HCI-Tolerant NoC Router Microarchitecture

Dean Michael Ancajas James McCabe Nickerson Koushik Chakraborty Sanghamitra Roy

USU BRIDGE LAB, Electrical and Computer Engineering, Utah State University
{dbancajas, jmnickerson}@gmail.com {koushik.chakraborty, sanghamitra.roy}@usu.edu

ABSTRACT

The trend towards massive parallel computing has necessitated the need for an On-Chip communication framework that can scale well with the increasing number of cores. At the same time, technology scaling has made transistors susceptible to a multitude of reliability issues (NBTI, HCI, TDDB). In this work, we propose an HCI-Tolerant microarchitecture for an NoC Router by manipulating the switching activity around the circuit. We find that most of the switching activity (the primary cause of HCI degradation) are only concentrated in a few parts of the circuit, severely degrading some portions more than others. Our techniques increase the lifetime of an NoC router by balancing this switching activity. Compared to an NoC without any reliability techniques, our best schemes improve the switching activity distribution, clock cycle degradation, system performance and energy delay product per flit by 19%, 26%, 11% and 17%, respectively, on an average.

1. INTRODUCTION

In the forthcoming era of many-core computing, fueled by the tremendous growth in on-chip resources from technology scaling, Network-on-Chip (NoC) architectures have emerged as the design of choice for on-chip communication. On the other hand, rapid technology scaling has severely undermined the device level reliability, forcing the chip designers to critically consider long term sustainability in system design. While a large body of recent works targets on-chip computing resources (processing cores), many-core systems must consider reliability and sustainability of NoCs. Various aging mechanisms such as *Negative Bias Temperature Instability (NBTI)*, *Hot Carrier Injection (HCI)*, *Time Dependent Dielectric Breakdown (TDDB)*, and *Electromigration* play a major role in degrading performance characteristics of NoCs over time. Such a performance degradation can have a massive system level impact in NoCs, and may ultimately shorten the chip lifetime prematurely [2, 3].

To extend the period of fault-free execution, few recent works have addressed aging challenges in NoCs by mitigating NBTI or Electromigration. For example, Fu et al. propose techniques to mitigate NBTI aging in NoCs by balancing the duty cycle in the Virtual Allocator circuits [10]. Bhardwaj et al. propose aging-aware adaptive routing to throttle NBTI and Electromigration degradation [3]. NBTI is a critical but recoverable device aging mechanism. In contrast, HCI is an unrecoverable aging phenomena [15], which affects the components due to their dependence on switching activity [17]. Due to aggressive transistor scaling, the thinner gate dielectric in CMOS transistors increases the probability of HCI degradation. In fact, HCI can account for a major component of aging in a 10-year product lifetime [19]. To the best of our knowledge, none of the existing works consider HCI aging in the NoC architecture.

In this work, we perform a holistic cross-layer analysis of HCI degradation in the NoC router microarchitecture. We focus on the crossbar structure of the router microarchitecture, due to its profound significance in dictating the router frequency [16]. Combining application level traffic profile with bit level logic analysis, we find that the crossbar structure is highly vulnerable to HCI aging. Due to the data communication patterns in many-core applications, we observe that a majority of gate-level switching activities are restricted to a small portion of the entire crossbar circuit topology, resulting in a large HCI degradation. To throttle HCI aging in the crossbar, we propose a series of low-overhead techniques that evenly distribute the switching activity in the crossbar, without affecting the architecture level routing latency and bandwidth.

We make the following contributions in this paper.

- We develop a cross-layer framework for HCI aging analysis of an NoC router. Our framework combines application traces, RTL gate-level simulation of a crossbar circuit, logic analysis, and HSPICE simulation of HCI degradation effect (Sections 3.2 and S1).
- We analyze the switching activity of the crossbar, a major circuit in an NoC router using real-world applications and find that only a small group of gates account for most of the switching activity. On an average for PARSEC benchmarks, only 25% of the gates account for more than 75% of the switching activity, severely damaging some gates while leaving others unscathed (Section 3).
- We propose four schemes using low overhead techniques to evenly distribute the switching activity and minimize HCI degradation (Section 4). Our four schemes are: *Bit Cruising* that distributes the high activity bits around the channel; *Distributed Cycle Mode* that exploits idle cycles in the NoC; *Crossbar Lane Switching*

DAC '13, May 29 - June 07 2013, Austin, TX, USA.

that manipulates the port in the crossbar by utilizing the virtual channels; and a combination of Bit-Cruising and Crossbar Lane Switching.

- We present a holistic evaluation of our proposals spanning full-system simulation down to RTL and gate-level HSPICE simulation (Section 6). Our best schemes improve the switching activity distribution by up to 31% (ave: 19%). We also see a maximum of 30% (ave: 26%) improvement in the clock cycle degradation, while the system performance degradation is reduced by up to 17.6% (ave: 11%) compared to the baseline scheme. The Energy Delay Product per Flit is improved by up to 27% (ave: 17%).

2. BACKGROUND

In this section, we introduce our HCI model that correlates threshold voltage degradation with the time spent by a transistor in stress.

HCI occurs when a carrier overcomes the potential barrier between silicon and the gate oxide and leaves the channel. A portion of the carriers (hole/electrons) that leave the channel are deposited into forbidden regions in the transistor such as the gate oxide. Throughout a transistor's lifetime, these deposited carriers change the conductive properties of the transistor and ultimately lead to degradation of the threshold voltage (V_{th}), drain saturation current (I_{on}) and transconductance (Δg_m).

The HCI effect on the transistor parameters described above can be modeled as a power-law with respect to the stress time (t) [6, 22]. We only discuss the V_{th} model as the one for I_{on} is similar. The model for Δg_m can be seen in [6].

$$\Delta V_{th} = A \cdot t^n \tag{1}$$

where A and n are technology dependent parameters. Parameter n has been widely accepted as $\sim$0.5 over a wide range of processes [1]. Parameter t is the time the transistor is under stress, while A is derived as:

$$A = \frac{q}{C_{ox}} K \sqrt{C_{ox}(V_{GS} - V_{th}} \cdot e^{\frac{E_{ox}}{E_0}} e^{\frac{\varphi_{it}}{q\lambda E_m}} \tag{2}$$

All relevant parameters in Equation 2 can be obtained from Wenping et al. [23]. The stress time of a transistor is derived from the transition density and the pertinent transitions, since not all inputs that cause switching have a significant contribution to HCI aging [13]. We give a brief background of how we estimate pertinent transitions in our framework in Section S1.1.

3. MOTIVATION

In this section, we motivate the need for HCI-aware design of components in an NoC router. We first discuss major reliability concerns in the datapath of an NoC router. We then explain our framework for holistic HCI aging analysis of the NoC crossbar. Lastly, we discuss our results, demonstrating the need for HCI-aware techniques in the design of resilient NoC routers.

3.1 HCI Degradation in the NoC Crossbar

Massively parallel programs running in the many-core use the NoC as an interconnect fabric due to scalability demands. Processors communicate with each other through messages sent as packets in the NoC. Since on-chip wiring is abundant, a lot of these packets that were previously sent over narrow off-chip buses now cannot fully utilize the whole channel bandwidth available. Coupled with the fact that most data sent through the network are narrow width [8], this trend leads to uneven sensitization of transistors, eventually causing unbalanced HCI degradation across the channel.

Circuit	Logic Depth	# of gates
Crossbar Switch	4	5760
64-bit ALU	46	4728
Address Generator	43	491
Issue Queue Logic	33	189

Table 1: Logic Depth of Various Modules

The crossbar switch is at the heart of the communication infrastructure in an NoC router[1], largely dictating the cycle time [16]. There are three critical reliability issues in an NoC crossbar. First, the gate level activity in a crossbar is only concentrated in a very few bits of the channel width, due to the bit patterns being sent. This asymmetry causes unbalanced HCI degradation. Second, since most upper bit transistors do not switch and only maintain their values, they can undergo NBTI degradation. Third, since the crossbar is a wide circuit with a shallow logic depth (Table 1), minor delay variations caused by both HCI and NBTI will have a profound effect on its overall critical path delay.

3.2 Aging Analysis Framework for the NoC Crossbar

Figure 1 shows the methodology we employ in assessing HCI degradation in the crossbar circuit. Our cross-layer approach comprises system level simulation of 16-thread parallel programs and their gate-level HCI degradation in a crossbar circuit. Since HCI depends on switching activity, we acquire the switching activity of each gate by capturing cycle-by-cycle actual data values traversing the crossbar. We then evaluate its overall degradation effect for each transistor in the circuit using our model discussed in Section 2. However, using real-world applications to assess gate-level degradation is a computationally intensive task. As such, we have adopted several important steps to efficiently avoid long simulation times, while still providing a holistic analysis of HCI aging effect.

First, we pick multiple sample points in different phases of execution of the program. The sample points are chosen according to traffic intensity in the NoC. Each sample phase contains about 1 million flits. Second, we run our simulation setup (Section 5) and take the traces of data traffic at the specified points. Third, we feed these data traces to an Open Source RTL Verilog model of a 16-core NoC and gather cycle-by-cycle inputs in the crossbar circuit. Lastly, we use our novel HCI Aging Analyzer Framework (Section S1) to analyze degradation in the circuit.

3.3 Results

3.3.1 Logic Depth Analysis

Table 1 shows the results for the logic depth analysis we perform on major circuits from NoC and processor systems. We analyzed the crossbar switch from an NoC, the Arithmetic and Logic Unit (ALU), the memory address generator

[1]We provide an NoC primer on Section S3.

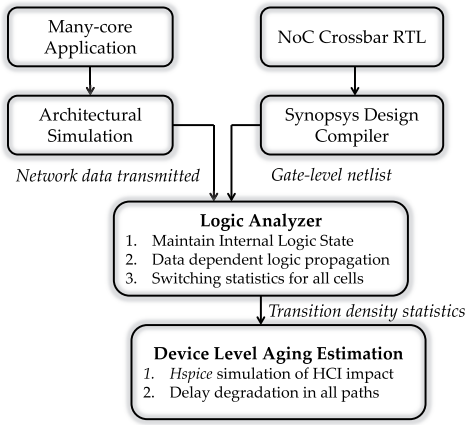


Figure 1: HCI Aging Analysis Framework

and the issue queue selector of a Fabscalar core [7]. Among all these modules, the crossbar has the shallowest logic depth that can be 10× lower than the other circuits. This characteristic makes it more susceptible to aging as there is little chance that a different signal path can hide the delay incurred by degraded transistors. Thus, we need to implement efficient aging mitigation techniques in the crossbar data path.

3.3.2 HCI Degradation Results

Figure 2 shows the switching activity data in the crossbar circuit. The x-axis shows the percentage of gates while the y-axis shows its accumulated switching activity as a percentage of the total activity. Ideally, a 1:1 ratio between the percentile gates and the switching activity is optimal for HCI aging (i.e. a straight line with a 45° slope). However, it can be seen that on an average, only 25% of the gates account for 75% of the total switching activity. This large asymmetry leads to unbalanced HCI degradation between different parts of the circuit and can accelerate failure of NoCs before their rated lifetime.

We also show the corresponding clock cycle degradation of an NoC router (22nm, 7 years) in Figure 3 as a result of the unbalanced HCI degradation. From this data, *swaptions* experiences the most clock cycle degradation at 10.51%, while *canneal* has the least at 8.99%. We can also verify this trend from Figure 2, where swaptions (left most curve) has the most concentrated switching activity among all programs.

Both the results above show that the inherent imbalance in switching activity caused by data patterns sent over the network causes non-uniform HCI aging in the crossbar circuit. This asymmetrical aging causes some path delays to increase disproportionately and will eventually lead to premature router failure. In the succeeding sections, we will discuss our proposed designs that primarily shift the switching activity from one part of a circuit to another in order to slow down HCI degradation and balance aging impact.

4. DESIGN OVERVIEW

In this section, we discuss our proposed techniques for mitigating HCI effect in the router crossbar. Our techniques aim to balance HCI degradation by distributing the switching activity. We explore four techniques in the router microarchitecture: *Bit Cruising (BC)*; *Distributed Cycle Mode (DCM)*; *Crossbar Lane Switching (CLS)*; and *BCCLS* that is a combination of schemes BC and CLS. Apart from DCM,

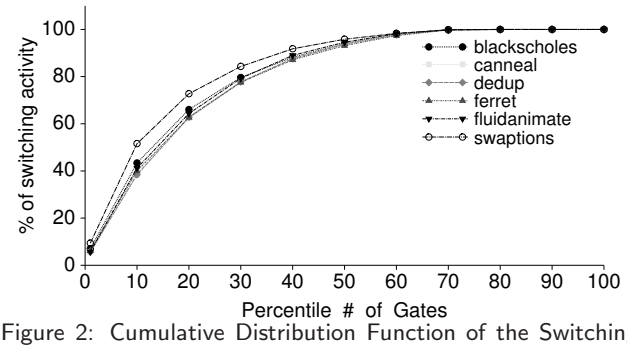


Figure 2: Cumulative Distribution Function of the Switching Activity vs Gate Count

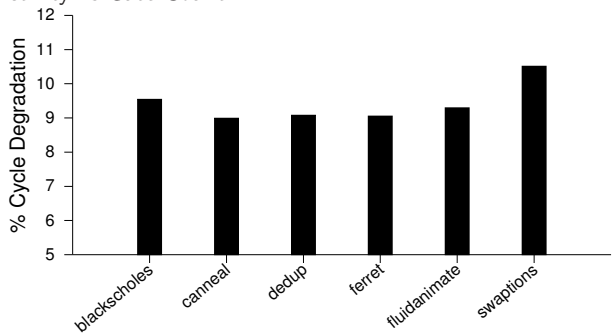


Figure 3: Clock Cycle Degradation of a 22nm NoC Router due to HCI after 7 years

all of our schemes involve minimal modifications at the front-end of the router and do not affect the critical path of the pipeline (crossbar traversal).

4.1 Bit Cruising (BC)

Bit Cruising swaps the different portions of the data being transmitted in the crossbar. This technique is largely motivated by two properties of the programs. First, most data traversing the NoC do not occupy the full channel width of the network because most data in the cache line are aggregated at the lower bits. In some cases, all data bits are actually zero. Recent works have also exploited this characteristic by compressing flits and sending only those that have important data [8]. Second, control requests, while being sent as a single flit also do not store information in the most significant portions of the channel as the routing information can fit in the first few bytes of the whole channel. In our setup, the control flit only utilizes 25% of the channel width, leaving the remaining 75% constant. Together, these two characteristics radically lower the switching activity in certain bits while emphasizing others.

To prevent this asymmetry in HCI degradation, the data being sent across the network must be such that the switching activity across the channel is distributed. By passing different data values each time a gate is used, it will balance the switching activity and hence also uniformly degrade all gates. This is the primary working philosophy of Bit Cruising, where highly changing bits are being *cruised* around the channel. The Bit Cruiser circuit is situated in the Network Interface (NI) and **does not add any overhead in the critical path of the pipeline** of an NoC. We explain in detail the functionality and the circuit implementation of the BC circuit in Section S2.

4.2 Distributed Cycle Mode (DCM)

The Distributed Cycle Mode aims to balance out degrada-

tion of transistors by latching an input value in the crossbar during idle times such that unswitched transistors in previous cycles will transition and experience equivalent aging. As such, it does not relieve any HCI aging compared to our other schemes but can be beneficial as equally aged transistors have smaller leakage power. The DCM mode can also be coupled with NBTI recovery schemes such as [21]. We explain the DCM mode in more detail in Section S4.

4.3 Crossbar Lane Switching (CLS)

Our two previous techniques focused on distributing the switching activity across an entire channel of an input port to balance HCI degradation. However, another asymmetrical degradation also occurs in the crossbar lanes that are immune to techniques applied in the channel level.

This type of asymmetric degradation arises when some input-output pairs are used more than others. We demonstrate this occurrence with an example in Figure 4 where there are two paths (p0 and p1) that both use the same *East* output port. For instance, if path p0 is used more than p1, then the transistors along the path p0 will be sensitized more and hence, experience more HCI degradation.

Our third technique, CLS, is also situated at the front-end of the router pipeline and aims to balance the usage of the crossbar lanes[2]. In the canonical router model, an input port directly forwards flits to the output ports by establishing a physical connection between the two via the crossbar switch. As such, flits coming from the same input port will always use the same crossbar lane to connect to different output ports. However, the introduction of Input Buffers (IB) and Virtual Channels (VC) in modern router architectures decouples this one-to-one association because the flits are first stored in the IB before being transmitted to the output ports. With trivial modifications in the VC allocator and the Route Calculation part of the pipeline, we can control which crossbar lane an input port will utilize at any given time.

This new allocation and routing policy will now cause the crossbar circuit to use a different path and activation circuit, but still send the same data as if it were coming from the original input port. Thus, we preserve the correctness of the flit and the route. Similar to the Bit Cruising technique's *cruise setting*, CLS will need a *knob* input to indicate the new mapping between input ports and crossbar lanes. We expand on this and explain the required overheads in implementing CLS in Section S5.

4.4 Bit Cruising and Crossbar Lane Switching (BCCLS)

Our last technique is a combination of the BC and CLS schemes. BCCLS combines both the benefit of switching distribution inside a channel (BC scheme) and the distribution of activity across many channels (CLS scheme). The implementation of BCCLS comes naturally because both BC and CLS tackle different portions of the router circuit. BC reshuffles the data sent through the network while CLS effectively changes the port a flit is coming from by modifying the VC allocation and route calculation.

5. METHODOLOGY

[2]a lane is the path taken by an input port to the output port

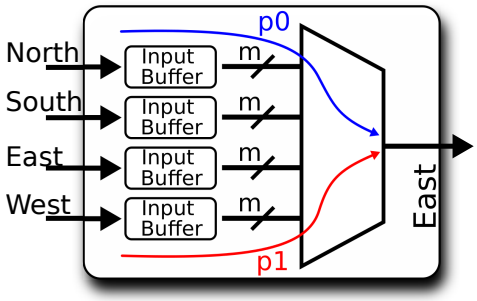


Figure 4: East Section of A Crossbar Switch. CLS works on the **inter**-lane[3] (by changing the path of the data) level while BC works only on the **intra**-lane level (by changing the bit ordering within a path).

In this section, we discuss our simulation infrastructure that combines multiple tools across different abstraction layers. Our methodology can be broadly classified into three categories: Architectural Setup, RTL and Switching Activity Simulation and HCI degradation analysis using SPICE.

5.1 Architectural Setup

Our simulation setup is composed of a 16-node mesh system arranged in a 4×4 grid. Each node in the system is composed of 1 processor, 1 L1 Cache and a slice of a system-shared L2 cache. Each router in the system has seven sets of input and output ports including the ones for the processor and caches. The flit size is configured at 16-bytes (128 bits). A single control request fits in a single flit while data flits needed to transfer a 64-byte cache line are sent in five (4 data + 1 control) consecutive flits. Each processor's L1 and L2 cache sizes are 64kB and 512kB, respectively.

5.2 RTL and Switching Activity Simulation

The first step in obtaining an accurate switching activity is to produce real-word data vectors from standard benchmark programs as inputs to the RTL circuits. We use the PARSEC [4] benchmark suite (large inputs) running on gem5 [5] to collect data traces. We collect data traces for the four center most routers in a 16-node mesh.

After the traces are taken, we implement a trace feeder through a Verilog VPI based functional verification framework called Teal [18]. This module allows us to easily obtain cycle-by-cycle values in any sub-module of the router such as the crossbar.

5.3 HCI Degradation Analysis

Using the outputs from the previous step, our logic analysis tool is then used to obtain the transition densities of each transistor (Figure 1). We post-process all the results in our HAAF (Section S1) to calculate V_{th} degradation and simulate them in HSPICE to obtain clock cycle degradation data for all paths and for different benchmarks. In all our analysis, we use the 22nm [24] technology and an aging period of 7 years.

6. RESULTS

In this section we present the effectiveness of our schemes across different metrics.

6.1 Comparative Schemes and Evaluation Metrics

We compare the following five schemes:

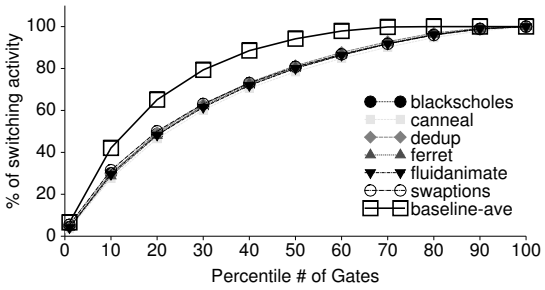


Figure 6: CDF for BCCLS scheme.

- **BASE**: Baseline configuration where the system is unmodified.
- **BC**: Bit Cruising scheme, the channel is divided into four segments and a bit cruiser circuit is placed between the NI and the router.
- **DCM**: Distributed Cycle Mode technique presented in Section S4.
- **CLS**: Crossbar Lane Switching scheme discussed in Section 4.3.
- **BCCLS**: Combination of BC and CLS schemes.

We evaluate all these schemes in terms of switching activity distribution through Cumulative Distribution Function (CDF) plots, clock frequency degradation, Energy-Delay Product Per Flit (EDPPF) and System Performance. The circuits used to facilitate all these schemes (except DCM) are added in the front-end of the pipeline without affecting the actual crossbar circuit, and as such do not incur any additional timing overhead in the critical path[3].

6.2 Switching Activity Distribution

We show the CDF plots (Figures 5 and 6) of the switching activity distribution of all schemes. The average of the baseline scheme is superimposed in each figure. All of our schemes outperform the baseline by having a lower value (y-axis) at any percentile point. Hence, it is evident that our schemes achieved their aim of distributing the switching activity. At an evaluation point of 20 percentile, our best performing scheme (BCCLS in Fig. 6) shows 31% less switching activity compared to the baseline.

6.3 Clock Cycle Degradation

Figure 7(a) shows the cycle degradation for the NoC router at the end of a 7 year aging period using the ASU 22nm predictive technology model [24] operating at 1 Ghz. On an average, the base scheme degrades the clock cycle by 9.4%. Our schemes improve this degradation by 20.6%, 0%, 12% and 25.5% for BC, DCM, CLS and BCCLS, respectively. Combining both BC and CLS schemes results in the least amount of clock cycle degradation while DCM provides no improvement from the baseline. As HCI is an unrecoverable degradation [15], any damage done during normal operation cannot be rectified. The difference between DCM and all other schemes is that it is reactive while the others are proactive (preventing aging beforehand). However, DCM improves other aspects of the circuit such as the EDPPF which will be discussed next.

6.4 Energy Delay Product Per Flit (EDPPF)

[3]DCM's cycle degradation is taken without the timing of the additional multiplexers as we want to show the timing degradation in the crossbar circuit only across all schemes.

We show in Figure 7(b) the EDPPF of all schemes. The base scheme is shown as a line at 100%. Most schemes have lower EDPPF compared to the baseline except for some outliers. For the BC scheme, *dedup* and *ferret* have larger EDPPFs while for CLS, *swaptions* has a slightly larger EDPPF than the baseline. Upon further investigation, although BC has helped achieve less degradation and a more distributed switching activity, its dynamic switching activity for benchmarks *dedup* and *ferret* are actually 63% and 30% more compared to the average of all other programs. This unusual activity increase is due to the workload-dependent bit patterns being sent across the network. For *swaptions*, the switching activity for the benchmark is unusually high in all schemes except for BC.

Even though DCM does not provide any improvement in the clock cycle, it provides consistent reduction in EDPPF. This reduction is because optimally aged transistors have higher threshold voltages and will have lesser leakage power. Leakage power cannot be ignored in small technologies such as the one we are using (22nm). On an average, DCM improves the EDPPF by 18% compared to the baseline.

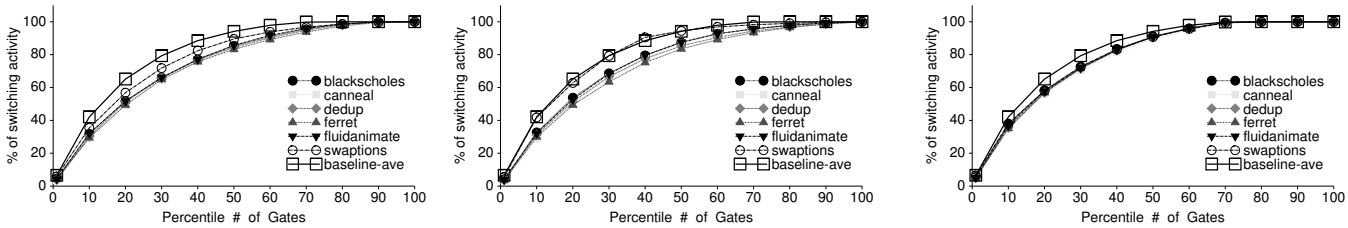
6.5 System Performance

Figure 7(c) shows the overall system performance impact of all schemes relative to the baseline. DCM shows no improvement because it has the same clock degradation as the baseline. On an average, performance degradation is reduced by 9.3%, 8% and 11% for BC, CLS and BCCLS schemes. Maximum is 17.6% for the BCCLS scheme running *ferret*. Overall, the system performance improvement is less than the clock cycle degradation improvement due to the sublinear dependence of clock frequency and performance.

7. RELATED WORK

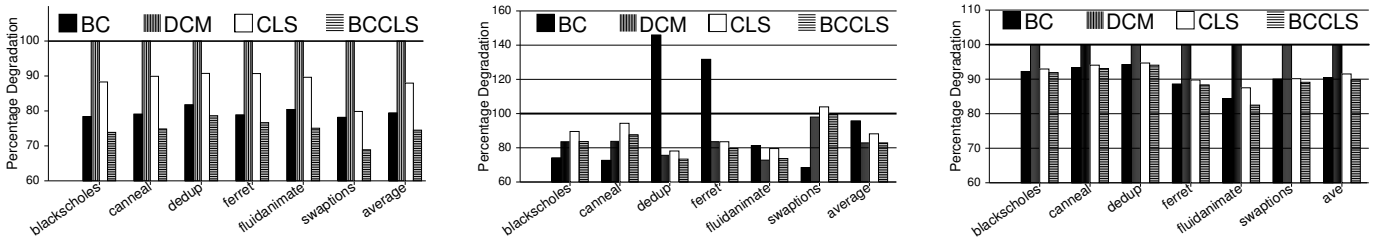
The aggressive scaling in CMOS technology has made reliability a primary design constraint in modern computing systems. While there has been a wide scope of studies tackling different reliability issues (NBTI, TDDB, HCI) in processing elements [11, 21], there is only a limited number of works which address wear-out mitigation in the on-chip communication infrastructure of such systems. Bhardwaj et al. implemented a dynamic routing algorithm to equalize NBTI and electromigration aging across the on-chip network [3]. Fu et al. created new virtual channel allocation and routing algorithms in order to improve process variation and NBTI effects in key components of the router [10]. Park et al., Fick et al. and Kim et al. explored fault tolerant NoC architectures by decoupling modules and having redundancies in order to recover from intermittent errors in the network or provide graceful degradation [9] [14] [20].

Most of the studies mentioned above focus on recovering from intermittent errors or minimizing NBTI effect on storage elements by balancing the duty cycle. On the contrary, our work focuses on HCI, an unrecoverable aging phenomena that affects combinational components. HCI mitigation presents a different set of challenges because of its dependence on the switching activity of transistors, as opposed to NBTI which depends only on the input bias. To the best of our knowledge, our study is the first work to tackle HCI in an NoC router microarchitecture.



(a) Bit Cruising (b) DCM (c) Crossbar Lane Switching

Figure 5: Cumulative Distribution Graph of BC, DCM and CLS schemes. Solid line in each graph is the baseline average. (Lower is better.)



(a) Cycle Degradation (b) EDPPF (c) System Performance Degradation

Figure 7: Router Cycle time Degradation, Energy Delay Product Per Flit and System Performance Degradation comparison. Solid line indicates baseline. (Lower is better).

8. CONCLUSION

In this paper, we find out that Network On Chip architectures running parallel programs produce communication patterns that lead to unbalanced HCI degradation through asymmetrical gate switching activity. We exploit this property and present four novel proposals in HCI mitigation in the crossbar circuit, a major component in an NoC router which dictates the operating frequency of the network. Overall, our schemes distribute the switching activity, improve the clock cycle degradation, energy delay product per flit and system performance.

Acknowledgments

This work was supported in part by National Science Foundation grants CNS-1117425 and CAREER-1253024, and donation from the Micron Foundation.

9. REFERENCES

[1] Bertolini, C. and others Relation between HCI-induced performance degradation and applications in a RISC processor. *Proc. of OLTS* (2012), 67–72.

[2] Bhardwaj, K. and others An MILP Based Aging Aware Routing Algorithm for NoCs. In *Proc. of DATE* (2012), pp. 326–331.

[3] Bhardwaj, K. and others Towards Graceful Aging Degradation in NoCs Through an Adaptive Routing Algorithm. In *Proc. of DAC* (2012), pp. 382–391.

[4] Bienia, C. and others The PARSEC benchmark suite: characterization and architectural implications. In *PACT* (2008), pp. 72–81.

[5] Binkert, N. and others The gem5 simulator. *SIGARCH Comput. Archit. News 39*, 2 (Aug. 2011), 1–7.

[6] Bravaix, A. and others Hot-Carrier acceleration factors for low power management in DC-AC stressed 40nm NMOS node at high temperature. In *Proc. of RPS* (2009), pp. 531 –548.

[7] Choudhary, N. K. and others FabScalar: composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proc. of ISCA* (2011), pp. 11–22.

[8] Das, R. and others Performance and power optimization through data compression in Network-on-Chip architectures. In *HPCA* (2008), pp. 215–225.

[9] Fick, D. and others Vicis: a reliable network for unreliable silicon. In *Proc. of DAC* (2009), pp. 812–817.

[10] Fu, X. and others Architecting reliable multi-core network-on-chip for small scale processing technology. In *Proc. of DSN* (2010), pp. 111–120.

[11] Gupta, S. and others StageNetSlice: a reconfigurable microarchitecture building block for resilient CMP systems. In *Proc. of CASES* (2008), pp. 1–10.

[12] Hestness, J. and others Netrace: dependency-driven trace-based network-on-chip simulation. In *Proc. of WNOCA* (2010), pp. 31–36.

[13] Kamal, M. and others An Efficient Reliability Simulation Flow for Evaluating the Hot Carrier Injection Effect in CMOS VLSI Circuits. In *ICCD* (2012), pp. 352–357.

[14] Kim, J. and others A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. In *Proc. of ISCA* (2006), pp. 4–15.

[15] Kufluoglu, H. *Mosfet Degradation due to NBTI and HCI and its Implications for Reliability-Aware VLSI Design*. PhD thesis, Purdue University, 2007.

[16] Kundu, P. On-Die Interconnects for Next Generation CMPs. In *Proc. of WOCIN* (2006).

[17] Lorenz, D. and others Aging analysis at gate and macro cell level. In *Proc. of ICCAD* (2010), pp. 77–84.

[18] Mints, M., and Ekendahl, R. *Hardware Verification with C++: A Practitioners Handbook*, vol. 1. Springer, 2006.

[19] Nigam, T. and others Accurate product lifetime predictions based on device-level measurements. In *Proc. of RPS* (2009), pp. 634 –639.

[20] Park, D. and others Exploring Fault-Tolerant Network-on-Chip Architectures. In *Proc. of DSN* (2006), pp. 93–104.

[21] Siddiqua, T., and Gurumurthi, S. Enhancing NBTI Recovery in SRAM Arrays Through Recovery Boosting. *IEEE Trans. on VLSI Systems. 20*, 4 (2012), 616–629.

[22] Takeda, E., and Suzuki, N. An empirical model for device degradation due to hot-carrier injection. *Electron Device Letters 4*, 4 (1983), 111 – 113.

[23] Wang, W. and others Compact Modeling and Simulation of Circuit Reliability for 65-nm CMOS Technology. *IEEE Trans. on Device and Materials Reliability* (2007), 509 –517.

[24] Zhao, W., and Cao, Y. *Predictive Technology Model*. `http://ptm.asu.edu/`.

Supplemental Materials

S1. HCI AGING ANALYZER FRAMEWORK (HAAF)

In this section, we discuss our Aging Analyzer Framework used to evaluate HCI degradation of all gates in a circuit. We first give an overview of pertinent transitions of a gate and then discuss our simulation framework.

S1.1 Pertinent Transitions

HCI affects a transistor during a switching activity. However, for a reliability evaluation of a VLSI circuit consisting of thousands of transistors operating for years (typically 7-10), accurate HCI degradation analysis using HSPICE takes too long. As such, it has been determined by [13] that only certain type of transitions in a logic gate generate interface traps in its transistors. Hence, we only calculate the HCI impact of these transitions, allowing for a practical simulation time. We list the pertinent transitions of the gates we used in our design (INV, NAND, NOR) in Table 2, the transitions indicated in the second column and third column induce HCI degradation for NMOS and PMOS transistors, respectively. We simulate all these transitions and evaluate their HCI aging impact on the logic gates. Only transitions that affect the transistor near the output node are counted as they contribute the most to HCI [13].

GATE	NMOS	PMOS
INV	↑A	↓A
NOR	(↑A,$\overline{B}$) ($\overline{A}$,↑B)	(↓A,B)
NAND	(↑A,B)	(A,↓B) (↓A,B)

Table 2: Pertinent Transitions of Various Gates

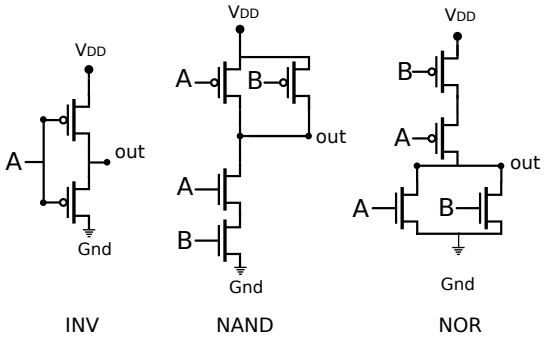


Figure 8: Basic Gates

S1.2 Aging Framework

In the gate level, HCI degradation is manifested during transistor switching. We developed a tool to examine the possible HCI impact on all gates of a circuit through extensive logic analysis. Our HAAF tool works by taking the input in every clock cycle and propagating the logic in a domino fashion until it reaches the output. During the course of this propagation, some gates will switch while others will not. We record all these transitions in all clock cycles and use them to calculate the transition density of the gate. Note that we post-process all the transition events to determine if they are pertinent transitions before calculating the transition densities.

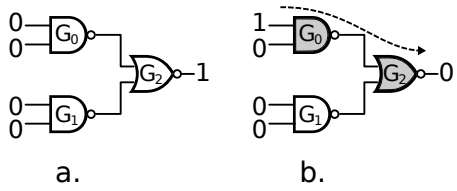


Figure 9: HCI Analysis

Figure 9a shows a detailed example of how this analysis is done on a circuit with three gates indicated as G_0, G_1, G_2, with initial states as shown[4]. Figure 9b shows a new set of inputs being fed and denotes the specific gates that will change (highlighted in gray). G_0 and G_2 changes in this cycle while G_1 does not. We calculate the transition density (TD_g) of a gate g as follows:

$$TD_g = \frac{\sum_{n=1}^{x} S_{gn}}{x} \tag{3}$$

where x is the total number of cycles simulated and $S_{gn} = 1$ if gate g made a pertinent transition at cycle n (0 otherwise). We then use the transition density to calculate the new V_{th} using our model in Equation 1. A new propagation delay t_g is then obtained for each gate g using HSPICE simulation. Note that we simulate t_g for all gates and not just the ones in the critical path because the critical path can change depending on the extent of degradation in different parts of the circuit. Finally, we calculate the new propagation delay (T_P) of the whole circuit as:

$$T_P = max(X_0, X_1, ..., X_Y) \tag{4}$$

$$X_y = \sum_{g=1}^{H_y} t_g \tag{5}$$

where Y is the set of all paths in the circuit and X_y is the total propagation delay of path y. H_y is the set of all gates in path y.

The process discussed above forms the bulk of our evaluation framework and although it is very computationally intensive, its thoroughness allows us to accurately evaluate the benefits of our architectural techniques at a circuit-level accuracy.

S2. CIRCUIT IMPLEMENTATION OF BC

We discuss the implementation and overhead of the Bit Cruiser circuit as a continuation of the discussion in Section 4.1.

Figure 10 shows the Bit Cruiser circuit that is responsible for cruising the bits around the channel. Bit Cruising can be implemented at different granularities. However, in this work we use a granularity of four (i.e. the whole channel is segmented into four equal parts) because the most lower quarter of the channel bits have the most activity based on our input traces. The Bit Cruiser circuit takes in as inputs

[4]initial states are a result of a previous execution

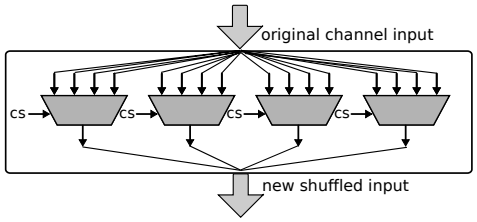


Figure 10: Bit Cruiser Circuit

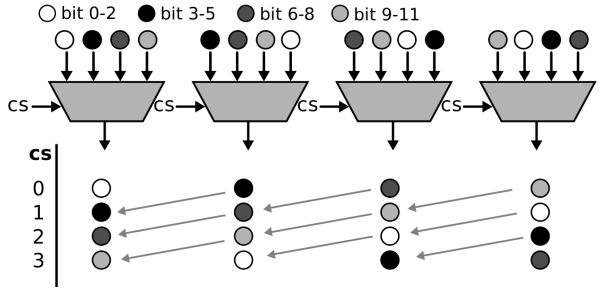


Figure 11: Time Lapse Example of a Bit Cruiser Circuit for a 12-bit channel. Signal cs is the cruise setting.

the *channel bits* and a 2-bit *cruise setting*. The *cruise setting* is then used as an input to a 4-to-1 multiplexer in order to reshuffle the bits as desired.

We show in Figure 11 an example of the effect of bit cruising on channel bits. The *cs* signal in the figure represents the cruise setting. In this example, we assume that in each clock cycle, *cs* is increased by one[5]. The shaded circles are used to indicate one segment of a channel. In the figure, when *cs* is equal to zero, the BC circuit output is the same as the channel input (i.e. or when there is no BC circuit at all). When *cs*=1 the lowermost segment is transferred to the uppermost and the second lowermost segment is shifted to the former's place (direction indicated by an arrow). All other segments follow in unison.

S2.1 Overhead of BC

The circuit in Figure 10 will be placed in the Network Interface right before sending the flit to the router of the source node. Since the bits being sent through the network are now jumbled, the router front end must be able to appropriately identify the header flit bits in order to route the circuit correctly. To this end, we introduce a Routing Information Extraction (RIE) circuit. The RIE circuit extracts the appropriate bits from the shuffled channel bits and places it in a Routing Information Register (RIR), which will be accessed by the Routing Calculation module in the succeeding pipeline stages.

Figure 12 shows the implementation of the RIE circuit. Every time a flit arrives and is about to be written in the virtual channel, the RIE circuit (using the cruise setting information) will determine if the flit is a head flit. If it is, the routing information is latched into the RIR. The RC module in the next pipeline stage will then use the contents of the RIR to route the flit in the corresponding VC. Since there is only going to be one packet in each virtual channel, the overhead for the RIR is minimal. We next calculate its overhead in a typical modern NoC configuration.

In modern NoCs, the network width is often large as on-chip wiring is abundant and bandwidth is also important.

[5]Note that the cruise setting can also be altered for larger time granularities.

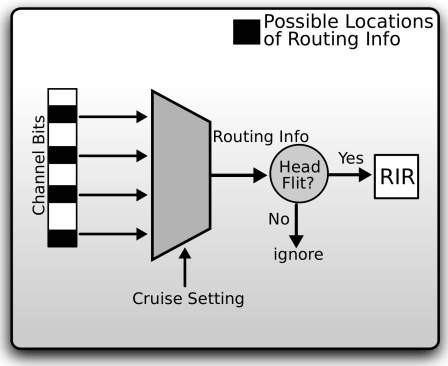


Figure 12: Routing Information Extraction Circuit

For a 128-bit flit width in an 8×8 network with an algorithmic routing algorithm that uses the number of hops in the x and y directions, the RIR will only require 3 bits in each direction for a total of 6 bits. If there are 5 flits in each buffer, then the overhead is 0.9%. For deeper flit buffers, the overhead further goes down.

S3. NOC AND CROSSBAR CIRCUIT PRIMER

In a network on chip, the crossbar circuit acts as the heart of communication of all inputs and outputs in the router. Hence, it's reliability is of extreme importance in the functioning of the whole NoC. Moreover, the critical latency delay of a crossbar circuit dictates the clock frequency of the whole NoC [16]. Thus, any minor deviations in its critical path could corrupt the transmitted data and will lead to total failure in the router and the NoC itself. This makes the crossbar circuit a good candidate as a case study for our HCI mitigation techniques.

Packets in NoCs are sent from one node to another by hopping through intermediate nodes. Figure 13 shows a 16-node mesh where node 0 sends a packet to node 8. Before reaching node 8, the packet will traverse through nodes 1,4 and 7. The router in each node is responsible for the correct transmission of a packet to its adjacent nodes.

Figure 14 shows a simplified model of a typical router. The end goal of a router is to transfer flits from an input port to a specified output port. In each cycle, the router will receive multiple requests to route, an allocator circuit will then determine an optimal connection of input and output ports. In the succeeding cycle, the allocator signals the crossbar and the flits traverse the switch, and are sent to the next adjacent router. This process continues in each node until the flit reaches the destination node.

As the crossbar is at the focal point of packet transmission in an NoC, we choose the crossbar circuit as a case study for our HCI mitigation techniques, our techniques could be easily adapted to work on other parts of an NoC.

S4. DISTRIBUTION CYCLE MODE (DCM)

The Distribution Cycle Mode technique utilizes idle cycles to balance out the HCI degradation of transistors. Most real-world applications spend a considerable time waiting for information from the NoC. Moreover, cache coherence requests are self-throttling or that succeeding requests are not sent unless a reply is received [12]. As such the crossbar spends most time (average of 85% in our setup) sending no data through the crossbar. This presents us with tremen-

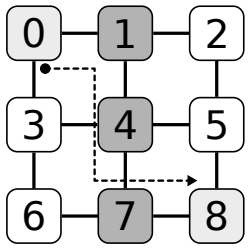


Figure 13: 3x3 Mesh NoC. Node 0 sending a packet to node 8.

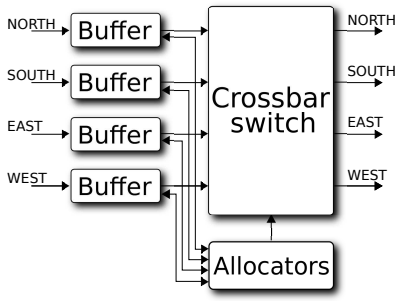


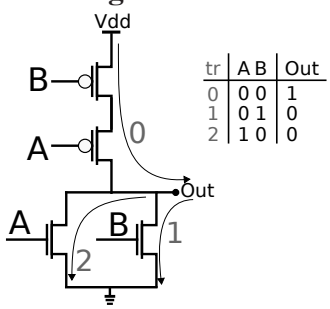
Figure 14: Simple NoC Router

dous opportunities in using reliability techniques without incurring a latency overhead. Our second technique, DCM, introduces a new mode of operation for the crossbar circuit during idle phases of execution. In DC mode, the crossbar uses optimized inputs to achieve a near-uniform switching activity across the data channel.

To explain the idea behind DCM, we introduce a simple example of a 2-bit NOR-gate in DC mode. Figure 15 shows a 2-bit NOR gate in standard CMOS executing optimized inputs. For each set of inputs, the transition number is indicated along with an arrow in the circuit that indicates the corresponding path that the output signal has taken. For instance, transition 0: inputs A=0 and B=0, switches both the PMOS transistors while transition 1 (A=0, B=1) only switches NMOS B. The key to successfully implementing the DC Mode is to balance out the number of switching transitions that each transistor makes.

In this example, executing transitions 0-2 once gives each transistor a balanced switching count. Note that there are four possible inputs for the NOR gate but we only need three in order to get a balanced HCI degradation. Being able to accurately determine the needed inputs, rather than exhaustively using all possible, is the key to optimal DCM operation.

S4.1 Implementing DCM in the Crossbar



tr	A B	Out
0	0 0	1
1	0 1	0
2	1 0	0

Figure 15: Two-bit NOR gate showing the different transitions with respect to inputs.

Applying DCM to a big circuit such as a router crossbar poses some major challenges because optimal HCI degradation is only achieved when the inputs are carefully constructed to balance the switching activity. However, despite the enormity of the crossbar, its regular structure allows us to analyze a small subset of the circuit and use our results to optimize the whole component.

There are three key requirements to seamlessly applying DCM while maintaining the correct and unobstructed execution of the NoC Router. We outline them here and discuss each one in detail. They are:

1. **Idle time identification** - To engage the crossbar in the Distributed Cycle Mode, idle cycles must be correctly identified or else the correct value that is supposed to be transferred during the *switch traversal* stage of an NoC is going to be overwritten. This overwriting can corrupt a running program.
2. **Identification of optimal inputs** - The optimal inputs to the crossbar circuit are derived using an offline analysis similar to the one discussed in the previous subsection. This is a one time effort that can be used throughout the lifetime of the NoC router.
3. **Feeding mechanism of customized inputs** - The crossbar must have an option of using the inputs provided by the analysis above in order distribute HCI aging in all of its transistors.

In a typical router in an NoC, the crossbar switch has multiple lanes to handle simultaneous demands of multiple inputs to multiple outputs (NORTH, SOUTH, EAST, WEST). As such, when no input port is scheduled to transfer a data to a specific output port in a particular time, that output port (or lane) is considered idle. Thus, correctly identifying the idle cycles of a crossbar depends mostly on the output of the scheduling algorithm of the switch.

The main mechanism to identify idle cycles is already present in any Switch Allocator (SA) implementation as it outputs a schedule of the switches every clock cycle. Figure 16 shows an NoC router along with the supplementary logic and components to identify idle cycles and implement DCM. Aside from the main DCM module that serves as the control unit for DCM operation, a lookup table and an additional multiplexer is added for the purpose of storing the optimized values and to have the ability to load them when desired, respectively.

In each clock cycle, the SA takes in as input the requests of different virtual channels and input ports and gives the permission to specific input ports to use the output ports in the next cycle. If there are no contention of requests, all requests could be permitted to traverse in the crossbar the next cycle. However, if there is, it is resolved based on a scheduling priority. In other cases though, there simply are not enough requests to keep the switch/crossbar fully utilized. When this happens, our DCM module immediately senses this and queries the lookup table and instructs the multiplexer to load an HCI aging-optimized value in the subsequent cycle.

S5. CROSSBAR LANE SWITCHING

In this section, we elaborate in more details the implementation of the Crossbar Lane Switching scheme discussed in Section 4.3. We first discuss the baseline implementation

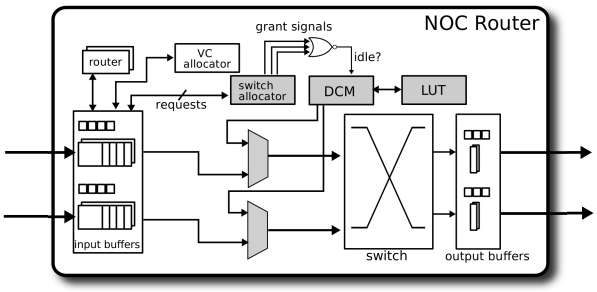


Figure 16: Modified NoC Router to Accommodate DCM operation.

of a modern NoC Router and then explain our modifications in order to implement CLS.

Figure 17 shows a logical diagram for a traditional Virtual Channel (VC) flow NoC Router with two input ports and two virtual channels per input port. The virtual channels are used to handle multiple concurrent streams per input port, each waiting for its turn to use the crossbar switch, hence improving the overall bandwidth of the network. In our example, the north input port can only utilize VCs 1 and 2, while the south utilizes 3 and 4. In each clock cycle, all VCs request usage of the crossbar for the succeeding cycle. The switch allocator will then determine a winner and subsequently connect the virtual channel to the desired output buffers.

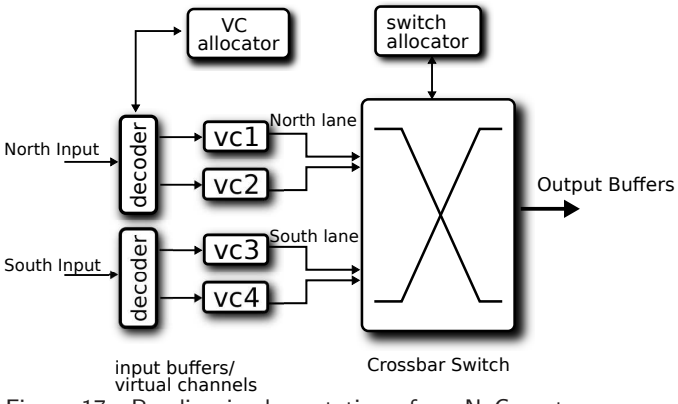


Figure 17: Baseline implementation of an NoC router showing the virtual channels, input ports and the crossbar switch. Output Ports and Output Virtual Channels are not shown.

As we have discussed in Section 4.3, the lanes of the crossbar can undergo uneven degradation when certain input and output pairs are used more. CLS aims to balance this degradation by evenly distributing the paths taken by a flit. Figure 18 shows the necessary modifications on the NoC Router to be able to implement CLS. Also, the VC allocator must be able to assign any incoming flit to any virtual channel (additional lines in the decoder)[6]. As the virtual channels are implemented as SRAM arrays [SR1] similar to a register file in a processor, there will be no additional logic needed to access the different virtual channels. The only extra logic needed will be for the VC allocator to distribute

[6]Note that there are many possible implementations of the Input Buffers. Our overhead is analyzed with respect to an open-source RTL implementation of a modern NoC router [SR1].

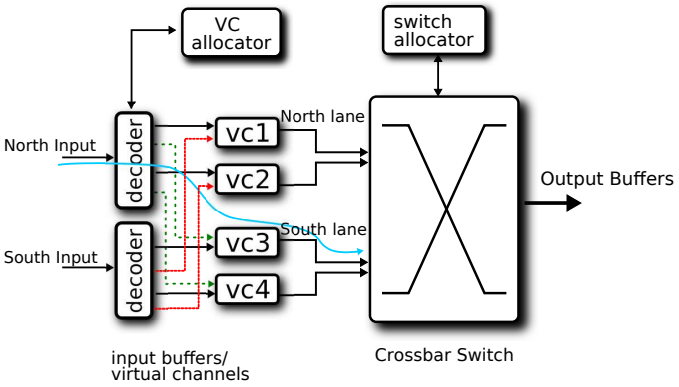


Figure 18: CLS Implementation. VC Allocator Can Assign Incoming Flits to Any Virtual Channel.

the flits across the many virtual channels which can be accomplished by a simple counter circuit which is added to the offset of the decoding stage. The Route Calculation (RC) stage will automatically determine the route of the flit since the routing information is stored in the head flit. The RC will then send the SA the appropriate commands, preserving the correctness of the flit and its route.

The light blue line in Fig. 18 shows the path taken for a flit arriving at the North input and traversing the South lane of the crossbar. This is made possible by storing the flit in virtual channels 3 or 4 and then informing the SA to use the same channel as input to the crossbar. In summary, an incoming flit uses the same input port, a different virtual channel and crossbar lane, and the same output port.

S6. REFERENCES

[SR1] Daniel Becker and William Dally Open Source NoC Router RTL *https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router.*