Detailed Routing Architectures for Embedded Programmable Logic IP Cores

Peter Hallschmid and Steven J.E. Wilton
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada
{peterh, stevew}@ece.ubc.ca

ABSTRACT
As the complexity of integrated circuits increases, the ability to make post-fabrication changes to fixed ASIC chips will become more and more attractive. This ability can be realized using programmable logic cores. These cores are blocks of programmable logic that can be embedded into a fixed-function ASIC or a custom chip. Such cores differ from stand-alone FPGAs in that they can take on a variety of shapes and sizes. With this in mind, we investigate the detailed routing characteristics of rectangular programmable logic cores. We quantify the effects of having different x and y channel capacities, and show that the optimum ratio between the x and y channel widths for a rectangular core is between 1.2 and 1.5. We also present a new switch block family optimized for rectangular cores. Compared to a simple extension of an existing switch block, our new architecture leads to an 8.7% improvement in density with little effect on speed. Finally, we show that if the channel widths and switch block are chosen carefully the penalty for using a rectangular core (compared to a square core with the same logic capacity) is small; for a core with an aspect ratio of 2:1, the area penalty is 1.6% and the speed penalty is 1.1%.

Categories and Subject Descriptors
D.7.1 [**Integrated Circuits**]: Types and Design Styles – VLSI (Very large scale integration)

General Terms
Design, Performance, Experimentation.

Keywords
Detailed Routing, Programmable Logic, FPGA, Embedded Cores, SoC Design.

1. INTRODUCTION
Field-Programmable Gate Arrays have rapidly become the implementation medium of choice for many digital circuits. Recent years have seen a plethora of research aimed at improving the speed and density of these devices. Unfortunately, despite significant progress, for many applications FPGAs still do not provide the required speed or density. For these applications, designers can still enjoy the benefits of flexibility and configurability by creating a custom chip or cell-based ASIC, and incorporating a programmable logic IP (Intellectual Property) core. A chip designed this way would contain both fixed logic and programmable logic. Parts of the chip that are unlikely to change can be implemented using the fixed ASIC circuitry or fixed IP (Intellectual Property) cores, while functions that may change can be implemented in the programmable logic core.

There are several scenarios in which incorporating a programmable logic region within a fixed ASIC would be advantageous:

1. Some design details can be left until late in the design cycle. In a communications application, for example, the development of the chip can proceed while standards are being finalized. Once the standards are set, they can be incorporated in the programmable portion of the chip. This is important, since time-to-market is so critical in industry today.
2. As products are upgraded, or as standards change, it may be possible to incorporate these changes using the programmable part of the chip, without fabricating an entirely new device.
3. In many cases, it may be possible to fabricate a single chip for an entire family of devices; the characteristics that differentiate each member of the family can be implemented

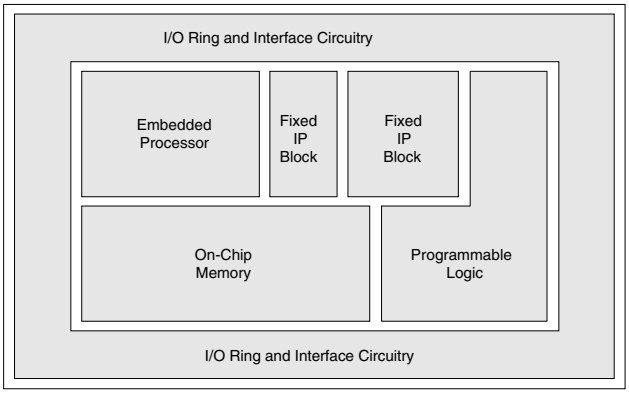

Figure 1: Hypothetical System-on-Chip with Programmable Logic IP Core.

FPGA 2001, February 11-13, 2001, Monterey, California, USA.

using the programmable logic. This would, in effect, amortize the cost of developing the ASIC over several products. In a similar way, standard products can be customized for different customers by implementing the parts of the circuit specific to that customer in the programmable logic portion of the chip.

4. Testing structures can be implemented using the programmable logic. Consider a chip with a faulty functional unit. To debug the functional unit (so changes can be made in later spins of the chip), it may be very valuable to construct a small circuit that can stimulate or analyze the faulty functional unit. The nature of this circuit would be unknown when the chip was initially designed, and thus would be a natural candidate for the programmable logic portion of the chip.

The use of programmable logic cores fits well with the emerging system-on-a-chip (SoC) design methodology, in which third-party cores are combined on a single chip. The chip in Figure 1 is a hypothetical chip that follows this design style. As the figure shows, the chip contains a processor core, some on-chip memory, two fixed IP blocks, and a programmable logic core. In this case, the programmable logic core is simply another core that the system-on-a-chip designer can buy from a third party. Already, several companies offer such cores [1][2].

The potential benefits of integrating fixed and programmable logic described above are so compelling that we feel that the ability to make post-fabrication changes in a fixed-function ASIC will eventually become as important as design-for-testability is today. In order for this to happen, however, there are a number of challenges that must be overcome. Most programmable logic architecture and CAD tool research has focused on stand-alone FPGAs. Much of this will not carry over to embedded programmable cores, which can take on many different shapes and aspect ratios. A second issue is the integration of the FPGA CAD flow into the existing ASIC design flow. Yet another challenge is the pre-tape-out verification of an ASIC with a programmable logic core; it is unclear how such a chip can be verified if the circuit to be included within the programmable logic core is unknown. Finding solutions to these problems is part of a larger project at the University of British Columbia where we are studying the System-on-a-Chip design style, and the use of programmable logic within that design style.

In this paper, we focus on just one of these problems: the architecture of the programmable logic core, specifically the detailed routing architecture. One of the most significant differences between embedded cores and stand-alone FPGAs is that embedded cores can take on a variety of shapes and sizes to better mesh with the fixed ASIC circuitry. One SoC design may need a relatively square programmable logic block, while another may need a long narrow programmable logic block (perhaps placed along an entire edge of the chip to provide a programmable interface to the I/O ports). Yet another design may need several small cores. The example in Figure 1 uses an L-shaped core. Each of these sizes and shapes makes unique demands on the detailed routing architecture. These demands and their implications to the detailed routing architecture are the focus of this paper. Specifically, this paper focuses on two aspects:

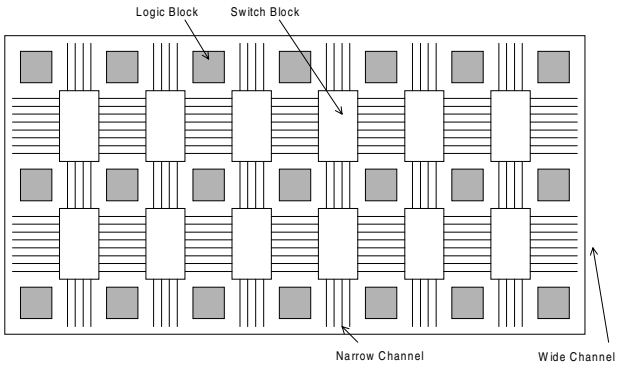

Figure 2: an FPGA with unequal horizontal and vertical channel capacities.

1. In [3], it was suggested that for a rectangular FPGA, channels in the long direction should have more tracks than channels in the narrow direction (as shown in Figure 2). On average, signals will have to travel further in the long direction of the FPGA so channels in this direction should have a higher track capacity. This is quantified in [3]; however, [3] approaches the problem using global routing only. However, due to the limited connectivity with programmable logic routing, the detailed routing architecture will have a significant effect on the optimum channel widths for a given programmable logic core. In this paper, we revisit this question, considering the detailed routing architecture.
2. At each intersection between a horizontal and vertical channel lies a switch block. Since switch blocks use up a large portion of area and are a very significant part of the routing flexibility, the design of a good switch block is of the utmost importance. Thus, there has been considerable work developing efficient switch block architectures [4][5][6][7]. All of the previous switch blocks are square; in other words, they assume the same number of incident tracks for all sides. In our environment, however, vertical and horizontal channels will often have different widths. A second contribution of the paper is the presentation of a new switch block optimal for cores with different vertical and horizontal channel widths.

Thus, the contribution to this paper is two-fold: we quantify the effects of different channel widths in rectangular programmable logic cores, and we propose a new switch block specifically for programmable logic cores that are directionally biased.

2. ARCHITECTURAL FRAMEWORK

In this paper, we restrict our attention to rectangular island-style programmable logic cores. Each core consists of $n_x * n_y$ clusters, each containing r logic elements (in the results of Section 4, we fix $r = 4$). Each logic element consists of a 4-input lookup table and a flip-flop. The logic clusters are surrounded by a grid of routing channels; each vertical channel contains t_y parallel tracks, and each horizontal channel contains t_x parallel tracks. In this paper, we assume, without loss of generality, that $t_y > t_x$. We assume a segmented architecture in which every track spans 4 logic clusters. Tracks of the same channel are staggered relative to each other [9]. At the intersection of each horizontal and vertical

channel is a switch block that comprises of 50% pass-transistors and 50% tri-state buffers (see [10]). Connection-block and switch-block populations are both 100% (see [11]).

3. RECTANGULAR SWITCH BLOCK

At the intersection of each horizontal channel and each vertical channel lies a switch block. Each switch block provides for programmable connections between tracks of the incident channels. Switch blocks consume a large portion of the area of an FPGA and their structure has a significant effect on the routablility and speed attainable by the device. Thus, optimizing the architecure of the switch block structure is very important.

Previously proposed switch blocks such as the Disjoint [4], Universal [5], and Wilton [6] have all been square; an equal number of tracks enters each side of the block. In each of these previous switch blocks, each incoming track can be connected to one or more of three outgoing tracks (ie. $F_s = 3$). In [7], a switch block was proposed for FPGAs with segmented routing. In such an FPGA, each track spans s switch blocks, and the track start points are staggered, so that at each switch block, $1/s$ of the tracks end, while the remaining pass through. The switch block in [7] distinguishes between these two types of tracks; tracks that ended were connected using a Wilton pattern, while tracks that passed straight through were connected using a Disjoint pattern. It was shown that this scheme helped improve the routability of the device significantly. Thus, we use the switch block from [7] (which we will call the Imran block) as a building block in this paper.

Since a programmable logic core can take on so many different shapes and sizes, it makes sense to define a *family* of switch blocks, where each family member is best for certain shapes and sizes of the core. In the following subsection we describe a baseline rectangular switch block and in Subsection 3.2 we show how a family of switch blocks can be derived from this baseline.

3.1 Baseline Rectangular Switch Block

Our baseline rectangular switch block is a natural extension of a square Imran switch block. As shown in Figure 3, our block is composed of several subblocks. With t_x tracks incident to the switch block from the x-channel and t_y tracks incident from the y-channel, there are $N_s = ceil(t_y/t_x)$ subblocks (called SSBn where $0 \le n < N_s$). The $(N_s\text{-}1)^{th}$ subblock will be a partial subblock if the number of vertical channels is not evenly divisible by the number of horizontal channels. The switch block can be represented by a graph $M(T,S)$ where each node in T represents a terminal (incident track) of the switch block and each edge in S represents a programmable switch that connects two terminals. T is partitioned into $2+2N_s$ subsets, each with $W=t_x$ terminals; two subsets represent the tracks incident to the short sides of the switch block and $2N_s$ subsets represent the tracks incident to the long sides.

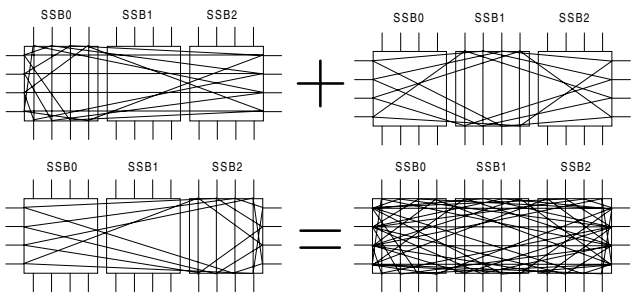

Figure 3: The tracks of a rectangular switch block are the summation of tracks from three individual Imran Patterns.

Each terminal in T is labeled $t_{m,n}$ where m is the subset number $(0 \le m \le 2N_s+1)$ and n is the terminal number within the subset $(0 \le n \le W-1)$.

The programmable connections within our baseline rectangular switch block can be obtained by replicating the Imran pattern N_s times. As an example, Figure 4 shows the switching pattern assuming $t_y=12$ and $t_x=4$ and assuming all tracks end at this switch block. As the figure shows, the final switching pattern is the sum of three individual Imran switch block patterns. Note that, in general, the last sub-block may have fewer than t_x vertical inputs; this is called a partial sub-block. The connection pattern for a partial sub-block is the same as a full sub-blocks with the exception that a connection is not made if, due to it being a partial sub-block, one or more of its incident tracks does not exist. This exception is incorporated into the algorithm using the function, Exists($t_{m1,n1}$, $t_{m2,n2}$), where $t_{m1,n1}$ and $t_{m2,n2}$ represent incident tracks. Figure 5 shows an algorithmic description of the baseline switch block.

3.2 Family of Rectangular Switch Blocks

As N_s becomes large, the baseline switch block becomes unsuitable. Consider the block in Figure 4. In this example, the effective F_s for horizontal tracks (number of choices for each incident vertical track) is 7. In general, if there are N_s sub-blocks, then $F_s=1+2N_s$. This high value for F_s is harmful for two reasons:

1. The routing delay of a net using a horizontal track will be high because of the large number of switches incident to the track at each switch block.
2. The large number of switches translates to a large layout area, which will reduce the achievable logic density of the device.

This motivates us to develop a new switch pattern with fewer potential connections for each track.

Rather than defining a single switch block, we define a family of switch blocks, where each member of the family is obtained by depopulating the baseline switch block by a different amount. Members of the family will be referred to as *MODn* where n is the proportion of diagonal switches that have been removed. Figure 7 shows an example MOD2 and MOD4 block, assuming $t_y/t_x=4$. Note that in Figure 7, t_x parallel connections are shown by a solid line for clarity (the pattern within each solid line is the Imran pattern as shown in Figure 4).

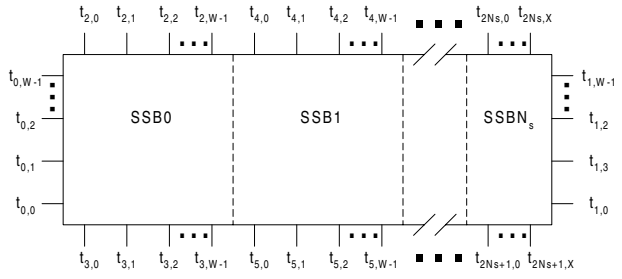

Figure 4: Illustration of terminal labeling scheme.

```
S = ∅
 for j = 0 to Ns-1
     for k = 0 to W-1

         if incident wires t0, k, t1, k, t2, k, t3, k end at switch block

             // Wilton Switch Block Pattern
             S = S ∨ { (t0, k, t1, k), (t2j+2, k, t2j+3, k) }

             if ( Exists(t0, k, t2j+3, (W-k) mod W) )
                 S = S ∨ { (t0, k, t2j+3, (W-k) mod W) }

             if ( Exists(t2j+3, k, t1, (k+1) mod W) )
                 S = S ∨ { (t2j+3, k, t1, (k+1) mod W) }

             if ( Exists(t1, k, t2j+2, (2W-2-k) mod W) )
                 S = S ∨ { (t1, k, t2j+2, (2W-2-k) mod W) }

             if ( Exists(t2j+2, k, t0, (k+1) mod W) )
                 S = S ∨ { (t2j+2, k, t0, (k+1) mod W) }

         else if incident wires t0, k, t1, k, t2, k, t3, k pass through switch block

             // Disjoint Switch Block Pattern
             S = S ∨ { (t0, k, t2j+3, k) }

         end
     end
 end
```

Figure 5: Connection algorithm for the baseline rectangular core.

```
S = ∅
 for j = 0 to Ns-1
     for k = 0 to W-1

         if incident wires t0, k, t1, k, t2, k, t3, k end at switch block

             // Wilton Switch Block Pattern
             S = S ∨ { (t0, k, t1, k), (t2j+2, k, t2j+3, k) }

             if ( ( j mod n = α0 ) && Exists(t0, k, t2j+3, (W-k) mod W) )
                 S = S ∨ { (t0, k, t2j+3, (W-k) mod W) }

             if ( ( j mod n = α1 ) && Exists(t2j+3, k, t1, (k+1) mod W) )
                 S = S ∨ { (t2j+3, k, t1, (k+1) mod W) }

             if ( ( j mod n = α2 ) && Exists(t1, k, t2j+2, (2W-2-k) mod W) )
                 S = S ∨ { (t1, k, t2j+2, (2W-2-k) mod W) }

             if ( ( j mod n = α3 ) && Exists(t2j+2, k, t0, (k+1) mod W) )
                 S = S ∨ { (t2j+2, k, t0, (k+1) mod W) }

         else if incident wires t0, k, t1, k, t2, k, t3, k pass through switch block

             // Disjoint Switch Block Pattern
             S = S ∨ { (t0, k, t2j+3, k) }

         end
     end
 end
```

Figure 6: Connection algorithm for a MODn rectangular switch block.

An algorithmic description of the family of switch blocks is given in Figure 6. Depopulating the switch block removes connections in a pattern dictated by what we call sub-block coefficients which are identified by a 4-tuple, $A_{MODn}=(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$. The values of each tuple depends on n; for a MOD2 switch block, we have arbitrarily chosen $A_{MOD2}=(0, 1, 0, 1)$, while for a MOD4 block, the pattern is $A_{MOD4}=(0, 3, 1, 2)$. These patterns can be seen in the example of Figure 7.

4. EXPERIMENTAL RESULTS

To investigate the suitability of the new switch block family, we used an experimental approach. Twenty benchmark circuits from the Microelectronics Corporation of North Carolina (MCNC) were used [13]. Because we were primarily interested in circuits which would fit into an embedded programmable logic core, we chose circuits which were approximately 10% of the size of circuits which could be implemented in commercially available stand-alone FPGAs. Half of the circuits were sequential, and half were combinational. Each circuit was optimized and technology-mapped using SIS [14] and Flowmap/Flowpack [15]. The logic elements were then packed to logic clusters using Vpack [16] and timing-driven placement and routing was performed using a modified version of VPR [16]. For each circuit and architecture, the minimum number of tracks required for 100% routability was found; the number of tracks in each channel was then increased by 20%, and the routing repeated. In all cases, we assumed a 0.18 µm CMOS process from TSMC.

First consider the effect that the track ratio (t_y/t_x) has on the overall density and speed of the programmable logic core. Figure 8 shows results assuming the core aspect ratio is 4:1 (ie. the core is four times as tall as it is wide). The upper graph in Figure 8 shows the area of the core as a function of the track ratio. The curve is a result of two competing trends. The first trend marks an increase in the number of transistors in each switch block as the track ratio increases (since there are more sub-blocks). The second trend shows that as the track ratio increases, channel capacity increases in the long direction of the core, thus increasing the

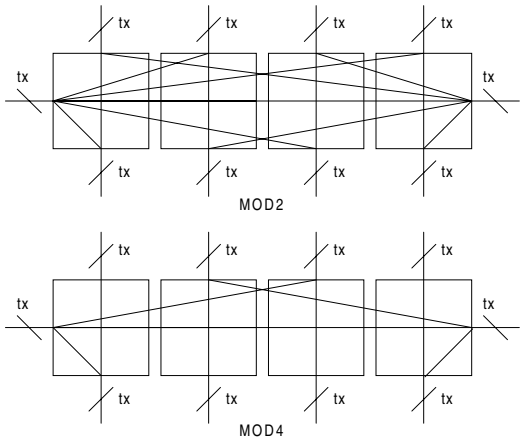

Figure 7: MOD2 and MOD4 rectangular switch patterns.

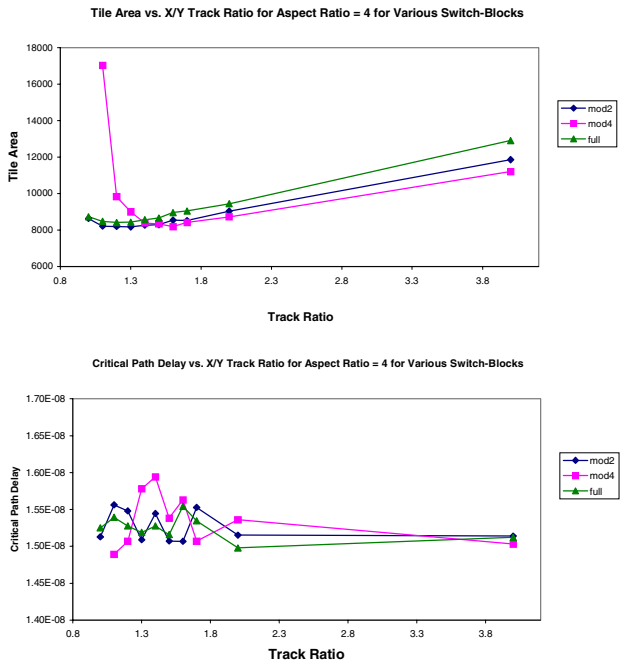

Figure 8: Area/Delay as a function of track ratio for various switch blocks. The core ratio is 4:1.

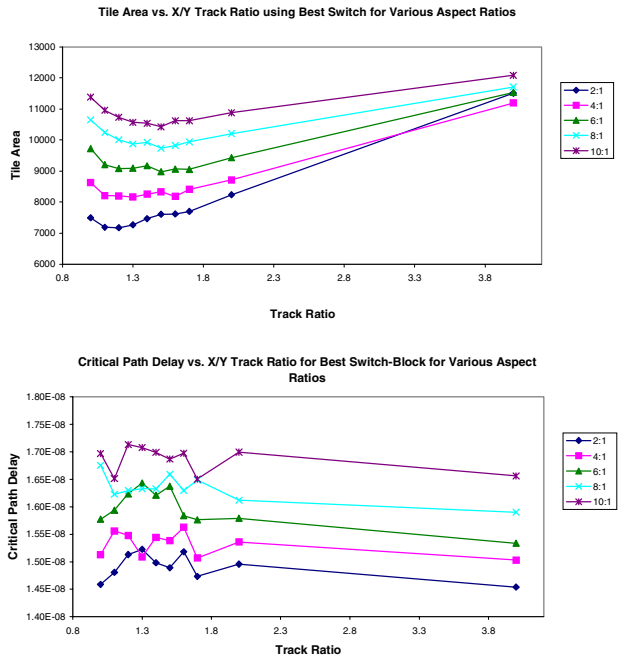

Figure 9: Plot of tile area and critical path delay versus track ratio for each aspect ratio. The ideal switch block is used for each data points.

overall routing flexibility. This leads to a reduction in the overall number of tracks required in the core and consequently a reduction in its overall area (or equivalently, an increase in the achievable density). Together, these trends cause a distinct minimum in the area curve, as shown in Figure 8. For all switch block types, the best track ratio was between 1 and 1.6. The average critical path delay is not significantly affected by the choice of t_y/t_x as shown in the lower graph of Figure 8. This experiment was repeated for aspect ratios of 2, 6, 8, and 10, with consistent results.

Figure 8 illustrates another interesting result: different members of our switch block family are appropriate for cores with different track ratios. As the figure shows, MOD2 is the best (in terms of tile area) for track ratios less than 1.5. For ratios greater than 1.5, MOD4 is the best choice and the baseline switch block should never be used.

Figure 9 shows the results across several aspect ratios. For each track ratio, the best switch block (from MOD2 and MOD4) was chosen. As the aspect ratio increases from 2:1 to 10:1, the curve minimum changes from approximately 1.2 to 1.5. This shows that the demand for channel capacity in the long direction of a rectangular programmable logic core increases with the aspect ratio. Note, however that this increase is not as dramatic as might be expected; the demand for extra tracks in the long direction needs to be balanced with the larger size of a highly non-square switch block.

Figure 10 compares the best results of Figure 9 with those obtained assuming $t_y/t_x=1$. The dotted line represents an architecture that maintains a unity track ratio and uses a baseline switch block for all aspect ratios. The solid line represents an

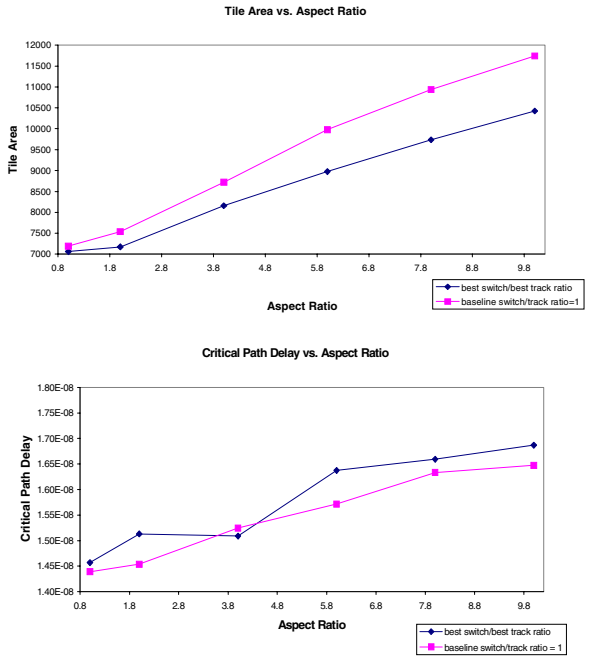

Figure 10: Area/Critical path delay as a function of track ratio. The dotted line represents the baseline switch block and an aspect ratio of 1:1. The solid line represents best the best switch block and the best track ratio for a given track ratio.

architecture that has the best switch block and track ratio for each aspect ratio. On average, the tile area for a programmable logic core with the best track ratio is 8.7% less than a core with a track ratio of unity; this improvement increases as the core aspect ratio increases. In terms of delay, a programmable logic core with the best track ratio is (with one exception) roughly 2.7% slower than a core with a track ratio of unity.

Figure 10 also shows that a square core is always best in terms of both area and critical path delay. This does not mean that rectangular cores are a bad idea; in many applications, the fixed shapes and sizes of the other cores will dictate that a rectangular programmable logic core is to be used. For small aspect ratios, the area density penalty for using a rectangular core is small; for an aspect ratio of 2:1, there is a 1.6% penalty. As the aspect ratio increases, however, the penalty increases. For an aspect ratio of 10:1, the penalty is 47.7%. In terms of delay, the penalty of using an apsect ratio of 2:1 is 1.1%, and is 14.5% for an aspect ratio of 10:1.

5. CONCLUSIONS

In this paper, we have investigated the detailed routing architectures of programmable logic cores. Specifically, we have focused on rectangular cores, and have made three contributions:

1. We have quantified the improvement in density and speed when using different channel widths in the x and y direction of rectangular cores. We showed that for a core aspect ratio of 2:1, a track ratio of 1.2 is appropriate, while for a core aspect ratio of 10:1, a track ratio of 1.5 is best.
2. We have shown that for small apsect ratios, there is only a small penalty associated with using a rectangular core (compared to a square core). For an aspect ratio of 2:1, the area penalty is 1.6% and the delay penalty is 1.1%. For very rectangular cores, however, this penalty becomes larger; for an aspect ratio of 10:1, the area penalty is 47.7% and the delay penalty is 14.5%.
3. We have presented a new family of switch blocks that work well in rectangular programmable logic cores. Compared to a simple extension of a previous switch block, we have shown that our family of switch blocks results in an area savings of 8.7%.

As mentioned in the introduction, this work is only a first step at examining the use of programmable logic cores in the SoC design style. More work is needed to quantify the benefits of merging small amounts of programmable logic onto a fixed ASIC chip, across all application domains. We feel that, eventually, the use of programmable logic in this way will become an essential tool in every chip designer's arsenal. When this happens, switch blocks and architectures such as the one described in this paper will become critical.

6. ACKNOWLEDGMENTS

This work was supported by the Micronet, the British Columbia Advanced Systems Institute, and the Natural Sciences and Engineering Research Council of Canada. The authors wish to thank the Canadian Microelectronics Corporation for providing process data, and Mike Sheng and Elias Ahmed for encapsulating this data into a VPR-readable format.

7. REFERENCES

[1] C. Matsumoto, “LSI Logic ASICs to add Programmable Logic Cores,” Electrical Engineering Times, August 29, 1999.

[2] C. Matsumoto, “Actel Plans to Produce FPGAs as ASIC Cores,” Electrical Engineering Times, September 20, 1999.

[3] V. Betz and J. Rose, “Directional and Non-Uniformity in FPGA Global Routing Architectures,” IEEE/ACM International Conference on Computer-Aided Design, 1996, pages 652-659.

[4] Xilinx, Inc., The Programmable Logic Data Book, 2000.

[5] Y.-W. Chang, D. Wong, and C. Wong, “Universal switch modules for FPGA design, “ACM Transactions on Design Automation of Electronic Systems, vol. 1, pp. 80-101, January, 1996.

[6] S.J.E. Wilton, “Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memory,” Ph.D. thesis, University of Toronto, 1997.

[7] M.I. Masud and S.J.E. Wilton, “A New Switch Block for Segmented FPGAs,” in Lecture Notes in Computer Science 1673, Springer-Verlag, pages 274-281.

[8] G.G. Lemieux and S.D. Brown, “A Detailed Router for Allocating Wire Segments in Field-Programmable Gate Arrays,” Proceedings of the ACM Physical Design Workshop, April, 1993.

[9] V. Betz and J. Rose, “Automatic Generation of FPGA Routing Architectures from High-Level Descriptions,” ACM*/SIGDA International Symposium on Field Programmable Gate Arrays*, 2000.

[10] V. Betz and J. Rose, “FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density,” *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 1999.

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

[12] A. Marquardt, V. Betz and J. Rose, ``Timing-Driven Placement for FPGAs," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2000.

[13] S. Yang, “Logic Synthesis and Optimization Benchmarks, Version 3.0,” *Tech. Report*, 1991.

[14] E.M. Sentovich et al, “SIS, A System for Sequential Circuit Analysis,” *Tech. Report No. UCB/ERL M92/41*, 1992.

[15] J. Cong and Y. Ding, “Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs,” IEEE *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 1, January 1994, pages 1-12.

[16] V. Betz and J. Rose, “VPR: A New Packing, Placement and Routing Tool for FPGA Research,” *Int. Workshop on Field-Programmable Logic and Applications*, August, 1997.