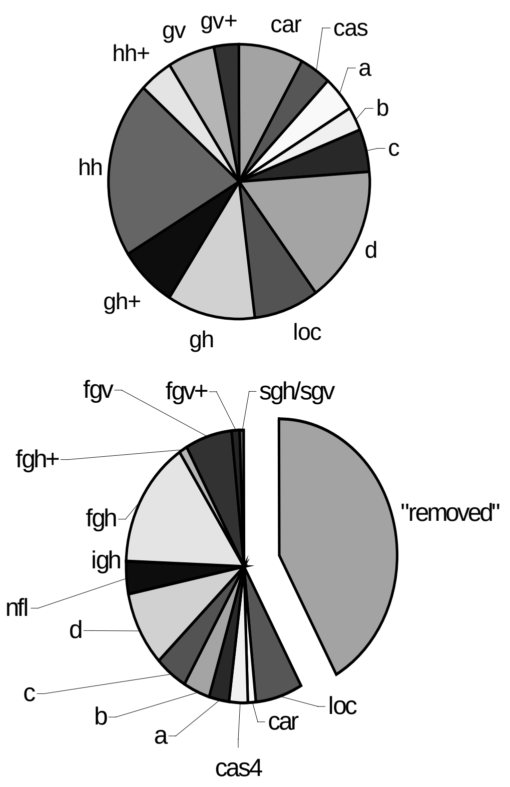

Figure 11. Dali Speedup relative to the base architecture.

Because of the new wire-types, we also require changes to the router to use all available wires. The router must prioritize the critical path connections onto the fast wiring resources or staggered lines where appropriate to achieve both performance and fitting. In production we would want to have placement also consider the availability of fast resources. Even though this leaves potential upside we chose, as previously mentioned, to not introduce greater experimental variation by significantly modifying placement. In the production-quality software we address a number of these new issues and problems. For example, the simple "promotion" algorithm outlined in Section 3 contains a full timing-analysis in its inner loop and is clearly not time-efficient.

Our results are presented in the following manner. For each design in the evaluation set, we take the 10K critical path length, normalized to 100 units. We then compare the length of the critical path of the same design after the introduction of the Dali timing-model, implementation of the modified cascade, placement modifications for new wire-types and prioritized routing algorithm. Figure 10 shows this result for a sample of user designs. All designs are between 70% and 100% full in terms of LE count. On average, the Dali device achieves a critical path that is 58% the delay of the beginning 10K path, translating to a 1.72X increase in internal clock-speed. The most affected design is almost 2X the speed, and the least affected has 70% the critical path, or 1.43X clock-speed of the original.

Figure 10 is the exact, un-edited result used in presentation of the prototype architecture. Thus it compares simulation-to-simulation on the same process rather than commercial benchmarking. It does not reflect differences in the parts based on process shrinks or future layout constraints, either positive or negative.

To further illustrate the areas where the primary benefits were achieved, we can examine the breakdown of the critical path in the "before" and "after" architectures, as shown in Figure 11. The delay types are GH (full-length global horizontal), HH (half horizontal), GV (full length vertical), carry, cascade, local, and LE delays through inputs A, B, C, and D of the 4-LUT. The distinction between GH and GH+ is the base delay vs. the additional delay due to fanout loading.

After conversion to the Dali architecture (lower pie-chart), we see 42% of the overall delay removed, and the breakdown of the remaining 58 units of delay has significantly moved away from the higher delay global wires to a mixture of the most appropriate wire types. The new wire types NFL and IV have appeared, and the GH and GV lines have been divided into the new wire types sGH – standard GH, and fGH – fast-GH.

One of the major effects of the detailed design of the fast wires is the removal of fanout effects. Overall, we can see that virtually all wire types were sped up, confirming our previous research experiment that we could speed up all critical paths with only a fraction of the basic routing architecture requiring additional cost. In particular, we note that the standard GH and GV wires comprise the majority of the routing architecture, yet they very rarely appear on any critical paths.

7. CONCLUSIONS

In this paper we have outlined the routing architecture of a new PLD architecture, Dali, which is specifically designed to achieve high-performance, both in core clock-speed and in I/O bandwidth. In addition to the overall architecture, we have made a number of independent research contributions, including justification that only a proportion of routing resources need be fast to achieve high-performance routing, effective use of dedicated connections for special-purpose logic, and discussion of the use of different routing types.

Overall, our prototype benchmarking shows the Dali routing architecture to achieve approximately 1.7X the performance of an equivalent process FLEX 10K device, at a relatively cheap 1.3X die-area cost.

The Dali architecture forms the basis for Altera's Mercury family. Mercury was originally implemented on a 8LM .18 Al, 1.8V CMOS process, and now commercially available on .15 Cu, 1.8V CMOS, the first two family members comprising the 4800 and 14,400 LE size devices. We firmly believe that the Mercury implementation of our Dali architecture represents the fastest PLD in its process and density range by a significant margin.

Though not discussed in this paper, many of the important features of the Mercury commercial family relate to multi-standard and voltage I/O structures, and Gbit high-speed differential interfaces with clock-data recovery as a primary feature. CDR, in particular, is a novel and important structure that is critical to modern high-bandwidth communications designs.

ACKNOWLEDGMENTS

The authors on this paper were the primary technical contributors to the high-level interconnect design of the Dali architecture but only a small part of the overall IC design, layout and software effort required to produce the family. During the architecture definition phase described in this paper Jay Schleicher lead the software team, and Tony Ngai the IC design team.

REFERENCES

[1] E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA00), pp. 3-11.

[2] Altera Corp. http://www.altera.com.

[3] V. Betz and J. Rose, "Circuit Design, Transistor Sizing and Wire Layout of FPGA Interconnect", in Proc. IEEE Custom Integrated Circuits Conference (CICC), 1999.

[4] K. Chung and J. Rose, "TEMPT: Technology Mapping for the Exploration of FPGA Architectures with Hard-Wired Connections", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA92), pp. 21-26, 1992.

[5] J. Cong and Y-Y. Hwang, "Boolean Matching for Complex PLBs in LUT-based FPGAs with Application to Architecture Evaluation", Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA98), pp. 27-34, 1998.

[6] S. Hauck, M. Hosler and T. Fry, "High Performance Carry Chains for FPGAs", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA98), pp. 223-233, 1998.

[7] F. Heile and A. Leaver, "Hybrid Product Term and LUT-Based Architectures Using Embedded Memory Blocks", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA97), pp. 13-16, 1997.

[8] F. Heile, A. Leaver and K. Veenstra, "Programmable Memory Blocks Supporting Content-Addressable Memory", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA00), pp. 13-21, 2000.

[9] W-J. Huang, M. Hutton, V. Maruri, T. Ngai, R. Patel, B. Pedersen, J. Schleicher and S. Shumarayev, "PLD Routing Architecture with Both Fast and Regular Routing Resources", US Patent Application Pending.

[10] M. Hutton, "Interconnect Prediction for Programmable Logic Devices", in Proc. ACM/SIGDA Int'l Workshop on System-Level Interconnect Prediction (SLIP01), pp. 125-131, 2001.

[11] M. Hutton, K. Adibsamii and A. Leaver, "Timing-Driven Placement for Hierarchical Programmable Logic Devices", In Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA01), pp. 3-11, 2001.

[12] S. Kaptanoglu, G. Bakker, A. Kundu and I. Corneillet, "A new high-density and very low cost reprogrammable FPGA architecture", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA99), pp. 3-12, 1999.

[13] A. Marquardt, V. Betz and J. Rose. "Using Cluster-Based Logic Blocks and Timing-Driven Packing to Improve FPGA Speed and Density", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA99), pp. 37-46, 1999.

[14] T. Ngai, B. Pedersen, S. Shumarayev, J. Schleicher, W-J. Huang, M. Hutton, V. Maruri, R. Patel, P. Kazarian, A. Leaver, D. Mendel and J. Park. "Interconnection and Input/Output Resources for Programmable Logic Integrated Circuit Devices", US Patent Applicatoin Pending.

[15] E. Ochotta, *et. al.*. "A Novel Predictable Segmented FPGA Routing Architecture", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA98), pp. 3-11, 1998.

[16] J. Park, B. Pedersen and W-J. Huang, "Carry-lookahead", US Patent Application Pending.

[17] B. Pedersen and J. Park, "Dedicated Multiplier", US Patent Application Pending.

[18] J. Rose, R.J. Francis, D. Lewis and P. Chow. "Architecture of Field-Programmable Gate Arrays: The Effect of Logic Block Functionality on Area Efficiency," In. IEEE J. Solid-State Circuits, 1990.

[19] J. Schleicher and M. Hutton. "Fast Cascade". US Patent Application Pending.

[20] S. Trimberger, K. Duong and B. Conn. "Architecture Issues and Solutions for a High-Capacity FPGA", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA97), pp. 3-9, 1997.

[21] K. Veenstra, B. Pedersen, J. Schleicher and CK Sung. "Optimizations for a Highly Cost Efficient Programmable Logic Architecture", in Proc. ACM/IEEE Int'l Conference on FPGAs (FPGA98), pp. 20-26, 1998.

[22] Xilinx Corp. http://www.xilinx.com.