THE EFFECT OF SPARSE SWITCH PATTERNS ON THE AREA EFFICIENCY OF MULTI-BIT ROUTING RESOURCES IN FIELD-PROGRAMMABLE GATE ARRAYS

Ping Chen and Andy Ye

Department of Electrical and Computer Engineering
Ryerson University
350 Victoria Street, Toronto, Ontario, Canada M5B 2K3
Email: pepe_chen@hotmail.com, aye@ee.ryerson.ca

ABSTRACT

The increased use of multi-bit processing elements such as digital signal processors, multipliers, multi-bit addressable memory cells, and CPU cores has presented new opportunities for Field-Programmable Gate Array (FPGA) architects to utilize the regularity of multi-bit signals to increase the area efficiency of FPGAs. In particular, configuration memory sharing has been traditionally used to exploit multi-bit regularity for area. We observe that the process of creating configuration memory sharing routing resources often leads to the use of much sparser switch patterns for connecting multi-bit elements to their routing tracks. In this work, we empirically evaluate the effect of these sparse switch patterns on the area efficiency of FPGAs. It is shown that the sparse switch patterns alone contribute significantly to the area reduction observed in configuration memory sharing FPGAs. In particular, our experiments show that, without configuration memory sharing, sparse switch patterns can reduce the implementation area of multi-bit routing resources by 10.4% while configuration memory sharing contributes to an additional 1.2% in area savings. The observation holds over a wide range of connection block flexibility values and demonstrates that efficient switch pattern designs can be effectively used to increase the area efficiency of FPGA routing resources.

1. INTRODUCTION

Field-Programmable Gate Arrays (FPGAs) are programmable devices that are designed to implement digital systems. They are optimized for hardware algorithms and have the added advantage of being able to change their functionalities in a fraction of second. Being both hardware-oriented and programmable, FPGAs provide a unique blend of performance and flexibility, which has been proven essential in many applications. Typically, only 25% of FPGA area is actually used to perform computation while the remaining 75% is used to connect the computing elements together [1][2]. Due to this vast area consumption, the design of these interconnects (called routing resources) is as important as the design of the computing elements.

As the logic capacity of FPGA increases, there has been a corresponding increase in the variety of FPGA computing elements. From a mere collection of logic blocks, FPGAs now can include digital signal processors, multipliers, multi-bit addressable memory cells, and even processor cores. One of the common characteristics of these new computing elements is their multi-bit design, where each element is designed to process several bits of data at a time.

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 processing element are logically organized to represent multiple-bit wide data. In this organization, pins that represent a datum are often used at the same time. Similarly, routing resources are routinely used to transport multiple-bit wide data from a common source to a common destination.

To transport a multi-bit wide datum, one can either treat the datum as a set of independent signals and transport these signals individually through a set of conventional routing resources [3], or view the entire datum as a single coherent unit and transport the unit collectively through a set of specialized routing resources [4]-[11]. Called multi-bit routing resources, these specialized routing resources can be more area efficient than the conventional routing resources [4][10][11]. Their increased area efficiency is partly due to the more efficient use of configuration memory. In particular, multi-bit routing resources can be configured one datum at a time while conventional routing resources must be configured one bit at a time. This increase in configuration granularity can lead to a significant reduction in the amount of memory that is required to configure these resources and hence increases their area efficiency.

The correlated behaviors of multi-bit signals, however, can affect the area efficiency of FPGAs in other ways. In particular, the sharing of configuration memory requires routing resources to be grouped into multi-bit wide groups. Connecting two groups together usually consists of

978-1-4244-1961-6/08/$25.00 ©2008 IEEE.

connecting each bit in one group to a corresponding bit in the other [11]. This one-to-one mapping of routing resources often results in a much sparser distribution of routing switches than the conventional routing architectures. The analytical work in [11] suggests that the sparser connection patterns can contribute significantly to the increase in area efficiency for multi-bit routing resources. The analysis, however, has made several simplifying assumptions and the effect has not been empirically studies before. In this work, we empirically evaluate the effect of the sparse connection patterns on the overall area efficiency of the multi-bit routing resources.

The remainder of this paper is organized as follows: Section 2 describes the routing architectures used in this investigation, Section 3 presents the experimental results, and Section 4 concludes.

2. ROUTING ARCHITECTURE

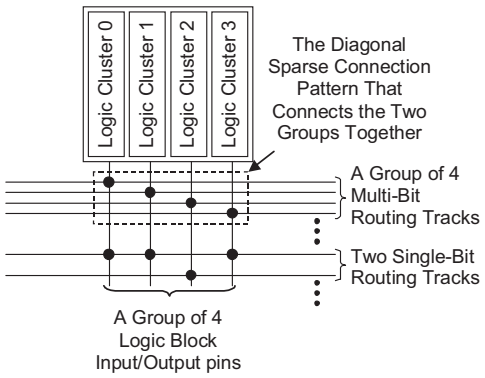

Fig. 1. Multi-Bit Logic Block

```
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] = step * i + increment * j
  }
}
```

Fig. 2. Connection Distribution Algorithm

To investigate the effect of sparse connection patterns on the area efficiency of FPGAs, we mapped a set of datapath circuits onto a set of specialized logic blocks. Each block, as shown in Figure 1, consists of four logic clusters [2], each of which contains four 4-input look-up tables, four d-type flip flops, ten cluster-level inputs and four cluster-level outputs.

As in [11], the logic clusters are used to implement the adjacent bit-slices of a datapath. This implementation creates a large number of inter-logic-block signals that share common sources and destinations. In particular, for benchmarks used in this work, 48% of all two terminal connections that connect the logic blocks together can be grouped into 4-bit wide groups, where each group shares a common source logic block and a common destination logic block [12].

We investigated three architectures for connecting the logic blocks to their routing tracks. These architectures include the conventional routing architecture, the configuration memory sharing routing architecture, and the sparse routing architecture. In the conventional routing architecture, the logic blocks are connected to the routing tracks using the conventional connection patterns as proposed in [3]. In particular, all logic block input/output signals are treated as independent signals. The algorithm shown in Figure 2 is used to distribute the connections as uniformly as possible across all routing tracks. Note that, in the figure, W is set to be the number of routing tracks that a logic block connects to, P is set to be the number of input/output pins per logic block, and F_c is set to be the percentage of tracks in W that each pin connects to. Each entry in the resulting matrix, switch_pattern[i][j], contains the index of a routing track that is connected to logic block pin i through the jth connection of i.

The configuration memory sharing routing architecture as proposed in [11], on the other hand, groups a portion of the routing tracks (called the multi-bit routing tracks) into 4-bit wide groups. Similarly, corresponding input/output pins from the logic clusters are also grouped into 4-bit wide groups. These input/output pins are then connected to the routing tracks one group at a time instead of one bit at a time.

Note that when connecting two groups of signals together the sparse diagonal connection pattern as shown in Figure 1 is used. The algorithm shown in Figure 2 is again used to distribute groups of connections (instead of bits of connection) as uniformly as possible across the routing tracks. In this case, W is set to be the number of groups of routing tracks that a logic block connects to. P is set to be the number of input/output pin groups. F_c is set to be the percentage of track groups in W that a pin group connects to. Finally, each entry in the matrix, switch_pattern[i][j], contains the index of a track group that is connected to pin group i.

To further minimize area, as in [11], at logic block outputs, all four switches in every diagonal connection pattern share a single set of configuration memory. Within each group of multi-bit routing tracks, the corresponding switches in the switch blocks also share a single set of configuration memory [11]. Finally, the remaining routing tracks (called the single-bit routing tracks), are connected to the logic block input/output pins using the conventional switch patterns.

To isolate the effect of the sparse connection patterns on the area efficiency of FPGAs, the sparse routing

architecture employs the same connection patterns as the configuration memory sharing routing architecture. Each switch in this architecture, however, is independently controlled by its own configuration memory instead of being collectively controlled by shared configuration memory.

3. EXPERIMENTAL RESULTS

We varied F_c values for multi-bit tracks and measured their effect on the routing area of the configuration memory sharing and the sparse routing architectures. Throughout the experiment, F_c values are kept constant for single-bit tracks for both architectures as well as the conventional routing architecture. In particular, F_c is set to be 0.4 for input pins and 0.25 for output pins. These values were determined to be the best in [11] for single-bit tracks. The same set of benchmark circuits from [11] were used in this work. The set consists of 15 datapath circuits from the Pico-Java processor [14]. As in [11], we varied the number of multi-bit tracks employed in both the configuration memory sharing and the sparse architectures. For a given number of multi-bit tracks, we then searched for the minimum number of single-bit tracks that are required to successfully implement each circuit.

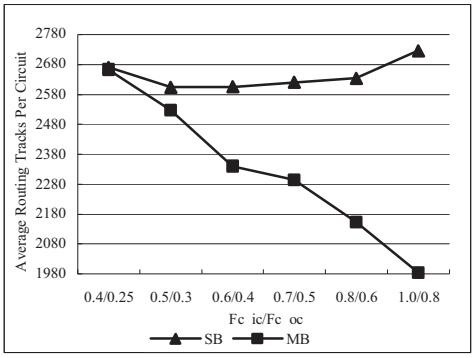

Fig. 3. # of Track Segments Vs. F_c for the Configuration Memory Sharing Architecture

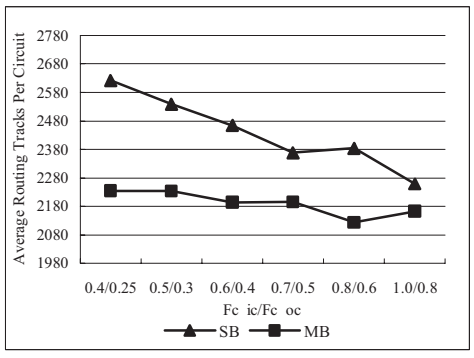

Fig. 4. # of Track Segments Vs. F_c for the Sparse Architecture

Figure 3 and 4 shows 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. In the figure, *Fc_ic* denotes the F_c values for the input pin to multi-bit track connections and *Fc_oc* denotes the F_c values for the output pin to multi-bit track connections. There are two curves in each figure – one shows the average number of single-bit tracks that are required to implement a circuit and the other shows the average number of multi-bit tracks. As shown, for the configuration memory sharing architecture, the utilization of the multi-bit tracks increases with the increasing values of F_c. In particular, when F_c is set to 0.4 for the input pins and 0.25 for the output pins, 2663 multi-bit track segments are required to implement a circuit. When F_c is increased to 1.0 for the input pins and 0.8 for the output pins, 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 (signals that can be grouped into 4-bit wide groups) since the number of single-bit track segments stays largely the same over all values of F_c.

Figure 4 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. This decrease is due to the increase in multi-bit track flexibility. As the values of F_c 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 (signals that can not be grouped into 4-bit wide groups). Consequently, the total number of single-bit track segments decreases with increasing values of F_c.

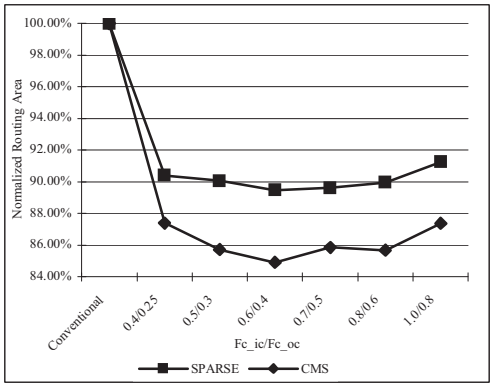

Fig. 5. Routing Area Vs. F_c

Figure 5 shows the effect of the utilization and the sparseness of multi-bit tracks on routing area. In the figure, area is normalized against the routing area of the conventional routing architecture. The curve above is for the sparse routing architecture while the curve below is for the configuration memory sharing routing architecture. As shown, the sparse architecture alone gains over 10% routing

area savings over the conventional architecture. Configuration memory sharing, on the other hand, gains an additional 4-5% area savings.

In Figure 5, the same routing algorithm is used for both the sparse and the configuration memory sharing architectures. The algorithm, however, is specialized for configuration memory sharing [15]. Since configuration memory sharing makes multi-bit tracks very specialized for multi-bit signals, this algorithm 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 [15]. This penalty, however, often causes congestion on multi-bit tracks.

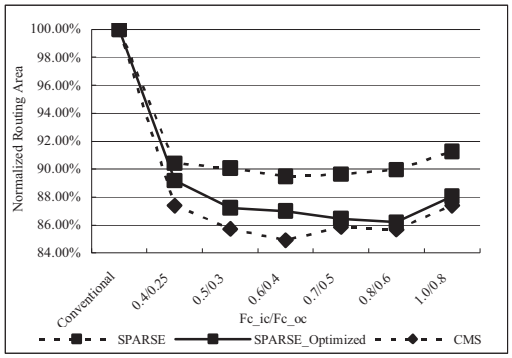

Fig. 6. Routing Area Vs. F_c with Routing Algorithm Optimized for the Sparse Architecture

We removed the penalty for the sparse architecture. This modification results in a slight increase in single-bit track segment count and significant reduction in multi-bit track segment count. The routing area of the sparse architecture is further reduced as shown in Figure 6.

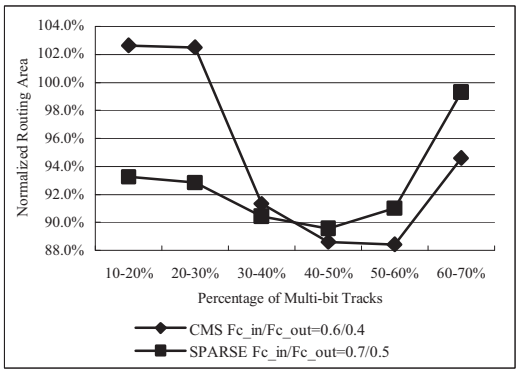

Fig. 7. Routing Area Vs. % of Multi-Bit Tracks

To determine the best proportion of multi-bit routing tracks for the configuration memory sharing and the sparse architectures, we repeated the above experiment by fixing the percentage of routing tracks [11]. The result is shown in Figure 7. As shown the best number of multi-bit tracks as a percentage of the total number of tracks is between 50 to 60% for the configuration memory sharing architecture and 40 to 50% for the sparse architecture. The best sparse architecture achieves 10.4% routing area savings, while the best configuration memory sharing architecture achieves an additional 1.2%.

4. CONCLUSION

This paper evaluates the relative contribution of configuration memory sharing and sparse connection patterns on the area efficiency of multi-bit routing tracks. It is shown that the sparse connection patterns contribute to 10.4% reduction in routing area while configuration memory sharing contributes an additional 1.2%. It is also shown that the sparse architectures consume significantly less routing tracks than the configuration memory sharing architectures.

5. REFERENCES

[1] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," *Proc. IEEE,* vol. 81, no. 7, pp. 1013-1029, Jul. 1993.

[2] E. Ahmed and J. Rose, "The Effect of LUT and Cluster Size on Deep-Submicron FPGA Performance and Density," in *TVLSI,* vol. 12, no. 3, pp. 288-298, Mar. 2004.

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

[4] D. Cherepacha and D. Lewis, "DP-FPGA: An FPGA Architecture Optimized for Datapaths," *J VLSID,* vol. 4, no. 4, pp.329–343, Apr. 1996.

[5] C. Ebeling, D. Cronquist, and P. Franklin, "RaPiD – Reconfigurable Pipelined Datapath," *FPL,* 1996, pp.126–135.

[6] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," *FCCM,* 1997, pp.12–21.

[7] A. Marshall, et. al, "A Reconfigurable Arithmetic Array for Multimedia Applications," *FPGA,* 1999, pp.135–143.

[8] A. Alsolaim, et. al, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems," *FCCM,* 2000, pp.205–214.

[9] S. Goldstein, et. al, "PipeRench: A Reconfigurable Architecture and Compiler," *IEEE Computer,* vol. 33. no. 4, pp.70–77, Apr. 2000.

[10] K. Leijten-Nowak and J. van Meerbergen, "An FPGA Architecture with Enhanced Datapath Functionality," *FPGA,* 2003, pp.195–204.

[11] A. Ye and J. Rose, "Using Bus-Based Connections to Improve Field-Programmable Gate Array Density for Implementing Datapath Circuits," *TVLSI,* vol. 15, no. 5, pp.462–473, May 2006.

[12] A. Ye, J. Rose, and D. Lewis, "Synthesizing Datapath Circuits for FPGAs with Emphasis on Area Minimization," *ICFPT,* 2002, pp. 219-226.

[13] A. Ye and J. Rose, "Measuring and Utilising the Correlation Between Signal Connectivity and Signal Positioning for FPGAs Containing Multi-Bit Building Blocks," *IEECDT,* vol. 153, no. 3, pp.146-156, May 2006.

[14] Pico-Java Processor Design Documentation, Sun Microsystems, 1999.

[15] A. Ye, "Field-Programmable Gate Array Architectures and Algorithms Optimized for Implementing Datapath Circuits," *Ph.D. Thesis*, University of Toronto, Nov. 2004.