# A 128 × 128 × 24Gb/s Crossbar
# Interconnecting 128 tiles in a single hop and Occupying 6% of their area

Giorgos Passas, Manolis Katevenis, and Dionisis Pnevmatikatos
*Inst. of Computer Science (ICS)*
*Foundation for Research and Technology – Hellas (FORTH)*
*Heraklion, Crete, Greece*
*Email: {passas, kateveni, pnevmati}@ics.forth.gr*

***Abstract*—We describe the implementation of a 128×128 crossbar switch in 90nm CMOS standard-cell ASIC technology. The crossbar operates at 750MHz and is 32-bits wide to provide a port capacity above 20Gb/s, while fitting in a silicon area as small as 6.6mm$^2$ by filling it at the 90% level (control not included). Next, we arrange 128 1mm$^2$ "user tiles" around the crossbar, forming a 150mm$^2$ die, and we connect all tiles to the crossbar via global links that run on top of SRAM blocks that we assume to occupy three fourths of each user tile. Including the overhead of repeaters and pipeline registers on the global links, the area cost of the crossbar is 6% of the total tile area. Thus, we prove that crossbars are dense enough and can be connected "for free" for valencies exceeding by far the few tens of ports, that were believed to be the practical limit up to now, and reaching above one hundred ports. Applications include Combined Input-Qutput Queued switch chips for Internet routers and data-center interconnects and the replacement of mesh-type NoC for many-core chips.**

## I. Introduction

Crossbar switches are basic building blocks for interconnection networks. Because their cost grows with the square of their port count, it is commonly believed that they become overly expensive for valencies above 32 or 64.

In particular, designers generally believe that many-core chips need a multi-hop Network-on-Chip (NoC) to interconnect a few tens of processing tiles because a crossbar would be prohibitively expensive in terms of wires and crosspoints. Moreover, in the domain of router systems for multiprocessors, clusters, or the Internet, several designers consider that internal speedup is expensive for high-valency crossbars, and that crosspoint queueing is an effective method to eliminate it, thus replacing combined input-output queueing.

To our surprise, when we set out to quantitatively measure the above costs by doing real VLSI layouts, we discovered gross misconceptions in the above common beliefs. Using a conservative 90nm CMOS standard-cell ASIC technology, we layout a 128×128 crossbar switch with a port capacity of 24Gb/s in an area of just 6.6mm$^2$ (control is not contained in that area).

Gate count and wiring is so large that the standard EDA tools were unable to automatically place and route the crossbar components. However, owing to the regularity of the circuit, we wrote a script that algorithmically places the gates in an orderly layout. Although the standard cells are packed together at an area utilization of 90%, regularity and wiring resources are such that the EDA tools are then able to successfully route all the required wires above the standard cells. To our knowledge, this is denser than any previously published high-valency crossbar layout.

We assume that our crossbar is surrounded by 128 IP tiles, of size 1mm×1mm each, arranged in a 12×12 = 144 square array, where the crossbar and its control replace the 16 = 4×4 centermost tiles. Assuming that three fourths of each tile contain SRAM blocks (e.g. cache memories next to a processor, or queues in a switch chip), we find out that all 128 input and 128 output links of the crossbar can be easily routed over the SRAM blocks, thus incurring virtually no area overhead to the IP tiles for wiring –actually, the area overhead for repeaters and pipeline registers on these wires is 0.5% of the total tile area. Including this overhead, the area cost of the whole crossbar network is 6% of the total tile area.

Following the above discussion, the rest of this paper is organized as follows. We start by discussing related work in the next section II. In section III, we examine various site plans of the crossbar in its context of IP tiles and we explain why we opted for a centralized crossbar. In section IV, we describe the crossbar circuit and its layout. Section V presents area cost numbers of a baseline crossbar scheduler to demonstrate the feasibility of the control circuit. Finally, section VI is a conclusion.

## II. Related work

Kim e.a. [1] have recently shown that high-valency switch chips reduce the diameter of the interconnection network, and with it, its latency and power consumption, thus being a good tradeoff. Switch chips usually employ a crossbar to interconnect their ports because the crossbar is the simplest non-blocking topology. However, as the valency of the switch increases, the scheduling of the crossbar becomes harder. To simplify and improve the performance of crossbar scheduling, Kim e.a. adopted crosspoint queueing (CQ) as an alternative to the traditional input queueing (IQ). However,

©ACM/IEEE Proceedings of the 4th Int. Symposium on Networks on Chip – NOCS 2010, Grenoble, France, May 2010

CQ was found expensive due to the high partitioning of the switch memory; since there is one memory per crosspoint, the total number of memories grows as $O(N^2)$, which is costly for flow and congestion control algorithms –see references [2][3] for a further discussion on this cost.

Thus, Kim e.a. proposed the hierarchically-queued crossbar (HQ) as an organization that lowers memory partitioning. In this organization, an $N \times N$ crossbar is partitioned in $(N/k)^2$ $k \times k$ sub-crossbars and memories are placed only at the inputs and outputs of the sub-crossbars. Hence, the total number of memories is reduced from $O(N^2)$ to $O(N^2/k)$. Unfortunately, this organization has a major disadvantage: Although partitioning is lowered, it remains unacceptably high, especially when $N$ is large. The reason is that each sub-crossbar has to be relatively small in order to be efficiently scheduled, which, in turn, implies a small $k$ and a quick growth rate of the total number of memories.

High memory partitioning also increases the switch implementation cost. Switch memories are usually implemented with SRAM blocks to increase memory density. Given that technology bounds the maximum on-chip memory capacity, the higher the partitioning, the smaller the SRAM blocks. For small SRAM blocks, the area overhead of the peripheral control circuitry becomes comparable to the area of the memory-cell array [4]. Scott e.a. [5] showed an implementation of HQ for a 64×64 switch with 8×8 sub-crossbars. Due to the high partitioning, they implemented the memories with registers to avoid the small and costly SRAM blocks. Thus, they lowered memory density at least by an order of magnitude.

We are studying the combined input and output queued (CIOQ) switch organization. CIOQ places memories only at the inputs and outputs of the crossbar and compensates for scheduling inefficiencies by over-provisioning the throughput of both the crossbar and the memories –this over-provisioning is usually referred to as *internal speedup*. Thus, in CIOQ, memory partitioning grows as $O(N)$, i.e. by a factor of $N/k$ slower than in HQ.

We focus on the cost of speeding up the crossbar. We consider a reference 128×128×10Gb/s CIOQ switch and we study the implementation of a 128×128×20Gb/s crossbar, which gives a speedup of two. On the other hand, the cost of speeding up the memories is obvious and low, as their throughput easily expands by arranging some SRAM blocks in parallel[1].

Our work is also related to [6]. Pullini e.a. showed that using flat EDA flows, it is impossible to place and route a 32×32 32-bit-wide crossbar with area utilization above 50%. They considered this utilization unacceptable, thus they suggested avoiding excessively large crossbars in NoC. By contrast, in the same technology, we demonstrate an hierarchical flow which gives an area utilization of 90% for the same crossbar width and four times higher valency.

Last but not least, a high-valency crossbar was also used in the many-core chips of the IBM Cyclops64 supercomputer. The only publicly available information we could find on this crossbar, though, is in reference [7]. According to reference [7], this is a 96×96 96-bit-wide crossbar implemented in a 90nm technology, running at 533MHz, and occupying 27mm$^2$, including the circuits for queuing, arbitration, and flow control. However, reference [7] provides insufficient information on the structure of the crossbar circuit and layout, hence a direct comparison is difficult. Judging from this information, the crossbar organization we propose here is radically different. In particular, the IBM Cyclops64 processor chips appear using a crossbar with an one-dimensional, port-sliced layout, while we describe a more scalable two-dimensional layout, studying in detail the organization of the crossbar links over the SRAM memories, and we show that bit slicing significantly reduces area. Furthermore, we show why and how the metal tracks over both the standard cells and the SRAM blocks suffice to route the whole wiring of the crossbar.

## III. Crossbar Site Plans

We describe and compare several alternative locations of the crossbar on the chip in its context of *user tiles*. Typical designs use cores consisting of some processing element, $P$, and one or more memory blocks, denoted \$ in the figures below. We call these user tiles, and we do not care whether their memory is used as cache or local memory, and whether it is physically built as a couple of large blocks or a collection of several smaller ones. Similarly, in CIOQ switch chips, each user tile contains the input and output queues associated with one of the switch ports, and thus consists mostly of SRAM blocks, plus a small area for port control.

We denote by $N$ the number of user tiles; $N$ will also be the crossbar valency. We are interested in $N = 128$, but, in this section, we will be treating $N$ as a parameter for the sake of presentation.

We consider dies with an aspect ratio of 1 (i.e. square), as this is the best in terms of balancing the distances in the two axes; however, our discussion applies equally well to dies with any other reasonable aspect ratio. We denote by $\alpha$ the die edge size, so the total die area is $\alpha^2$, excluding chip I/O (pad and associated) circuitry. We consider $a = 12mm$.

Finally, we assume that the memory of each user tile comprises a significant portion of the tile area –up to three fourths thereof– and we use that area for routing wires on top of SRAM. This assumption is in accordance with most general purpose processing cores where the actual processing part is rather small compared to the memories and caches needed to support this processing. On the other

[1] 32-bit-wide, 4K-word-tall, single-port SRAM blocks have a worst-case latency of 2.9ns and an area of 0.3mm$^2$ and are optimal both latency and area wise [4]. For a throughput of 30Gbps, it suffices to arrange just three such blocks in parallel.

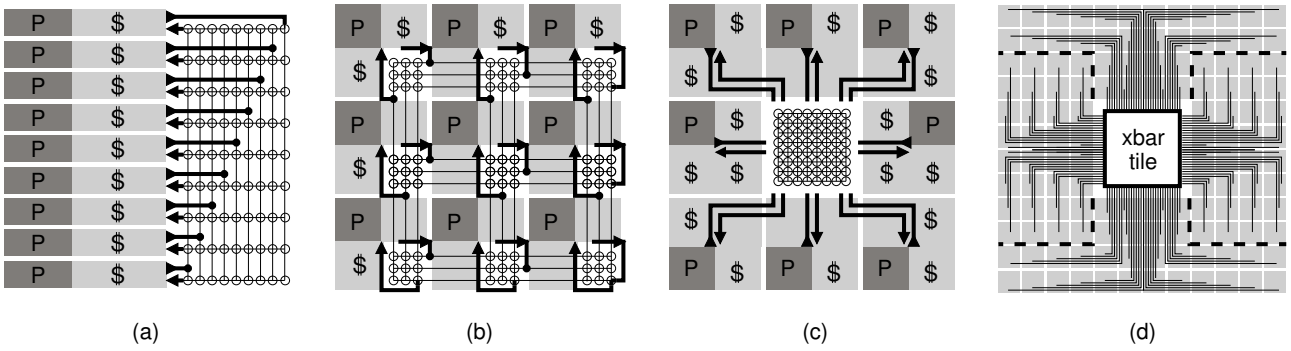


Figure 1. Crossbar site plans: (a) column of user tiles using side crossbar, (b) matrix of user tiles using distributed crossbar, (c) matrix of user tiles using centralized crossbar, (d) balanced link density of (c) in a "12×12 minus 4×4" matrix of user tiles.

hand, processing elements use substantial logic and complex interconnections and benefit from the use of all metal layers. In contrast, SRAMs are much simpler in their wiring needs; in our reference process [4], they obstruct only the four lower of nine metal layers, thus leaving the five upper metal layers available for routing in both directions.

### A. Column (1-D) of User Tiles using Side Crossbar

Figure 1(a) shows the first site plan considered. The user tiles are arranged in a single column, with the crosspoints to each of them lying on its side.

While being simple, this arrangement is non-scalable to large $N$. It can be easily shown that when the area of the crossbar is much smaller than the total area of the die, which is the usual case for non-trivial user tiles, the aspect ratio of each user tile converges to $1/N$. User tiles with such aspect ratios are clearly problematic for large $N$. One reason is that memory is often built using a few, relatively large SRAM blocks, in order to optimize memory density; another disadvantage is that very long tiles incur longer wire delays.

A straightforward improvement is to place the crossbar in the middle of the die area, and arrange half of the user tiles on each side thereof; a similar arrangement was proposed in [8][7]. In this way, the aspect ratio is improved by a factor of two, and this is probably the best arrangement for small $N$. However, it remains non-scalable to large $N$.

### B. 2-D Matrix of User Tiles using Distributed Crossbar

Figure 1(b) depicts a more scalable plan. The user tiles are Γ-shaped with the processor at the corner and the memory occupying at least the two thirds. The user tiles are placed in a $\sqrt{N} \times \sqrt{N}$ matrix where each tile owns a crossbar input line spanning its row and a crossbar output line spanning its column. Thus, each tile also embraces the $\sqrt{N} \times \sqrt{N}$ crosspoints between the $\sqrt{N}$ (horizontal) crossbar input lines that are owned by the tiles in its row and the $\sqrt{N}$ (vertical) crossbar output lines that are owned by the tiles in its column.

Providing space for the crossbar lines can be achieved in two ways. The traditional way is to leave wiring channels in both directions between the user tiles. This approach has considerable overhead, and it has been used for interconnection topologies that do not require many wires, such as 2-D meshes. However, in figure 1(b), we show an alternative approach, where the crossbar lines are routed above the SRAM blocks

The size of the over-the-tile routing channel is $0.5 \times \alpha/\sqrt{N}$. This channel can be utilized according to the routing pitch on each metal layer. Using an average mid and higher metal layer routing pitch of 500nm and assuming $a = 12mm$, we find that there are on the order *two thousand* available wires per direction on top of each user for an $N$ up to 144. Since $\sqrt{N}$ crossbar lines are routed over each tile in each direction, a line width up to 160 wires is feasible.

Given the availability of many wires, this topology comes as a natural extension of the 2-D mesh used frequently in many-core designs [9] [10] [11]. From another point of view, it can be considered as an *enriched mesh*, where each tile uses a router with more crosspoints and links.

### C. 2-D Matrix of User Tiles using Centralized Crossbar

Figure 1(c) shows an alternative two-dimensional plan. The user tiles are now square and the memory occupies at least three fourths of their area. The crosspoints are concentrated in the central region of the matrix replacing some of the user tiles. Hence, the crossbar lines are shorter, as all the crosspoint logic is kept in close proximity, while the wires connecting the user tiles with the crossbar are extended to reach the center. Essentially, instead of routing over the user tiles the crossbar lines, we route the links that connect the user tiles to the crossbar.

The over-the-tile link density can be balanced as in figure 1(d). The user tiles are divided in four isometric groups and the tiles of each group are routed to a distinct edge of the crossbar region[2]. The density varies from its lowest

[2]Care must be taken to avoid unwanted swastika-like configurations.

at the corners of the die, to its highest at the periphery of the crossbar. The highest density over a tile is $\frac{N}{2*\gamma}$ unidirectional links, where $\gamma$ the ratio of the edge length of the crossbar region to the edge length of a user tile.

In figure 1(d), $N = 128$ and $\gamma = 4$, thus at most 16 links are routed on top of a user tile. On the other hand, the wiring resources over the user tiles remain the same as in the example of the previous subsection, assuming the same die edge and routing pitch. Thus, we can permit a link width up to 120 wires.

The examples and analysis in the last two subsections clearly demonstrate that even for a conservative technology, there are enough wiring resources to route either the distributed or the centralized crossbar.

### D. Distributed versus Centralized Crossbar Comparison

While both crossbars are feasible, each has advantages and disadvantages. In particular, when distributing the crossbar across the die, communication distance scales better and over-the-tile wiring is better balanced, but, then, the cost of the global links increases too.

First, the total length of the global links is higher. In the distributed crossbar plan, the total length is $2 \times N \times \alpha$. On the other hand, in the centralized version, it can be approximated by the sum of the distances of all tiles to and from the center

$$2 \times \sum_{i=1}^{\sqrt{N}} \sum_{j=1}^{\sqrt{N}} (|i \times \frac{\alpha}{\sqrt{N}} - \frac{\alpha}{2}| + |j \times \frac{\alpha}{\sqrt{N}} - \frac{\alpha}{2}|), \quad (1)$$

which converges to $N \times \alpha$. Hence, the distributed crossbar approximately doubles the total length of the global links.

Assuming the total length of the global links is much higher than the distance between successive repeaters or pipeline registers, by doubling it, the distributed crossbar also doubles the number of repeaters and pipeline registers. As we will show in section IV, the overhead of repeaters and pipeline registers nears 40% of the whole crossbar area in our implementation.

A second disadvantage of the distributed crossbar has to do with the speed of the global links. In the distributed crossbar, half of the global links, the vertical ones, are actually multiplexor circuits, while in the centralized version, all global links are "logicless", driving a single value to the crossbar region. Because repeaters are faster than multiplexor gates, by distributing the crossbar, we also lower the bandwidth and increase the latency of the global links.

## IV. The Crossbar Circuit and Layout

We implemented the centralized crossbar in a 90nm CMOS standard-cell standard-performance ASIC process [4]; standard-cell height is 2.8um, supply voltage ranges from 0.9 to 1.1V, junction temperature varies from -40 to 125°C, and there are 9 layers of interconnect (M1-M9).

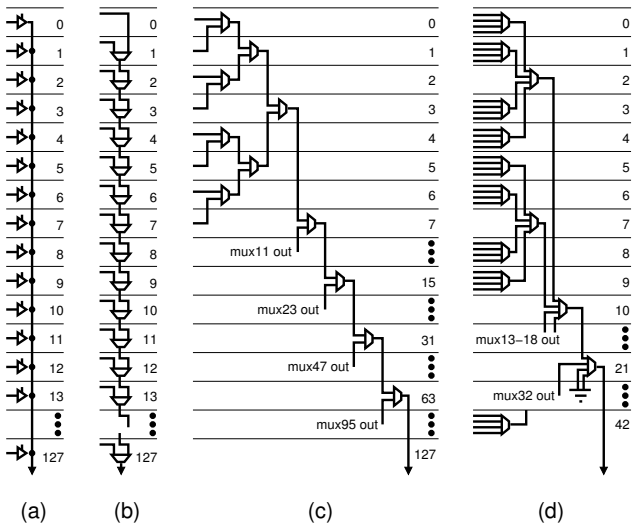


Figure 2. Four circuits for the implementation of a crossbar output line

In order to synthesize, place, and route the circuit, we used some popular industrial tools. Initially, we were experimenting with flattened designs, but, due to the large size of both the circuit and the die, the result we got was long tool running times and low performance, in terms of both area efficiency and timing closure; this is in accordance with the results presented in reference [6]. Thus, given the regularity of crossbars, we decided to specify the structure of the circuit ourselves, using hierarchical prototyping. Below, we describe the process we followed.

### A. Circuits for Crosspoint Implementation

Figure 2 depicts four alternatives for crosspoint implementation; 128 crossbar input lines and a single crossbar output line are shown, assuming each crossbar input line fans out to 128 circuits same as the above.

In (a), each input line crosses the output line through a tristate driver. The advantages of this circuit are that: *(i)* it is simple; and *(ii)* it requires a single metal track to implement the output line. The disadvantages are that: *(i)* each driver is loaded with the large parasitic capacitance of the whole output line; and *(ii)* EDA flows encounter problems in designs that use tristate drivers[3].

(b-d) describe approaches using multiplexor gates. In our reference process, such gates can be compiled to either transmission or logic gates.

In (b), the crossbar output line is implemented with a chain of two-to-one multiplexors. As in (a), a single metal track suffices, but a prohibitively long delay of 128 multiplexors in series is needed.

[3]For example, optimization through buffer insertion is impossible, unless the bus is explicitly segmented. Furthermore, Automatic Test Pattern Generation (ATPG) is complicated by the requirement that a single tristate driver drives the bus at any point in time, so that bus contention is avoided.

In (c), multiplexing is parallelized by connecting the multiplexors in a binary tree. The levels of multiplexors are reduced to 7 (=$\log_2 N$), at the cost of 7 metal tracks. However, as we will show in section IV-C, these tracks introduce zero area overhead, as they are already available over the multiplexor standard cells. Thus, (c) is preferable to (a) and (b).

Finally, in (d), a shorter and wider circuit is shown, as an alternative to (c). The multiplexors have a valency of four and are connected in a quad tree.

### B. Bit Slicing versus Port Slicing Comparison

The bit width, $W$, of the multiplexors is derived from the ratio of the desired port throughput to the achieved clock frequency. Two groupings of gates are then possible. One grouping, called *port slicing* [12], places all bits of each multiplexor close to each other; for example, they can be arranged as a $\sqrt{W} \times \sqrt{W}$ matrix. This grouping minimizes the span of each control wire, since all gates that it controls are placed in close proximity. The other grouping, called *bit slicing* [12], places together all multiplexor gates for a given bit position, and replicates this structure $W$ times, for all bit positions. This grouping minimizes the distance among gates that contribute to the generation of each crossbar output signal. Intermediate solutions are also possible (see e.g. reference [13] for byte slicing), but they are not studied here.

In order to compare bit and port slicing, we have to contrast: *(i)* the wiring for the connection of the data and control ports of the crossbar with the slices; and *(ii)* the sizing of the multiplexor gates.

As we described in section III, the data ports of the crossbar have to be uniformly distributed at its periphery in order to balance the over-the-tile link density. Hence, no matter how the crossbar is sliced, analogous wiring is needed for the connection of the crossbar data ports with the slices. Besides, for the control ports, the critical problem is their fan-out capacitance, which is the same in both port and bit slicing, rather than the length of their path to the slices. Thus, we focused on gate sizing.

We first emulated one bit-sliced crossbar output line by implementing the binary-tree circuit of figure 2(c); we considered each row corresponds to a standard-cell row and we empirically set its width to 80um to provide sufficient space for in-place optimization. We placed the circuit, performed trial routing to extract its parasitic capacitances, in-place optimized it under a range of timing constraints, and finally routed it. For each timing constraint that was satisfied after final routing, we measured the actual delay and the total standard-cell area of the circuit. The resulting delay-area curve is plotted in figure 3(a), labeled "bit width = 1". Notice that the inverse plot is infeasible, as EDA algorithms prioritize speed over area.

By increasing the distance between the rows of multiplexor gates, we emulated port-sliced crossbar output lines

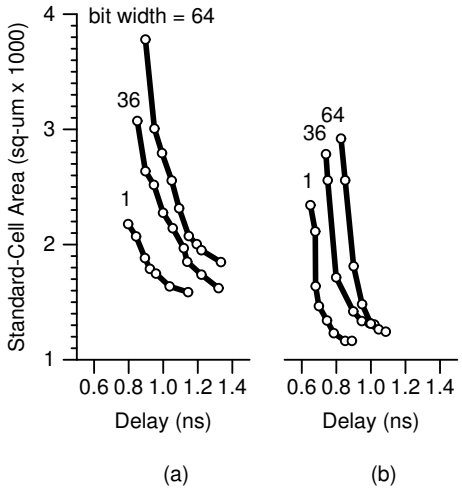


Figure 3. Delay-area curves of a port-sliced (a) binary- and (b) quad-tree crossbar output line as a function of its bit width. Bit-slicing coincides with port slicing when bit width = 1, and stays unaffected by bit width.

of various bit widths. A bit width $W$ corresponds to $\sqrt{W}-1$ empty standard-cell rows between successive rows of multiplexor gates, or a distance of ($\sqrt{W}$-1)×2.8um. Figure 3(a) plots curves for $W$ = 1, 36, and 64. We observed that for distances up to 200um ($W$<5000) the EDA algorithms optimize the circuit by increasing the size of the multiplexor gates, while they switch to repeater insertion for longer distances.

We repeated the same experiment for the quad tree. Figure 3(b) shows the resulting performance curves.

We conclude that, for both trees, the closer the multiplexor gates are clustered together, the faster and the more area efficient the circuit that results. Then, the output load of the gates is reduced, thus their performance increases, and smaller-size gates suffice, thus reducing area; reduced area further improves performance. Moreover, we observe that the quad tree is more area efficient being three times shorter[4]. Last, for a circuit delay of 0.8ns, a 36-bit wide port-sliced implementation would increase the standard-cell area of a crossbar output line by at least 40% compared to a bit-sliced one. Hence, bit slicing is preferable to port slicing.

### C. Organization of the Bit Slice

We implemented a crossbar bit slice by: *(i)* structuring a crossbar output line as one of the circuit instances that generate the curve "bit width = 1" of figure 3(b); *(ii)* replicating the line for each crossbar output; *(ii)* placing the standard cells of all lines with a custom script[5]; *(iii)* defining IO circuits and similarly placing them; and finally *(iv)* routing the whole circuit using the standard EDA tools.

[4]An additional, but minor, factor increasing the area efficiency of the quad over the binary tree, is the better area efficiency of the four- over the two-to-one multiplexor gates.

[5]The script generates the coordinates of standard cells and writes a placement definition file in a standard format

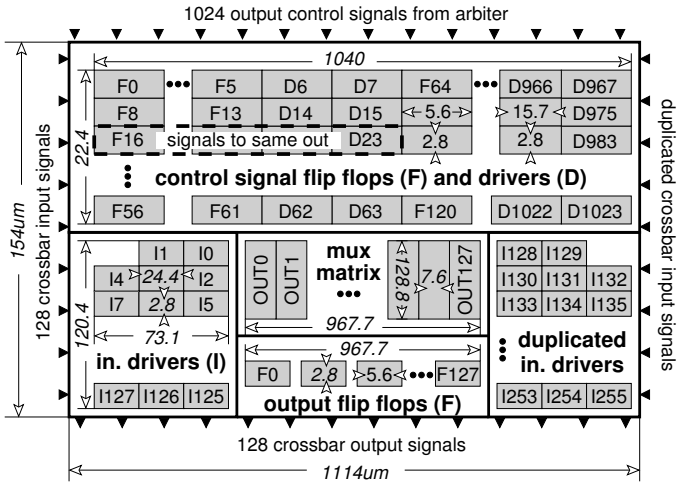


Figure 4. Floor plan of the crossbar bit slice

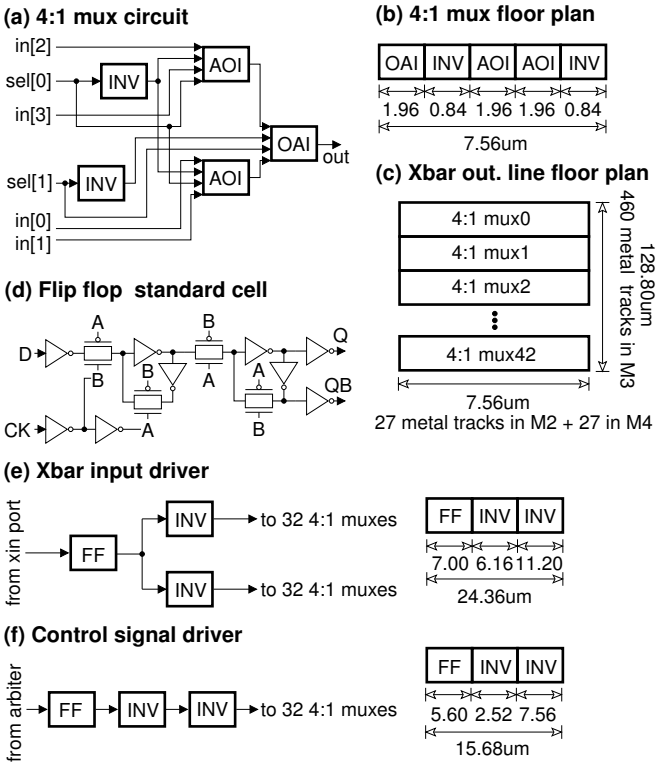


Figure 5. Elements of the crossbar bit slice

Our approach was to implement an as-fast-as-possible bit slice, while pipelining its IO links. The floor plan and the elements of the bit slice are shown in figures 4 and 5 respectively.

We implemented a crossbar output line with the instance (*bit-width=1*, *delay=0.8ns*) of figure 3(b). This instance uses identical four-to-one multiplexors for all levels of the multiplexor tree except for the root. The non-root multiplexors are built with And-Or-Invert standard cells as in figure 5(a). We placed them in a wide area, as in (b), to provide for the quad tree wiring. The root multiplexor is bigger, built with NAND gates, and takes up four standard-cell rows; however, it is not further described here for brevity. We stacked the multiplexors as in figure 5(c), forming a crossbar output line. By replication, we placed all crossbar output lines in a 967.68um×128.8um area, which is labeled "multiplexor matrix" in figure 4.

Each crossbar input drives the flip flop of figure 5(d). In turn, the flip flop drives a 1-mm-long wire, which spans the width of the multiplexor matrix and fans out to each crossbar output line, plus one multiplexor at each crossbar output line. We halved this load by duplicating the crossbar inputs –half on the left and half on the right sides of the multiplexor matrix. By approximating the resulting capacitance, we synthesized the driver of figure 5(e) and by replication, we generated drivers for all crossbar inputs. We placed the drivers as in figure 4.

Except for data, inputs to the bit slice are control signals that configure the crossbar output lines, i.e. control their multiplexors. We assigned the control signals at the top of the multiplexor matrix, as in figure 4. We used two bits for each level of each multiplexor tree, for a total of 1024 signals; each bit fans out to each multiplexor at that level. The signals to leaf multiplexors have the largest loading capacitance; they fan out to 32 multiplexors each, while spanning the height of the multiplexor matrix. For these signals, we specified and placed drivers, similarly to the crossbar inputs; figure 5(f) shows their circuit and floor plan. For the rest of the control signals, we placed just flip flops.

We also placed a row of flip flops at the bottom of the multiplexor matrix to register the outputs of the bit slice.

Finally, we routed the circuit using the standard EDA routers. We budgeted M2 and M4 for the wiring of the multiplexor trees and their control signals; the crossbar inputs were routed in M3. Thus, the bit slice obstructs M2-M4 from the higher levels of our hierarchy. Simple calculations suffice for one to verify the routability of the layout[6].

The whole bit slice takes up a 1114um×154um area, while running at up to 750 MHz under worst-case operating conditions of 0.9 Volts and 125°C. At this clock frequency, the maximum length of a repeated and pipelined link is –nota bene– 6mm at the M5-M8 layers.

### *D. Organization of the Crossbar Tile*

To give a port throughput of 20Gbps, a crossbar clocked at 750MHz suffices to be 27-bits wide; we made it 32-bits

[6]The layout should have remained routable with a single metal layer in the vertical direction, but due to artifacts of the routing tool –few DRC violations and slower critical paths were reported, we provided both M2 and M4.

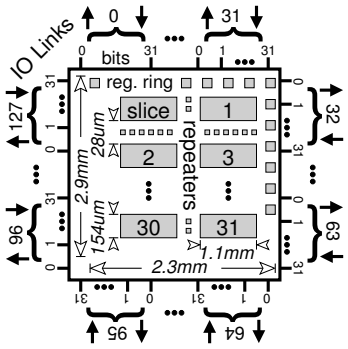


Figure 6. Floor plan of the crossbar tile

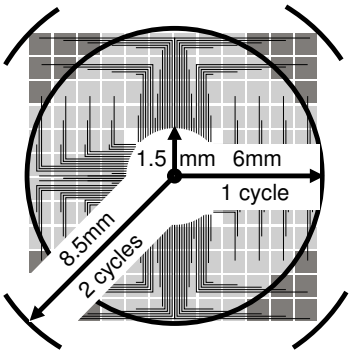


Figure 7. Division of the die into three conceivable circles: (inner) crossbar tile, (median) user tiles that reach the periphery of the inner circle in at most one clock cycle, (outer) user tiles that reach the periphery of the inner circle in at most two cycles.

wide for orthodox data alignment, thus also increasing the port capacity to 24Gbps.

To compose the crossbar, we placed 32 bit slices in two stacks as in figure 6. We also uniformly distributed the data ports of the crossbar at its periphery and the control ports at the center of the eastern side.

Each bit of each data port needs to be connected with the corresponding pin of the homonym bit slice, while each control bit to all bit slices. Thus, the longest route equals the half perimeter of the crossbar, which is 15% shorter than the maximum length of a pipelined and repeated link. Hence, we surrounded the bit-slice stacks with a ring of flip flops to register their IO.

We left an empty space of 28um between the slices for repeater and buffer insertion; we came up with it by empirically specifying a value to get timing closure and then repeatedly lowering it. Thus, the whole crossbar area is 2.3mm×2.9mm.

A simple solution that verifies the routability of the layout is the following. Each bit is routed in M7 and M8 to the proximity of its bit slice, and from there, in M5 and M6, to the corresponding pin of the slice. Using an average routing pitch of 500nm, there are above 4500 metal tracks in each dimension of the crossbar, thus each bit can be allocated its own metal track in M7 and M8; routing in M5 and M6 is trivial. The wires are routed on top of the bit slices, which obstruct the M1-M4 layers. Thus, the whole crossbar obstructs all metal layers from M1 to M8.

The control signals can be routed similarly and are also buffered to lower their fan-out.

### E. Organization of the Global Links

We emulated a user tile with a small circuit which serves as a traffic source and sink, while obstructing area and metal layers in accordance with the user-tile configuration of figure 1(c). Nevertheless, the obstructed area is 985.6um×985.6um, leaving 14.4um in each direction for repeaters and flip flops on global links. The global links are routed in M5 and M6, over the user tiles, as in figure 1(d).

The die can be conceivably divided by three homocentric circles as in figure 7. The inner circle is the circumcircle of the crossbar tile; the median and outer circle contain all user tiles that reach the periphery of the inner circle in at most one and two clock cycles respectively.

The corner-to-corner latency is 8 clock cycles, composed by: *(i)* a two-cycle trip from the corner of the die to the crossbar tile; *(ii)* a single-cycle trip from the edge of the crossbar tile to the corresponding bit slice; *(iii)* a single-cycle trip inside the bit slice; and *(v)* the symmetrical trip from the output of the bit slice to the destination corner.

The careful reader should have observed that the control signals are also pipelined; they propagate in a flip-flop tree. In the first cycle, a control signal arrives at the crossbar tile from the crossbar control circuit; in the second cycle, the value of the signal is loaded to all 32 slices; and in the third cycle, it is loaded to the multiplexors inside the bit slice.

### F. Quantitative Data

We profiled the crossbar circuit with respect to: *(i)* gate, buffer/repeater, and flip-flop count and standard-cell area – see table I; *(ii)* power consumption –see table II; and *(iii)* metal-track utilization –see table III.

**Area Cost:** The area numbers for the bit slice come up directly from the description of its floor plan in subsection IV-C. We observe that the 75% of the total standard-cell area within a bit slice is allocated to multiplexors, while the remaining is equally shared by IO circuits, i.e. buffers and flip flops; not shown in the table is that the data wires account for 65% of the buffer area and 15% of the flip-flop area, while the control wires account for the rest. The total standard-cell area of the bit slice is 0.16mm$^2$. Thus, given that the bit slice is floor-planned in a 1.1mm×0.154mm box, its area utilization is above 95%.

Moving one level higher in the hierarchy, the crossbar tile utilizes the standard cells of 32 bit slices plus the flip flops, buffers, and repeaters on the wires that connect the

| | Count ($\times 10^3$) | | | Standard-Cell Area ($mm^2$) | | | |
|---|---|---|---|---|---|---|---|
| | Logic Gates | Buffer/Repeater Standard Cells | Flip Flops | Logic Gates | Buffers/Repeaters | Flip Flops | Total |
| Bit Slice | 63.0 | 1.0 | 1.4 | 0.121 | 0.020 | 0.020 | 0.161 |
| Bit Slice X 32 | 2000 | 33 | 45 | 3.872 | 0.640 | 0.640 | 5.152 |
| Crossbar Tile | 2000 | 60 | 54 | 3.872 | 1.196 | 0.824 | 5.892 |
| Whole | 2000 | 92 | 56 | 3.872 | 1.855 | 0.862 | 6.589 |

Table I
CUMULATIVE AREA COST

ports of the bit slices with the ports of the crossbar tile. The repeaters and buffers take up 0.56mm$^2$ and the flip flops another 0.18mm$^2$; not shown in the table is that the data and the control ports account for 65% and 35% of this overhead area respectively. Hence, the total standard-cell area of the crossbar tile is 5.9mm$^2$. Given that the crossbar tile is floor-planned in a 2.3mm$\times$2.9mm box, its area utilization nears 90%.

In the top level of the hierarchy, the actual area cost of the global links is 3% of the total tile area, corresponding to the empty space that we leave between the user tiles for insertion of flip flops and repeaters. However, this cost could be lowered to the actual standard-cell area by pushing the repeaters and flip flops inside the user tiles. Hence, we consider that their actual overhead coincides with their standard-cell area, which is 0.66mm$^2$ for repeaters and 0.04mm$^2$ for flip flops. Thus, the global links add a 10% overhead to the area of the crossbar tile.

Overall, the whole crossbar circuit utilizes a standard-cell area of 6.6mm$^2$ and a silicon area of 7.5mm$^2$, or 6% of the total tile area. Furthermore, 60% of the total standard-cell area is occupied by multiplexor gates, while the remaining 40% by repeaters, buffers, and flip flops on multiplexor wires –15% control and 25% data.

Finally, we observe from table I that the number of repeaters needed to route the IO of the bit slices is approximately 60000; around 35000 of them are on the data wires. On the other hand, the total length of the global data links can be calculated by multiplying equation 1 (section III-D) by the width of the crossbar, which gives a total length of 50cm. Thus, the distance between successive repeaters on the same link is approximately 1.5mm, which is in accordance with reference [14].

**Power Cost:** Table II shows power consumption; we break it down into power consumption due to wires, standard cells, and leakage; leakage was always negligible. For our measurements, we simulated a permutation communication pattern, circularly updated in every clock cycle; we also assumed a toggle rate of one and a power supply of 0.9 Volts.

First, we observe that power consumption within a bit slice is almost equally due to the capacitance of the interconnect and the standard cells. Second, the wires connecting the bit slices with the ports of the crossbar tile add an overhead of 40% to the power consumption of all slices; this is mostly due to the capacitance of the wires. Likewise, the global links add a further 30% to the power consumption of the crossbar tile.

| | Wires | Standard Cells | Total |
|---|---|---|---|
| Bit Slice | 0.073 | 0.085 | 0.158 |
| Bit Slice X 32 | 2.336 | 2.720 | 5.056 |
| Crossbar Tile | 3.842 | 3.100 | 6.942 |
| Whole | 5.462 | 3.590 | 9.052 |

Table II
CUMULATIVE POWER COST (WATTS)

| | Bit Slice | Crossbar Tile | Global Links |
|---|---|---|---|
| M2 | 23 | $< 5$ | $< 5$ |
| M3 | 60 | $< 5$ | $< 5$ |
| M4 | 20 | $< 5$ | $< 5$ |
| M5 | 0 | 38 | $< 5$ |
| M6 | 0 | 47 | $< 5$ |
| M7 | 0 | 46 | 0 |
| M8 | 0 | 30 | 0 |
| M9 | 0 | 0 | 0 |

Table III
METAL TRACK UTILIZATION (%)

Overall, the whole crossbar power is 9 Watts; 30% is consumed on the multiplexor gates and 70% on the multiplexor wires and their repeaters, buffers, and flip flops.

**Metal Track Utilization:** We define the metal track utilization on a metal layer as $((L \times R)/A) \times 100\%$, where L and R the total wire length and routing pitch on that layer, and A the area of the reference region. Table III shows numbers for each level of the crossbar hierarchy. We observe that the lower two levels utilize approximately half of the available wiring resources, while the top level utilizes a negligible amount. As a consequence, the area of the crossbar is a linear, rather than quadratic [15], function of its width.

## V. FEASIBILITY OF CONTROL

The control of the crossbar datapath mainly comprises a scheduler circuit, which updates the crossbar configuration based on the traffic at the crossbar network interfaces. A

thorough study of the queueing stradegy at the network interfaces and the scheduler circuit will appear in a follow-up paper. Here, we give brief area cost estimates of a baseline configuration.

We placed and routed a centralized iSLIP scheduler. The implementation is similar to [16], with the difference that, instead of pipelining successive iterations, we pipeline the phases of a single iteration. Our implementation consists of 256 identical round-robin arbiters, implementing the 128 grant and the 128 accept arbiters of iSLIP. Each arbiter is placed in an 154um×154um area and routed in M2-M4. The area of each arbiter is dominated by the parallel prefix trees needed to implement priority encoding. The arbiters are placed in a 3mm×4mm grid and interconnected with point to point links in M5-M9. The whole scheduler circuit is wire limited, featuring an area utilization as low as 50%. Thus, the whole scheduler is by 30% larger than what could fit in the central 4mm×4mm region. Despite that, we conclude that a baseline configuration is very close to fit in our configuration. Wiring, and thus area, could be lowered e.g. by compromising some scheduling performance.

## VI. Conclusion

Though designers generally believe that high-valency crossbars are expensive due to their quadratic area cost, we demonstrated that they are implementable with a minimal area budget even in a conservative 90nm technology. We showed a crossbar implementation interconnecting 128 $1mm^2$ tiles in a single hop, while providing a bidirectional channel capacity of 48Gbps and occupying 6% of the total tile area. Our implementation featured the following points:

- The centralized organization allows area-efficient crosspoints and global links;
- the custom placement of the crosspoints guides the EDA tools to regular and compact routing solutions;
- the plentiful of wiring resources allows all crossbar lines to be routed over the crosspoint standard cells;
- the plentiful of wiring resources on top of SRAM blocks allows all global links to be routed over the user tiles.

Hence, we also proved that the area of the crossbar scales linearly with its width. Applications of our implementation include CIOQ switch and many-core chips.

## Acknowledgments

This work was supported by the European Commission in the context of the SARC Integrated Project (FP6 #27648) and the HiPEAC Network of Excellence (FP7 #217068). We also thank Christos Sotiriou for useful discussions on EDA flows and algorithms.

## References

[1] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," *SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 420–431, 2005.

[2] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *Proceedings of the Int'l Symp. on High-Performance Computer Architecture (HPCA-11)*, 2005.

[3] G. Mora, J. Flich, J. Duato, P. López, E. Baydal, and O. Lysne, "Towards an efficient switch architecture for high-radix switches," in *Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS '06)*, 2006.

[4] FARADAY Technology Corporation, "UMC's 90nm Logic SP-RVT Standard Cell Process," www.faraday-tech.com.

[5] S. Scott, D. Abts, J. Kim, and W. J. Dally, "The BlackWidow high-radix Clos network," *SIGARCH Computer Architecture News*, vol. 34, no. 2, pp. 16–28, 2006.

[6] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini, "Bringing NoCs to 65 nm," *IEEE Micro*, vol. 27, no. 5, pp. 75–85, 2007.

[7] Y. Zhang, T. Jeong, F. Chen, H. Wu, R. Nitzsche, and G. R. Gao, "A study of the on-chip interconnection network for the IBM Cyclops64 multi-core architecture," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[8] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," *SIGARCH Comput. Archit. News*, vol. 33, no. 2, pp. 408–419, 2005.

[9] W. J. Dally and B. Towles, "Route packets, not wires: On-chip inteconnection networks," in *DAC '01: Proceedings of the 38th annual Design Automation Conference*, 2001.

[10] M. B. Taylor *et al.*, "The RAW microprocessor: A computational fabric for software circuits and general-purpose programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[11] Tilera, "TILE-Gx processors family," *http://www.tilera.com/products/TILE-Gx.php*, 2009.

[12] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[13] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny-Tera: A packet switch core," *IEEE Micro*, vol. 17, no. 1, pp. 26–33, 1997.

[14] G. Michelogiannakis, D. Pnevmatikatos, and M. Katevenis, "Approaching ideal NoC latency with pre-configured routes," in *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2007.

[15] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitectures in on-chip networks," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2003.

[16] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, 1999.