# High Radix Self-Arbitrating Switch Fabric with Multiple Arbitration Schemes and Quality of Service

Sudhir Satpathy, Reetuparna Das, Ronald Dreslinski, Trevor Mudge, Dennis Sylvester, David Blaauw
University of Michigan, Ann Arbor
sudhirks@umich.edu

## ABSTRACT

A scalable architecture to design high radix switch fabric is presented. It uses circuit techniques to re-use existing input and output data buses and switching logic for fabric configuration and supporting multiple arbitration policies. In addition, it integrates a 4-level message-based priority arbitration for quality of service. Fine grain clock gating, tiled fabric topology and self-regenerating bit-line repeaters enable scaling the router to 8k wires. A 64×64(128b data) switch fabric fabricated in 45nm SOI CMOS spans 4.06mm$^2$ and achieves a throughput of 4.5Tb/s at 3.4Tb/s/W at 1.1V with a peak measured efficiency of 7.4Tb/s/W at 0.6V.

## Categories and Subject Descriptors

B.4.0 [**INPUT/OUTPUT and Data Communications**]: General

## General Terms

Algorithms, Design

## Keywords

Switch Fabric, Radix, Arbitration, Quality of Service

## 1. INTRODUCTION

Technology scaling has made billion transistors design feasible on a single die. With transistors getting cheaper and faster, the core count in multi-processor systems has been steadily increasing [1,2]. High end servers [3], gigabit Ethernet routers [4,5] and multimedia processors [6,7] now serve workloads dealing with terabytes of data flow every second. Even medium throughput applications now prefer multi-core architectures over a single core implementation for better energy efficiency and fault tolerance [8]. These system need a network to communicate data among processing and storage elements in the chip. Although processing units are getting smaller and simpler, the dramatic rise of their number in a single die has resulted in the growing complexity of interconnect. As a result, the interconnect fabric has become a bottleneck in improving overall system efficiency. Thus, the design paradigm for multi-core chips is gradually shifting from a core-centric architecture towards an interconnect-centric architecture, where overall system performance is limited by the bandwidth of the interconnect fabric rather than the processing ability of any individual core.

A generic switch fabric comprises of three key modules: 1) A data routing module to transfer information among different IPs connected to the fabric. This could be a single or a collection of shared buses for systems where processing units rarely communicate [9] or could be a fully connected crossbar [10]. 2) An arbiter that receives requests from processing units and configures the fabric to ensure data sent from a source reaches the appropriate destination. 3) A priority management module that monitors traffic flow pattern within the fabric and assists the arbiter to ensure fairness in resource allocation. The overall efficiency of the switch fabric relies on how efficiently each of these independent modules function and how seamlessly they communicate among one another. With a growing number of input and output ports, each module gets physically bigger and hence farther from the others. Beyond a size, the latency and energy overhead due to communication between these modules starts limiting overall fabric efficiency. Existing circuit techniques to build high radix switch fabrics rely on assembling together smaller switches that are usually 5×5 in dimension [11]. This approach has certain limitations: 1) The elementary switches are built using multiplexers that select bits rather than buses. Hence, as buses get wider, routing at the fabric input ports involves a lot of interleaving among the wires incurring additional area penalty. 2) A switch with ports far exceeding 5 would require multiple of these switches to be connected in stages. Data has to traverse through multiple stages to reach its destination thereby increasing latency and energy dissipation. They would also require additional data storage elements in the data routing path for higher throughput resulting in further latency and power overhead. 3) Each source sends requests to the arbiter before it could access a destination, and eventually the arbiter sends an acknowledgement back to the source after setting up the routing path. Configuring all the switches along the routing path incurs latency. In systems that have well defined communication patterns and usually operate on massive sets of data, the latency and energy cost for configuring the fabric can be amortized by setting up the routing path ahead of time or by sending multiple chunks of data once the path is established. However, in generic multiprocessor systems most traffic patterns are not pre-defined. Hence, the fabric configuration cost becomes a bottleneck. 4) A non-blocking switch supports all possible permutations and hence can guarantee starvation free communication for all applications. However many applications (like FFT, LDPC, Color space conversion etc.) can be sped up significantly by incorporating multicast and broadcast features in the switch fabric [12].

Arbitration policies have a noticeable impact on throughput and fairness of interconnection networks. Some arbitration policies like the greedy allocation policy tend to maximize network throughput at the cost of quality of service. At the other extreme, some complex schemes like probabilistic distance [13] based arbitration can guarantee quality of service at the price of more complex logic and hence additional arbitration latency and power overhead. Hence, adaptive and hybrid resource allocation schemes are preferred in general over static schemes because of their ability to mitigate congestion.

In this paper, we propose a fabric architecture called swizzle switch network (SSN) to accomplish a variety of arbitration policies with very minimal overhead as shown in Fig. 1. For proof of concept, we fabricated a 64x64 SSN prototype with 128b data bus in 45nm SOI CMOS with the following key features: 1) A novel, single cycle least recently granted (LRG) priority arbitration technique that re-uses the already present input and output data buses and their drivers and sense amps. 2) An additional 4-level message-based priority arbitration for quality of service (QoS) with 2% logic and 3% wiring overhead. 3) A new bidirectional bit-line repeater that allows the router to scale to >8000 wires. These features result in a compact fabric (4.06mm$^2$) with throughput gain of 2.1× over [5] at 3.4Tb/s/W efficiency which improves to 7.4Tb/s/W at 600mV.

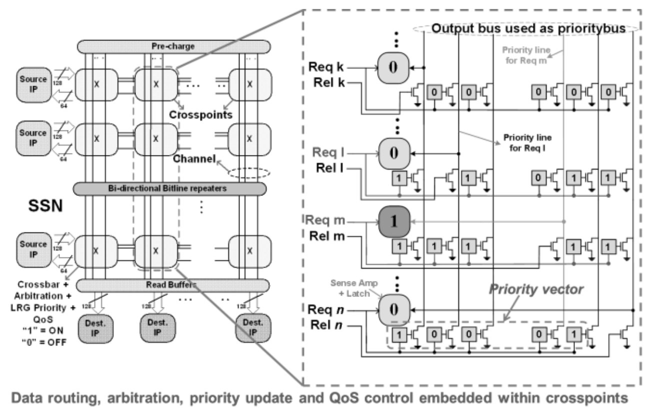

Pre-charge
Source IP
Crosspoints
Channel
Bi-directional Bitline repeaters
SSN
Read Buffers
Crossbar + Arbitration + LRG Priority + QoS "1" = ON "0" = OFF
Dest. IP
Output bus used as prioritybus
Priority line for Req m
Req k
Rel k
Req l
Rel l
Priority line for Req l
Req m
Rel m
Sense Amp + Latch
Priority vector
Req n
Rel n
Data routing, arbitration, priority update and QoS control embedded within crosspoints

**Figure 1. Swizzle Switch Network**

The rest of the paper is organized as follows: In section 2, we present the SSN fabric architecture. Section 3 explains the various arbitration policies that can be implemented in SSN. In section 4, we present SSN's message based QoS arbitration scheme. Detailed circuit level implementation and design choices are then presented in section 5. Measurement results from SSN test prototype fabricated in 45nm have been described in section 6 before conclusion in section 7.

## 2. SSN ARCHITECTURE

SSN is a matrix-type fabric as shown in Fig. 1 with input buses running horizontally and output buses vertically. When data is routed, the input and output buses transfer data traffic. During arbitration the input bus routes a multi-hot code indicating which output channel(s) are requested by that input, and the output bus is used for conflict detection and arbitration [14]. Each crosspoint stores a connectivity status bit indicating whether the input bus was granted access to the output channel. An *n-1* bit priority vector is also stored to represent the priority of the input bus with respect to all other inputs for that output bus. Fig. 1 shows the priority vector at each crosspoint in a blow-up of a *single* output channel. Each input bus is assigned a unique bit line from the channel as its *priority line* which, if high, indicates it as the winner in a particular arbitration cycle. Similarly, each bit in the priority vector at a crosspoint indicates whether the input bus at that crosspoint has higher or lower priority than the input bus associated with the priority line. For instance, in Fig. 2 priority line *m* corresponds to input bus *m* while the *m*-th priority bit of bus *n* is a 1, indicating that *n* has higher priority than *m*. When input *n* requests the output channel this high bit results in the discharge of priority line *m*, suppressing access by input *m*. In contrast, input *l* stores a 0 at its *m*-th priority bit and hence does not suppress an access request from input *m*, meaning that *l* has lower priority that input *m*.

Priority vectors need to be set consistently and indicate the same priority order. In Fig. 2, the 0 at bit *m* of input *l* must be mirrored with a 1 at bit *l of* input *m*. Furthermore, the priority bits need to be correctly updated after each arbitration cycle to implement LRG policy. We propose a new, simple mechanism to accomplish this. In Fig. 1, inputs *l* and *m* request the output channel in an arbitration cycle. Input *m* wins owing to its higher priority and its

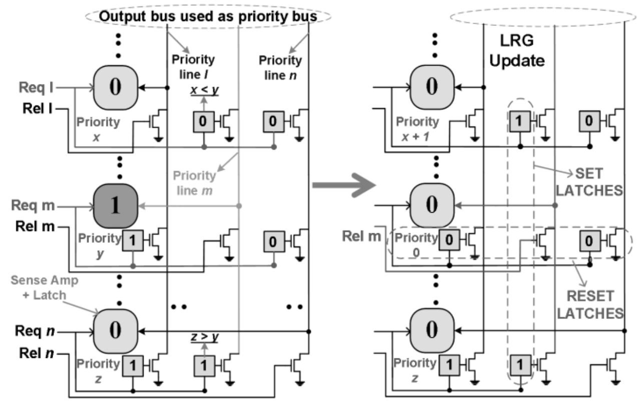


**Figure 2. Arbitration and priority update.**

connectivity status bit is set to 1. After data transfer, input *m* releases its channel during a channel release cycle. In this cycle, input *m* first *resets* all its priority bits. This guarantees that *m* now has lowest priority, as required by the LRG algorithm. At the same time, input *m* also lowers its *priority line m*, which is a signal to other crosspoints in the output channel to *set* their *m*-th priority bit. This ensures that all other input buses now have higher priority than *m*. Input buses with higher priority than *m* remain unchanged and o*nly* inputs with lower priority than input *m* are increased in their priority by exactly one level. This simple and fast update mechanism provably guarantees both consistency of all priority vectors and correct implementation of the LRG arbitration scheme, which enables efficient and deadlock-free routing.

## 3. ARBITRATION SCHEMES

The SSN is capable of supporting multiple arbitration schemes. The priority vectors at various crosspoints along a single output bus form a priority matrix. A priority matrix with 6 inputs (arranged top to down numerically from *input1* to *input6*) is shown in Fig. 3. Here, the priority line connections are denoted as Xs. The priority matrix satisfies the following criteria: 1) The total number of 0s equals the total number of 1s. 2) Each row has a unique number of 1s which represents the corresponding input's priority. 3) Each column has a unique number of 1s. 4) At any priority line connection (denoted as X in Fig. 3.), the sum of the number of 1s (or 0s) in its corresponding row and column must add to one less than the total number of inputs.

### 3.1 Least Recently Granted (LRG) scheme

Though priority lines can be randomly assigned to inputs without limiting the generality of the priority update schemes, a diagonal assignment as shown in Fig. 3 makes the priority matrix skew (or anti) symmetric and easy to understand. As shown in Fig. 3, the

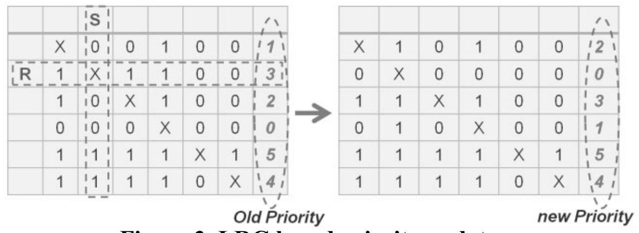


**Figure 3. LRG based priority update**

input corresponding to second row used the channel most recently. It is assigned a priority level 3. An LRG priority update is accomplished by resetting all priority bits along the second row (denoted by R) and by setting all priority bits (denoted by S) along the second column (which is the priority line for *input2*). *Input2* is thus downgraded to have the least priority while all inputs with lower priorities get upgraded by exactly one level. In the rest of this section, the other priority update schemes will be explained using the priority matrix notation.

## 3.2 Most Recently Granted (MRG) scheme

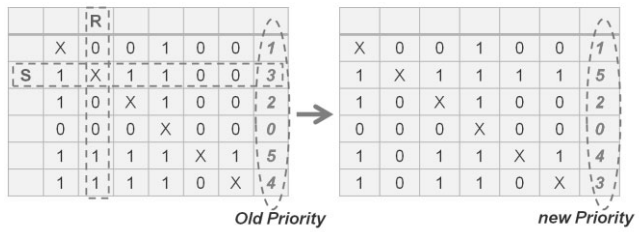


**Figure 4. MRG based priority update**

For accomplishing an MRG based update, we set all priority bits along the second row (denoted by S) and reset all priority bits (denoted by R) along the second column (which is the priority line for the *input2*) as shown in Fig. 4. By setting bits in the second row, *input2* now gets the highest priority. Inputs that previously had higher priorities than *input2* get downgraded by exactly one level. Inputs that previously had lower priority than *input2* retain their old priorities.

## 3.3 Round Robin schemes

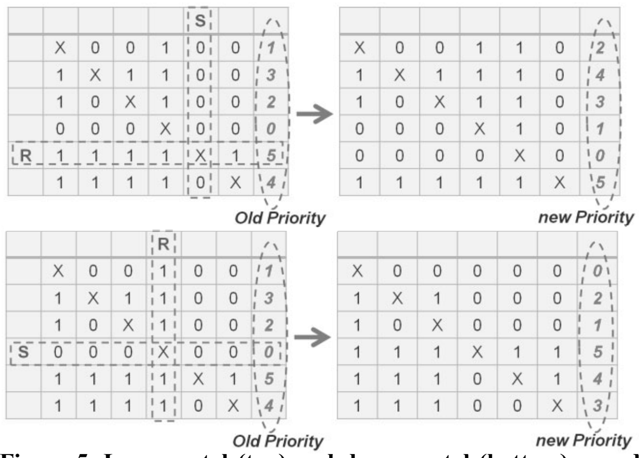


**Figure 5. Incremental (top) and decremental (bottom) round robin based priority updates**

For accomplishing an incremental Round Robin based update, we pick the row with the highest priority. This can be identified by a logical AND operation of all the priority bits. In this case *input5* has the highest priority. We reset all priority bits along the fifth row (denoted by R) and set all priority bits (denoted by S) along the fifth column (which is the priority line for *input5*) as shown in Fig. 5 top. By resetting bits in the fifth row, *input5* now gets the least priority. All other inputs get upgraded by exactly one level.

For accomplishing a decremental Round Robin based update, we pick the row with the lowest priority. This can be identified by a logical OR operation of all the priority bits. In this case *input4* has the highest priority. We set all priority bits along the fourth row (denoted by S) and reset all priority bits (denoted by R) along the fourth column (which is the priority line for *input4* ) as shown in Fig. 5 bot. By setting bits in the fourth row, *input4* now gets the highest priority. All other inputs get downgraded by exactly one level.

## 3.4 Priority swap and reversal

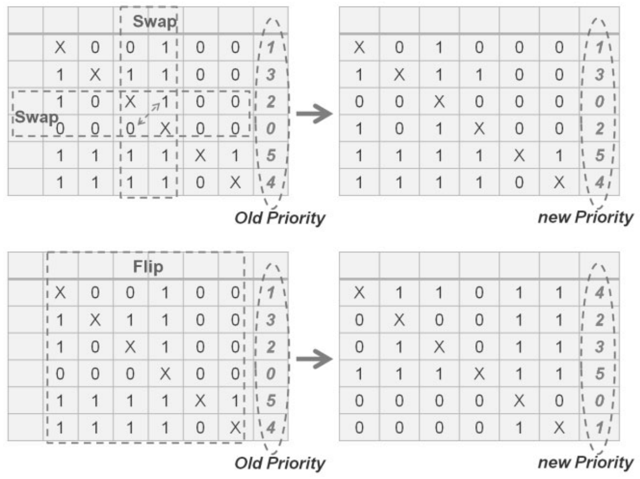


**Figure 6. Priority swap (top) and priority reversal (bottom)**

The priorities of 2 inputs can be swapped (without affecting priorities of other inputs) by swapping the priority bits in their corresponding rows and those in the columns corresponding to their priority lines as shown in Fig. 6 top. Here, we intend to swap the priorities of *input3* and *input4*. In the physical realization of this technique, already existing word-lines will be used to swap priority bits between columns and bit-lines to swap priority bits between rows. In a single cycle, any two priorities can be swapped.

The unique priority encoding scheme also allows reversing the priority of all inputs instantaneously by flipping all the priority bits as shown in Fig. 6 bottom. In physical circuit level implementation, rather than flipping all the bits, a multiplexer can be used to select the inverted priority. Hence, this functionality can be achieved without the expense of a clock cycle. The consistency of the new priority vectors is guaranteed because this transformation ensures that the priority matrix still satisfies all the criteria mentioned before.

## 3.4 Selective LRG and MRG

In this scheme LRG update is applied to a selective section of inputs. In the standard LRG scheme, the input that used the output bus most recently is downgraded to have the least priority while all inputs with lower priorities get upgraded by exactly one level. In this case *input6* with a priority level 4 used the channel most recently. However, in the selective LRG scheme, instead of downgrading *input6* all the way down to 0, we intend to downgrade it to some intermediate priority (say priority level of *input0* which is 1). To accomplish this, before setting/resetting priority bits we identify certain rows and columns that need to be frozen. In this case, all columns corresponding to priority bits that are high in the first row are frozen as shown in Fig. 7 top. Simultaneously, all rows corresponding to priority bits that are low in the first column (which is the priority line for *input0*) are also frozen. Following this the priority bits in the sixth row are reset (except the bits in frozen columns) and those in the sixth column are set (except the bits in frozen rows). This ensures that the new priority matrix is consistent and the intended priority update is achieved.

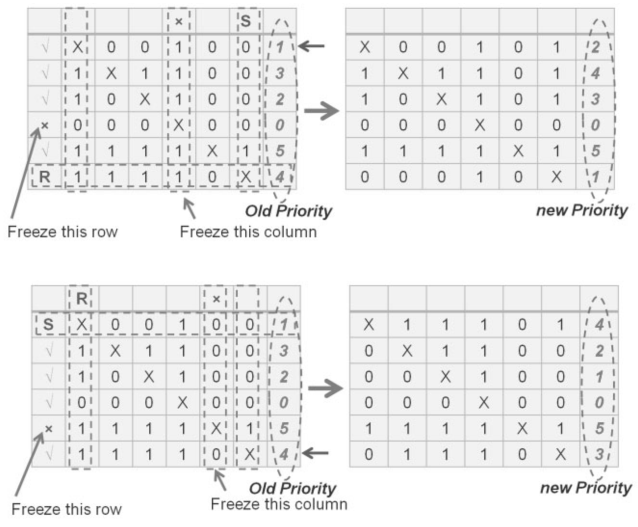


**Figure 7. Selective LRG (top) and selective MRG (bot)**

In this scheme MRG update is applied to a selective section of inputs. In the standard MRG scheme, the input that used the output bus most recently is upgraded to have the highest priority while all inputs with higher priorities get downgraded by exactly one level. In this case *input1* with a priority level 1 used the channel most recently. However, in the selective MRG scheme, instead of upgrading *input1* all the way up to 5, we intend to upgrade it to some intermediate priority (say priority level of *input6* which is 4). To accomplish this, before setting/resetting priority bits we identify certain rows and columns that need to be frozen. In this case, all columns corresponding to priority bits that are low in the sixth row are frozen as shown in Fig. 7 bot. Simultaneously, all rows corresponding to priority bits that are high in the sixth column (which is the priority line for *input6*) are also frozen. Following this the priority bits in the first row are set (except the bits in frozen columns) and those in the first column are reset (except the bits in frozen rows). This ensures that the new priority matrix is consistent and the intended priority update is achieved.

## 4. QoS ARBITRATION

In a 64×64 SSN it might take a message 64 cycles to win arbitration in the worst case when all inputs collide. To assist critical messages to reach destination early, SSN also features a 4-level message-based QoS arbitration technique that allows only input buses with the highest message priority to arbitrate for the channel as shown in Fig. 8. A 2-bit *message priority* is decoded into a 4-bit thermometer code at the crosspoint, which is used to selectively discharge priority bit-lines comprising the *QoS priority bus*. A multiplexer samples one of those priority bit-lines using its own *message priority* and the input bus progresses to the LRG arbitration cycle if the monitored priority bit is not discharged. Using separate wires for QoS arbitration incurs 3% area overhead. However, the additional QoS arbitration cycle can be overlapped with the prior routing operation for the output bus, avoiding a latency penalty.

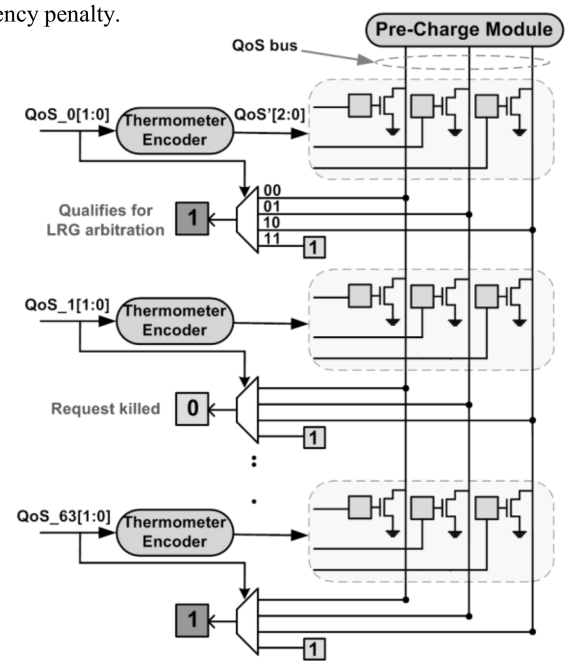


**Figure 8. QoS arbitration technique**

## 5. PROTOTYPE IMPLEMENTATION

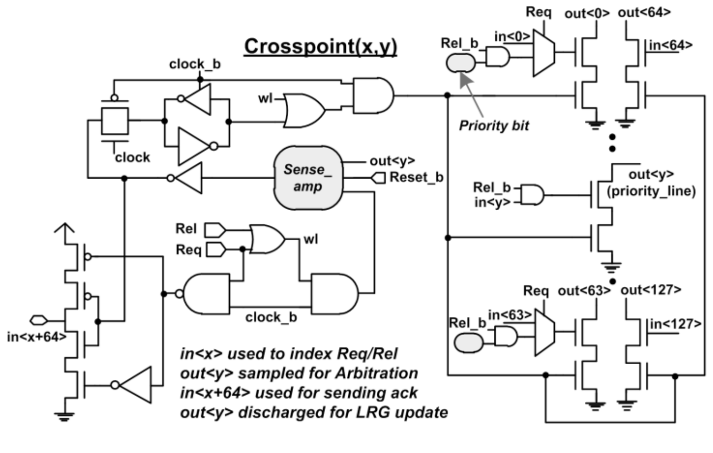


**Figure 9a. Crosspoint circuit**

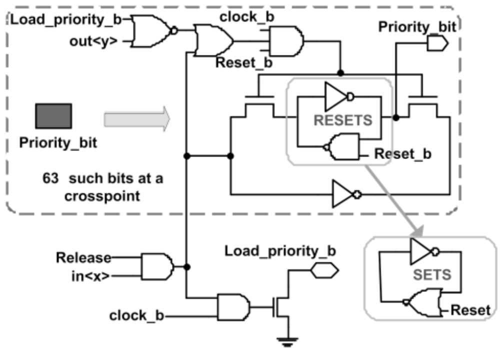


**Figure 9b. Priority storage latch**

Fig. 9 shows the SSN crosspoint circuit and the priority storage latch. *Load_priority_b* is an additional bit-line provided per channel that is discharged during the release cycle. This triggers the priority update mechanism. During a request/release cycle the channels are indexed using the lower 64 bits from the input bus. Crosspoints send acknowledgements over the upper 64 bits.

Fig. 10 shows the test prototype fabricated in 45nm SOI CMOS with a 64×64 SSN as the communication network between the traffic generators and traffic analyzers. SSN is laid out using a semi custom design flow. A generic crosspoint cell is laid out as a parameterized cell. A skill script parses the generic crosspoint by taking in the x coordinate, y coordinate and the priority vector (at reset) as the arguments and generates crosspoint specific to each location. These crosspoints are then tiled using a compiler that appropriately sizes the word line drivers and precharge transistors to generate the switch fabric.

The SSN features 8448 word-lines and 8576 bit-lines spread across 4096 crosspoints. The integration of the LRG and QoS control within this fabric with very low overhead greatly improve SSN's scalability and makes it possible to realize a fabric of this large size. In addition, new bi-directional repeaters (Fig. 11) are used for bit-lines that use a regenerative sensing element to improve delay despite high slew rates on long bit-lines. The proposed repeater uses a thyristor element to detect and amplify a transition on the bit-line. Once a transition is detected the repeater enters a self regeneration mode where it decouples itself from the slow transitioning bit-line. This allows the internal nodes in the thyristor to switch faster and reduces delay. The regeneration and self-decoupling mechanism improves bit-line delay by 32% and allows for a 50% smaller bit-line driver compared to a conventional repeater. Simulated fabric latency with increasing SSN size shows 1.6× performance improvements over an SSN with un-repeated bit-lines as shown in Fig. 11 due to the near-linear latency increase with radix size rather than quadratic dependency without repeaters. Bit-lines are pre-charged within every 16×16 SSN macro. This improves pre-charge time by 59% over a similar sized lumped driver and results in more uniform current drawn from the power grid. Bit-line delay degrades more rapidly than word-line delay under voltage scaling. Hence, the bit-cell aspect ratio (1:0.73) is chosen to shorten bit-lines, improving fabric latency at low $V_{dd}$. Fine grain clock gating reduces clock power by 94% at each crosspoint. A crosspoint is clocked only if its connectivity status is ON, a request is asserted, or an LRG priority update occurs. These events are registered in the positive clock phase, allowing gating of the negative (active) phase with 2.3% delay penalty. Adjacent SSN input ports are driven from opposite directions, reducing routing congestion and local Ldi/dt drop when repeaters on the 2.5mm long word-lines switch.

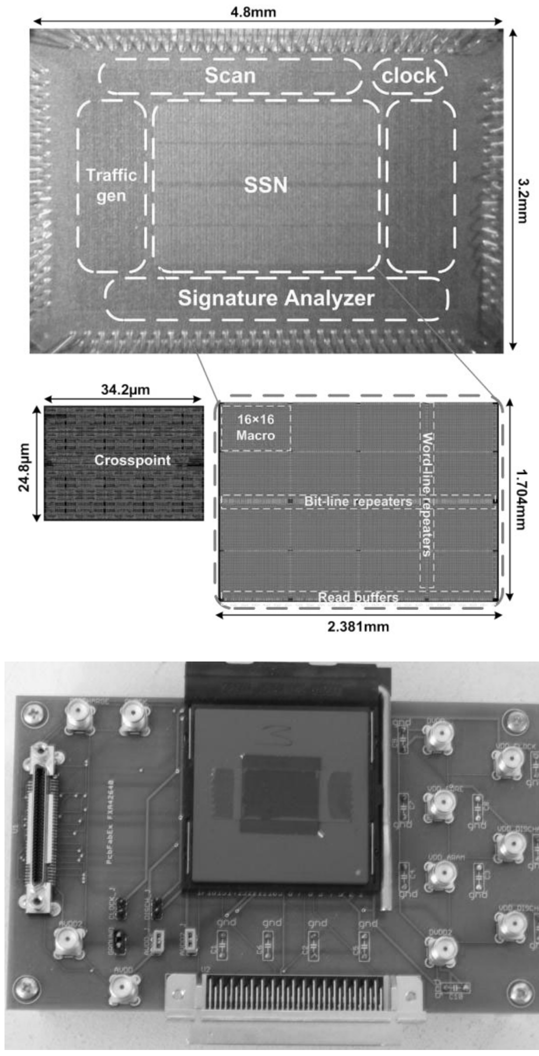


**Figure 10. SSN die photo (top) and printed circuit board hosting SSN test prototype (bottom)**

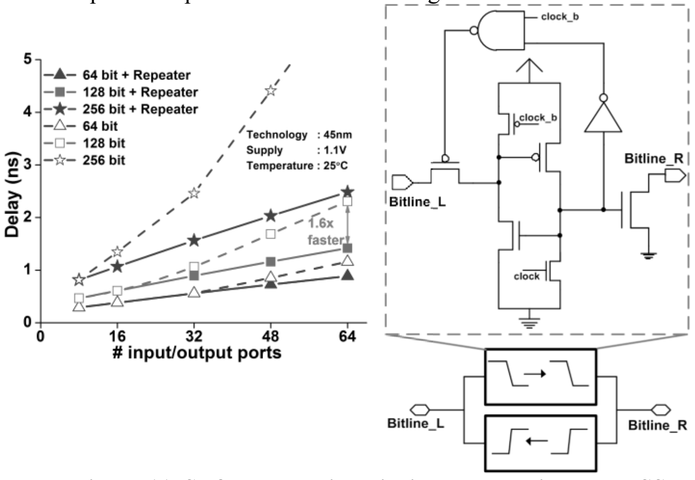
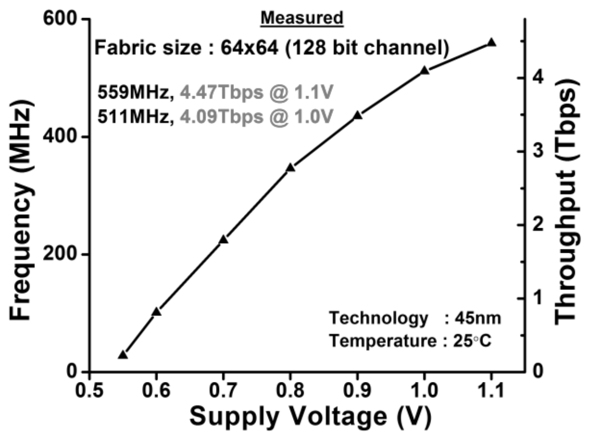


**Figure 11. Self-regenerating bit-line repeater improves SSN delay by 1.6×**

## 6. MEASUREMENT RESULTS



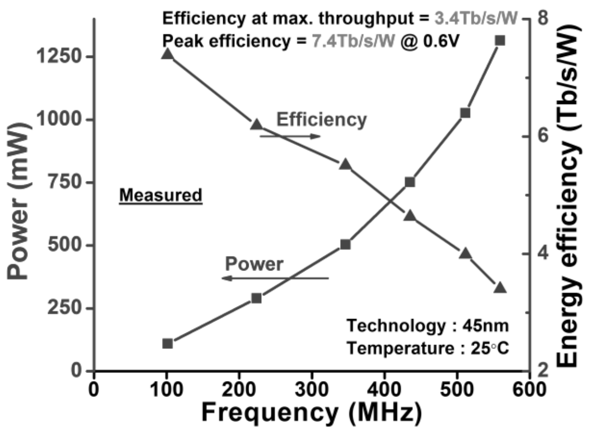


**Figure 12. Measured performance and power for 64×64 SSN**

The traffic generators in the test prototype can be tuned to produce traffic patterns with varying switching activity and collision patterns. The SSN is tested for functionality by streaming various data streams through it and verifying the signatures. Fig. 12 (top) shows the measured SSN's operating frequency and the aggregate throughput at varying supply voltage. SSN's power consumption at different operating frequencies is shown in Fig. 12 (bottom). At 1.1V, the SSN operates at 559MHZ with a throughput of 4.47Tb/s while consuming 1.32W. This translates into an efficiency of 3.4Tb/s/W which is which is 3.7× higher than [4] at similar bandwidth. The work in [4] uses an 8×8 mesh topology based on 5×5 routers at each node to connect 64 units, whereas the SSN uses a 64×64 single-stage fabric. The SSN is fully functional down to 550mV with a measured peak efficiency of 7.4Tb/s/W at 0.6V.

## 7. CONCLUSION

In this paper, we present a self-arbitrating fabric called SSN that leverages a novel priority encoding scheme that re-uses existing logic and interconnect resources in switch fabric to locally store priorities at router crosspoints resulting in a compact implementation. A 64×64 SSN with 128b data bus achieves a peak throughput 4.5Tb/s at an energy efficiency of 3.4Tb/s/W while spanning only 4.06mm$^2$ in 45nm SOI CMOS. It features a single cycle least recently granted arbitration technique that re-uses data buses and switching logic, a 4-level message based priority arbitration for quality of service and unique bidirectional bit-line repeaters to aid scalability. The unique priority encoding scheme also allows seamless implementation of many other arbitration policies in addition to LRG with very minimal overhead.

## 8. ACKNOWLEDGEMENTS

The authors gratefully acknowledge funding and support from ARM Ltd.

## 9. REFERENCES

[1] D.Truong *et al.*, “A 167-processor 65nm Computational Platform with per-Processor Dynamic Supply Voltage and Dynamic Clock Frequency Scaling,” International symposium of VLSI Circuits, pp. 22-23, 2008.

[2] S. Vangal *et al.*, “An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS,” International Solid State Circuits Conference, pp. 98-99, 2007.

[3] S.Tremblay *et al.*, “A Third-Generation 65nm 16-Core 32-thread Plus32-Scout-Thread CMY SPARC Processor,” International Solid State Circuits Conference, pp. 82-83, 2008.

[4] S. Vangal *et al.*, “A 5.1 GHz 0.34mm2 Router for Network-on-Chip Applications,” International Symposium on VLSI Circuits, pp. 42-43, 2007.

[5] M.Anders *et al.*, “A 4.1Tb/s Bisection-Bandwidth 560Fb/s/W Streaming Circuit-Switched 8×8 Mesh Network-on-Chip in 45nm CMOS,” International Solid State Circuits Conference, pp. 110-111, 2010.

[6] K.Kim *et al.*, “A 211 GOPS/W Dual-Mode Real-time Object Recognition Processor with Network-on-Chip,” European Solid State Circuits Conference, pp. 462-465, 2008.

[7] J.Kim *et al.*, “A 118.4 GB/s multi-casting Network-on-Chip with hierarchical star-ring combines topology for real-time object recognition,” Journal of Solid State Circuits, vol.45 pp. 1309-1409, 2010.

[8] E. Karl *et al.*, “ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon,” IEEE Design and Test of Computers, Vol. 23, No. 6, pp. 484-490, 2006.

[9] D. Flynni *et al.*, “AMBA: enabling reusable on-chip designs,” IEEE Micro, pp. 20-27, 1997.

[10] P. Kongetira *et al.*, “Niagara: A 32-way multithreaded Sparc processor,” IEEE Micro, pp. 21-29, 2005.

[11] P. Salihundam *et al.* ,“A 2Tb/s 6×4 Mesh Network with DVFS and 2.3Tb/s/W router in 45nm CMOS,” International Symposium on VLSI Circuits, pp. 79-80, 2010.

[12] S. Satpathy *et al.*, “A 1.07 Tbit/s 128×128 Swizzle Network for SIMD Processors,” International Symposium on VLSI Circuits, pp. 81-82, 2010.

[13] M. Lee *et al.*, “Probabilistic distance-based arbitration: Providing equality of service for many-core CMPs,” IEEE MICRO, 2010.

[14] S. Satpathy *et al.*, “SWIFT: A 2.1Tb/s 32x32 self-arbitrating many-core interconnect fabric,” International Symposium on VLSI Circuits, pp. 81-82, 2010.