The Effect of Multi-Bit Correlation on the Design of Field-Programmable Gate Array Routing Resources

Phoebe Ping Chen and Andy Ye, *Member, IEEE*

***Abstract*—As the logic capacity of field-programmable gate arrays (FPGAs) increases, they are being increasingly used to implement large arithmetic-intensive applications. Large arithmetic intensive applications often contain a large proportion of datapath circuits. Since datapath circuits are designed to process multiple-bit-wide data, FPGAs implementing these circuits often have to transport a large amount of multiple-bit-wide signals from one computing element (such as a logic block, a DSP block, or a multi-bit addressable memory cell) to another. In this work, we investigate the area efficiency of FPGA routing resources for transporting multiple-bit-wide signals. It is shown that, for datapath circuits, the switch patterns used by the conventional routing architecture, which uniformly distribute routing switches across the routing tracks, are inefficient for connecting the computing elements to their tracks. The more efficient multi-bit aware patterns, which contain a densely populated single-bit region and a sparsely populated multi-bit region, can be effectively used to reduce the routing area of FPGAs for implementing arithmetic intensive applications by 6%–10%. It is also shown that the further sharing of configuration memory among the switches within the multi-bit aware patterns does not significantly increase their area efficiency since datapath circuits typically contain a mixture of multi-bit and single-bit signals—while configuration memory sharing can substantially increase the area efficiency of routing resources for transporting multi-bit signals, it also significantly reduces their ability for transporting single-bit signals. More importantly, configuration memory sharing can significantly reduce the effectiveness of the enhanced multi-bit aware patterns—patterns that incorporate both multi-bit aware and single-bit oriented switches within a single region in order to increase its ability for transporting both single-bit and multi-bit signals.**

***Index Terms*—Area efficiency, datapath-oriented FPGA, field-programmable gate arrays (FPGAs), routing resources.**

I. Introduction

As the logic capacity of field-programmable gate arrays (FPGAs) increases, they are being increasingly used to implement large arithmetic-intensive applications. Large arithmetic-intensive applications often contain a large proportion of datapath circuits. Since datapath circuits are designed to process multiple-bit-wide signals, FPGAs implementing these circuits are routinely used to transport these signals from one computing element (such as a logic block, a DSP block, or a multi-bit addressable memory cell) to another.

Manuscript received December 02, 2008; revised May 08, 2009.

The authors are with the Department of Electrical and Computer Engineering, Ryerson University, Toronto, ON M5B 2K3, Canada (e-mail: pepe_chen@hotmail.com; aye@ee.ryerson.ca).

Digital Object Identifier 10.1109/TVLSI.2009.2029232

To transport a multiple-bit-wide signal, one can either treat the signal as a set of independent bits and route each bit individually through a set of conventional routing resources, or view the entire signal as a single coherent unit and transport the unit collectively through a set of specialized routing resources. In this work, we investigate the detailed design of these specialized routing resources on the area efficiency of FPGAs and compare them to the conventional routing resources, which are designed to transport independent bits of signals.

Several FPGAs have been proposed to use multi-bit routing resources to connect their computing elements [1]–[9]. Except for the work in [8] and [9], however, none has investigated their detailed design. Both [8] and [9] are based on the observation that multi-bit routing resources can be configured one datum at a time while the conventional routing resources must be configured one bit at a time. Consequently, both have focused on utilizing the multi-bit nature of signals to share configuration memory. These studies observe that this increase in configuration granularity can lead to a significant reduction in the amount of memory that is required to configure the routing resources and an increase in FPGA area efficiency.

The correlated behaviors of multi-bit signals, however, can affect the area efficiency of FPGAs in other ways. In particular, multi-bit routing resources can be grouped into multiple-bit-wide groups. Connecting two groups of resources together requires one to connect each bit in one group to a corresponding bit in the other. This one-to-one mapping of resources can result in a much sparser distribution of routing switches than the distribution employed in the conventional routing resources—which must use denser switch patterns to connect independent bits of signals. In this work, the sparse distribution is used to construct two multi-bit aware routing architectures—the sparse and the enhanced sparse architectures. The area efficiency of these architectures is then compared to the area efficiency of the conventional and the configuration memory sharing architectures.

The remainder of this paper is organized as follows. Section II motives the research and describes the routing architectures used in this investigation, Section III presents the detailed architectural parameters and examines their effects on the implementation area of the various FPGA components, Section IV presents experimental results, and Section V gives concluding remarks.

II. Multi-Bit Routing Architectures

Consider implementing a datapath circuit on FPGAs by mapping each bit slice of the circuit into a set of logic clusters. These clusters can then be grouped into a set of multi-bit computing elements that are connected by multi-bit signals. For example,

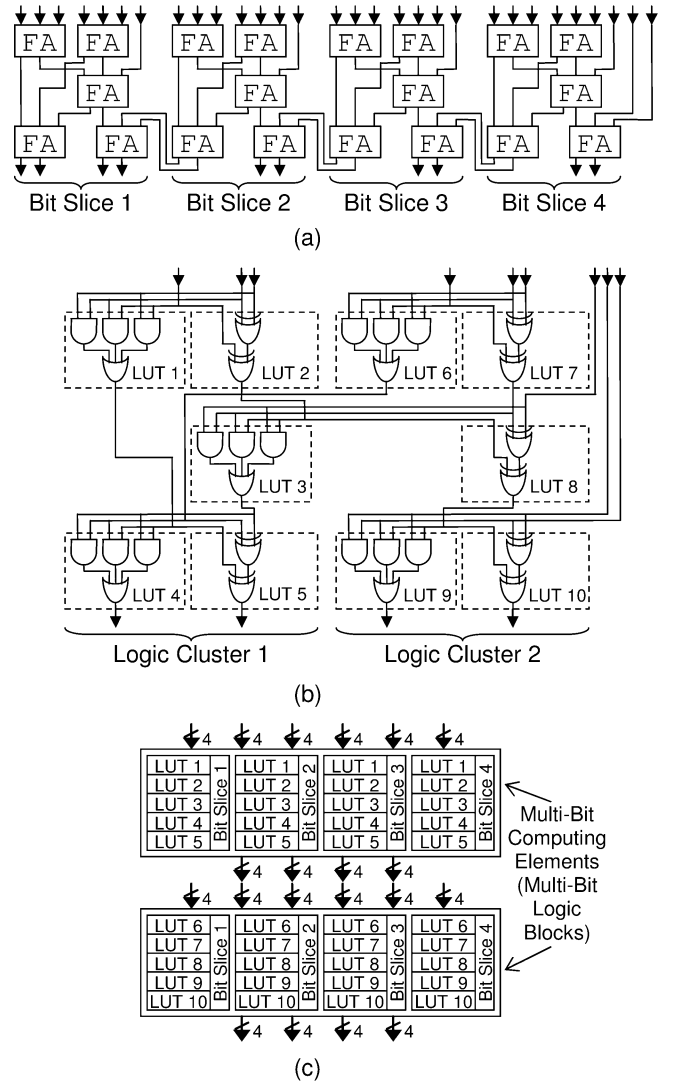


Fig. 1. Four-bit 7:2 compressor: (a) 7:2 compressor and its bit slices; (b) mapping each bit slice into two clusters; (c) mapping clusters into multi-bit computing elements.

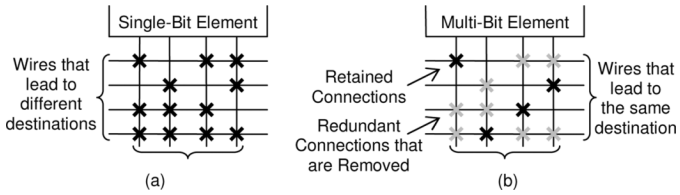


Fig. 2. Connection patterns for single-bit and multi-bit elements: (a) independent signals and (b) correlated signals.

as shown in Fig. 1, each slice of a four-bit-wide 7:2 compressor [10] can be mapped into ten three-input lookup tables (LUTs), which in turn can be grouped into two logic clusters. The eight clusters from all bit slices can be organized into two multi-bit computing elements with each element containing four clusters, six four-bit-wide inputs and four four-bit-wide outputs.

The multi-bit processing nature of the multi-bit computing elements is significantly different from the single-bit design of the traditional logic blocks. While the input and output pins of a conventional logic block carry independent bits of information, the input and output pins of a multi-bit computing element are logically organized to represent multiple-bit-wide data. In this organization, pins that represent a datum are often get used at the same time. Similarly, routing resources are often used to transport multiple-bit-wide signals from a common source to a common destination. These correlated behaviors can significantly affect the efficiency of FPGA routing resources.

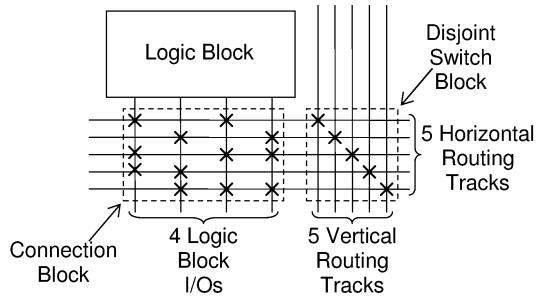
In particular, if all signals are independent, there are

$$\left(\sum_{i=0}^{m}\binom{m}{i}\right)^{n} = 2^{(m\times n)}$$

different ways of using programmable switches to connect a group of m wires to another group of n wires. For example, Fig. 2(a) shows four vertical wires (which represent four output pins of a computing element) and four horizontal wires (which represent four FPGA routing tracks). These wires are connected by 12 programmable switches, which form one of the over 65 000 connection patterns. Taking into account of correlation, however, the situation changes. For example, if the vertical wires always simultaneously carry valid signals and these signals always share a common destination, connecting a vertical wire to a horizontal wire automatically implies three similar connections (one for each of the remaining vertical-horizontal pairs). Consequently, as shown in Fig. 2(b), each vertical wire only requires one connection for every four horizontal wires and the total number of connection patterns is reduced to 24. The reduction of eight connections per pattern can result in significant routing area savings, and the reduction in the number of patterns enables a more targeted search of the design space. The exact number of connection point reduction, however, is closely related to the detailed placement of these points, the interaction between the patterns (there could be tens of thousands of connection points arranged using several patterns in a single architecture) and the routing demand of the applications.

The *conventional routing architecture* as defined in [11] exploits only a smaller number of design alternatives from this vast design space. The design space reduction is achieved through the use of disjoint [12]–[15] and near-disjoint [16]–[19] patterns in the switch blocks and by distributing logic block input/output connections uniformly across the routing tracks. In particular, as shown in Fig. 3(a), the disjoint topology is used to connect the horizontal tracks to the vertical tracks, where each horizontal track is connected to a distinct vertical track. The logic block input and output pins, on the other hand, are uniformly distributed based on the algorithm shown in Fig. 3(b). As shown, W is equal to the number of routing tracks per channel (where a channel is defined as the group of horizontal/vertical tracks located between two rows/columns of logic blocks). F_c is equal to the percentage of tracks per channel that can be connected to a logic block input/output pin. P is equal to the number of logic block input/output pins, and the switch pattern matrix contains the index of a routing track that the jth connection of the ith logic block input/output pin connects to.

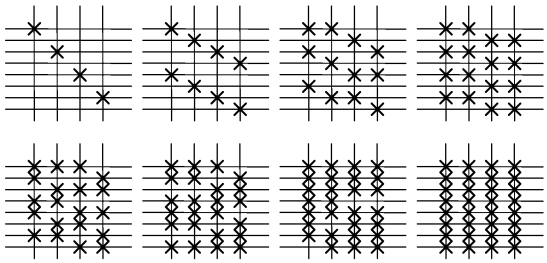
Note that the algorithm connects each logic block input/output pin to the same number of tracks and given W tracks, it can generate W distinct patterns. As an example,



(a)

```
For Logic Block Output Pins: C = ceil (Fc * W)
For Logic Block Input Pins: C = floor (Fc * W)
step = W / P / C
increment = W / C
for (i = 0; i < P; i++) {
  for (j = 0; j < C; j++) {
    switch_pattern[i][j] = floor (step * i + increment * j)
  }
}
```

(b)



(c)

Fig. 3. Conventional FPGA routing architecture: (a) connection and switch blocks; (b) switch distribution algorithm for connection blocks; and (c) patterns for connecting four logic block inputs/outputs to eight routing tracks.

Fig. 3(c) shows the eight patterns that the algorithm can generate to connect eight tracks to four logic block input/output pins. These eight (W) patterns represent a very small fraction of the total $2^{32}(2^{(W\times P)})$ switch patterns that can be used to connect the pins to the tracks.

This work expands the design space of the conventional connection blocks by exploiting the correlated behaviors of multi-bit signals while still preserving the tile-based design of conventional FPGAs. In particular, the disjoint pattern is applied to two groups of multi-bit routing resources as shown in Fig. 4(a), where a bit in one group is connected to a corresponding bit in the other. For routing channels designed to transport multiple multi-bit signals, the algorithm shown in Fig. 3(b) is used to distribute the disjoint patterns (instead of individual bits of switches) uniformly across a channel. For example, the distribution algorithm can connect 24 routing tracks (which are grouped into six four-bit-wide groups) to eight logic block input/output pins (which are grouped into two four-bit-wide groups) using the switch distributions shown in Fig. 4(b). The disjoint patterns are then used to complement the uniformly distributed switches, as shown in Fig. 4(c), to increase the area efficiency of routing resources for transporting multi-bit signals.

In this paper, the routing architecture shown in Fig. 4(c) is called the *sparse routing architecture*. Comparing to the

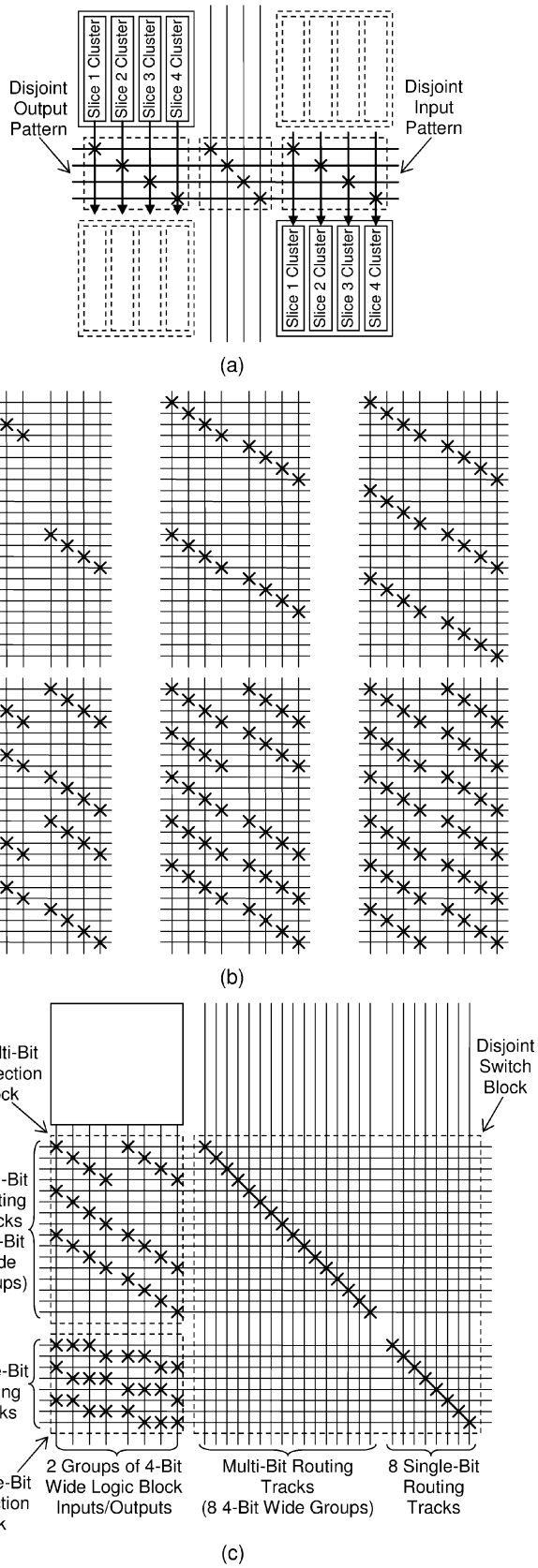


Fig. 4. Disjoint switch patterns and sparse routing architecture: (a) disjoint input and output connections; (b) connecting six four-bit buses to two four-bit buses; and (c) sparse routing architecture.

W connection block designs that can be generated by the

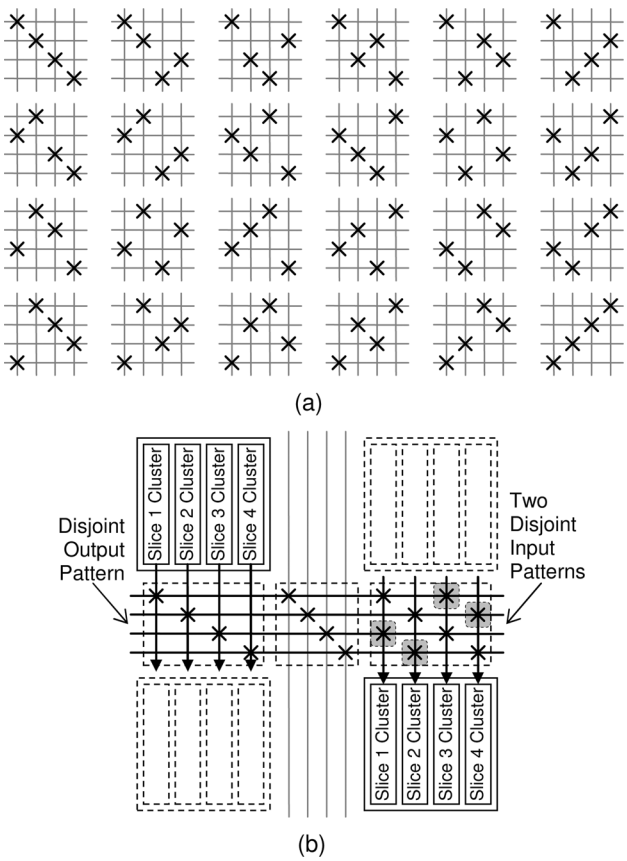


Fig. 5. Enhanced sparse architecture: combining disjoint patterns: (a) 24 disjoint patterns for 4×4 connections and (b) combining two disjoint patterns at logic block inputs.

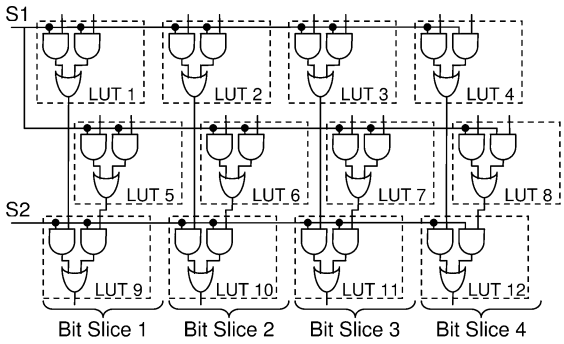


Fig. 6. Four-bit-wide 4:1 multiplexer and its control signals (S1 and S2).

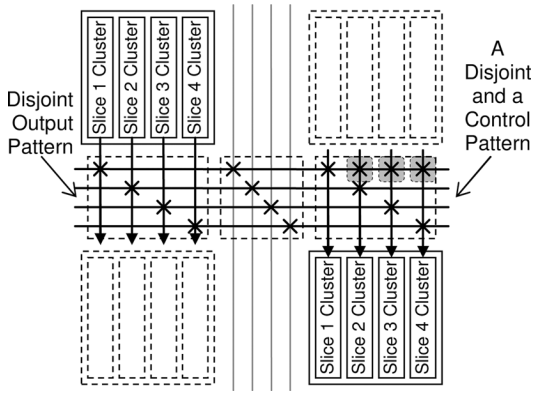


Fig. 7. Enhanced sparse routing architecture: combining disjoint and control patterns at logic block inputs.

conventional routing architecture, the sparse architecture can generate $\sum_{i=0}^{\lfloor W/M \rfloor} (\max(1,i) \times \max(1, W - i \times M))$ distinct connection blocks. For example, when W is equal to 24 and M is equal to four, the sparse architecture can generate 170 distinct connection blocks while the conventional architecture can only generate 24. (Note that while the work in [8] and [9] employs similar disjoint patterns, these patterns were used in conjunction with configuration memory sharing. The independent effects of these patterns on FPGA area efficiency have not been examined previously.)

The functionalities of the sparse routing architecture can be further enhanced by incorporating additional programmable connections into the disjoint pattern. For example, the basic disjoint pattern shown in Fig. 4 is just one of the $\prod_{i=1}^{M} \binom{i}{1}$ disjoint patterns that can be used to connect an M-bit-wide signal to another. Specifically, Fig. 5(a) shows the set of 24 disjoint patterns that can be used to connect two four-bit-wide signals. As shown each pattern provides a different amount of shift. These patterns can be used to enhance the basic disjoint pattern. For example, as shown in Fig. 5(b) two disjoint patterns are combined into a new pattern. In combination, the pattern provides two types of shifting capabilities—shift-by-0 and shift-to-the-right-by-two. The resulting patterns allow multi-bit routing tracks to rearrange as well as transport multi-bit signals.

Note that high fan-out signals, such as the control signals of the 4:1 multiplexer shown in Fig. 6, are routinely used to coordinate the operations of multiple bit slices in a datapath circuit. These signals also consist of a significant amount of connections in datapath circuits. The basic disjoint pattern can be further enhanced to support these control signals with the addition of routing switches that distribute a signal from one multi-bit routing track to all bit slices in a multi-bit logic block. As shown in Fig. 7, with the addition of control patterns, a multi-bit routing track can be used to distribute high fan-out signals as well as route multiple-bit-wide signals.

The routing architectures that contain additional disjoint or control patterns are called the *enhanced sparse architectures*. In the remainder of this paper, we evaluate the area efficiency of both sparse and enhanced sparse architectures and compare them to the conventional and configuration memory sharing architectures [8]. Note that, like the sparse architecture, the configuration memory sharing architecture, as shown in Fig. 8, also uses disjoint patterns to connect multi-bit routing tracks. Instead of relying on the sparseness of the disjoint patterns to reduce FPGA routing area, however, the architecture is designed to reduce configuration memory. Consequently, instead of using independently controlled switches within each disjoint pattern, each set of multi-bit routing tracks are driven by switches that are collectively controlled by a single set of configuration memory. Consequently, the configuration memory sharing architecture is less efficient at routing single-bit signals such as the high fan-out control signals.

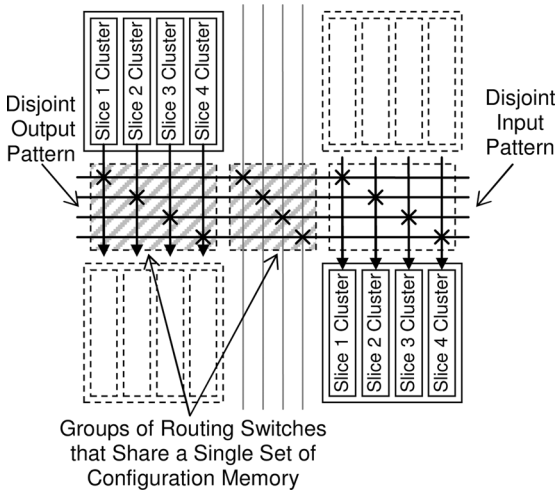


Fig. 8. Regions of configuration memory sharing in the configuration memory sharing routing architecture.

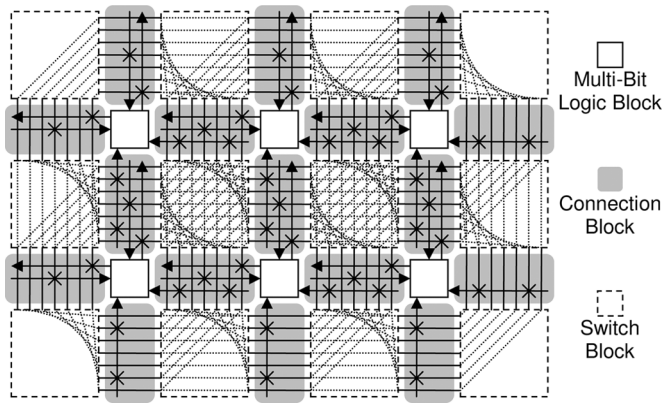


Fig. 9. Overall architecture of an FPGA.

III. Architectural Models and Parameters

The architectural model from [11] is used to evaluate the area efficiency of the sparse and enhanced sparse architectures. Area is measured in terms of the minimum-width transistor area and is based on the formula shown at the bottom of the page.

All architectures investigated in this work use the same layout as Fig. 9. As in Fig. 4(a), each multi-bit logic block consists of four logic clusters. Each cluster contains four fully connected basic logic elements (each basic logic element contains one LUT and one register), ten inputs, and four outputs. The inputs/outputs of each cluster are directly connected to the inputs/outputs of the multi-bit logic blocks. To capture multi-bit regularity, the logic clusters in each multi-bit logic block are used to implement identical portions of adjacent bit-slices from datapath circuits using datapath-oriented synthesis [20] and packing [21] tools. Note that the cluster size of four with ten inputs and four outputs was found to be the most area efficient for conventional FPGAs in [22]. Multi-bit logic blocks containing four logic clusters were shown to be the most area efficient for FPGAs with configuration memory sharing routing resources [8].

In the conventional routing architecture, both the input and output connection blocks can be parameterized by two parameters—W, the number of routing tracks per channel and $F_{c\_\mathrm{in}}/F_{c\_\mathrm{out}}$, the percentage of these tracks that a logic block input/output pin connects to. An input connection block from the sparse routing architecture is parameterized by $W_{\mathrm{single}}, F_{c\_\mathrm{in}\_\mathrm{single}}, W_{\mathrm{multi}}$, and $F_{c\_\mathrm{in}\_\mathrm{multi}}$. W_{single} and $F_{c\_\mathrm{in}\_\mathrm{single}}$ are the number of single-bit tracks per channel and the percentage of these tracks that a logic block input pin can connect to. W_{multi} is the number of multi-bit tracks per channel (where the tracks are grouped into four-bit-wide groups) and $F_{c\_\mathrm{in}\_\mathrm{multi}}$ is the percentage of these groups that a logic block input pin can connect to. Similarly the output connection block is parameterized by two additional parameters: $F_{c\_\mathrm{out}\_\mathrm{single}}$ and $F_{c\_\mathrm{out}\_\mathrm{multi}}$.

The enhanced sparse architecture differs from the sparse architecture in its input connection block design. In this work, the difference is characterized by the parameter F_{c_enh}, which is equal to the percentage of multi-bit tracks that are connected through the enhanced disjoint patterns. For example, Fig. 10(a) shows the input connections for a group of four input pins. As shown, 16 of the 24 tracks are connected through disjoint patterns that are enhanced by the control pattern ($F_{c\_\mathrm{enh}} = 0.67$). Note that the control pattern is distributed uniformly across all bit positions, where the first, second, third, and fourth disjoint patterns are enhanced on the first, second, third, and fourth tracks, respectively. Three of the 24 disjoint patterns shown in Fig. 5(a) are also used to enhance the basic disjoint pattern. As shown in Fig. 10(b), these patterns correspond to shift-to-left by one, two, and three bits, respectively, and are uniformly distributed across all enhanced patterns.

As shown in Fig. 8, the sparse and enhanced sparse architectures differ from the configuration memory sharing architecture in their output connection block and switch block designs. (Note that, to accommodate connections from both multi-bit and single-bit tracks, configuration memory is not shared within the input connection blocks [8].) As shown in Fig. 11, in a configuration memory sharing output connection block, switches from a disjoint pattern share a single set of configuration memory.

Configuration memory is similarly shared in the switch blocks. In particular, the sparse and enhanced sparse architectures share the same switch block design as shown in Fig. 12(a) (Note that for clarity two-bit-wide groups instead of four-bit-wide groups are shown in the figure.) Each switch block consists of two types of routing switches—the full switches and the half switches. A full switch connects four

$$\mathrm{Area} = \sum_{\mathrm{All\_Trans}} \left(0.5 + \frac{\mathrm{Drive\_Strength\_of\_the\_Current\_Trans}}{2 \times \mathrm{Drive\_Strength\_of\_Min\_Width\_Trans}}\right)$$

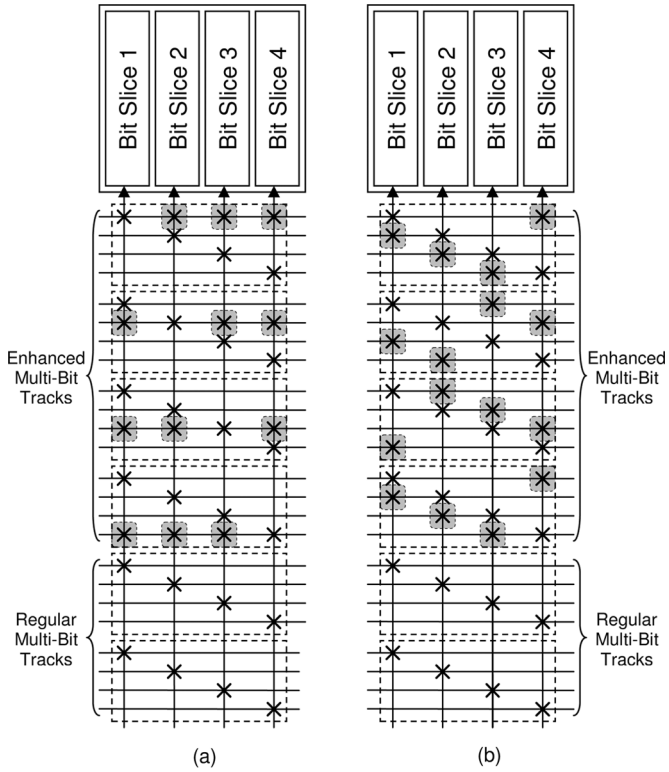


Fig. 10. Enhanced disjoint patterns ($F_{c\_\mathrm{enh}} = 0.67$): (a) disjoint patterns enhanced by control patterns and (b) disjoint patterns enhanced by shift patterns.

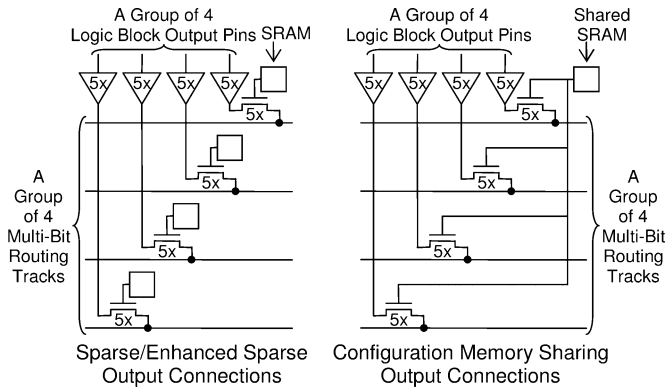


Fig. 11. Configuration memory sharing in output connection blocks.

track segments (two vertical and two horizontal) that terminate at the switch block. A half switch (required for connecting track segments that span multiple logic blocks), on the other hand, connects a vertical non-terminating segment to a horizontal non-terminating segment. The sparse/enhanced sparse architecture differs from the conventional architecture [as shown in Fig. 12(c)] in the placement of the full and half switches. As in the conventional architecture, in the sparse/enhanced sparse architecture each routing switch is independently controlled by its own configuration memory. In a configuration memory sharing switch block [as shown in Fig. 12(b)], on the other hand, configuration memory is always shared among the routing switches within each group of multi-bit routing tracks.

Figs. 13 and 14 show the total routing area, including the switch block area, the input connection block area, and

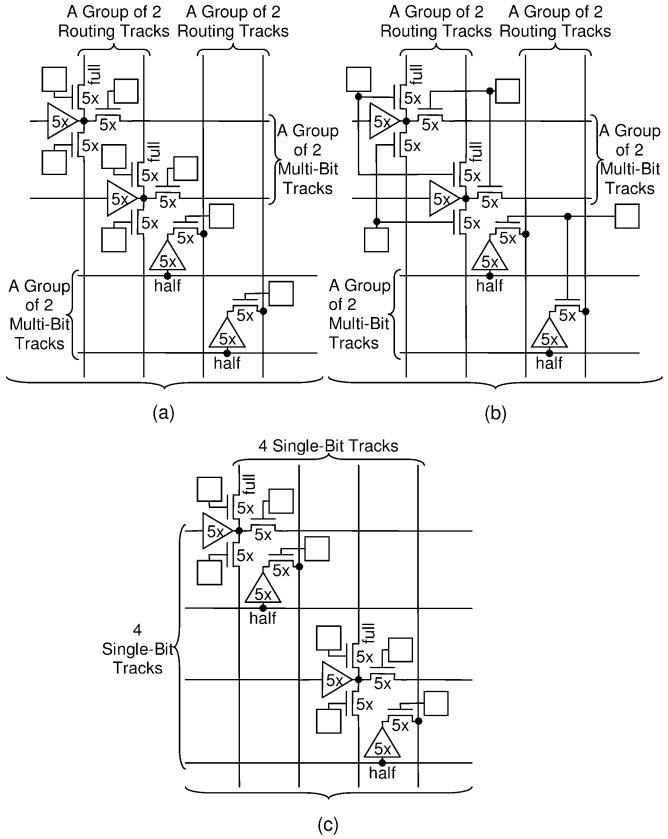


Fig. 12. Configuration memory sharing in switch blocks (segment $\mathrm{Length} = 2$, only one buffer per track is shown). (a) Sparse/enhanced sparse switch block; (b) configuration memory sharing switch block; and (c) conventional switch block.

the output connection block area, as a function of F_{c_in} and $F_{c\_\mathrm{in\_multi}}$. In both figures, the conventional architecture contains 40 single-bit tracks while the sparse, enhanced sparse, and configuration memory sharing architectures contain 40 multi-bit tracks grouped into four-bit-wide groups. In Fig. 13, F_{c_out} (for the conventional routing architecture) and $F_{c\_\mathrm{out\_multi}}$ (for the sparse, enhanced sparse, and configuration memory sharing routing architectures) are set to 0.3. In Fig. 14, F_{c_out} and $F_{c\_\mathrm{out\_multi}}$ are set to 0.8. As shown, for the same number of tracks, the sparse routing architecture consumes significantly less area than the conventional architecture, and the configuration memory sharing architecture consumes the least amount of area. These figures also show that the control and shift enhancements only slightly increase the implementation area of the enhanced sparse routing architecture over the sparse routing architecture.

IV. Experimental Results

To investigate the relationship between routing flexibility and routing switch density, we employ 15 datapath circuits from the Pico-Java processor [23]. As in [8] and [9], the circuits are first mapped onto multi-bit logic blocks using a set of datpath-oriented synthesis and packing tools [20], [21]. The logic blocks are then placed using the conventional simulated annealing placement [11], [24]. The placed circuits are then implemented on both the conventional and the multi-bit aware

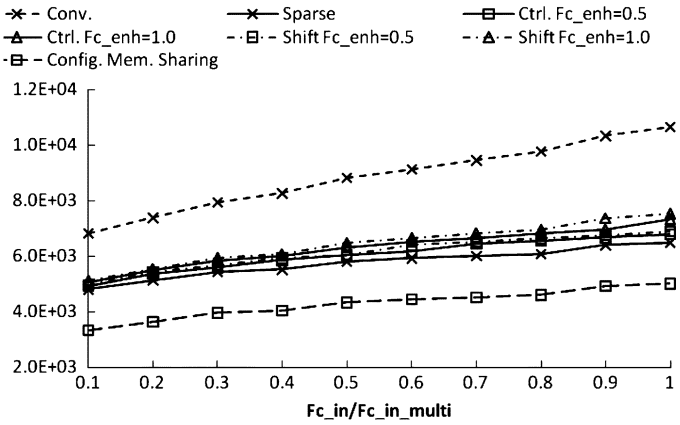


Fig. 13. Routing area comparison ($F_{c\_\mathrm{out}} / F_{c\_\mathrm{out\_multi}} = 0.3$).

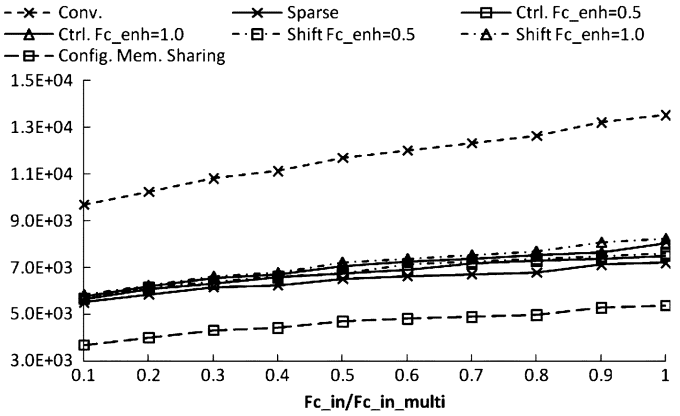


Fig. 14. Routing area comparison ($F_{c\_\mathrm{out}} / F_{c\_\mathrm{out\_multi}} = 0.8$).

routing architectures using the conventional [25], [26] or the multi-bit aware [9], [24] routing tools. Through these implementations, we attempt to address the following three questions: 1) What is the effect of configuration memory sharing on the overall area efficiency of FPGAs? 2) What is the most efficient method of increasing the area efficiency of the sparse routing architecture—increasing the flexibility of the input connection blocks through control and shift enhancements or reducing the implementation area of the output and switch blocks through configuration memory sharing? and 3) How does the area efficiency of the multi-bit aware routing architectures compare to the area efficiency of the conventional routing architecture?

Note that among the two routing tools only the multi-bit aware router is designed to utilize configuration memory sharing. To accomplish this, the router first differentiates multi-bit signals from single-bit signals. It then routes the multi-bit signals based on a modified negotiated-congestion algorithm [9]. Unlike the conventional negotiated-congestion algorithm [26], however, the multi-bit router rips up and reroutes one multi-bit signal (instead of one bit) at a time. By ripping up an entire signal, the router ensures that configuration memory sharing switches can always be identically configured when the signal is rerouted through configuration memory sharing resources [9].

If the signal is rerouted through non-configuration memory sharing resources, however, the ripping up of multiple bits can reduce routing efficiency—one bit might simply move to the tracks that are vacated (or made less costly) by the other bits [27]. Consequently, in this work, we employ the multi-bit aware routing tool only in the investigations that involve configuration memory sharing.

TABLE I
PARAMETER SETTINGS FOR THE SPARSE AND THE CONFIGURATION MEMORY SHARING ROUTING ARCHITECTURES

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.4	1.0
	$F_{c\_out\_multi}$	0.25	0.8
	W_{multi}	4	60
Common Parameters	L (Segment Length)	2	N.A.
	Multiplexer	Minimum Area	N.A.
	Routing Tool	Multi-Bit Aware [9]	N.A.

In each investigation, however, we do employ the same router across all architectures in order to eliminate routing tool variations. In particular, we observe the multi-bit aware routing tool is significantly different from the conventional routing tool due to its unique routing schedules (caused by the differentiation of multi-bit and single-bit signals), its special penalties for routing multi-bit/single-bit signals on single-bit/multi-bit tracks and its capability of deciding if multi-bit signals should be routed on multi-bit or single-bit tracks based on the routing results of test bits [9].

A. Effect of Configuration Memory Sharing on Area Efficiency

To investigate the effect of configuration memory sharing on the area efficiency of FPGAs, we measure the amount of routing area that is required to implement the benchmark circuits for both the configuration memory sharing and the sparse routing architectures. The parameter settings used in this investigation are shown in Table I. As shown, the F_c values ($F_{c\_\mathrm{in\_single}}$ and $F_{c\_\mathrm{out\_single}}$) are kept constant for single-bit tracks in both architectures. These values were found to be the most efficient for the configuration memory sharing architecture in [8]. We assume that they are equally efficient for the sparse routing architecture. Similarly, the routing track segment length was set to two and the minimum area implementation of the multiplexer is used.

We varied F_c values ($F_{c\_\mathrm{in\_multi}}$ and $F_{c\_\mathrm{out\_multi}}$) for the multi-bit tracks for both the configuration memory sharing and the sparse routing architectures. For each set of $F_{c\_\mathrm{in\_multi}}$ and $F_{c\_\mathrm{out\_multi}}$ values, we varied the number of multi-bit tracks from 4 to 64 tracks. For a given number of multi-bit tracks, we search for the minimum number of single-bit tracks that are required to implement each circuit.

Figs. 15 and 16 show the average number of routing track segments that are required to implement a circuit for the configuration memory sharing and the sparse routing architectures, respectively. There are two curves in each figure-one shows the average number of single-bit track segments that are required to implement a circuit while the other shows the average number of multi-bit track segments. As shown, for the configuration memory sharing architecture, the utilization of the multi-bit

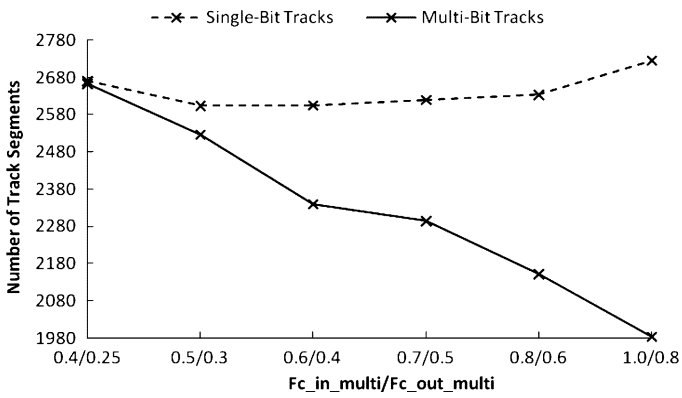


Fig. 15. Number of single-bit and multi-bit routing tracks in the configuration memory sharing routing architecture.

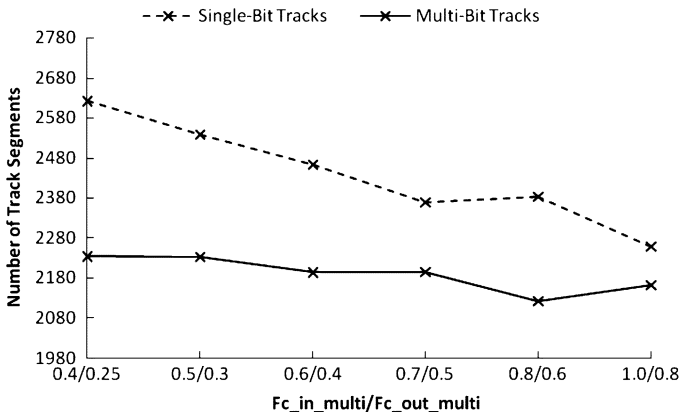


Fig. 16. Number of single-bit and multi-bit routing tracks in the sparse routing architecture.

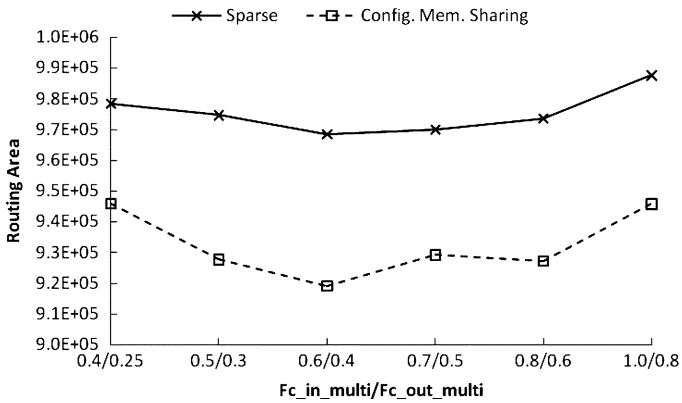


Fig. 17. Routing area for sparse and configuration memory sharing routing architectures.

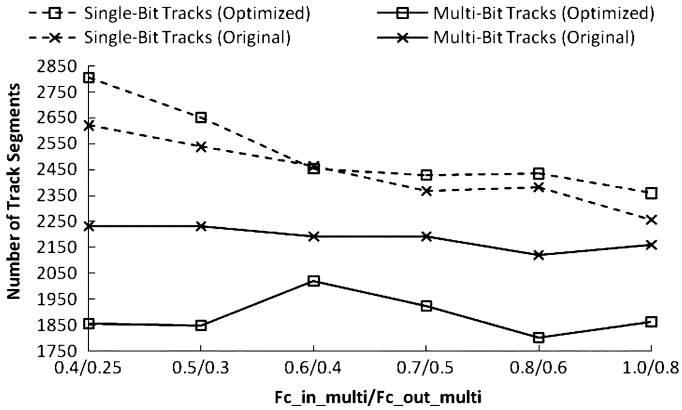


Fig. 18. Multi-Bit Track Reduction Due to Routing Algorithm Optimization.

tracks increases with the increasing values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$. In particular, when $F_{c\_in\_multi}$ is set to 0.4 and $F_{c\_out\_multi}$ is set to 0.25, 2663 multi-bit track segments are required to implement a circuit. When $F_{c\_in\_multi}$ is increased to 1.0 and $F_{c\_out\_multi}$ is increased to 0.8, only 1985 multi-bit track segments are required. Note that this reduction is largely due to the more efficient use of the multi-bit tracks by the multi-bit signals since the number of single-bit track segments stays largely the same over all values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$.

Fig. 16 shows that when configuration memory sharing is removed from the multi-bit tracks, there is a substantial reduction in the number of multi-bit track segments for small values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$. This reduction is due to the increase in multi-bit track flexibility and their per-track implementation area. As the values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$ increase, the utilization of the multi-bit tracks increases as well. This increase, however, is due to the more efficient use of multi-bit tracks by multi-bit signals as well as single-bit signals. Consequently, the total number of single-bit track segments decreases with increasing values of $F_{c\_in\_multi}$ and $F_{c\_out\_multi}$.

Fig. 17 shows the effect of multi-bit track utilization on routing area. In the figure, area is the minimum average routing area across 15 benchmarks. The curve above is for the sparse routing architecture while the curve below is for the configuration memory sharing routing architecture. As shown, the best sparse architecture consumes 5% more routing area than the best configuration memory sharing architecture.

In Fig. 17, the same multi-bit aware routing algorithm [9] is used for both the configuration memory sharing and the sparse routing architectures. The algorithm, however, is specialized for configuration memory sharing. Since it is very expensive to route a single bit of signal on a set of configuration memory sharing routing resources, this algorithm encourages multi-bit signals to use multi-bit tracks as much as possible. It heavily penalizes the act of breaking a multi-bit signal into individual bits and routing some of the bits on single-bit tracks and the remaining bits on multi-bit tracks [9], [24]. Without configuration memory sharing, however, the penalty still forces the single-bit and multi-bit signals to stay on their respective routing resources even when one type of resource is much more congested than the other.

We modified the routing algorithm for the sparse architecture by removing the penalty. With the penalty removed, more multi-bit signals can be routed on single-bit tracks. This results in a slight increase in the number of single-bit track segments and a significant reduction in the number of multi-bit track segments as shown in Fig. 18. Consequently, the routing area of the sparse architecture is further reduced as shown in Fig. 19.

To determine the best proportion of multi-bit routing tracks for the configuration memory sharing and the sparse routing

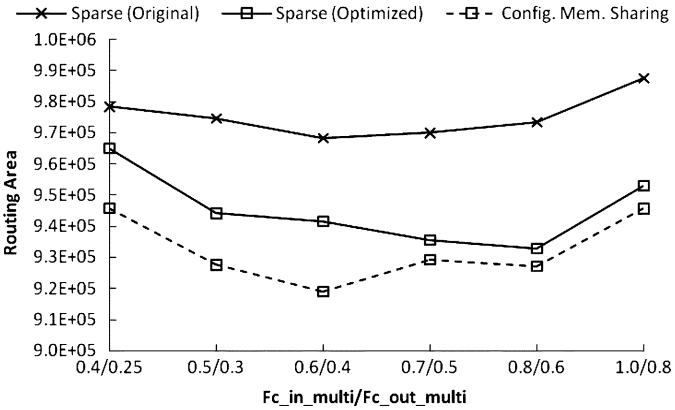


Fig. 19. Reduction in routing area for the sparse routing architecture due to routing algorithm optimization.

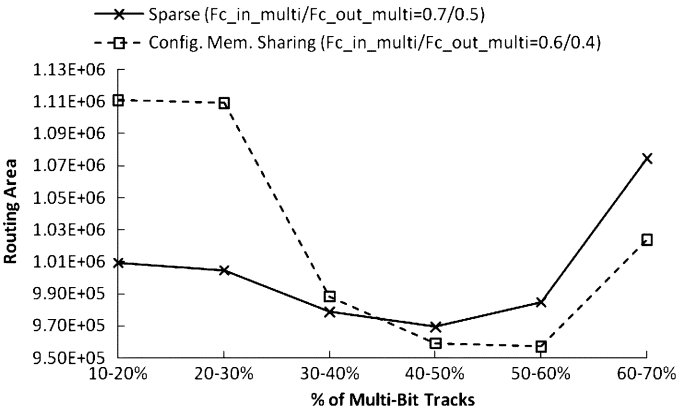


Fig. 20. Routing area versus % of multi-bit tracks.

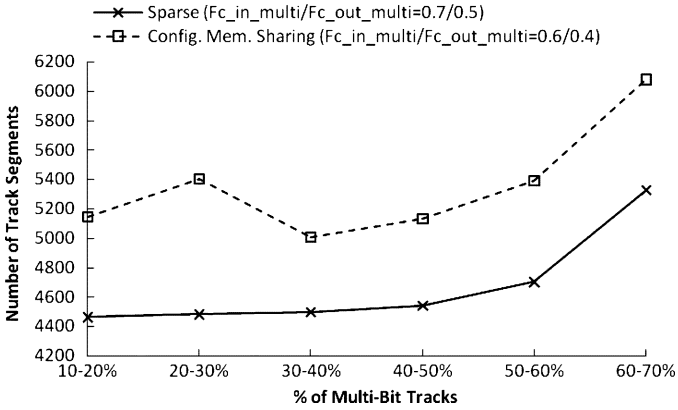


Fig. 21. Track segments versus % of multi-bit tracks.

TABLE II
PARAMETER SETTINGS FOR THE ENHANCED SPARSE AND THE CONFIGURATION MEMORY SHARING ROUTING ARCHITECTURES

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.6 (Configuration Memory Sharing)/ 0.7 (Enhanced Sparse)	N.A.
	$F_{c\_out\_multi}$	0.4 (Configuration Memory Sharing)/ 0.5 (Enhanced Sparse)	N.A.
	W_{multi}	4	60
	F_{c_enh}	0	1.0
Common Parameters	L (Segment Length)	2	N.A.
	Multiplexer	Minimum Area	N.A.
	Routing Tool	Multi-Bit Aware [9] (Optimized)	N.A.

architectures, we repeat the above experiment by fixing the percentage of multi-bit routing tracks. The result is shown in Fig. 20. As shown the best number of multi-bit tracks as a percentage of the total number of tracks is between 50% and 60% for the configuration memory sharing architecture and 40% to 50% for the sparse architecture. The best sparse architecture consumes 1.2% more routing area than the configuration memory sharing routing architecture. Fig. 21 shows the average number of track segments as a function of the percentage of multi-bit tracks. As shown the most area efficient sparse routing architecture uses 16% less track segments than the most area efficient configuration memory sharing routing architecture.

B. Shift and Control Enhancements Versus Configuration Memory Sharing

In this section, we evaluate the effect of the shift and control enhancements on the area efficiency of FPGAs. The parameter settings used in this investigation are shown in Table II. In particular, the same parameter values from Section IV-A are used for the single-bit tracks. For multi-bit tracks, the $F_{c\_\text{in}\_\text{multi}}$ and $F_{c\_\text{out}\_\text{multi}}$ values are set to 0.6 and 0.4 for the configuration memory sharing architecture and 0.7 and 0.5 for the enhanced sparse architecture. These values are shown to be the most area efficient in Section IV-A. For each set of $F_{c\_\text{in}\_\text{multi}}$ and $F_{c\_\text{out}\_\text{multi}}$ values, we vary the F_{c_enh} value from 0 to 1.0. For each value of F_{c_enh}, we vary the number of multi-bit tracks from 4 to 64 to find the minimum number of single-bit tracks that is required to route a circuit.

For the control-enhanced sparse architecture, Fig. 22 shows the effect of F_{c_enh} on the average number of routing track segments that is required support a circuit. As shown, as the value of F_{c_enh} is increased from 0 to 0.6, more and more single-bit signals are routed on the multi-bit tracks. Consequently the number of multi-bit tracks increases and the number of single-bit tracks decreases. As F_{c_enh} is increased beyond 0.6, the number of multi-bit and single-bit tracks stabilizes as most of the signals that can be efficiently routed through the multi-bit tracks have already been moved onto these tracks.

Fig. 23 shows the effect of control enhancements on area. As shown, the control-enhanced sparse architecture is more efficient than the configuration memory sharing architecture over all percentage values of multi-bit tracks. In particular, the best control-enhanced sparse architecture (at 60%–70% multi-bit tracks) is 1.8% smaller than the best configuration memory sharing architecture (at 50%–60% multi-bit tracks). Fig. 24 shows the best control-enhanced sparse architecture requires 18% less routing tracks than the best configuration memory sharing architecture.

Table III summarizes the best implementation area for each benchmark circuit for the configuration memory sharing, the sparse, and the control-enhanced sparse architectures. These circuits are divided into four groups according to the percentage

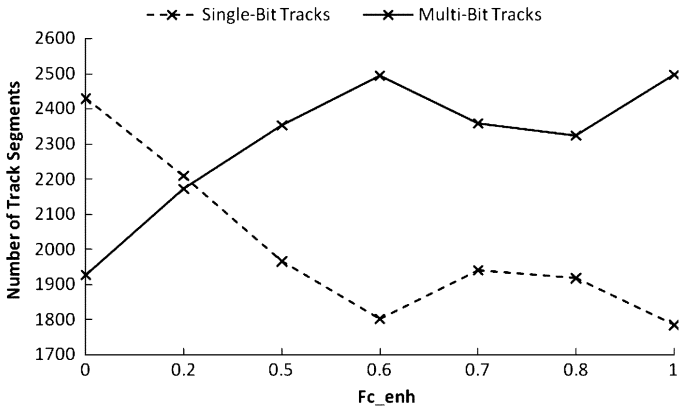


Fig. 22. Number of single-bit and multi-bit routing tracks in the control-enhanced sparse architecture.

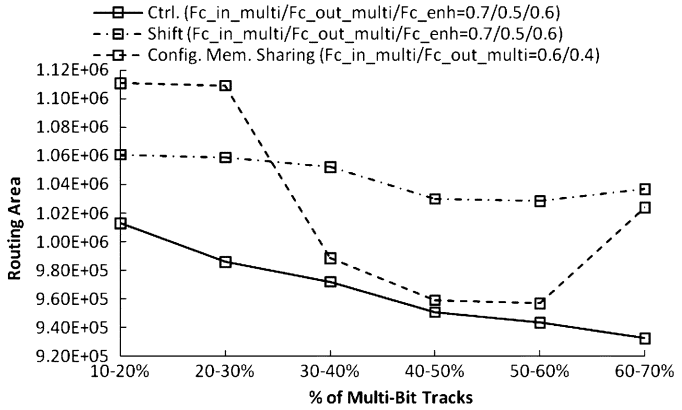


Fig. 23. Routing area versus % of multi-bit tracks.

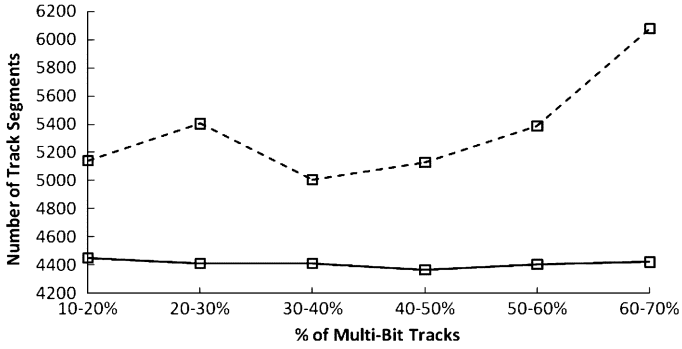


Fig. 24. Track segments versus % of multi-bit tracks.

of multi-bit signals that they contain. As shown, for circuits with less than 60% multi-bit signals, the enhanced sparse architectures are more area efficient than the configuration memory sharing architecture. For circuits with more than 60% multi-bit signals, the configuration memory sharing architecture is more area efficient.

Fig. 23 also shows the effect of shift enhancements on area. As shown, the shift-enhanced sparse architecture is less area efficient than the configuration memory sharing architecture. This is primarily due to the small amount of single-fanout shift signals that present in the benchmarks—unlike the multi-fanout signals targeted by the control switch patterns, the single-fanout signals targeted by the shift switch patterns can often be eliminated by mapping their source clusters onto the same bit positions as their sink clusters by the CAD tools [20], [24].

TABLE III
ROUTING AREA VERSUS % OF MULTI-BIT SIGNALS PER CIRCUIT

Percentage Range	Benchmark (% of Multi-Bit Signals)	Routing Area (1E+5)		
		Config. Mem. Sharing	Sparse	Enhanced Sparse (F_{c_enh}=0.6)
30-40%	multmod_dp (32%)	18.4	16.5	14.9
	Average	*18.4*	*16.5*	***14.9***
40-50%	prils_dp (42%)	2.87	2.70	2.64
	code_seq_dp (46%)	2.81	2.77	2.71
	exponent_dp (47%)	5.19	5.31	4.91
	incmod (48%)	7.86	7.18	6.86
	Average	*4.68*	*4.49*	***4.28***
50-60%	smu_dpath (51%)	4.29	4.06	3.89
	imdr_dpath (53%)	11.4	11.8	11.6
	pipe_dpath (56%)	2.72	2.56	2.63
	mantissa_dp (56%)	10.4	10.7	10.6
	Average	*7.19*	*7.28*	***7.18***
Over 60%	icu_dpath (61%)	32.2	32.2	31.1
	ucode_dat (61%)	9.36	10.5	9.86
	rsadd_dp (61%)	2.02	1.98	1.95
	ex_dpath (61%)	26.2	28.8	28.0
	dcu_dpath (65%)	7.33	7.87	7.64
	ucode_reg (74%)	0.58	0.51	0.64
	Average	***12.9***	*13.6*	*13.2*

C. Multi-Bit Aware Versus Conventional Routing Architectures

Having compared the area efficiency of the multi-bit aware routing architectures, this section compares two of the most area efficient architectures—the sparse and the control-enhanced sparse architectures—to the area efficiency of the conventional routing architecture. Table IV shows the parameter settings used in this investigation. In particular, the conventional routing tool [25], [26] is used since none of the architectures employs configuration memory sharing and our results show the conventional routing tool outperforms the multi-bit aware tool [9] for these architectures. For the conventional routing architecture, the F_{c_out} value is set to 0.25. It is shown to be the most area efficient in previous studies [28]. The F_{c_in} value is varied from 0.3 to 0.8 to measure the minimum routing area that is required to implement the benchmarks. Fig. 25 shows that 0.4 is the most area efficient F_{c_in} value for the conventional architecture.

For the sparse and enhanced sparse architectures, we vary $F_{c\_\mathrm{in\_multi}}, F_{c\_\mathrm{out\_multi}}, W_{\mathrm{multi}}$, and F_{c_enh} as in Sections IV-A and IV-B. We found that for the conventional routing tool, the most area efficient parameter values are $F_{c\_\mathrm{in\_multi}} = 0.6$ and $F_{c\_\mathrm{out\_multi}} = 0.4$ for the sparse architecture and $F_{c\_\mathrm{in\_multi}} = 0.6, F_{c\_\mathrm{out\_multi}} = 0.4$, and $F_{c\_\mathrm{enh}} = 0.8$ for the control-enhanced sparse architecture.

Fig. 26 shows the routing area as a function of the percentage of multi-bit tracks for the three architectures. As shown the control-enhanced sparse architecture is the most area efficient. At 70%–80% multi-bit tracks, it consumes 10% less area than the conventional routing architecture. The figure also shows that

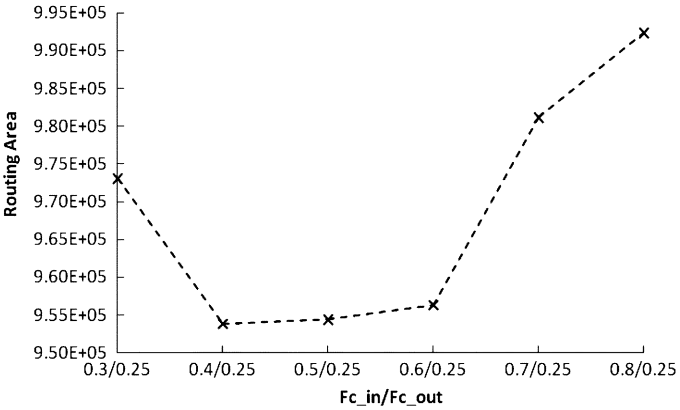


Fig. 25. Most area efficient F_{c_in} values for the conventional routing architecture.

TABLE IV
PARAMETER SETTINGS FOR THE CONVENTIONAL, SPARSE, AND CONTROL-ENHANCED SPARSE ARCHITECTURES

Classification	Parameter	Minimum Value	Maximum Value
Parameters for Conventional Architecture	F_{c_in}	0.3	0.8
	F_{c_out}	0.25	N.A.
	W	best	N.A.
Parameters for Single-Bit Tracks	$F_{c\_in\_single}$	0.4	N.A.
	$F_{c\_out\_single}$	0.25	N.A.
	W_{single}	best	N.A.
Parameters for Multi-Bit Tracks	$F_{c\_in\_multi}$	0.4	1.0
	$F_{c\_out\_multi}$	0.25	0.8
	W_{multi}	4	60
	F_{c_enh}	0	1.0
Common Parameters	L (Segment Length)	2	N.A.
	Multiplexer	Minimum Area	N.A.
	Routing Tool	Conventional [25][26]	N.A.

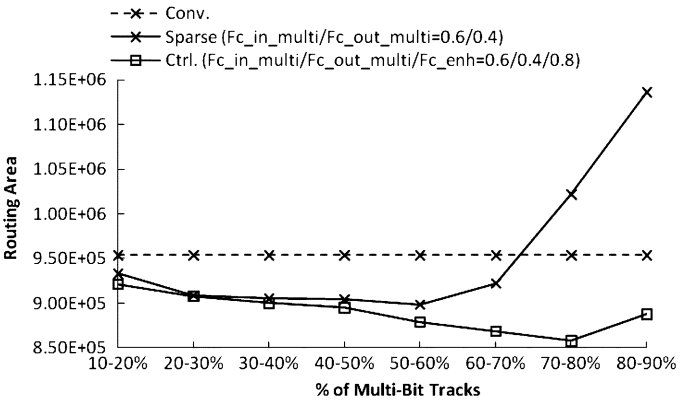


Fig. 26. Routing area versus % of multi-bit tracks.

the sparse architecture also consumes less area than the conventional architecture. At 50%–60% multi-bit tracks, the sparse architecture is 5.8% smaller than the conventional architecture.

Finally, Fig. 27 shows the number of track segments per circuit as a function of the percentage of multi-bit tracks. As shown both the sparse and the enhanced sparse architectures consume more track segments than the conventional architecture—with the best sparse architecture employing 8% more track segments and the best control-enhanced sparse architecture employing 4% more track segments.

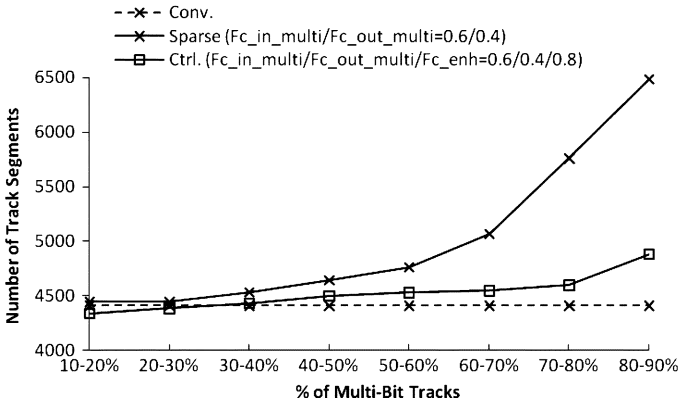


Fig. 27. Track segments versus % of multi-bit tracks.

TABLE V
ROUTING DELAYS OF 15 BENCHMARKS ROUTED ON THE FOUR BEST ARCHITECTURES

Benchmarks	Routing Delays (seconds)			
	Conventional	Config.Mem. Sharing	Sparse	Enhanced
code_seq_dp	9.413E-09	8.009E-09	8.711E-09	9.413E-09
dcu_dpath	3.797E-09	3.797E-09	3.797E-09	3.797E-09
ex_dpath	2.767E-08	2.767E-08	2.556E-08	2.626E-08
exponent_dp	1.558E-08	1.558E-08	1.558E-08	1.558E-08
icu_dpath	2.275E-08	2.275E-08	2.275E-08	2.275E-08
imdr_dpath	2.907E-08	2.907E-08	2.907E-08	2.907E-08
incmod	2.837E-08	2.767E-08	2.767E-08	2.837E-08
mantissa_dp	4.348E-09	4.348E-09	4.348E-09	4.348E-09
multmod_dp	2.345E-08	2.135E-08	2.345E-08	1.347E-08
pipe_dpath	1.067E-08	1.067E-08	1.067E-08	1.067E-08
prils_dp	1.152E-08	1.082E-08	1.152E-08	1.012E-08
rsadd_dp	2.626E-08	2.556E-08	2.626E-08	2.556E-08
smu_dpath	2.556E-08	2.556E-08	2.556E-08	2.556E-08
ucode_dat	3.797E-09	3.797E-09	3.095E-09	2.944E-09
ucode_reg	2.242E-09	2.242E-09	2.242E-09	2.242E-09
Geometric Mean	**1.224E-08**	**1.194E-08**	**1.193E-08**	**1.144E-08**

Table V shows the routing delays of the 15 benchmarks routed on the most area efficient conventional, configuration memory sharing, sparse and enhanced sparse architectures. The geometric means of the routing delays across the 15 benchmarks for the four architectures are listed at the bottom of the table. The geometric means of the routing delays show that the most area efficient sparse and enhanced sparse architectures have slightly better performance than the most area efficient conventional and configuration memory sharing architectures.

V. CONCLUSION

In this work we investigated the area efficiency of two multi-bit aware routing architectures—the sparse and the enhanced sparse architectures—and compared them against the configuration memory sharing and the conventional routing architectures for implementing arithmetic intensive applications. We found that, under the multi-bit aware router, both the sparse and the enhanced sparse architectures are as efficient as the configuration memory sharing architecture. In particular, our benchmarks show that the best sparse architecture only consumes 1.2% more area than the configuration memory sharing architecture while the best enhanced sparse architecture

consumes 1.8% less. The data show that the sharing of configuration memory does not increase the area efficiency of multi-bit aware routing architectures for the arithmetic intensive circuits investigated in this work. Instead, the multi-bit aware switch patterns are directly responsible for area reductions.

More importantly, the area efficiency of the sparse and the enhanced sparse architectures can be further improved by substituting the multi-bit aware router with the bit-oriented conventional router. We found that, using the conventional router, the sparse architecture is 5.8% more area efficient than the conventional routing architecture while the control-enhanced sparse architecture is 10% more area efficient. Our results suggest that as modern FPGAs are being increasingly used to implement large multi-bit processing circuits, FPGA architects should look beyond the conventional routing architecture in order to create more area efficient FPGAs.

Finally, here are the two suggested directions of future research. First, since the power efficiency of FPGAs can correlate directly to the active area that they consume, the effect of the new datapath-oriented switch patterns on the power efficiency of FPGAs should be investigated in the future. Second, future research should also include further investigations into how the configuration memory sharing routing architecture can be enhanced to better accommodate control signals. The enhanced architecture should then be compared against the current control-enhanced sparse architecture for area and power efficiency.

REFERENCES

[1] D. Cherepacha and D. Lewis, "DP-FPGA: An FPGA architecture optimized for datapaths," *J. VLSI Des.*, vol. 4, no. 4, pp. 329–343, Apr. 1996.

[2] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiD—Reconfigurable pipelined datapath," in *Proc. Int. Workshop Field-Programmable Logic Applications*, 1996, pp. 126–135.

[3] J. Hauser and J. Wawrzynek, "Garp: A MIPS processor with a reconfigurable coprocessor," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 1997, pp. 12–21.

[4] A. Marshall *et al.*, "A reconfigurable arithmetic array for multimedia applications," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 1999, pp. 135–143.

[5] A. Alsolaim *et al.*, "Architecture and application of a dynamically reconfigurable hardware array for future mobile communication systems," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2000, pp. 205–214.

[6] S. Goldstein *et al.*, "PipeRench: A reconfigurable architecture and compiler," *IEEE Comput.*, vol. 33, no. 4, pp. 70–77, Apr. 2000.

[7] K. Leijten-Nowak and J. van Meerbergen, "An FPGA architecture with enhanced datapath functionality," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2003, pp. 195–204.

[8] A. Ye and J. Rose, "Using bus-based connections to improve field-programmable gate array density for implementing datapath circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 5, pp. 462–473, May 2006.

[9] A. Ye and J. Rose, "Measuring and utilising the correlation between signal connectivity and signal positioning for FPGAs containing multi-bit building blocks," *Proc. Inst. Electr. Eng.—Comput. Digit. Tech.*, vol. 153, no. 3, pp. 146–156, May 2006.

[10] I. Koren, *Computer Arithmetic Algorithms*. Natick, MA: A. K. Peters Ltd., 2002.

[11] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*. Norwell, MA: Kluwer Academic, Feb. 1999.

[12] H. Hseih *et al.*, "Third-generation architecture boosts speed and density of field-programmable gate arrays," in *Proc. IEEE Custom Integrated Circuits Conf.*, 1990, pp. 31.2.1–31.2.7.

[13] G. Lemieux and S. Brown, "A detailed router for allocating wire segments in field-programmable gate arrays," in *Proc. ACM/SIGDA Physical Design Workshop*, 1993, pp. 215–226.

[14] *The Programmable Logic Data Book,* Xilinx, Inc., San Jose, CA, 1994.

[15] G. Lemieux, S. Brown, and D. Vranesic, "On two-step routing for FPGAs," in *Proc. Int. Symp. Physical Design*, 1997, pp. 60–66.

[16] Y. Chang, D. Wong, and C. Wong, "Universal switch modules for FPGA design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 1, no. 1, pp. 80–101, Jan. 1996.

[17] S. Wilton, "Architectures and algorithms for field-programmable gate arrays with embedded memory," Ph.D. dissertation, Univ. of Toronto, Toronto, ON, Canada, 1997.

[18] M. Masud and S. Wilton, "A new switch block for segmented FPGAs," in *Proc. IEEE Int. Conf. Field Programmable Logic Applications*, 1999, pp. 274–281.

[19] G. Lemieux and D. Lewis, "Analytical framework for switch block design," in *Proc. IEEE Int. Conf. Field Programmable Logic Applications*, 2002, pp. 122–131.

[20] A. Ye, J. Rose, and D. Lewis, "Synthesizing datapath circuits for FPGAs with emphasis on area minimization," in *Proc. Int. Conf. Field-Programmable Technology*, 2002, pp. 219–226.

[21] A. Ye and J. Rose, "Using multi-bit logic blocks and automated packing to improve field-programmable gate array density for implementing datapath circuits," in *Proc. Int. Conf. Field-Programmable Technology*, 2004, pp. 129–136.

[22] V. Betz and J. Rose, "How much logic should go in an FPGA logic block?," *IEEE Des. Test Comput.*, vol. 15, no. 1, pp. 10–15, Jan. 1998.

[23] Pico-Java Processor Design Documentation, Sun Microsystems, Santa Clara, CA, 1999.

[24] A. Ye, "Field-programmable gate array architectures and algorithms optimized for implementing datapath circuits," Ph.D. dissertation, Univ. of Toronto, Toronto, ON, Canada, Nov. 2004.

[25] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. Int. Conf. Field Programmable Logic Applications*, 1997, pp. 213–222.

[26] C. Ebeling, L. McMurchie, S. Hauck, and S. Burns, "Placement and routing tools for the triptych FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[27] P. Chen, "The effect of multi-bit correlation on the design of routing resources in field programmable gate arrays," M.A.Sc. thesis, Ryerson Univ., Torontao, ON, Canada, Oct. 2008.

[28] V. Betz and J. Rose, "FPGA routing architecture: Segmentation and buffering to optimize speed and density," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 1999, pp. 59–68.

Phoebe Ping Chen received the B.A.Sc. degree in electronic engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 1996 and the M.A.Sc degree in computer engineering from Ryerson University, Toronto, ON, Canada, in 2009.

From 1997 to 2001, she was an Assistant Engineer in the Guangxi TV station in Nanning, Guangxi, China. In 2001, she joined the Shenzhen branch of the ASK Engineering, Ltd., Guangdong, China, as a Hardware Engineer. From 2002 to 2004, she worked as a Software Engineer in the Guangxi Highland Digital Corp., Ltd., Guangxi, China. After immigrating to Canada in 2004, she joined the SAE Power in Toronto, ON, Canada, as a technician and entered Ryerson University in 2006. Her current research interests include field-programmable gate array (FPGA) architectures and FPGA applications in video processing.

Andy Ye (S'97–M'06) received the B.A.Sc., M.A.Sc., and Ph.D. degrees in computer engineering from the University of Toronto, Toronto, ON, Canada, in 1996, 1999, and 2004, respectively. He graduated first in class in the engineering science program in 1996.

From 1999 to 2000, he participated in the development of the Ultragizmo board for the University of Toronto Undergraduate Microprocessor Laboratory. Currently, he is an Assistant Professor in the Department of Electrical and Computer Engineering at Ryerson University, Toronto, ON, Canada. His research interests include field-programmable gate array (FPGA) architectures, computer-aided design (CAD) tools for FPGAs, logic synthesis, and hardware implementation of computer graphics algorithms.