Power, Interconnect, and Reliability Techniques for Large Scale Integrated Circuits

by

David A. Fick

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor David T. Blaauw, Co-Chair
Professor Dennis M. Sylvester, Co-Chair
Professor Trevor N. Mudge
Assistant Professor James W. Cutler
Assistant Professor Thomas F. Wenisch

I dedicate this dissertation to my parents.

Their love, support, and encouragement made this possible.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

CHAPTER 1

Introduction

This introduction briefly discusses large scale computing and why it is an important research topic. It shows how some of the major research problems came to be and how they are related, and highlights efforts in this area related to this dissertation.

1.1 Large Scale Computing

The first "computer", as we understand it today, was the ENIAC, a general-purpose electronic machine developed by The University of Pennsylvania in 1946. ENIAC was constructed for the United States Army during World War II, for the calculation of artillery firing tables, which require projectile simulations including factors such as distance to target, angle of elevation, and atmospheric conditions. At 8 by 3 by 100 feet in size, and 30 tons, it was a room sized machine. It was also expensive to operate: it drew 150 kW and required nearly daily replacement of components. To reduce failures and keep uptime high it was never shut off, which reduced thermal stress. It operated continuously for nearly a decade before being shut down. Today, a small portion of it remains on display at The University of Michigan in the Computer Science and Engineering building atrium. [1]

ENIAC serves as an example of the forefront of computing: big machines solving big problems. Development of large machines has been ongoing, with the current largest machines being the 16.32 petaflop "Sequoia" built by IBM and the 8.162 petaflop "K computer", built by Fujitsu in Japan [2]. While ENIAC used discrete vacuum tubes, diodes, relays, resistors,

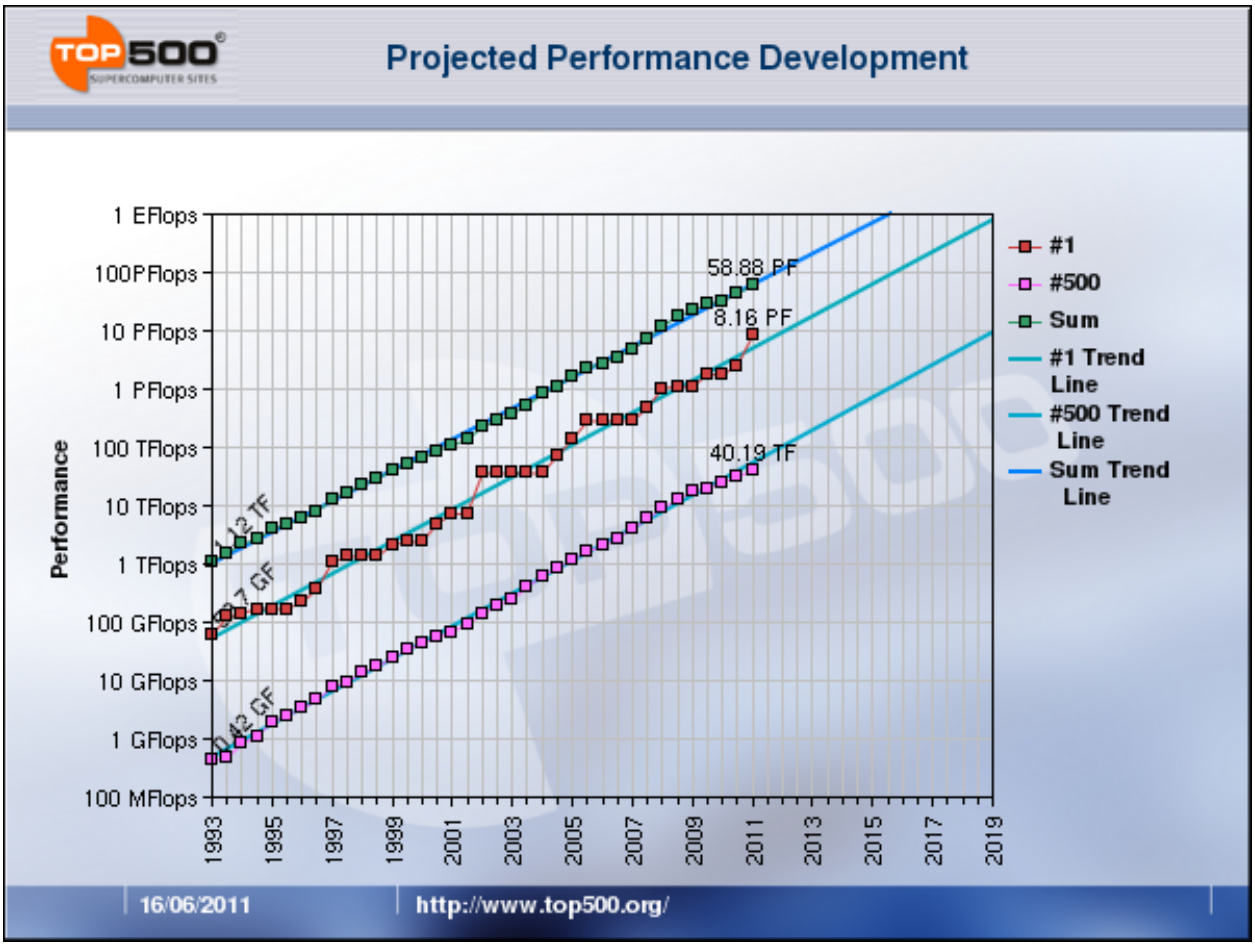


Figure 1.1: **Top 500 super computer performance projections.** Historic performance data of the top 500 performing non-distributed computers in the world. "#1" represents the fastest machine, "#500" represents the 500th fastest machine, and "Sum" represents the aggregate performance of the top 500 machines. Trend lines are also drawn showing steady growth, with an exaflop computer projected to exist before 2020. [2]

and capacitors, the K computer uses 68,544 SPARC64 VIIIfx processors, each containing eight 2.0 GHz cores [3, 4]. At 9.89 megawatts, the K computer also draws much more power than the ENIAC.

The growth for the largest scale machines has been steady for decades. Figure 1.1 shows historic data for the top 500 machines in the world, showing the trends for the fastest machine, the 500th fastest machine, and the aggregate performance of the top 500 fastest machines. It is projected that an exaflop computer will exist by 2020.

Another class of large scale computing is the datacenter. Datacenters have similar statistics - tens of thousands of multi-core processors - but their organization is designed to reduce

cost by using many separate machines in a distributed fashion. This allows for simpler construction, maintenance, and expansion, but the kinds problems that datacenters run must have a coarser form of parallelism. Indeed, in many cases the problems can be considered "embarrassingly parallel", a multi-part workload where each part can be computed independently [5].

Exceedingly popular, datacenters were estimated to use 1.5 percent of the total electricity in the United States in 2006, and are predicted to use double that currently [6]. Some of the organizations building these datacenters are investment firms for automated trading, mining companies for analyzing geological data, governments for simulating weather, and engineering firms for running precise simulations such as molecular interactions or vehicle collisions. While smaller personal computing devices such as smartphones have become popular, datacenters are providing the computational backbone remotely via web applications by companies such as Google, Facebook, and Apple.

1.2 VLSI Process Scaling

The growth of large scale computing has been due to advances of technology on many fronts, but one of the most visible is VLSI process scaling. Gordon Moore famously predicted that the number of transistors on a single die would double every two years with fixed cost [10]. As shown in Figure 1.2, Intel (and other integrated circuit fabricators) has had little trouble abiding Moore's law, and it later became a self-fulfilling prophecy since it guided process development timelines.

Exponential growth is a powerful thing: ENIAC had 17,468 vacuum tubes [1], while a single Intel Xeon E7-8870 has 2.6 billion transistors [8]. In the floor space of the ENIAC, one could fit 8,400 Intel Xeon E7-8870 processors, for a total of 21.8 trillion transistors. If Moore's law continues, in 26 years this would fit in a single die. Without considering anything else, the size of computers built today and the size of problems that they can solve much greater today than even a few years ago, and that growth shows no signs of slowing.

Number of transistors is different from *transistor performance* or *system performance*. Table 1.1 shows how CMOS devices perform under constant field and constant voltage style

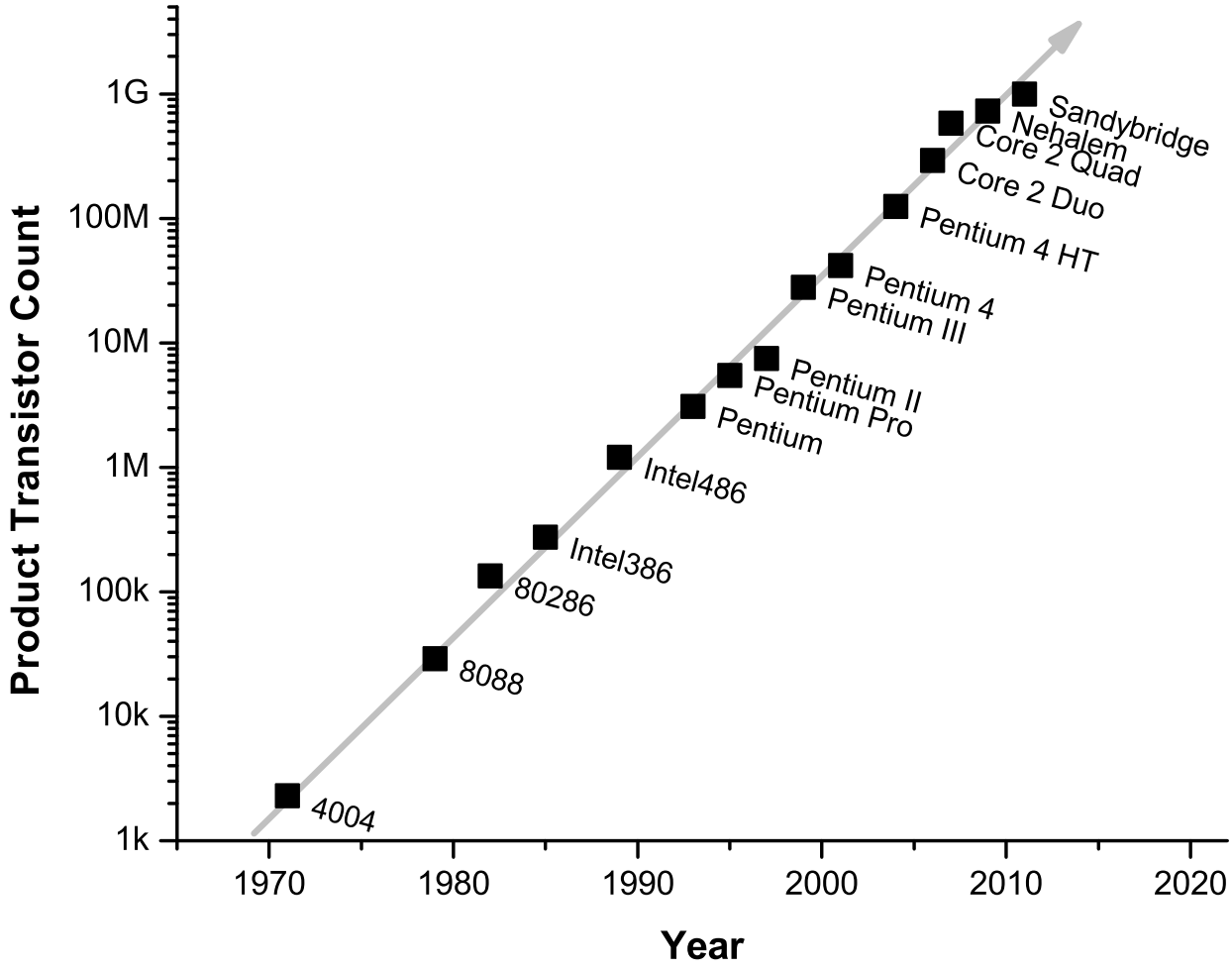


Figure 1.2: **Growth of transistor count for Intel consumer processors.** Transistors are for a single die, die size may not be the same for each. [7–9]

process scaling. Ideally, silicon manufacturers would like to follow constant field scaling, which improves performance but maintains the same power density. Maintaining the same power density is important: since the Pentium 4 was released in 2000, processors have reached the maximum power (per chip) that can be supported given heat dissipation (*e.g.*, 125W). This is called the thermal design power (TDP). Perfect constant field scaling has not been achievable, however, due to increased leakage caused by reducing V_t, increased V_t variation, and V_t reduction via drain induced barrier lowering (DIBL). This results in an even greater power density, which designers are required to offset this in other ways. This can include lowering frequency, increasing the amount of memories relative to active logic, or decreasing the die size. For instance, Intel's latest processor generations, Nehalem (45nm)

Table 1.1: **Influence of scaling on MOS device characteristics.** Due to velocity saturation, device lifetime, and power density limitations, semiconductor manufacturers currently follow constant field scaling as best they can. [11]

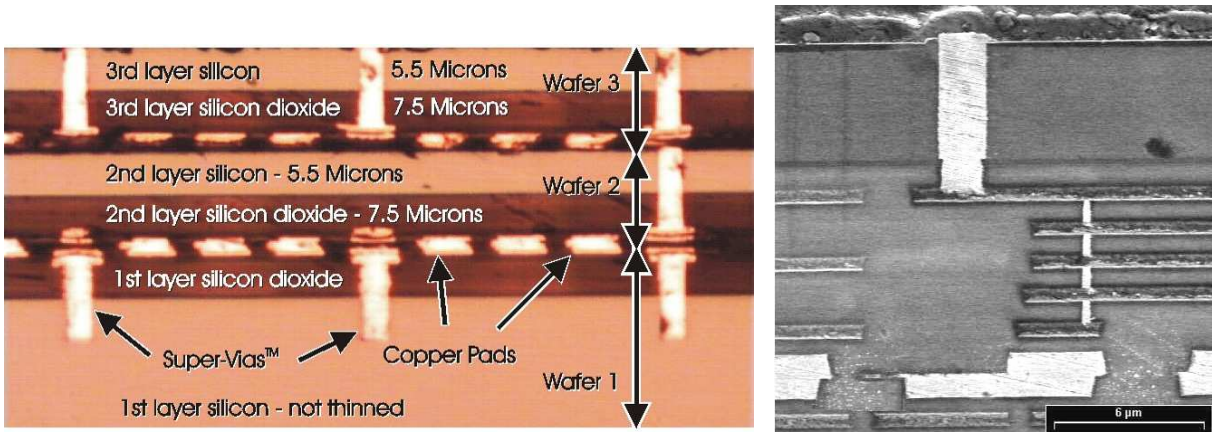
Parameter	Sensitivity	Constant Field	Constant Voltage
Scaling Parameters			
Length: L		$1/S$	$1/S$
Width: W		$1/S$	$1/S$
Gate oxide thickness: t_{ox}		$1/S$	$1/S$
Supply voltage: V_{DD}		$1/S$	1
Threshold voltage: V_{tn}, V_{tp}		$1/S$	1
Substrate doping: N_A		S	S
Device Characteristics			
β	$(W/L)(1/t_{ox})$	S	S
Current: I_{ds}	$\beta(V_{DD} - V_t)^2$	$1/S$	S
Resistance: R	V_{DD}/I_{DS}	1	$1/S$
Gate capacitance: C	WL/t_{ox}	$1/S$	$1/S$
Gate delay: τ	RC	$1/S$	$1/S^2$
Clock frequency: f	$1/\tau$	S	S^2
Switching energy: E	CV_{DD}^2	$1/S^3$	$1/S$
Switching power dissipation (per gate): P	Ef	$1/S^2$	S
Area (per gate): A		$1/S^2$	$1/S^2$
Switching power density	P/A	1	S^3
Switching current density	I_{ds}/A	S	S^3

and Sandybridge (32nm), show a substantial die shrink, $263mm^2$ to $216mm^2$ [8], which can also be seen in terms of transistor count in Figure 1.2.

System performance also depends on interconnect performance. Table 1.2 shows how interconnect RC delay scales for local and global connections each generation. Local connections benefit from improved driver strength while the wire RC delay remains unchanged since it is scaled in all dimensions. This improvement is still only proportional to $\sqrt{S}$, however, while device performance is proportional to S resulting in relatively slower local interconnect each generation. Global interconnections span the length of the entire chip, which does not does not scale with the process generations. Due to the R^2 increase in resistance, these global interconnections see a $\sqrt{S}$ *increase* in delay each generation, versus a S delay *decrease* for devices [12]. Additionally, since global interconnections scale only in width and not length, the relative number of available interconnections scales with $1/S$, while the number of available devices scales with $1/S^2$, resulting in an apparent reduction in available connectivity. To address this, processing core size is now scaled down with each generation so that the core itself does not have any global interconnections. For example, the Xeon E7-8870 mentioned

Table 1.2: **Influence of scaling on interconnect characteristics.** RC delay scaling for local and global interconnect is shown. While local wires improve in performance each generation, global interconnections become slower. [11]

Parameter	**Sensitivity**	**Scale Factor**
Scaling Parameters		
Width: W		$1/S$
Spacing: s		$1/S$
Thickness: t		$1/S$
Interlayer oxide height: h		$1/S$
Die Size		D_c
Characteristics per Unit Length		
Wire resistance per unit length: R_w	$1/wt$	S^2
Fringing capacitance per unit length: C_{wf}	t/s	1
Parallel plate capacitance per unit length: C_{wp}	w/h	1
Total wire capacitance per unit length: C_w	$C_{wf} + C_{wp}$	1
Unrepeated RC constant per unit length: t_{wu}	$R_w C_w$	S^2
Repeated wire RC delay per unit length: t_{wr} (assuming constant field scaling)	$\sqrt{RCR_wC_w}$	$\sqrt{S}$
Crosstalk noise	w/h	1
Energy per bit per unit length: E_w	$C_w V_{DD}^2$	$1/S^2$
Local/Semiglobal Interconnect Characteristics		
Length: l		$1/S$
Unrepeated wire RC delay	$l^2 t_{wu}$	1
Repeated wire delay	$l t_{wr}$	$\sqrt{1/S}$
Energy per bit	$l E_w$	$1/S^3$
Global Interconnect Characteristics		
Length: l		D_c
Unrepeated wire RC delay	$l^2 t_{wu}$	$S^2 D_c^2$
Repeated wire delay	$l t_{wr}$	$D_c\sqrt{S}$
Energy per bit	$l E_w$	D_c/S^2



(a) A Three-Wafer Stack (b) A Two Wafer Stack in Finer Detail

Figure 1.3: **(a)** A three-layer stack with face-to-face (wafers 1-2) and face-to-back (wafers 2-3) bonds. **(b)** A face-to-face bond shown in finer detail, with an exposed TSV at the top. [13]

above has 10 cores, each using 1/20th of the die area. With such a large number of cores, providing enough bandwidth to off-chip memory and between cores becomes problematic.

Process variation also increases with process scaling. Variation in effects such as line edge roughness, oxide thickness, and random dopant fluctuations do not scale directly with the feature length, causing a relative increase. This causes designers to increase margins for timing and lifetime, resulting in an increase in power and/or a decrease in performance. Some of this margin can be recovered using equipment to test the performance of each chip and "bin" the chip for either power or performance. With chips having 10s of cores presently, and possibly 100s in the future, testing each core becomes expensive.

1.3 3D Integrated Circuits

Three dimensional integrated circuits (3D-ICs) is an up and coming method of addressing the issues of interconnect scaling. 3D-ICs have multiple layers of silicon with vertical interconnect between them, typically in the form of through-silicon vias (TSVs). An example of a 3D-IC process is shown in Figure 1.3, which includes a picture of a Face-to-Face (F2F) connection, and a Face-to-Back (F2B) connection. The face side of a circuit is the side with devices, while the back is the side with bulk silicon. Since global interconnect can

be millimeters long, and silicon layers tend to be only tens of microns thick in 3D stacked processes, the power and performance gains by using vertical interconnect can be substantial.

At first, 3D-ICs are expected to be used to stack memories above cores to create a larger, closer memory, but eventually they could be used to create larger cores than the ones created today. Multiple logic layers stacked above each other would require addressing power density and hotspot issues. Although the TDP of a chip might be low, the highest temperature in the circuit would increase in this case. Thermal hotspots reduce circuit performance and lifetime, and can even cause mechanical stress that would damage the chip or package. One way to address this would be to include large thermally conductive cooling columns in the chip, although more novel approaches such as using integrated fluid-based microchannel cooling are being studied [14].

3D stacking also allows the ability to mix different process technologies (*e.g.*, digital CMOS, analog CMOS, DRAM, Flash, and optoelectronics) within the same die, which can improve performance, power, connectivity, and form factor. For instance, DRAM has conflicting process requirements for the bit cell access transistors and for the control logic and sense amps. By separating these structures into two different processes, DRAM performance can be improved. Meanwhile, the buses connecting to off-chip DRAM are typically very narrow (64-bit), low frequency (400 MHz), and high power. A semiconductor company, Tezzaron, was able to design a 3D-stacked DRAM which addresses these issues: it uses separate processes for the bitcell and controller layers and it uses a 1024-bit, 2 GHz bus for communication [15]. The per-bit power requirements were also much lower since each interface pin induces only a 125 fF load, instead of a multiple pico-farad load of a DRAM trace pin. A study from Georgia Tech analyzed Tezzaron's DRAM in the context of a multi-core processor, and found that it created a 75% performance improvement, with roughly equal contributions from the bus width, bus frequency, and DRAM speed [16]. A similar study from The University of Michigan showed that L2 cache becomes unnecessary due to the low-latency of stacked DRAM, improving computational area utilization and reducing power even further [17].

Another major benefit from 3D stacking is increased yield through "known good die" techniques, in which each layer of an IC is tested individually before integration [18]. The 3D

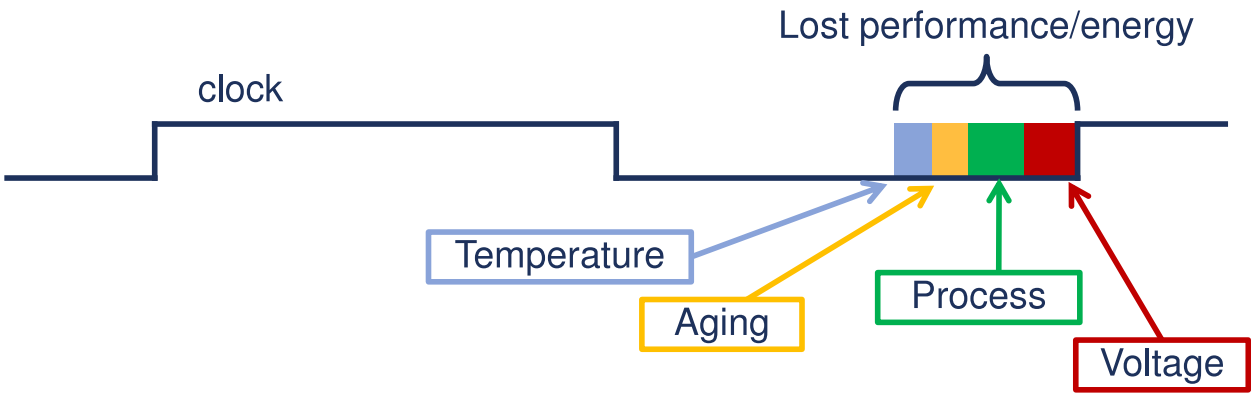


Figure 1.4: **Sources of timing margins in integrated circuits.** Timing margins reduce energy efficiency and system performance.

stacking process itself causes a small loss of yield, however, the total yield is still dominated by the total area of the die, of which yield has an inverse-exponential relationship [11]. By splitting an IC into multiple parts and testing those parts individually, an IC manufacturer could affordably build much larger chips than previously possible. Testing a die before it is bonded is difficult and expensive, so some efforts are in progress to obtain information with as few pins of possible, making a trade-off between accuracy and cost of testing, and cost of manufacturing and yield.

1.4 Timing Speculation

Timing speculation is a method of reducing timing margins, which reduce energy efficiency and system performance. Timing margins come from uncertainty or variation in process, voltage, and temperature (PVT) as well as data and lifetime (Figure 1.4). With process scaling each of these factors get a bit worse each generation. As mentioned in Section 1.2, the variation in process parameters does not scale directly with the parameters themselves, causing an increase in relative variation. As V_{DD} decreases, noise from outside sources increases relatively, and the relative effectiveness of decoupling capacitance decreases due to increased current draw to maintain the same TDP.

Increases in power density also cause increases in voltage and temperature uncertainty, even if the TDP is the same. In advanced processes it is possible for a relatively small

structure to suddenly draw a large amount of current, which causes IR droop, Ldi/dt noise, and thermal hotspots. The event can be data driven, such as the case for "power viruses" run on processors, making the variation even greater since the designer often does not have control over the programs that a customer runs. An event can have complicated and cascading effects. For instance, if a driver to one half of a clock tree suddenly saw a voltage droop, then the two halves of the tree would be skewed for one or more cycles. In this case a purely local effect can have catastrophic global consequences.

Two ways of reducing timing margins is through the use of "Razor" circuits, or through canaries. The modern-day understanding of Razor was first proposed by researchers at The University of Michigan in 2003 [19], although similar, less developed ideas were proposed as far back as 1993 [20]. Canary circuits were proposed by a number of researchers much earlier.

Razor is a "detect and correct" style strategy. The system operates at voltages and frequencies that may cause a timing failure, and a backup copy of any at-risk data is kept in case of emergency. If a timing failure occurs, then the backup copy of the data is restored, and operation continues. The data backup occurs primarily at two levels: circuit level and architectural level. The original Razor design used circuit-level data backup [19], where each flip flop had a "shadow latch" that stored known good data. Each cycle the shadow latch would need to be updated, meaning that a global rollback signal would need to be computed and distributed before the next clock cycle. It is common to have the following few clock cycles lengthened to give any undesirable electrical conditions time to pass; this also gives extra time to distribute the rollback signal. Forward progress is always made with this scheme since the program only stalls one cycle while the rollback occurs and computation continues at a guaranteed frequency.

RazorII introduced the now-prevalent architectural rollback which takes advantage of existing rollback mechanisms in modern processors [21]. Typically processors have a built-in rollback mechanism for branch mispredicts which can be leveraged. Care must be taken to make sure that no architectural state is ever corrupted; this includes the register file, program counter, and any external memories. The rollback mechanism itself must also be considered reliable so that the program can be properly restarted. Forward progress can be

stalled much longer in this technique since the rollback mechanism will undo multiple cycles of work. The lack of shadow latches at each critical flip-flop and the lack of a need for a global control signal makes this technique very popular.

Both forms of Razor require some sort of timing violation detector at each critical flip-flop. This can be a transition detector, which raises a flag if a transition comes after the end of the clock cycle, or it can be a comparison between the D and Q ports of a flip-flop at some fixed time after the end of the clock period. In either case, the input to a flip flop is expected to be stable for some time after the beginning of the clock cycle (additionally, before the flip-flop's setup time as well). This time period is called the "speculation window", which sets a new hold-time constraint on the design. If a flip-flop is being monitored for late transitions, then any "short paths" must be buffered so that transitions do not occur during the speculation window. These buffers cause additional power draw, but if not included correctly, the system will have false timing violations almost every cycle, potentially preventing the system from progressing. A wider speculation window causes additional power draw, but it must be wide enough to handle any possible timing violation, and calculating the worst case scenario may be difficult. Ultimately, this additional power is offset by the reduction in supply voltage by allowing violations to occur.

Canary circuits, or "replica paths", use a string of inverters or other gates to create a known timing path that triggers every cycle. This path is used to adjust the clock frequency or system voltage such that there is never any timing failures in the main circuit. This strategy can account for changing global conditions, such as global temperature, global process variation, or global voltage scaling. It cannot respond to local variation, local hotspots, or local supply noise, so a margin must still be included for this class of variation.

Large scale systems may have dozens of cores each with their own critical paths and variation. When IC manufactures fabricate a large scale system, they often choose to test and bin each core individually, or test the system as a whole. Razor and canaries provide a third option, where the cores speed-test themselves and adjust their own performance online. This means that the manufacturer only needs to check functionality, which reduces cost.

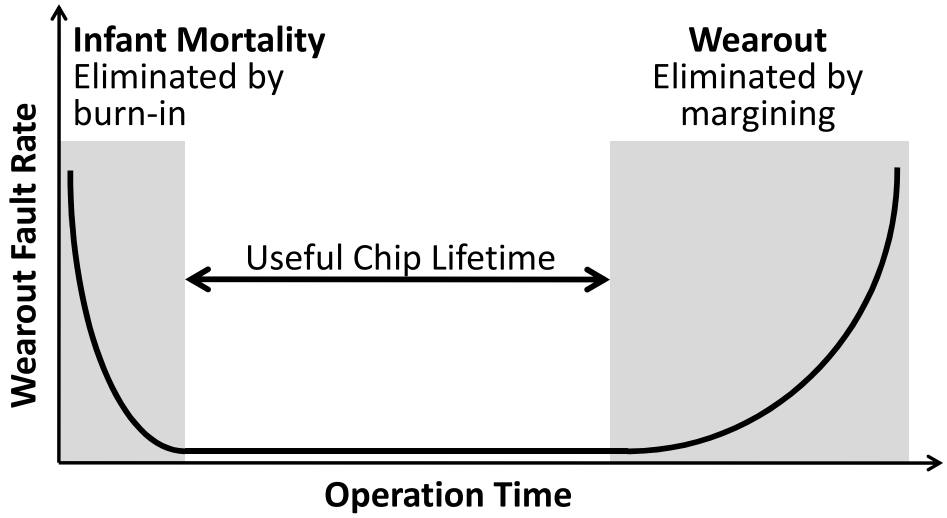


Figure 1.5: **The "bathtub curve" of device lifetime.**

1.5 Reliability

Integrated circuits are not infinitely robust. For instance, we cannot successfully "overclock" a processor by setting V_{DD} to 10V without quickly destroying it. While normal operation is not that extreme, integrated circuit designers must understand the lifetimes and sensitivities of the devices that they are using, so that the system remains functional for its advertised lifetime.

Device failures include hard breakdown and soft breakdown. Hard breakdown results in complete functional failure of a device. Some of these include oxide breakdown, where holes are punched in the oxide by leakage current passing through the oxide, and electromigration, where electrical current pulls metal atoms downstream until the wire is severed or it bulges causing a short. Soft breakdown causes devices to be slower, which can cause systematic timing failures, or functional failures in the case of structures like sense amplifiers. Soft-breakdown failures include hot carrier effects, where electrons get injected and stuck inside the gate oxide, and negative bias temperature instability, where dangling bonds form underneath the gate oxide. In both cases the threshold voltage of the device is adversely affected. [11]

Figure 1.5 shows the "bathtub curve" of device lifetime. Devices that are exceptionally weak due to local process variation are eliminated during the "burn-in" process, a technique

IC manufactures use that activates devices at increased voltage and temperature in order to stimulate faster IC aging. After burn-in the system enters its useful operating lifetime, where no failures are expected. Later, normally robust devices start failing due to the wearout mechanisms mentioned above.

Under normal operating conditions, the vast majority of devices in a system are expected to last many times longer than system's lifetime. Since a single device or wire failure can cause complete system failure, there is an inverse-exponential relationship between system size and lifetime. At the same time, process scaling is making devices smaller and less robust, while providing exponentially more of them. Together, these trends cause the lifetime margins to reduce the useful chip lifetime until it is nearly non-existent.

Due to the strong relationship of lifetime with temperature and voltage, the designer can reduce voltage and frequency to reduce wearout and extend lifetime. Eventually this lifetime margin will cause an unacceptable performance loss. Researchers from The University of Michigan proposed a way to recover some of this performance loss: by spending the majority of time in low-power modes, the lifetime is extended, and performance can be recovered by intelligently boosting performance when it is needed, beyond what could be sustained in continuous operation [22]. This style of techniques requires tracking operating conditions of the device over its lifetime and saving the history off-chip, or a "silicon odometer" which can indicate the wearout sustained during operation without requiring the use of an off-chip memory [23].

Another strategy is to allow portions of the system to fail instead of requiring that the entire system be functional to continue operation. While shutting off a faulty core can be relatively simple, determining when or if to shut off a core can be difficult. A built in self test (BIST) or other mechanism must be included to make this determination. Knowing when to run the BIST is also an issue; since we do not know the answer to the problem being computed, it may not be possible to know if the answer was computed incorrectly. One strategy would be to have a reliable, low power checker looking over the shoulder of the processor [24]. This can be done since checking if a computation was done correctly is typically an easier problem than computing the answer itself. If the BIST or checker has a finer granularity than entire cores, then functional parts of failed cores can be combined to

create a new, lower performance functional core [25].

1.6 Contributions of This Work

This dissertation attempts to address some of the major issues mentioned in this chapter. The remainder of the paper is outlined below.

Chapter 2 discusses Centip3De, a large scale 3D system, its fabrication, and how it addresses issues related to 3D design. Centip3De includes a large number of cores that operate near the threshold voltage of the process. This improves reliability and keeps the system within a reasonable power envelope for a 3D stacked environment. It includes 3D-stacked DRAM without L2 caches, and highlights the benefits of this approach given the low-voltage nature of the processors. Centip3De includes boosting mechanisms to address single-threaded performance requirements, providing a range of operating points between single threaded performance and energy efficiency.

Chapter 3 discusses "Safety Razor", a hybrid timing speculation technique that attempts to recover some of the margins that canaries could not recover, while avoiding the design complexity introduced by traditional Razor techniques. A novel time-to-digital converter necessary for this technique was fabricated and discussed.

Chapters 4 and 5 discuss interconnect reliability for large scale systems and how to efficiently provide it. In particular, Chapter 4 discusses how network-on-chip can reroute traffic around broken network components, and analyzes an algorithm to safely reconfigure without the aide of an outside processing element. Chapter 5 expands on this work and shows how network routers can be designed with a BIST that analyzes the router's own functionality, and then re-organizes its components to reduce the number of communication links in the network.

Chapter 6 discusses the use of stochastic computing as a method of processing multimedia information. By using neural-inspired processing elements, multimedia information can be efficiently processed. In this work, a stochastic computing enhanced image sensor with *in situ* edge detection and noise filtering is described and analyzed.

Chapter 7 summarizes and concludes the dissertation.

CHAPTER 2

Centip3De: A Cluster-Based NTC Architecture with 64 ARM Cortex-M3 Cores in 3D Stacked 130nm CMOS

2.1 Motivation

Process scaling has resulted in exponential growth of the number of transistors available to designers, with high-performance designs now containing billions of devices per chip [26]. However, with the stagnation of supply voltage scaling in advanced technology nodes, power dissipation has become a limiting factor in high-performance processor design. As a result, designers have moved away from a single, high-complexity, super-scalar processor and instead have opted for multiple, simpler, higher energy efficiency cores. In these system-on-chip (SoC) or chip-multiprocessor (CMP) designs, many components share the same chip. These systems typically include processing cores, memory controllers, video decoders, and other ASICs [27, 28].

Due to the large number of individual components in an SoC or CMP, the interconnect network between components has become critical to these systems. However, global interconnect has not scaled nearly as well as transistor count since global wires scale in only one dimension instead of two, resulting in fewer, high resistance routing tracks.

Three-dimensional (3D) integration seeks to address the global interconnect scaling issue by adding multiple layers of stacked silicon with vertical interconnect between them, typically

in the form of through-silicon vias (TSVs). Since global interconnect can be millimeters long, and silicon layers tend to be only tens of microns thick in 3D stacked processes, the power and delay gains by using vertical interconnect can be substantial, often 30-50% [29].

Additional benefits of using 3D integration include the ability to mix different process technologies (CMOS, bipolar, DRAM, Flash, optoelectronics, *etc.*) within the same die and increased yield through "known good die" techniques, where each layer is tested before integration [18]. Integrated DRAM in particular has shown significant performance improvements [16,17]. Recently, several industrial and academic 3D systems have been demonstrated [30–34].

Heat dissipation is a salient issue with 3D integration. High performance designs reached a maximum practical thermal design power (TDP) years ago. Since then, power density has been increasing *further* due to non-ideal process scaling [35], which is exacerbated by having multiple layers of silicon. In this chapter, we propose using near-threshold computing (NTC) in 3D design to address these issues [36, 37]. We show how NTC has a unique synergy with 3D design and propose a new clustered cache architecture that exploits the unique properties of NTC design. We demonstrate the proposed approaches in a 64-core 3D CMP design and present silicon measurements.

In previous work, subthreshold computing has been widely used for maximum energy efficiency. In this realm, the supply voltage is reduced below the threshold voltage of the devices down to V_{opt}, the optimal voltage that minimizes energy/cycle. By operating at V_{opt}, the leakage and dynamic power components become nearly balanced, maximizing energy efficiency($\sim$12-16x greater than nominal operation). The cost of this improvement is that performance is reduced by $\sim$1000x. This trade-off, however, is particularly suitable for environmental sensing applications [38, 39] and medical sensor applications [40], where low activity rates are needed.

NTC focuses on high performance applications. To accomplish this, the supply voltage is instead reduced from the wear-out limited nominal supply voltage to just *above* the threshold voltage of the technology (V_{NTC}), resulting in a $\sim$60-80x power reduction [36, 37]. This reduction in power facilitates heavily 3D stacked designs. An associated $\sim$10x performance loss is also incurred, resulting in $\sim$6-8x total energy savings. This can be shown to be true

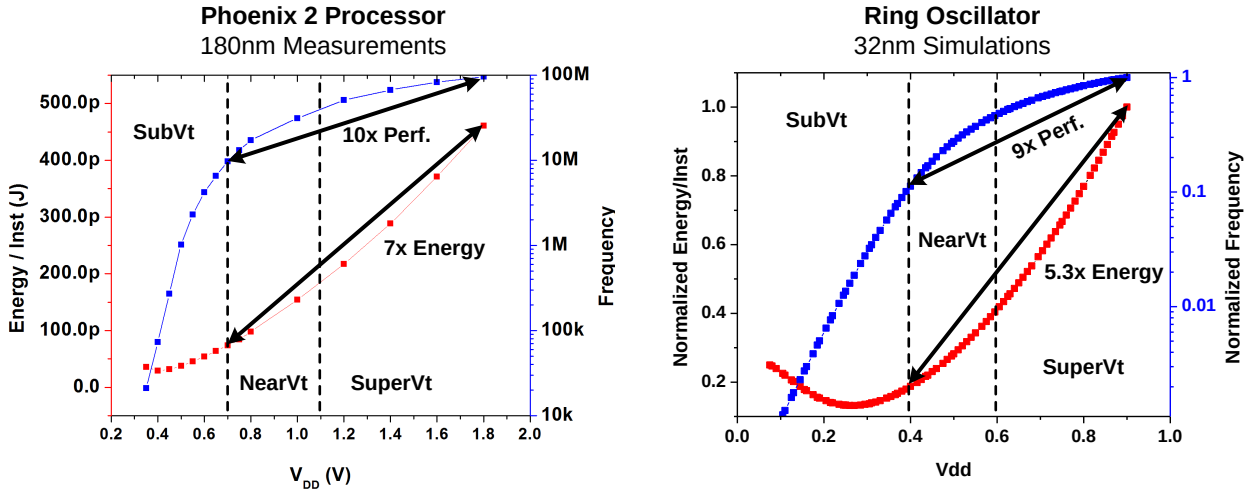


Figure 2.1: **Trade-offs of NTC operation in a range of process technologies.** Data from 180nm is measured, while the 32nm data is simulated. NTC operation is shown to be effective in both older and newer process technologies. Results from Centip3De are from a 130nm process.

across a range of processes, with measured data from a 180nm process and simulated data from a 32nm process shown in Figure 2.1. This loss of performance is more manageable, and can be recovered through parallelism and the performance benefits of 3D-design, resulting in both improved energy/operation and increased overall performance for a fixed TDP [41].

A key observation in NTC design is the relationship between activity factor and optimal energy point [41]. The energy consumed in a cycle has two main components: leakage energy and dynamic energy. At lower activity factors, the leakage energy plays a larger role in the total energy, resulting in higher V_{opt}. Figure 2.2 illustrates this effect in a 32nm simulation, where activity factor was adjusted to mimic different components of a high performance processor. Memories in particular have a much lower activity factor and thus a much higher V_{opt} than core logic. Although NTC does not try to achieve energy-optimal operation, to maintain equivalent energy↔delay trade-off points, the relative chosen NTC supply points should track with V_{opt}.2 Thus, V_{NTC_memory} should be chosen to be relatively higher than V_{NTC_core} for them to maintain the same energy↔delay trade-off point. This strategy does not conflict with other energy saving techniques such as "drowsy caches" [42]. But, since V_{opt} is lower than V_{min} (minimum functional voltage) for typical SRAM designs, a low-voltage memory design will have be used, such as an 8T design [43] or a specially designed 6T [44].

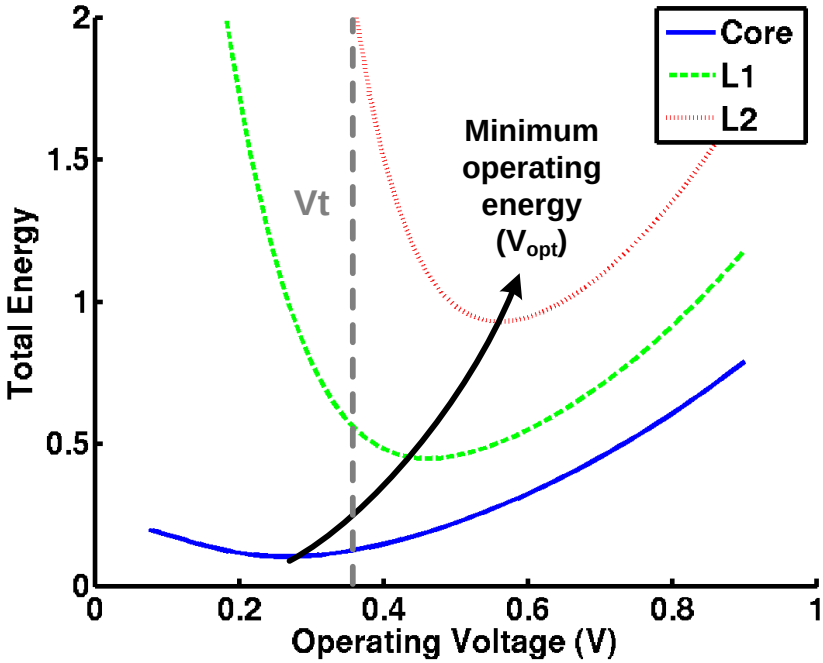


Figure 2.2: **Activity factor versus minimum operating energy.** As activity factor decreases, the leakage component of energy/operation increases thereby making the energy optimal operating voltage increase. To account for this, Centip3De operates caches at a higher voltage and frequency that the cores. An 8T SRAM design was used to ensure that V_{min} is below V_{opt}.

For this reason, 8T SRAMs have become popular in industry for lower level caches and register files [45, 46].

In Centip3De we have used this observation to reorganize our CMP architecture. Instead of having many cores each with an independent cache, the cores have been organized into 4-core clusters and their aggregate cache space combined into a single, 4x-larger cache. This larger cache is then operated at a higher voltage and frequency to service all four cores simultaneously. To do this, the cores are operated out-of-phase (described in detail in Section 2.2.3) and are serviced in a round robin fashion. Each core still sees a single-cycle interface and has access to a much larger cache space when necessary.

The NTC cluster architecture also provides mechanisms for addressing a key limiter in parallelization: intrinsically serial program sections. To accelerate these portions of the program, Centip3De uses per-cluster DVFS along with architecturally-based boosting modes. With two cores of the cluster disabled, the cluster cache can reconfigure its pipeline to access

tag arrays and data arrays in parallel instead of serially and change from a 4x core↔cache frequency multiplier to a 2x multiplier. The remaining core(s) are voltage boosted and operate at 2x their original frequency, roughly doubling their single-threaded performance. The performance of the boosted cores is further improved by the fact that they access a larger cache and hence have a lower miss rate. To offset the related increase in cluster power and heat, other clusters can be disabled or their performance reduced.

The Centip3De NTC cluster architecture has manufacturability benefits as well. By using lower voltages many of the components have improved lifetime and reliability. Redundancy in the cluster-based architecture allows faulty clusters to be disabled which known good die techniques can be coupled with to further increase yield. Centip3De's boosting modes in combination with "silicon odometer" techniques improve performance while maintaining lifetime [22, 23].

Additional benefits of the NTC cluster architecture include reduced coherence traffic and simplified global routing. Coherence between the cores is intrinsically resolved in the cache while the top level memory architecture has 4x fewer leaves. Drawbacks include infrequent conflicts between processes causing data evictions and a much larger floorplan for the cache. In architectural simulation, however, we found that the data conflicts were not significant to performance for analyzed SPLASH2 benchmarks, and we effectively addressed floorplanning issues with 3D design.

To demonstrate the proposed design concepts, we describe Centip3De [34], a large-scale 3D CMP with a cluster-based NTC architecture. Centip3De uses Tezzaron's 3D stacking technology in conjunction with Chartered 130nm process. Measured results for a two-layer, 16-cluster system are discussed. Section 2.2 starts with an overview of the architecture, then describes each of the components of the system in detail. Future expansions of the stack are described in Section 2.3. Section 2.4 describes Tezzaron's 3D integration technologies. Section 2.5 discusses how 3D integration factored into design reuse. Silicon results are presented in Section 2.6 and the chapter is summarized in Section 2.7.

2.2 System Architecture

Centip3De is a large-scale, 3D CMP containing clusters of ARM Cortex-M3 cores [47] designed with the NTC principles described in Section 2.1. A system containing two layers bonded F2F has been measured, where 16 clusters localize the cores to a single layer and the caches to another.

To determine the system architecture, analysis was performed using SPLASH2 benchmarks (Cholesky, FFT, FMM, LU, Radix, and Raytrace) on the gem5 simulator in studies similar to those performed by Dreslinski *et al.* [41]. A 128b, eight bus architecture was chosen based on cache miss bandwidth requirements, floorplanning constraints, and to match data widths of future connect memories (Section 2.3). A cluster size of four cores per cluster, a 1kB instruction cache, and a 8kB data cache were chosen to maximize energy efficiency while working within our area constraints. In this study, four-core cluster systems were found to be 27% more energy efficient while providing 55% more throughput than one-core cluster systems for Centip3De's design constraints.

For multi-program support, each of the 64 cores runs with an independent stack pointer in a non-virtualized memory space with access to a core ID and other core-specific registers programmed by JTAG. Each cluster contains a cluster-level memory-mapped semaphore to support fine-grained data locking. With these tools, Centip3De can support completely separate programs, or a single program with many threads.

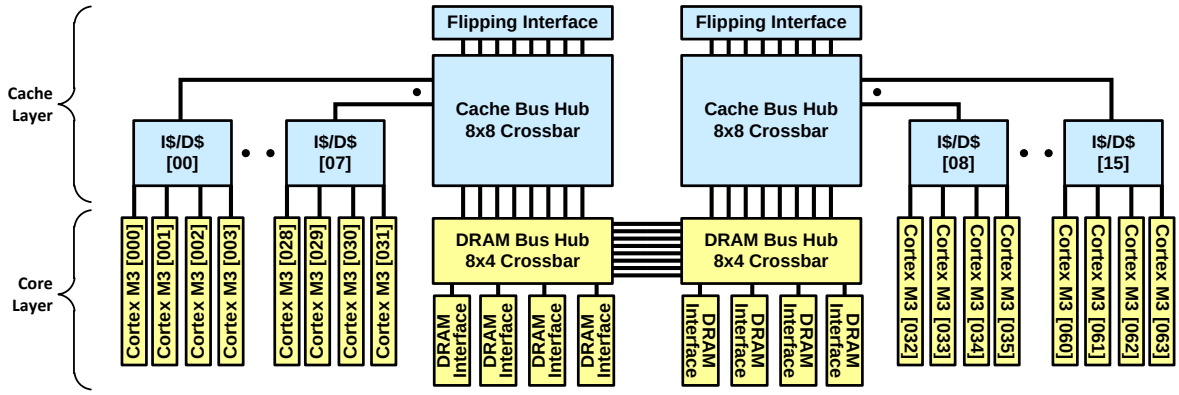


Figure 2.3: **System block diagram.** The block diagram is organized and shaded by layer, with F2F connections shown as lines crossing between these layers. The 8 buses each connect to all 16 clusters as well as a round-robin arbiter.

In addition to cores and caches, each cluster contains local clock generators and synchronizers. Figure 2.3 shows a block diagram for the architecture with the blocks organized by layer. Centip3De also has an extensive clock architecture to facilitate voltage scaling, which is discussed in detail in Section 2.2.5.

2.2.1 Floorplanning

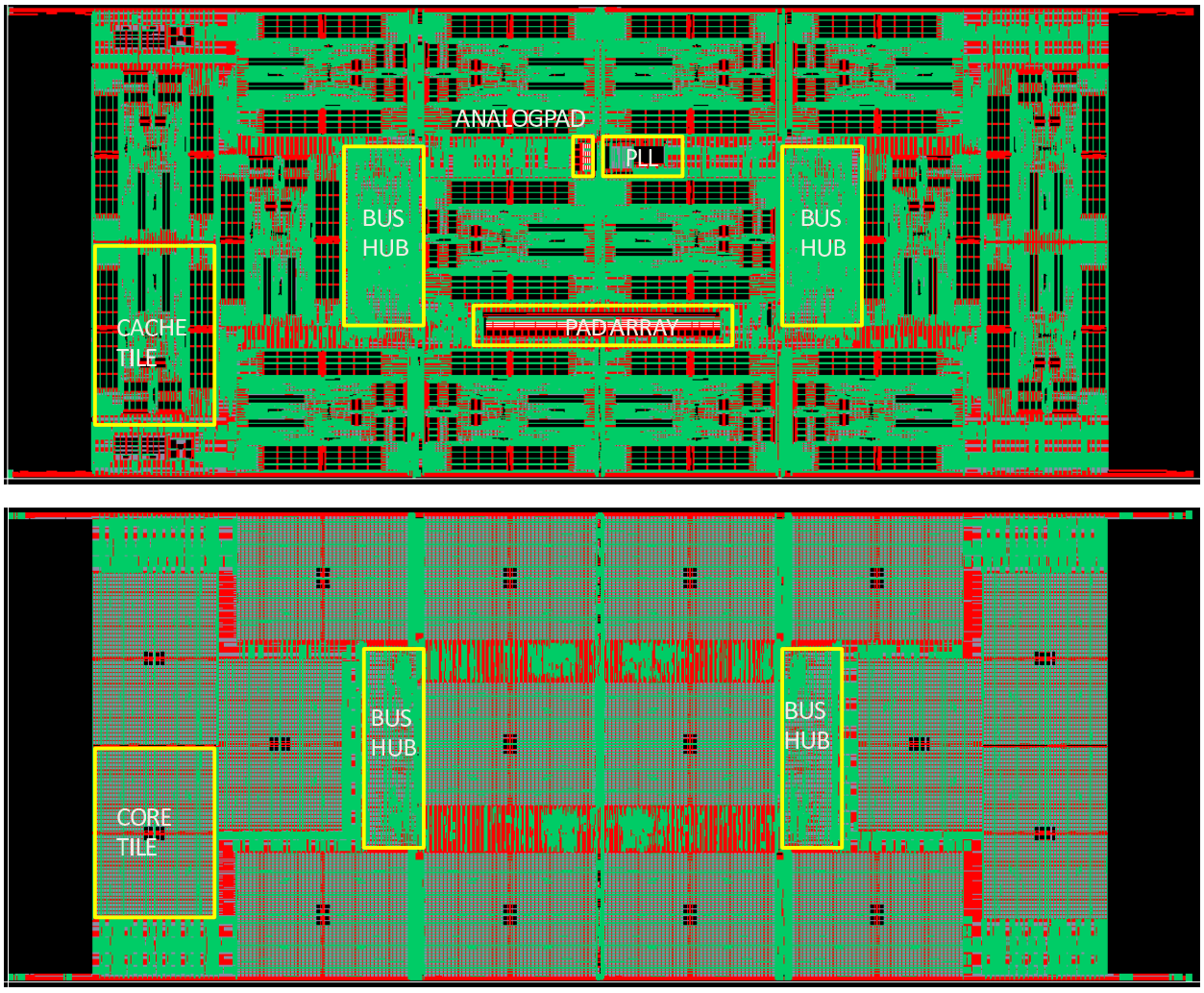


Figure 2.4: **GDS floorplan of core and cache layers.** The cache layer shown on top and the core layer is shown on bottom.

Floorplanning in 3D design addresses the main two classes of interconnect: signal routing and power delivery. Signal routing between layers can be through fine grained (transistor level) or coarser grained (bus level) vertical interconnect. Centip3De uses bus level connec-

tions for design simplicity. However, due to the low parasitics of TSVs in Tezzaron's process (Section 2.4), Centip3De places high-performance, routing-dense buses vertically, which in this system are between cores and caches, and within the bus architecture. A GDS screenshot of the top-level floorplan is shown in Figure 2.4.

To improve power routability, the cores and caches were separated into different layers since they use separate power supplies. Needing to route one less class of power in a given layer reduced resistive droop by approximately 15-25% for the same resources. Localizing the caches to a single layer also simplified the bus design. Careful planning of bus hub ports eliminated cache↔bus routing congestion on that layer. The twin bus hub columns are bridged with eight buses on the core layer, which has little additional global routing.

By building the bus architecture vertically, required routing resources reduced by approximately 50% compared to a single-layer floorplan. Similar gains are obtained in energy and performance, and were not offset by 3D interconnect loading due to its relatively small overhead.

2.2.2 Processing Core

Centip3De contains 64 ARM Cortex-M3 cores, which have a 3-stage, in-order, single issue pipeline. In this 130nm design, the core operates between 10MHz at 650mV and 80MHz at 1.15V.

The core has a JTAG interface that allows access to status registers within the core, reset and halt capabilities, and access to the memory system. JTAG also provides access to the memory system, which includes memory mapped IO (MMIO). The JTAG data signals are daisy-chained between the four cores of each cluster, then between multiple clusters, as described in Section 2.2.6.

MMIO registers control the stack pointer location, a general purpose registers, a hard reset, and core clock adjustment controls. The clock adjustment controls allow the measurement and control of core↔cache clock skew that is further described in Section 2.2.5. MMIO also allows access to a hardware based semaphore contained in the cache, which organizes memory operations within the cluster. Core 0 has additional MMIO registers (they exist

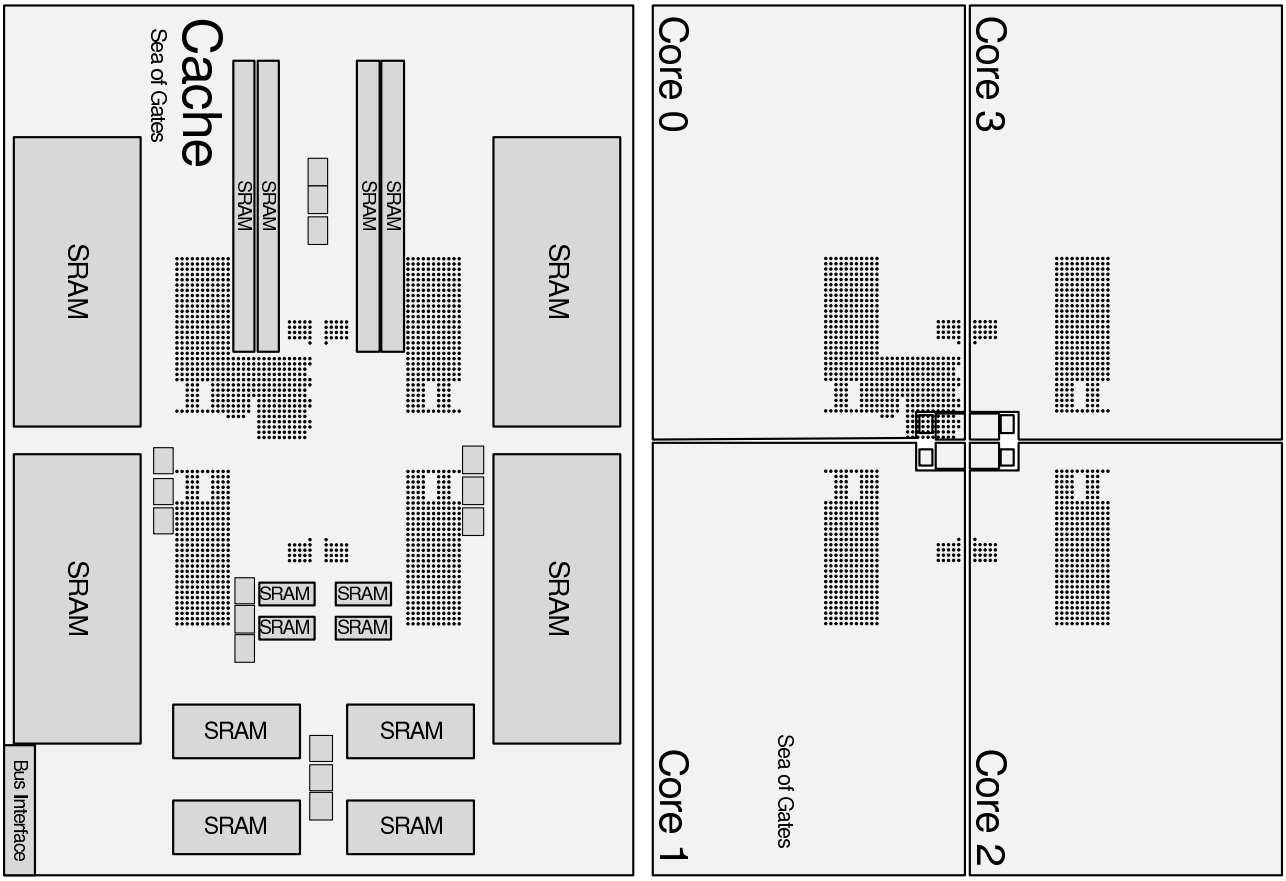


Figure 2.5: **Cluster floorplan with F2F connections represented as dots.** There are 1591 F2F connections per cluster, including 331 per core, plus an additional 267 cluster configuration signals for Core 0.

vestigially in cores 1-3) to access cache mode, cache clock skew measurement and control, and cache clock multiplier controls. The core 0 MMIO has additional registers to control clock gating for the other three cores and multiplexers to add or remove those cores from the JTAG daisy chain.

The cluster floorplan for the core layer is shown in Figure 2.5. Each of the four cores is identical - the same layout is mirrored four times to achieve the cluster layout. In the center of the cluster is the clock delay generator and the skew measurement unit. Core 0 has 598 signals, while cores 1-3 each have 331 signals, for a total of 1591 connections to the cache. This results in 25,456 core↔cache vertical interconnects for the 64-core system. The vertical interconnections are visualized in Figure 2.6. Outputs from the core are differential to facilitate level conversion in the cache.

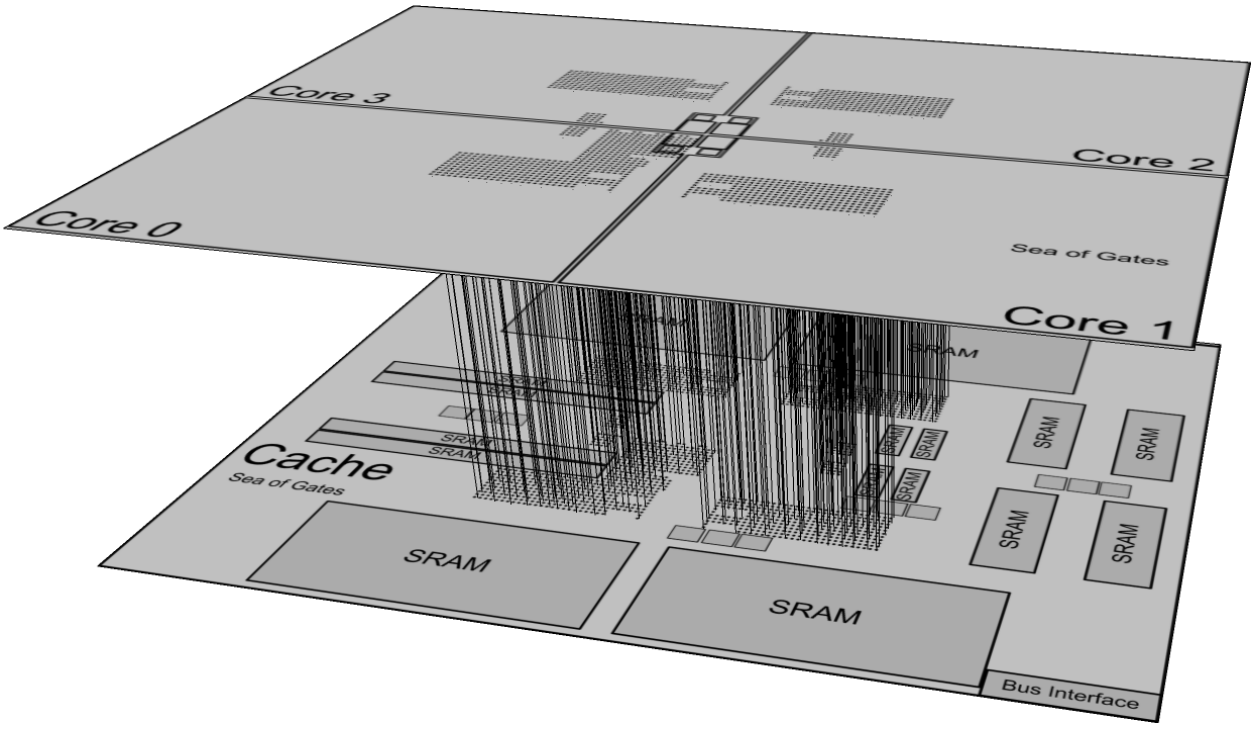


Figure 2.6: **Artistic rendering of a cluster.** Relative module size and placement are accurate. F2F connections are represented as dots with lines between. The floorplan can be seen more clearly in Figure 2.5.

All core signals connect F2F to the cache on the adjacent layer, with no direct connections to other modules on the same layer (Figure 2.3). The core also contains a square grid of dummy TSVs on a 50um pitch to meet TSV density rules, and are visible in Figure 2.7. Since the cluster is rotated to many orientations (as seen in Figure 2.12), both grids are square (x and y dimension matched) and can rotate together on the relevant copper honeycomb interface patterns discussed in Section 2.4.

2.2.3 Cluster Cache

The cluster cache contains a 1kB instruction cache, an 8kB data cache, clock generators, and hardware semaphores. The cache operates between 40 MHz at 800mV and 160 MHz at 1.65V. The cache can operate in high-efficiency four- and three-core modes or high-performance two- and one-core modes. In three-/four-core modes, the cache operates 4x the frequency of the cores, and the enabled cores operate with separate clocks that are 90° out-of-phase. In one-/two-core modes, the relationships are 2x and 180°, respectively. The

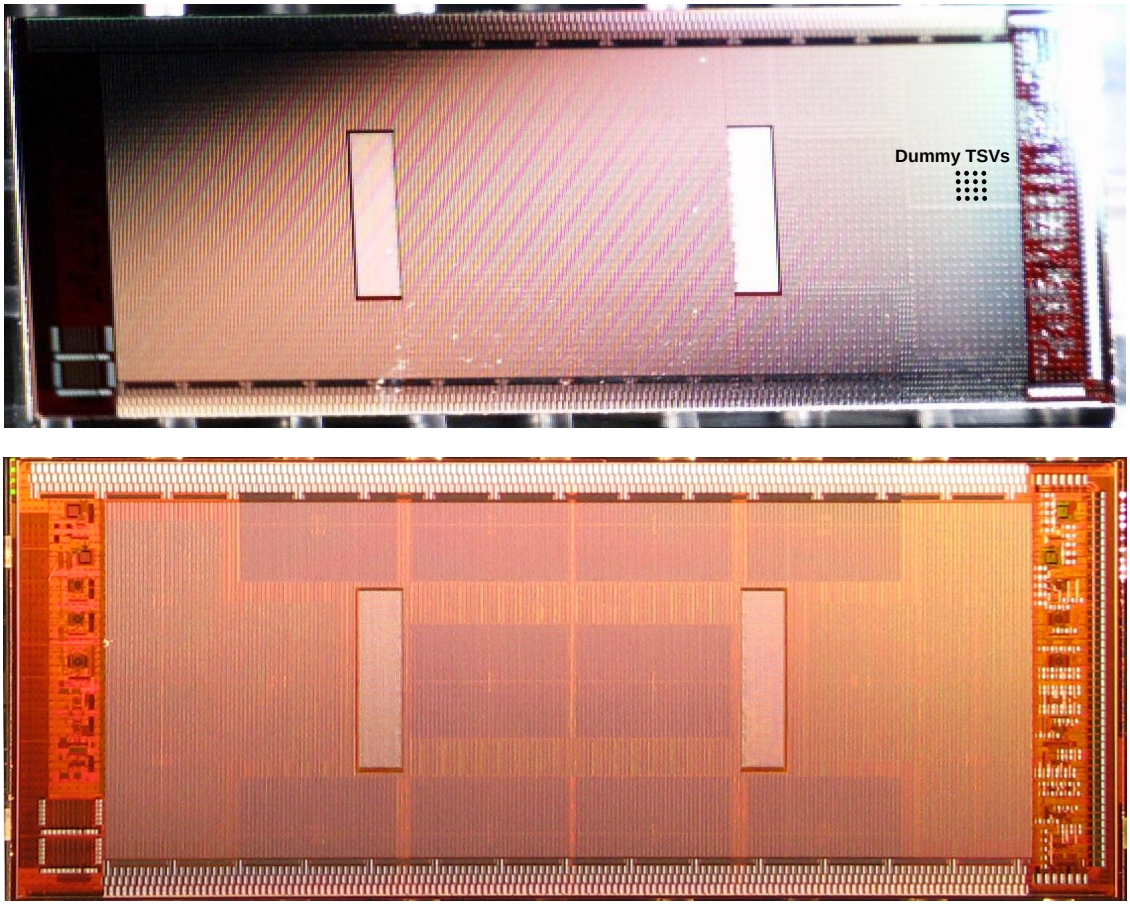


Figure 2.7: **Die micrographs of two-layer system.** Two layers are bonded face-to-face. The clusters can be seen through the backside of the core layer silicon. Wirebonding pads line the top and bottom edges. Above, the clusters can be seen by the pattern of dummy TSVs; the clusters have a square grid of TSVs (for rotational symmetry), whereas the space between cores has more densely packed rows of TSVs.

cache is pipelined as shown in Figure 2.8 such that in all modes the cores see a typical, single-cycle interface to the cache.

For robust low-voltage operation, the cache uses a custom 8T bitcell design with tunable pulse generators. A word length of 32b and a line length of 128b were chosen to match the core and bus datapath widths, respectively. The cache supports 4-way set association nominally and a direct mapped mode for debugging.

The cache state machines and pipeline diagrams for each of these modes are shown in Figure 2.8. In these diagrams the terms **EA**, **EB**, **EC**, and **ED** refer to the phase of the core clock, **Mem Available** refers to the availability of the shared bus interface, and **Conflict**

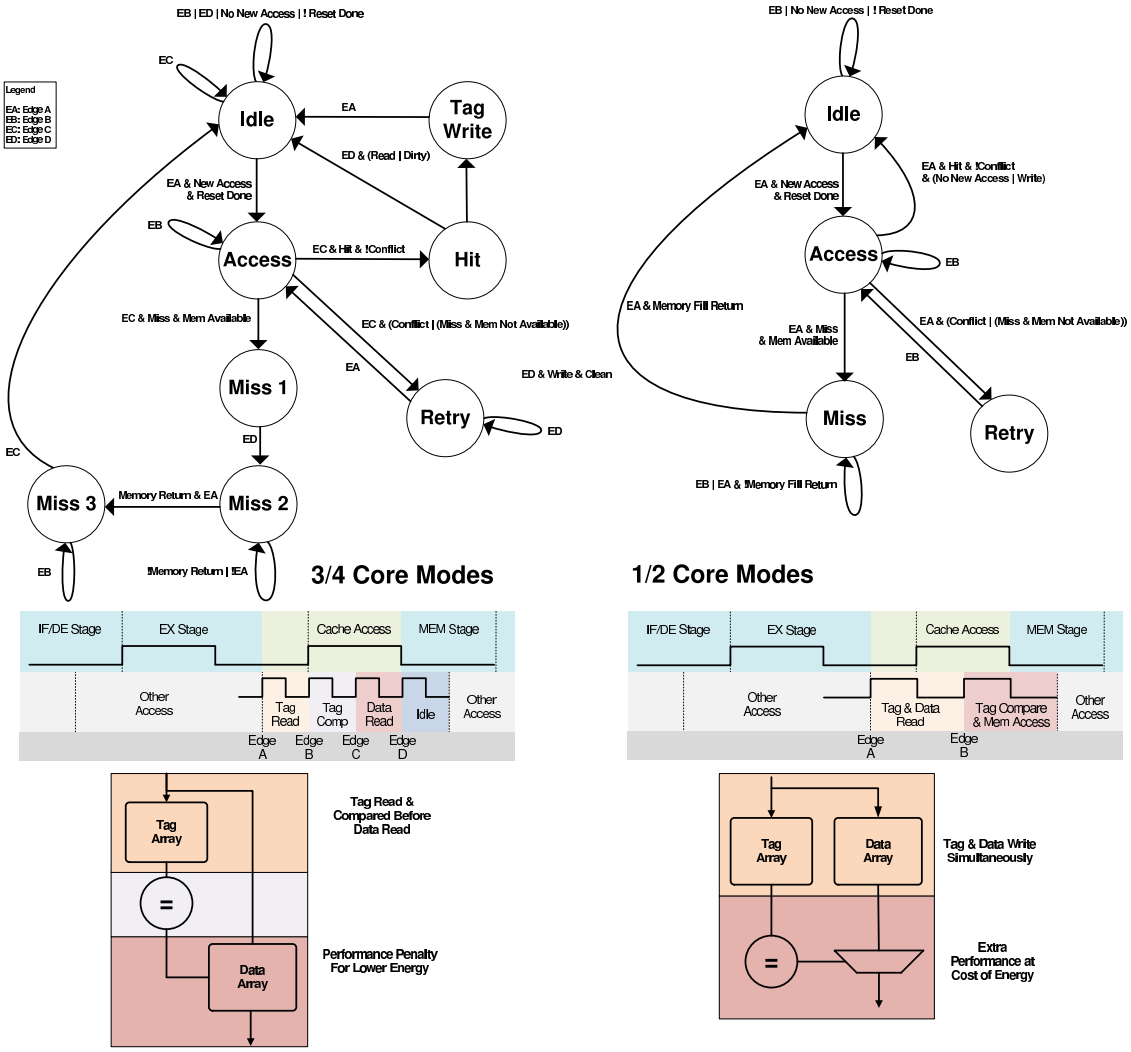


Figure 2.8: **Cache state machines and pipeline diagrams.** Four cache modes are supported. In three and four-core modes the high latency of the processors is used to improve efficiency by accessing tag and data arrays sequentially. Knowing the tag check results reduces the number accessed data arrays.

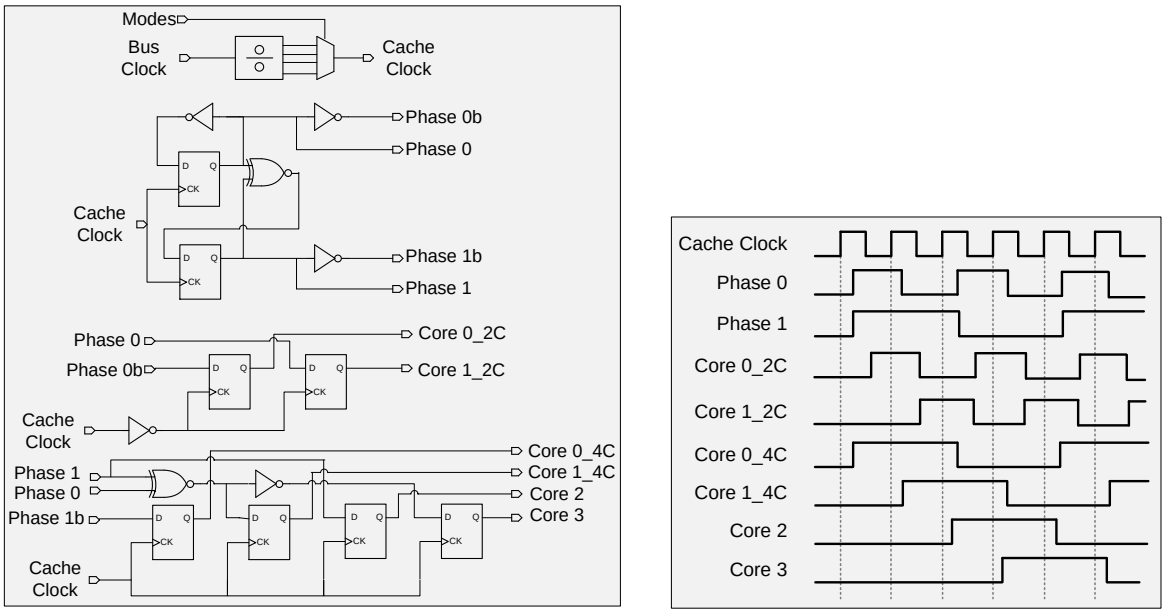


Figure 2.9: **Glitch-free clock generator and associated waveform.** The bus clock is divided to generate the cache clock. The core clocks are then generated from the cache clock, depending on the mode. Since Core 0 sets the clock mode but also uses one of the generated clocks, it is important that its clock never glitches. For safety, this unit is designed such that no clock glitches. Additional components include clocked multiplexers for selecting between modes and gating-off partial cycles post-reset.

refers to data conflicts between the cores of the cache.

The three/four-core mode supports higher efficiency than the one-/two-core mode by first reading and checking the tag arrays, then accessing only the necessary data array(s). By doing this, at most one data array is read per hit, or all four data arrays for a miss with eviction. In contrast, each of the four data arrays are read for every access in one/two-core mode, in case there is a hit. When a miss occurs, the other cores are able to continue operation until a conflict occurs.

In both modes, the cache is pipelined such that the core sees a typical single-cycle interface. Through synthesis, it was determined that cycle-stealing from the core is possible, as shown in the pipeline diagram in Figure 2.8. The cache first captures the access from each core three quarters of the way through the core cycle and returns halfway through the next core cycle.

The cluster clock generator is shown in Figure 2.9. These two modes require different frequency and phase relationships with the core clocks. These clocks are generated locally

based on configuration bits from core 0 and synchronized with the bus clock with a tunable delay buffer. Before generating the core clocks, the cache clock is first divided by a factor between one and eight. The clocks transition glitch-free during mode changes to prevent core logic corruption, which is particularly important for core 0. The individual core clocks can also be gated by configuration bits from core 0 to reduce energy consumption in unneeded cores.

To assist in multi-core programming, hardware-based semaphores are included in the cache to provide a read-modify-write operation to a select number of addresses. Inputs from the core are differential and are level converted upon entry into the cache. Similarly, outputs to the bus are also differential.

The floorplan in the cache is shown in Figure 2.5. The F2F connections from the cores appear in the center of the floorplan. This is particularly beneficial since the SRAMs use all five metal layers of this process, thereby causing significant routing congestion. By using 3D stacking, we estimate that cache routing resource requirements were reduced by approximately 30%. These benefits were not offset by the loading requirements of the 3D interface, since the parasitic capacitance and resistance of the F2F connections are small.

A square grid of dummy TSVs exists on a 50um pitch and is aligned with the TSV grid of the cores such that that the cluster design may be rotated. Within the SRAM arrays, some dummy TSVs are safely forgone while others are aligned with pre-existing power routing. The dummy TSV impact on array utilization is less than 1.5%.

2.2.4 Bus Architecture

The bus architecture includes eight independent, 128b buses that operate between 160 MHz at 1.05V and 320 MHz at 1.6V (all at the same frequency). The eight buses represent independent address spaces and service cache misses from all sixteen clusters. A round-robin arbiter determines the order of priority of the requests (Figure 2.3). The buses are each physically split into two columns that span the cache and core layers of the chip, as shown in Figure 2.4.

Each bus can service a single request at a time, which takes six to fifteen bus cycles.

Crossing from one column to the other induces a single-cycle penalty each way. For cores operating in higher efficiency modes, a cache miss and resulting memory request incur as little as a single-cycle penalty for the core. In the highest performance core modes a cache miss becomes a four-cycle minimum penalty.

The bus arbiter contains configuration bits to mask out faulty clusters, although this proved to be unnecessary in our testing. Configuration bits also control which three bits of the memory address select the bus, with options being either the top three or bottom three. These settings allow Centip3De to either distribute traffic to all of the memory buses, or localize traffic to particular buses.

The two communication columns are bridged on the core layer with eight buses, while the routing to the clusters is on the cache layer. Routing lanes exist between the clusters on both layers to facilitate this routing. The main portion of the logic exists in two bus hubs on each layer, as seen in Figure 2.4. Vertical interconnect in the bus hub modules alleviates routing congestion within the module, reducing the footprint and making additional perimeter space accessible for global routing.

2.2.5 Clock architecture

Centip3De is designed so that cores, caches, and the main bus operate on different power domains with level converters included between these power domains. Since each power domain can be scaled individually depending on the desired power and performance target, inter-clock-domain skew becomes an issue. For example, if a cluster that runs at half the frequency of the main bus is re-tuned to run at a quarter of the frequency, then the delay through its clock tree will change, resulting in timing violations when crossing clock domain boundaries.

To address inter-clock-domain skew caused by voltage scaling, each clock domain has a delay generator and each relevant pair of clock domains has a skew detector connected at the leaves. This method also simplified clock tree matching between separately designed modules. By tuning the clock tree once, the settings can be saved and restored, even for different dies, although re-tuning a particular die is possible if needed.

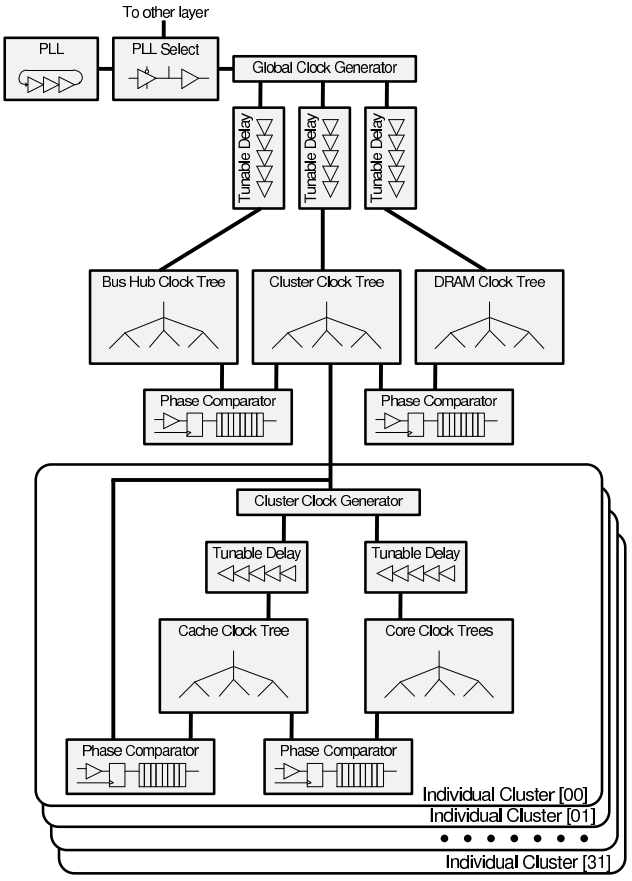


Figure 2.10: **Clock architecture.** To align clock phases after phase changes due to voltage scaling, the clock architecture includes clock phase comparators and digitally controlled tunable delay buffers.

As shown in Figure 2.10, an initial clock is generated by a phase-lock loop (PLL), which locks to an off-chip clock reference, or can be controlled directly (no-feedback) via an off-chip reference voltage. This clock is used to generate three global clocks that can be aligned using two phase comparators. Each computation cluster has its own clock generator for cache and core clocks. The core clocks are phase compared to the cache clock, which is also phase compared to the bus clock.

The phase comparator design is shown in Figure 2.11. Similar to simple PLL phase comparators, a flip flop is used to compare the phases of two clock trees. Two additional flip flops are used to protect against metastability. In lieu of an RC filter, an up/down counter counts for a preset number of cycles (2048). The phase comparators are controlled via scan chain at the system level or JTAG at the cluster level.

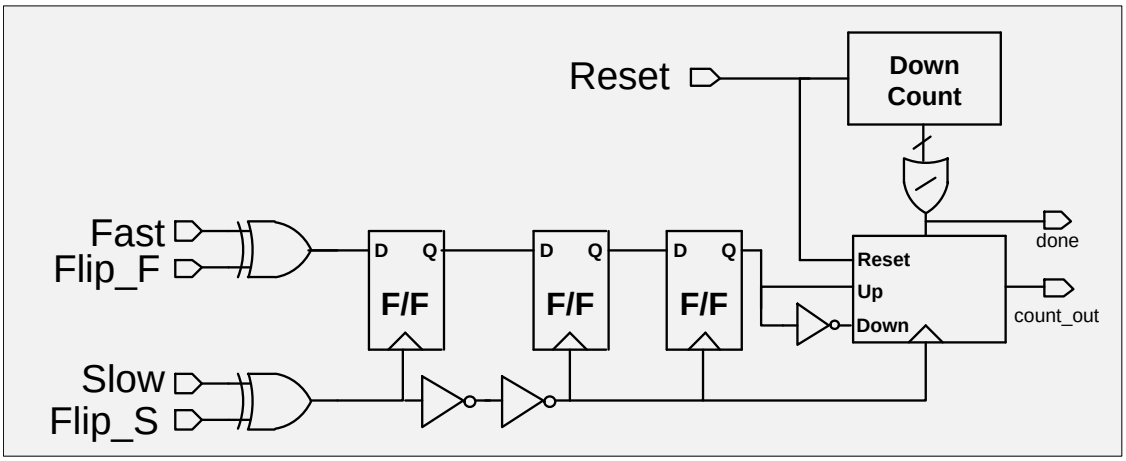


Figure 2.11: **Clock phase comparator.** Similar to simple PLL phase comparators, a flip flop is used to compare the phases of two clock trees. Two additional flip flops are used to protect against metastability. In lieu of an RC filter, an up/down counter counts for a preset number of cycles (2048).

2.2.6 Offchip I/O and Power

Aluminum wire bonding pads were added to the core backside for packaging. These structures are visible in the periphery of the die in Figure 2.7.

Due to floorplanning constraints, the digital and analog pads were moved to the center of the floorplan on the cache layer (Figure 2.4). On the core layer, these areas are used to route bridging buses between the two communication columns. Routing from these pads to the chip periphery used extra-wide wiring for robustness.

In a trade-off between testing time and number of IO pads, the JTAG data signals were daisy-chained through the four cores of a cluster and two clusters on the same cache layer. In this way, there are eight JTAG data interfaces, each connected to eight cores, which facilitates the parallel loading of data. A scan chain is included to control system-level configuration settings, such as clock delay chain settings and phase generator controls [48].

2.3 Future Directions

This chapter describes a measured two-layer, 64-core system, while the final system will include up to seven layers, 128-cores, and 256MB of DRAM, via already existing cross-layer

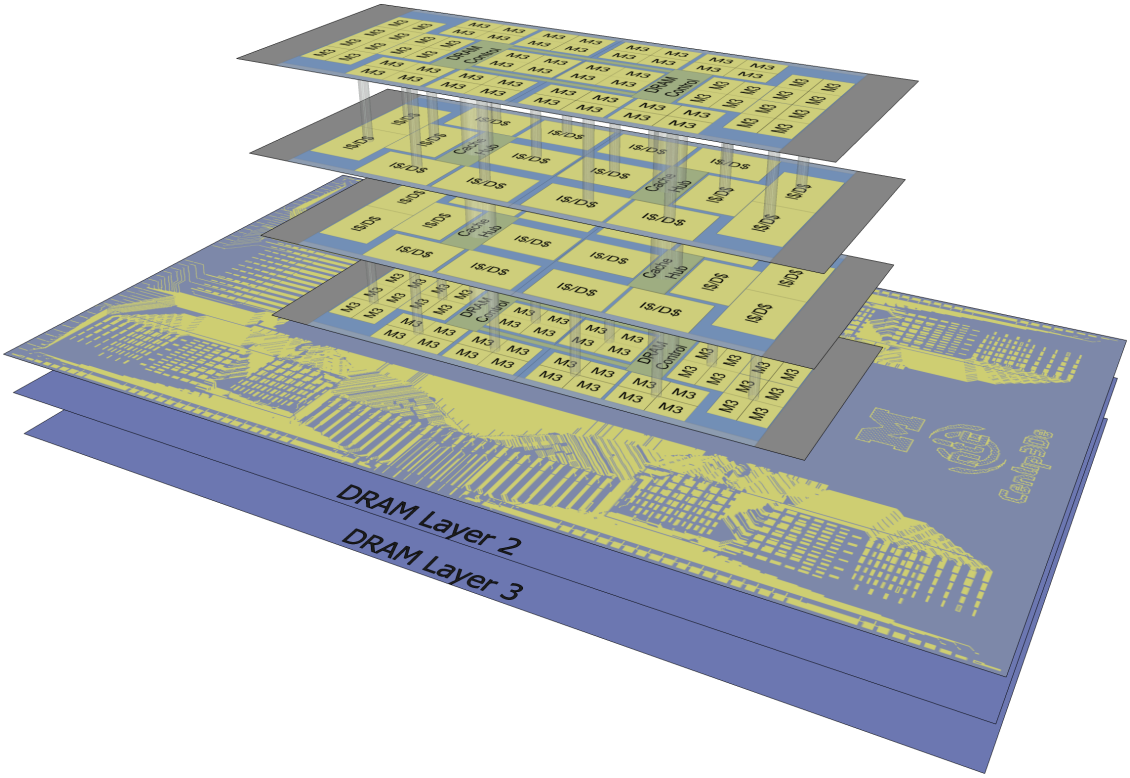


Figure 2.12: **Floorplan accurate artistic rendering of a seven-layer Centip3De system.** Centip3De includes up to seven layers in future versions, including two core layers, two cache layers, and three DRAM layers.

interfaces. These include a flipping interface to allow the addition of another core and cache layer pair, and a DRAM interface, which allow the addition of three layers of DRAM as shown in Figures 2.12 and 2.13. These additional layers use back-to-back (B2B), and face-to-back (F2B) bonds in addition to the F2F bonds used in the measured system.

2.3.1 Flipping Interface

The seven-layer system includes duplicated core and cache layers. These layers are designed to be reused, i.e., they are identical copies. To accomplish this, a flipping interface is designed into the bus hub on the cache layer, as seen in Figure 2.3. The two bus hubs are placed equidistant from the center of the chip so that when two cache layers are aligned B2B, the bus hubs also align. The flipping interface in the bus hubs includes pairs of tri-state buses.

The direction of each of these buses is determined by detecting which side of the B2B

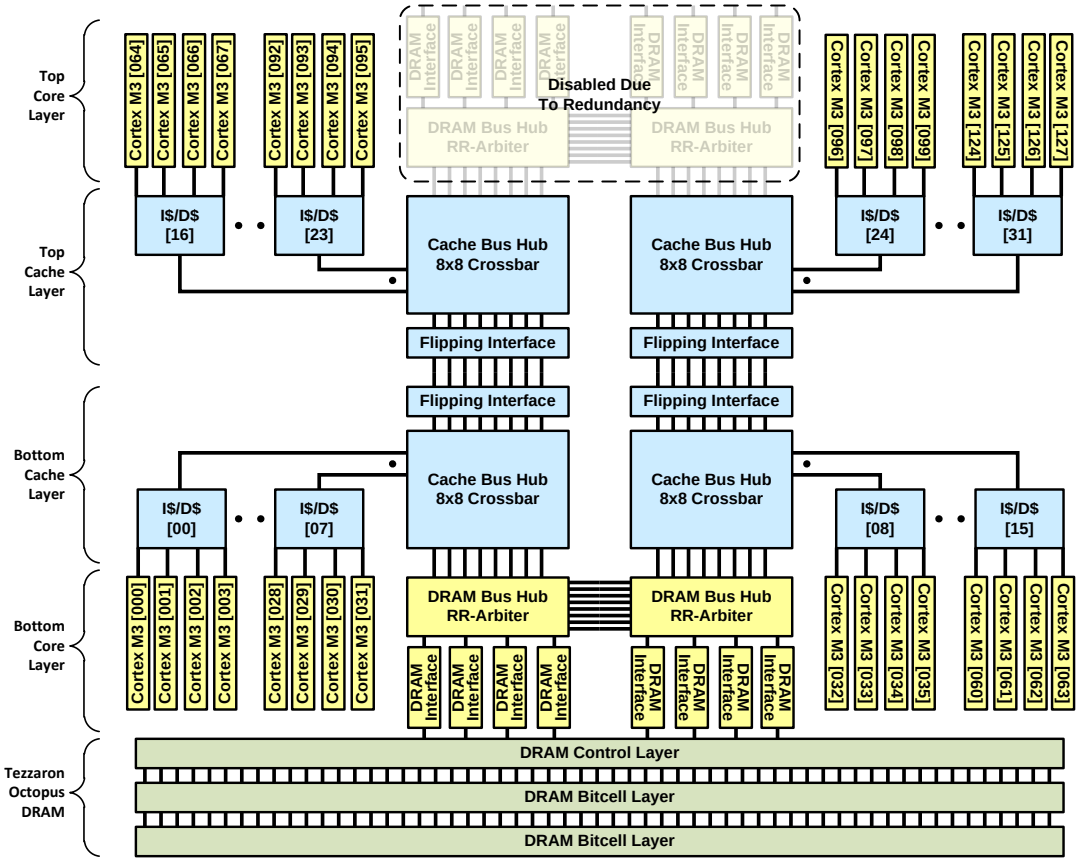


Figure 2.13: **Seven-layer system block diagram.** Centip3De includes up to seven layers in future versions, including two core layers, two cache layers, and three DRAM layers.

interface is connected to the DRAM and/or wirebonding pads. A single internally pulled-down IO pad is used to make this determination. The side-detector connected to the outside world will be wirebonded to V_{DD}, whereas the other will automatically pull-down to ground instead. By doing this, we can safely negotiate the directions of the tri-state buses. The side-detector also disables redundant units such as the unconnected DRAM interfaces and unnecessary bus hubs, labeled at the top of Figure 2.3.

2.3.2 DRAM Interface

In the seven-layer system, there will be 256MB of Octopus DRAM organized into eight banks, each with its own DRAM interface [15]. The DRAM interfaces are similar to DDR2 and use double edge clocking for data transfer to the DRAM. They operate between 160 MHz at 1.05V and 320 MHz at 1.6V, on the same frequency and voltage domain as the

bus architecture. They also use a double frequency clock at 320/640MHz to facilitate DDR operation. With the bus architecture, the DRAM interfaces provide 2.23-4.46 GB/s of memory bandwidth to the clusters.

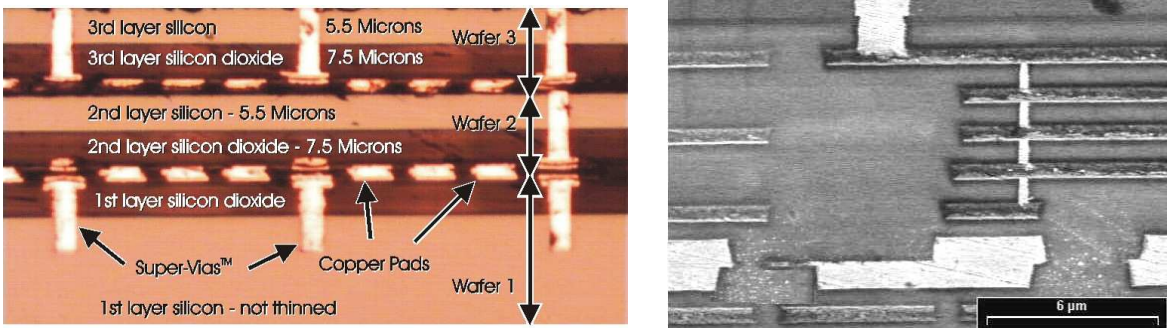
Similar to typical DDR2 interfaces, a page must first be opened before it can be accessed. The controller keeps track of which DRAM page is currently open, and opens new pages when necessary. Memory operations are performed in 4x128b bursts that are also cached. Depending if the page is closed, the page is open, or the needed data is already cached, a DRAM operation can take between one and eight bus cycles. DRAM interface settings include typical RAS and CAS settings, among others.

The DRAM controllers are synthesized with the bus and share the bus hub module area on the core layer. The bus hubs were placed directly on top of the DRAM interface. Large clusters of TSVs were used to connect to the DRAM through the die-to-wafer stacking process. These TSVs provide sufficient density to meet TSV density requirements.

The DRAM has one control layer and 1-4 gigabit bitcell layers; the seven-layer Centip3De system will include two bitcell layers. The control and sense amplifier layer was designed by Tezzaron in the same Chartered 130nm process used for the core and cache layers. The bitcell layers were provided by an outside vendor to Tezzaron, who then stacked all 2-5 wafers together for the final DRAM wafer stack. To configure the DRAM internal timing and program the number of bitcell layers, the DRAM contains a "message box" control interface that is accessible by the scan chain.

2.4 Tezzaron's 3D Technology

Tezzaron's FaStack® technology stacks wafers of silicon (as opposed to individual dies) using copper bonding [13]. Before each wafer pair is bonded, a layer of copper is deposited in a regular honeycomb pattern that is then thinned. Due to the homogeneity of the pattern, it is very flat after thinning, and when two wafers with this pattern are pressed together with heat, the copper bonds strongly. After a bond, one side of the resulting wafer is thinned so that only a few microns of silicon remains, which exposes TSVs for 3D bonding, flip-chip, or wirebonding patterns. The TSVs are made of Tungsten, due to preferable



(a) A Three-Wafer Stack

(b) A Two Wafer Stack in Finer Detail

Figure 2.14: **Cross-sections of Tezzaron's 3D stacking process.** Provided by Tezzaron, neither micrograph is exactly the same process used for Centip3De or from the same wafer. **(a) A three-layer stack with F2F (wafers 1-2) and F2B (wafers 2-3) bonds.** The honeycomb pattern is seen as the regular array of metals. **(b) An F2F bond shown in finer detail, with an exposed TSV at the top.** The exposed TSV is coated with backside aluminum. [13]

thermal-mechanical properties, and are created after the transistors, but before the lowest metal layers, preventing the loss of any metal tracks. Since their diameter is small (1.2um), their length is short (around six microns), and they're only coupling to the bulk silicon, their parasitic resistance and capacitance is very small (about as much capacitance as a small gate). The finished wafer stack has a silicon pitch of approximately 13 microns with a per-interface TSV density of up to 160,000/mm^2. The combination of thin silicon layers, a high density of tungsten TSVs, and planes of bonding copper together maximize heat dissipation in this stacking technology. Figure 2.14 shows two Tezzaron-provided photos of the technology.

A similar process is used to stack two dies of different sizes. In this design, the Tezzaron Octopus DRAM [15] is much larger than the core and cache layers (Figure 2.12). To stack these, the wafer of smaller dies is thinned, coated with copper, and then diced. The wafer of larger dies is also thinned and coated with copper. The smaller dies are put in a tray to hold them in place and the tray of smaller dies and the larger wafer are pressed together to finish the bond. A larger copper pattern is used to support less-precise alignment, using

27 TSVs for each connection. This process would not be needed if the two designs had the same physical dimensions.

2.5 Design Reuse

Design reuse in 3D ICs has challenges at both module and layer levels. At the module level, we reduced design effort for the core and cache designs by tiling instead of recreating each core and cache instance. This simplified verification, DRC, and LVS, however placing vertical interconnect in a design that may be rotated or flipped proved challenging. TSVs and F2F interconnects both connect to a pre-set honeycomb pattern. These designs could only use TSV and F2F interconnect locations that are rotationally symmetric around the origin of the module, relative to the honeycomb pattern. For the F2F interfaces, this created a 5um square grid of interconnect locations that could be used. For the B2B interface, however, this was a 50um grid. This restriction only applied to modules that needed to be flipped and rotated, which included the clusters only. The B2B interface in the cluster was needed for placing filler TSVs only.

At layer level a modification was created to expand the system from two logic layers to four. Since the two-layer logic stack is symmetric across the Y-axis, it is possible to combine two completed two-layer stacks such that their bus interfaces align. The core-side of the two-layer stack contained the DRAM interface, so the cache-side was used to expand the logic stack with B2B TSVs. This design strategy requires only one additional mask for fabrication, the copper pattern for the B2B interface. To complete the expansion, a flipping interface and side-detector were added, as discussed in Section 2.3.1.

2.6 Measured Results

A two-layer system was fabricated with results provided. Technology data and system information are shown in Table 2.1. A die micrograph is shown in Figure 2.7. In this system a core layer and a cache layer were bonded face to face. The backside of the core layer was then ground down, exposing TSVs for the DRAM interface and for offchip I/O. A

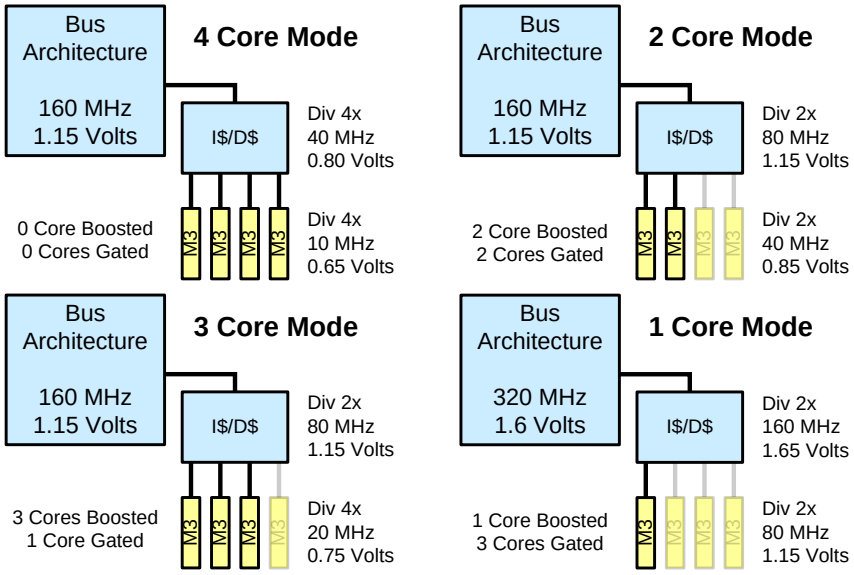


Figure 2.15: **Visualization of the four cluster modes.** Included are frequencies and voltages used for analysis for performance and power measurements.

layer of aluminum was then applied to create wirebonding pads for the TSVs and to cover exposed DRAM and dummy TSVs. The chip was then wirebonded in a 256-pin, ceramic PGA package, and tested using a custom-designed PCB and LabVIEW.

A measured frequency and energy profile of the core is included in Figure 2.16. Although Centip3De can support a wide variety of voltages, frequencies, and modes for each cluster, four configurations in particular were used in this analysis, as detailed in Table 2.3 and visualized in Figure 2.15. The modes were chosen to illustrate a wide range of operations, with maximum energy efficiency at low voltage in four-core mode, and maximum boosting performance at high voltage in one-core mode. Near ideal voltage scaling is achieved within the cluster, although one-core mode has an additional 8% of cache leakage power due to the bus voltage being $50mV$ higher than the cache voltage. Table 2.2 details the bus configurations used in these four modes, with 160MHz and 320Hz operations.

The test program was written in C, compiled into the ARM Thumb ISA, and loaded into the cluster via JTAG. It contains a number of arithmetic operations based on an example power virus program provided by ARM and also uses the hardware semaphores to ensure that all four cores start the main portion of the program simultaneously. It is designed to

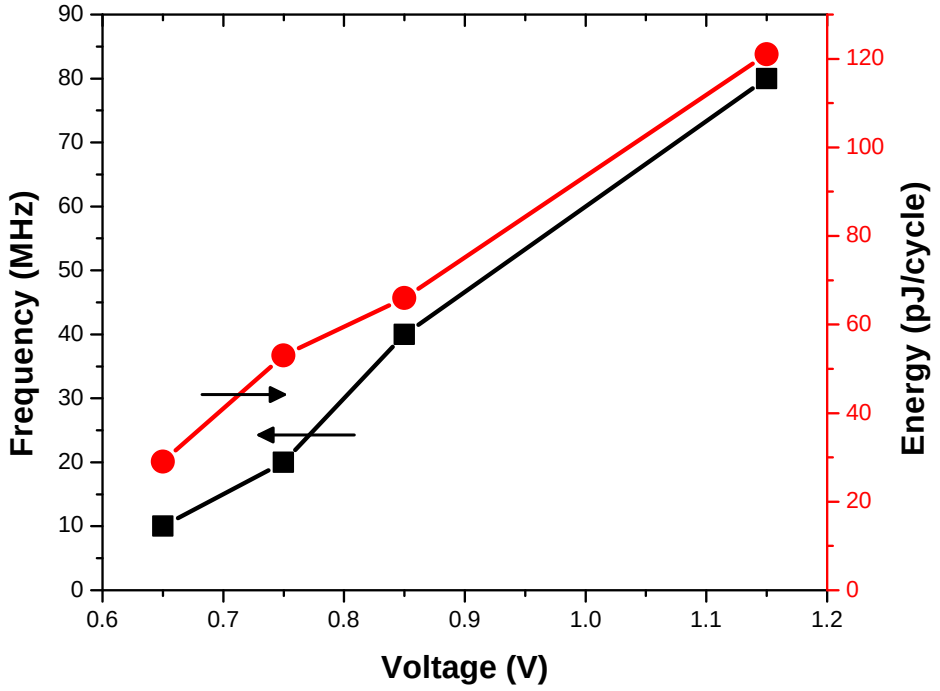


Figure 2.16: **Measured core frequency and energy profile.**

stress the caches, but also fit within them since the DRAM is currently unavailable for use. The direct-mapped cache mode facilitates operation without DRAM.

For each performance point the core voltage is adjusted first. Each core voltage requires a different clock tree alignment setting, which was applied using JTAG controls. After a particular operating voltage for the core is set, the minimum necessary cache voltage is determined. After the cache voltage is adjusted, the cache clock is first re-aligned to the bus clock and then the core clock is re-aligned to the cache clock. During clock phase alignment, the phase comparator counter typically returned one of the two extreme values, meaning that the clock jitter was significantly smaller than the delay generators FO1 delay increment, and a three or four bit counter would work as well as eleven bits. The phase generator and phase comparators were always able to align the clock trees across a variety of voltage scaling scenarios. In a commercial application, the phase information would be stored in a look-up table off-chip and would be managed by the system BIOS. The operating system would make advanced configuration and power interface (ACPI) requests for particular power states, and the BIOS would provide the voltage and configuration definitions of these states.

The highest efficiency four-core mode operates with cores at 10MHz and caches at 40MHz,

Table 2.1: **Design and technology data.** Connection numbers include only signals. F2F connections are for a single F2F bonding interface.

Logic Layer Dimensions	2.66 x 5mm
Technology	130 nm
Metal Layers	5
Core Layer Devices	28.4 M
Cache Layer Devices	18.0 M
Core Layer Thickness	12 um
F2F Connection Pitch	5 um
F2F Connections/Cluster	1591
Bus F2F Connections	2992
Total F2F Connections	28485
B2B Connection Pitch	5 um
Total B2B Connections	3024
DRAM Connection Pitch	25 um
DRAM Connections	3624

Table 2.2: **Bus configurations used in power and performance analysis.**

Frequency (MHz)	**Voltage (V)**	**Power (mW)**
160	1.05	113
320	1.60	471

achieving 8800 DMIPS/W (Table 2.3). Latency critical threads can operate in the boosted mode at up-to 8x higher frequency. The boosted one-core mode operates the core at 80MHz and the cache at 160MHz. The difference in energy efficiency and single-threaded performance between four-core and one-core modes is 7.6x and 8x, respectively.

Boosting clusters within a fixed TDP environment may require disabling or down-boosting other clusters to compensate for that cluster's increase in power consumption. A package with a 250mW TDP can support all sixteen clusters in four-core mode (configuration 16/0/0/0, with the number of clusters in each mode designated as 4C/3C/2C/1C). Up to five clusters can be boosted to three-core mode (11/5/0/0) while remaining within the budget. To boost a cluster to one-core mode, however, would require disabling other clusters, resulting in system configuration 9/0/0/1. By boosting clusters, Centip3De is able to efficiently adapt to processing requirements. Figure 2.18 shows a range of system configurations under a fixed TDP of 250mW. On the left are high-efficiency configurations, with more aggressively boosted configurations on the right, which provide single-threaded performance when needed.

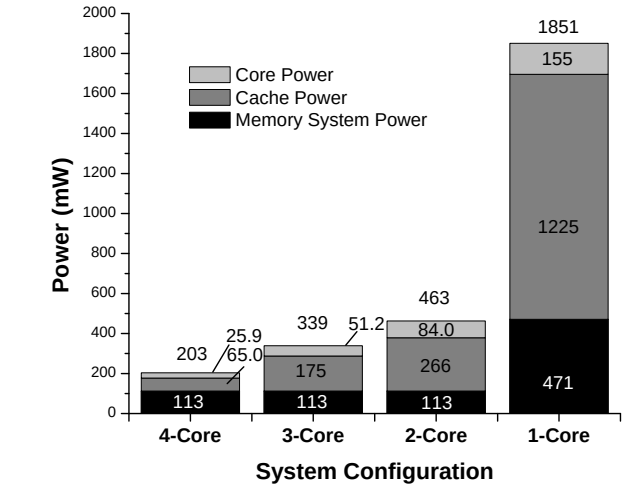
Table 2.3: **Cluster configurations used in power and performance analysis.**

Mode	Core (MHz)	DMIPS /Core	DMIPS /Cluster	DMIPS /W	Cluster (mW)	Core (V)	Cache (V)	Core (mW)	Cache (mW)
4C	10	12.5	50	8800	5.68	0.65	0.80	1.16	4.06
3C	20	25	75	5300	14.2	0.75	0.95	3.20	11.0
2C	40	50	100	4560	21.9	0.85	1.15	5.25	16.6
1C	80	100	100	1160	86.3	1.15	1.65	9.71	76.5

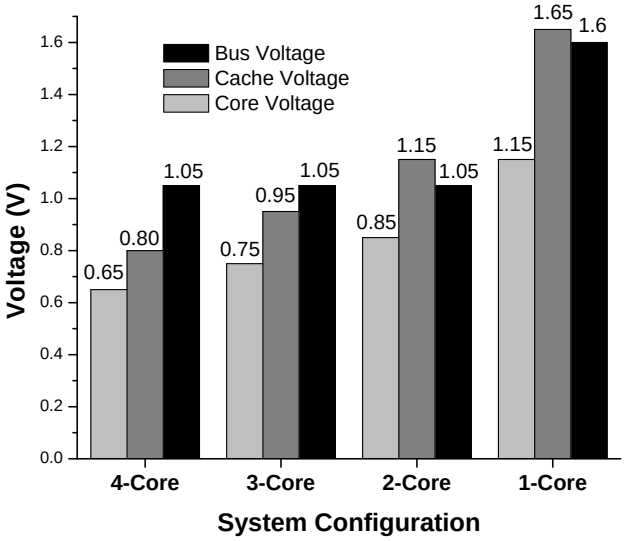
Table 2.4: **Power and performance analysis of selected system configurations.** Total number of clusters is 16 for each system configuration.

Bus Freq	#4C	#3C	#2C	#1C	DMIPS	mW	DMIPS/W	MHz/mW
160	16				800	203	3930	3.15
160	15	1			825	211	3890	3.12
160	12	4			900	237	3790	3.03
160	15		1		850	219	3870	3.10
160	14		2		900	235	3820	3.05
160		16			1200	339	3540	2.83
160			16		1600	462	3460	2.77
320	14		2		900	594	1510	1.21
320	15			1	850	642	1320	1.06
320			16		1600	821	1950	1.56
320				16	1600	1851	860	0.69

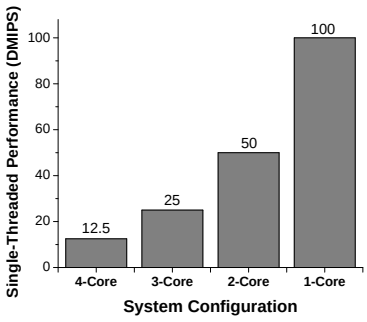
A wider system-level analysis is performed in Table 2.4. A variable number of clusters are boosted from four-core mode to other modes, providing a variety of trade-offs between single-threaded performance and energy efficiency. Four system-level modes are analyzed in detail in Figure 2.17, with power breakdowns, voltages, single threaded performance, energy efficiency, and throughput visualized. An ARM Cortex-A9 in a 40nm process is able to achieve 8000 DMIPS/W [49]. At peak system efficiency Centip3de achieves 3930 DMIPS/W.



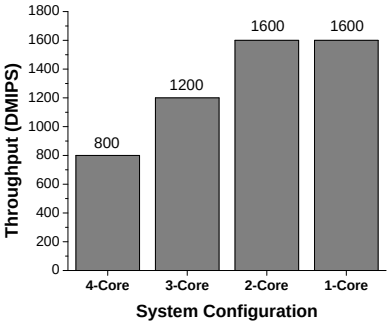
(a) System Power Breakdown



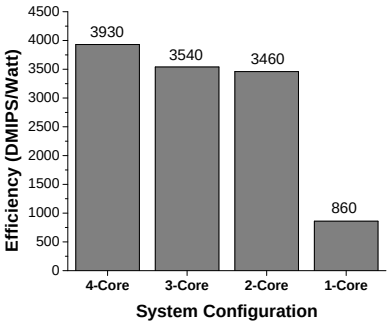
(b) System Voltage Configurations



(c) Single Threaded Performance



(d) System Throughput



(e) System Energy Efficiency

Figure 2.17: **Power and performance results of four system modes from Table 2.2.**

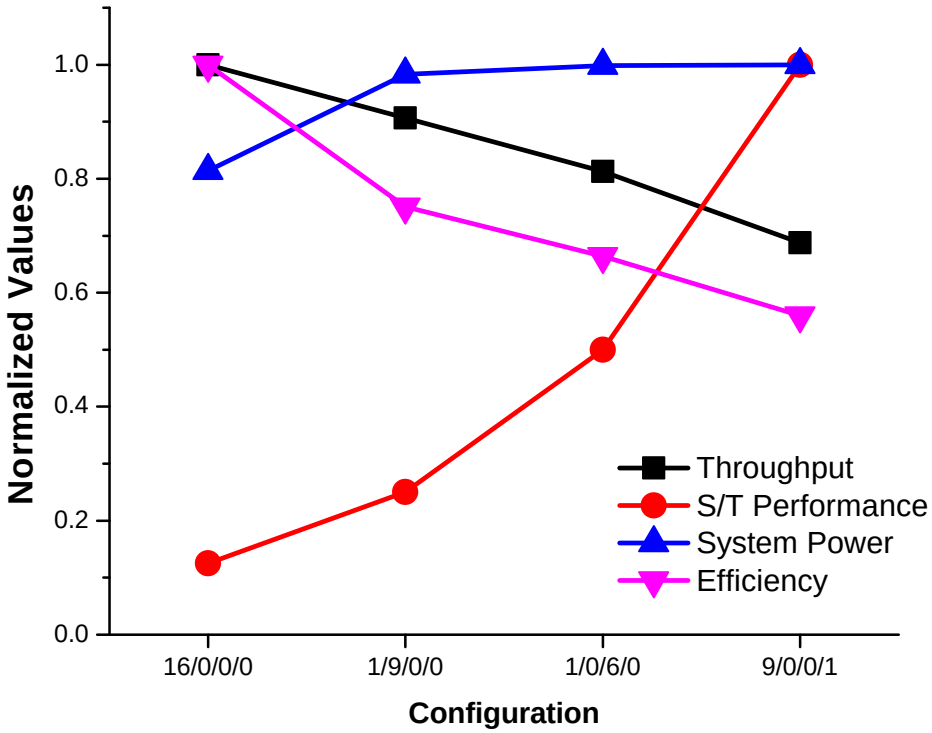


Figure 2.18: **Range of system configurations under a 250mW fixed TDP.** The number of clusters in each mode is listed as 4-core/3-core/2-core/1-core. Each configuration emphasizes a different cluster mode, with the most energy efficient configurations on the left, and the highest single-threaded performance on the right.

2.7 Summary

In this chapter we proposed a cluster-based NTC architecture as a solution for maximizing performance in a TDP-constrained environment. Centip3De was implemented in Tezzaron's 3D stacking process, demonstrating the feasibility of 3D design, particularly when coupled with energy-efficient computing. Issues such as design reuse, floorplanning, and voltage scaling in a 3D environment were discussed. A two-layer system was fabricated and measured, achieving 3930 DMIPS/Watt energy efficiency in a 130nm process. At 46.4M devices, Centip3De is one of the largest academic projects to date.

CHAPTER 3

Safety Razor: Reducing Operating Margins Without Failures

3.1 Motivation

Advanced CMOS technologies have become highly susceptible to process, voltage, and temperature (PVT) variations which cause performance uncertainty. Given these variations, we must add timing margins to guarantee correctness of operation. Figure 3.1 illustrates how these timing margins accumulate. Ultimately, these margins result in lost performance or energy: performance is lost directly through a reduced clock frequency, and energy is lost by operating with a higher voltage than needed to meet non-margined timing requirements.

To address this variation, some manufactures use a "binning" process where each integrated circuit (IC) is tested for power and performance and is assigned to a bin. Each bin

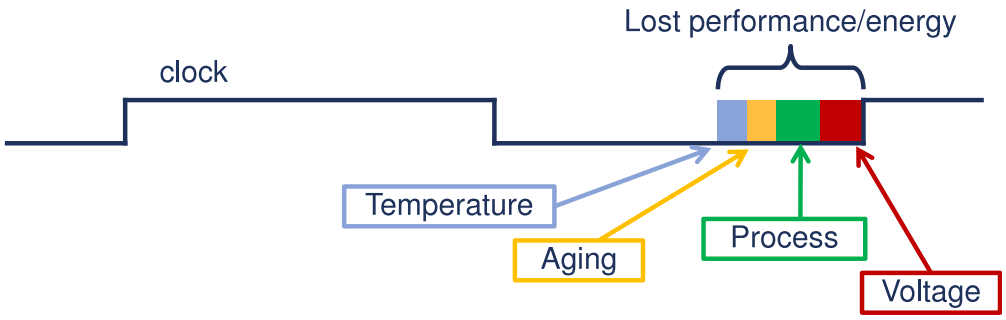


Figure 3.1: **Sources of timing margins in integrated circuits.** Timing margins reduce energy efficiency and system performance.

Table 3.1: **Methods of dynamic variation timing margin reduction.** Significant benefits are highlighted in *italics*. Static variation is accounted for by tester-based tuning in prediction techniques and intrinsically in others.

		Detect and Correct [50–54]	**Prediction via Canaries and Sensors [55–61]**	**Delay Slack Monitoring (This Work)**
Margins Reduced or Introduced	**Global Variation**	*Yes*	*Yes*	*Yes*
	Local Variation	*Yes*	No	*Yes*
	Fast Variation	*Yes*	No	No
	Introduced	None	Sensor Tracking & Sensor Variation	*TDC Accuracy*
Costs	**Area**	Large	*Small*	*Small*
	Invasiveness	Large	*Small*	*Small*
	Tester Time	*Small*	Typically Large	*Small*

is marketed and sold as separate products each with specific voltage and frequency settings. This technique accounts for *static* variations (*e.g.*, process) but it cannot account for *dynamic* variations (*e.g.*, voltage and temperature) which occur during runtime. Additionally, the tester time required to perform binning is cost-intensive so this technique is only suitable for expensive ICs.

Reducing timing margins for dynamic variation requires specialized structures within the IC. Systems with these structures fall into two main categories: "detect and correct" systems often known as "Razor" [50–54], and prediction based systems that use canary circuits [55–58] or environmental sensors [59–61]. A summary of these techniques is shown in Table 3.1.

Detect and correct systems monitor critical paths of the design with specialized registers. These registers detect late transitions at their inputs and trigger a state rollback. Earlier works saved a copy of the state at each register, while later works moved towards an architectural rollback, similar to a branch mispredict, which reduces complexity and overhead.

Although robust, detect and correct strategies have a few salient issues. First, care must be taken to not allow fast "short path" transitions to be flagged as late transitions. In many ICs there are a few registers that are immediately followed by another register in the pipeline. To protect this timing path a hold-time margin must be included, which increases area and power and mitigates some of the benefits of the technique.

Next, electronic design automation (EDA) tools reduce power consumption in designs by downsizing non-critical paths, sometimes with high-V_t device cells, until they are nearly all critical. The result is that nearly every register in the design must be onerously monitored or a select few paths must be monitored while adding a timing margin for non-critical paths. Determining which paths to monitor is non-obvious and is an open area of research [62, 63] and may not be possible for aggressively pipelined designs.

The final two challenges with Razor (and arguably the most challenging) come from the system architecture level. The memory and input/output (IO) interfaces of a processor cannot be easily rolled-back which means that a Razor processor must buffer these interfaces long enough to be sure that those accesses are correct. Depending on the system architecture, this may cause a significant performance impact. The other issue comes from performance guarantees; these are challenging to have when a rollback may occur at any time, perhaps without relation to the program being executed. Performance guarantees may be an issue at the microarchitectural level, where a processor needs to be able accomplish work on time to be in-sync with other parts of the system (*e.g.*, process an entire incoming memory stream), or the software level, where computations need to be completed on time for the user (*e.g.*, a video frame or sound bite).

Canary- and sensor-based circuits use on-die structures that attempt to predict the critical path delay of the design under dynamic variation. Canary-based designs replicate the critical path delay using delay-chain circuits that range from inverter chains to precise critical path replicas. Sensor-based designs instead monitor temperature, supply noise, or other relevant environmental conditions to predict the critical path delay.

The accuracy of these prediction-based strategies depends on canary/sensor variation and tracking. In the case of canaries, the fastest delay of the canary must *always* be slower than the slowest critical path delay under *all* variation corners and conditions. These margins are therefore compounding since the two distributions cannot overlap, which necessitates canary/sensor tuning during tester time to achieve any benefit. Similar to binning, tuning is cost intensive and is not suitable for commodity ICs. After tuning, the canary- and sensor-based designs can only account for global variation, but not local variation (*e.g.*, hotspots) or fast variation (*e.g.*, IR droop).

The advantage of canary- and sensor-based designs is the low design overhead. The microarchitecture of the design does not need to be aware of the canaries or sensors since no rollbacks occur and no additional hold-time constraints are needed. The canaries and sensors are separate from the architecture itself which means that they can be designed separately.

Safety Razor is a hybrid technique that attempts to recover some of the margins that canary and sensor based designs could not while avoiding the design complexity introduced by traditional Razor techniques. This is accomplished by monitoring the delay slack of the critical paths themselves. To measure the delay slack, a time-to-digital converter (TDC) is connected to the data and clock inputs of each critical register. This is similar to the off-chip interface work by Khel [20], while being flexible enough to address an entire processor. Since late transitions can never occur, additional hold time constraints are not required and rollback mechanism is not required, so the microarchitecture of the processor may remain unaware of the delay slack monitoring.

The system architecture is described in Section 3.2, measured results are provided in Section 3.3, and the chapter is summarized in Section 3.4.

3.2 System Architecture

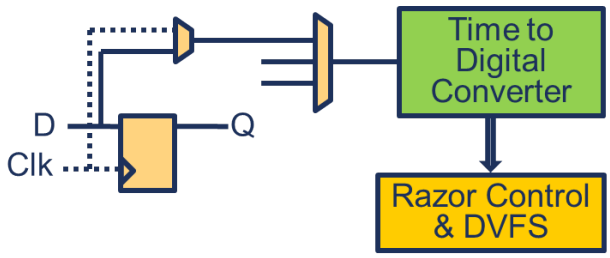


Figure 3.2: **Monitoring system methodology.** Each critical register is connected to a local multiplexer, which are then multiplexed in a tree to the time-to-digital converter.

Safety Razor contains an Alpha64 core as the circuit under test (CUT), with the 512 most critical registers monitored. This number was chosen for flexibility of analysis; fewer could be used for a final implementation depending on needed confidence of information. Each critical register has a multiplexer placed next to it to send the clock or data to a TDC

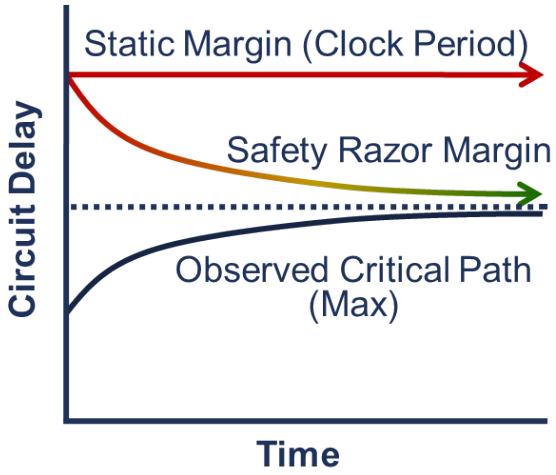


Figure 3.3: **Safety Razor timing margin methodology.** Over time Safety Razor gains confidence that it has seen the critical path and reduces timing margins, gaining energy and performance benefits over a traditional design.

(Figure 3.2). Two TDCs were included, however, only one would be needed for a final design.

The TDC measures the time difference between one of its inputs and its local clock. To determine the delay slack of a register, the delay of the clock from that register is measured first. Once the clock delay is measured, the data delay is measured and the two are subtracted, which yields the timing slack at that register. Since the critical path of that register may not have been exercised at the time that this register was monitored, this is a probabilistic process that gains confidence over time. As shown in Figure 3.3, as confidence increases the margin can be reduced, which improves energy and performance benefits over a traditional system.

To determine the 512 most critical registers, the core was synthesized using Synopsys Design Compiler and run through Cadence Encounter automatic place and route (APR) to generate an initial layout. Once the initial layout was generated, Synopsys PrimeTime was used to identify the 512 most critical registers. Cadence Encounter was then used to reload the layout, insert the needed multiplexers, and complete routing changes for those multiplexers. The locations of these 512 critical flops are shown in Figure 3.4.

For testing purposes a second core is included, which is visible in the die micrograph in Figure 3.5. This "control" core operates the TDC, which allows arbitrary algorithms to be performed. A production design would include a small state machine to operate the TDC,

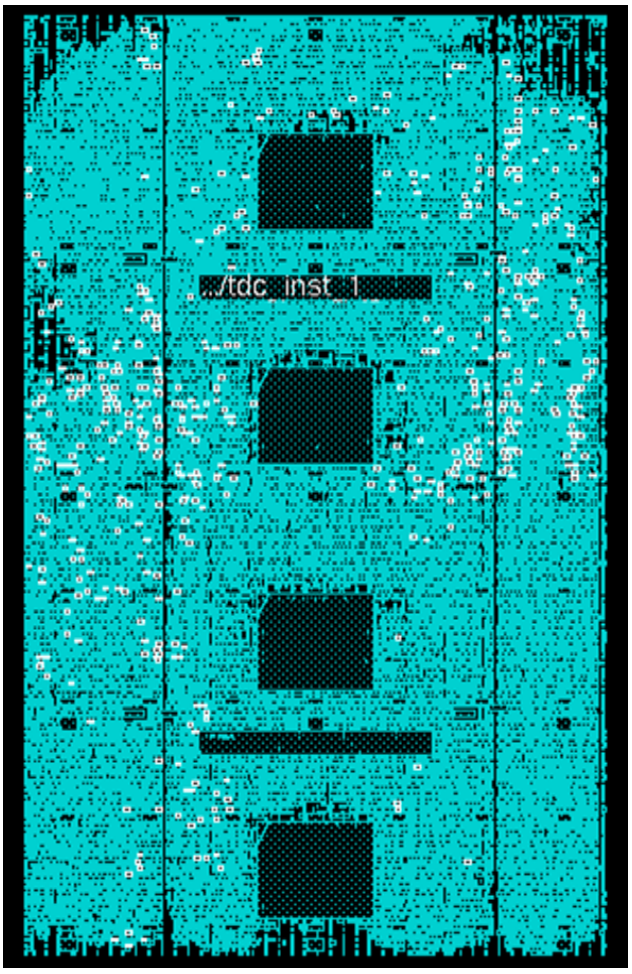


Figure 3.4: **Critical register locations within Safety Razor.** Monitored registers are highlighted in white.

which would be smaller than the TDC itself.

3.2.1 Time-To-Digital Converter

At the heart of the design is an all-digital, self-calibrating time-to-digital converter (TDC) consisting of a 30-bit Vernier chain (VC) with each stage tuned to 5ps increments. The overall measurement window is 150ps, which is sufficient for timing slack measurements in 2 Ghz+ processor systems. A tunable delay chain is also included to allow a wider range of measurements, if necessary. A key concern in TDCs is the need for time consuming and complex calibration, which is especially difficult given the very high resolution demanded by high-performance processors. To avoid expensive off-chip measurements and tester proce-

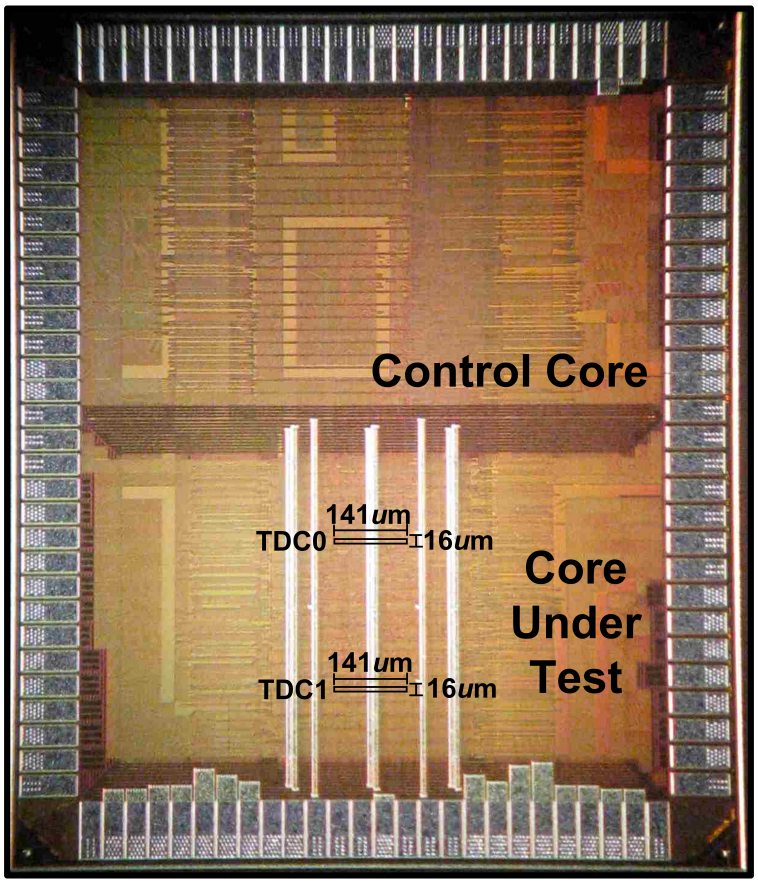


Figure 3.5: **Die micrograph.**

dures, we propose a new approach that allows the TDC to be automatically self-calibrated under the sole control of the processor using only the off-chip crystal oscillator. We use a process of statistical sampling to convert the relatively slow system clock to 1ps accurate calibration steps. The proposed approach was implemented on a 64-bit Alpha processor and can complete full self-calibration of the entire TDC in five minutes with approximately 1ps accuracy. Calibration can be performed in the field during idle periods, incurring no additional tester time or system downtime. Typical delay slack margins remain in place until the TDC is calibrated.

The TDC architecture is shown in Figure 3.6. A measurement is initiated using the start signal, which allows the next clock edge to pass through a latch and into the device. The clock passes through the tunable delay chain, which allows coarse alignment of the detection window. The VC is calibrated so that each stage delays the clock by 5ps relative to the data, resulting in a forward sampling of time. The clock then exits the VC and triggers a TDC

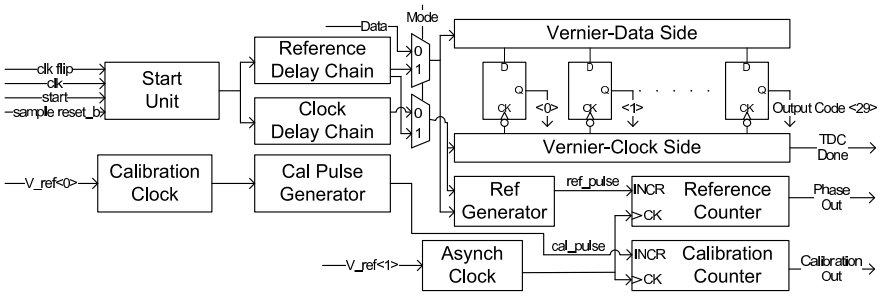


Figure 3.6: **Time-to-digital convert architecture.** The Vernier chains measure the time difference between *clk* and *data*. The counters and pulse generators are used to measure (and calibrate) the delays in the Vernier chains using statistical methods.

done signal, which requests that the processor read the latch states. Each delay element in the VC is tunable using eight identical processor-controlled capacitor loads, which are designed to induce 1ps shifts in delay. A uniform load structure was chosen in lieu of a more compact binary weighted scheme to ensure monotonicity, which greatly eases automation of calibration.

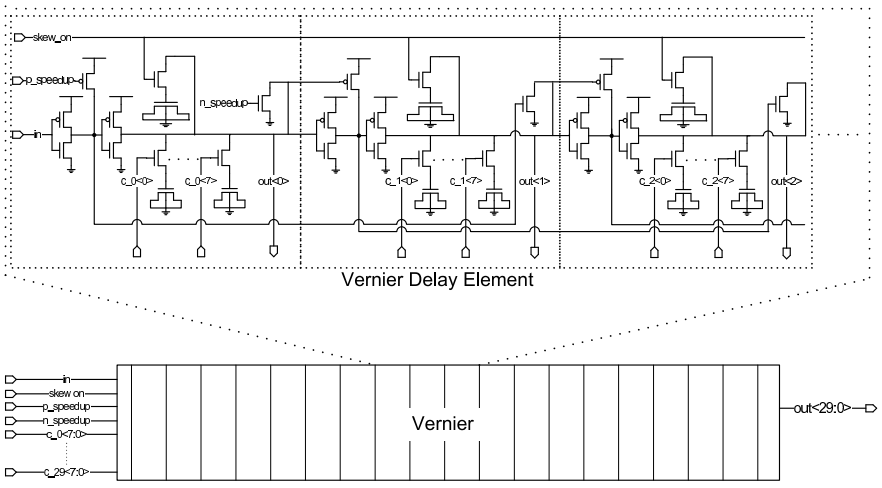


Figure 3.7: **Vernier chain schematic.** A single side of the Vernier chain is shown. Eight tuning capacitors each representing 1ps of delay are included for each stage. The speedup path allows falling transitions to skip every other stage of the chain.

Narrow pulses (glitches) can occur on data signals and must be monitored since they could be captured by a pipeline register if delay slack margins are reduced too far. To prevent pulses from collapsing in the VC due to rise-time/fall-time imbalance, the chain lengthens pulses by accelerating falling transitions forward using speedup signals as shown

in Figure 3.7. Because of this, the TDC is calibrated for rising transitions - we include an XOR gate in order to monitor falling transitions. The metric of interest is the slack between the latest arriving input signal and the clock edge at the register input. To compute this metric, a multiplexer local to the monitored register selects between clock and data, and we subtract the two measured delay values.

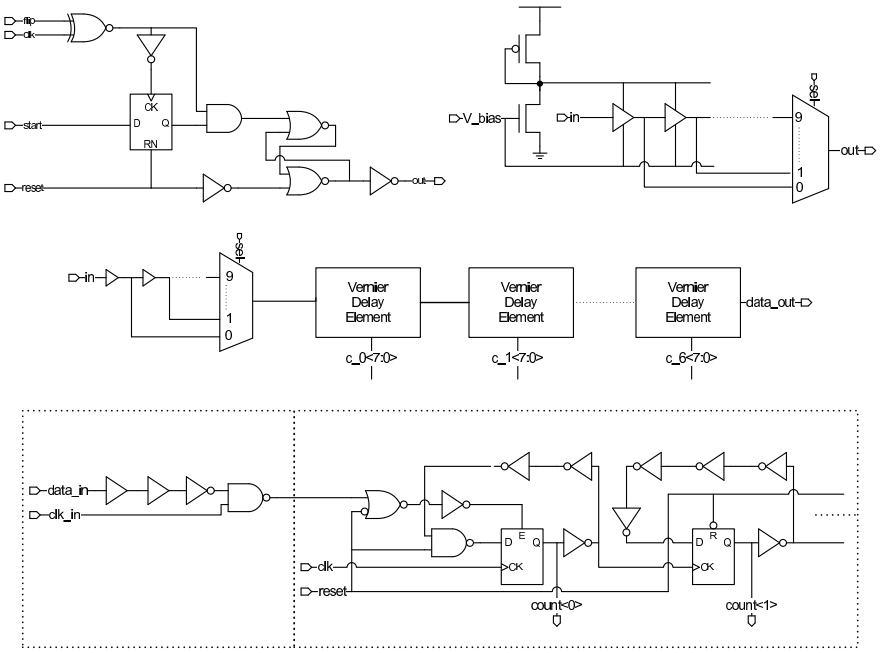


Figure 3.8: **Schematics of the start unit, clock delay chain, reference delay chain, and counters.** The start unit is designed to trigger an SR latch synchronously with the clock edge. The reset is asynchronous. The clock delay chain has a wide range of tunability via current starving and coarse grain multiplexing. The counters use a ripple based design for high-speed clocking, and the first stage is optimized to capture short pulses. A pulse generator is shown on its input.

The VC is calibrated using a reference data transition (*ref_tran*) generated by the reference delay chain (RDC in Figure 3.6). The RDC is trigged by *clk* and feeds test transitions into the data input of the VC. First, we find the zero point of the RDC (which is tunable in 1ps increments) by increasing delay until the zeroth bit of the VC shows equal probability of reading zero and one. The goal is then to shift *ref_tran* in exact increments of 5ps and tune the subsequent bits of the VC in the same manner. To accurately measure shifts in *ref_tran* under PVT variation, a measureable reference pulse (*ref_pulse*) is generated by subtracting *clk* and *ref_tran*. The signal *ref_pulse* is intentionally lengthened to ensure non-zero pulses

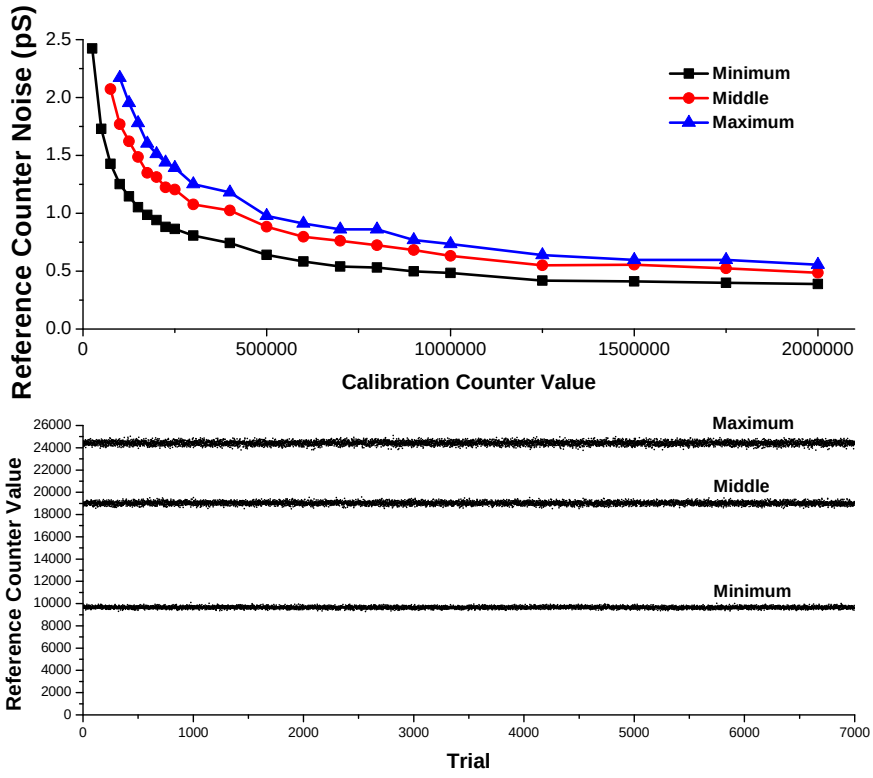


Figure 3.9: **Reference delay chain history and noise.** Above: the reference delay chain delay is measured 7000 times, each counter value is plotted showing the trend. Below: the noise in the reading depending on sample size is shown. Since the delay measurement is a ratio, *i.e.*, a fraction of a number of samples, the noise can be reduced through increased measurement length. A measurement of 500,000 samples yields sub-ps noise.

when *ref_tran* and *clk* are aligned. As explained below, the length of *ref_pulse* is computed by comparing it to a known calibration pulse (*cal_pulse*) using statistical sampling, which is generated using a reference off-chip crystal. Although the exact relationship between the length of *ref_pulse* and the alignment of *ref_tran* cannot be accurately determined, it is unimportant since we require only that a 5ps shift in *ref_tran* will result in an identical 5ps shift of the *ref_pulse* length. Hence, we search for an RDC setting that produces an exact 5ps increase in *ref_pulse*. This ensures that *ref_tran* is delayed by an exact 5ps step and allows the next bit of the VC to be tuned.

To determine the length of *ref_pulse*, a start signal generates a single *ref_pulse* and *cal_pulse*, which are each fed to the enable input of identical up counters. A slow asynchronous sampling clock generated locally provides the clock input for both counters. Since it has no relationship to either pulse source, it statistically samples both signals over many

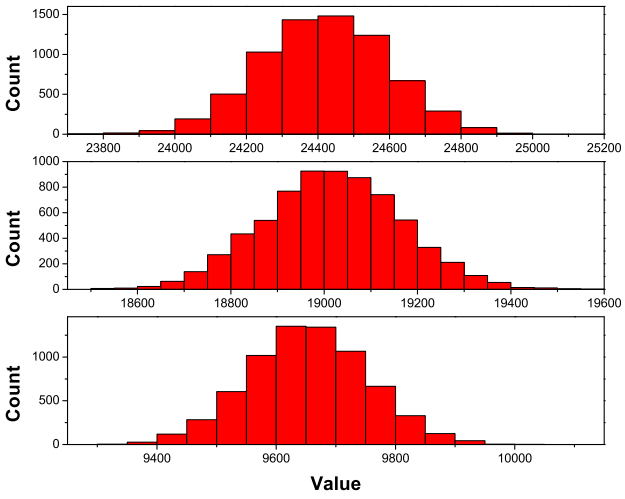


Figure 3.10: **Reference delay chain reading histograms.** Histograms are shown for different reference delay chain readings. The readings show proper Gaussian distributions, making them ideal for statistical noise reduction.

trials (1M in our tests). This technique allows us to compare the length of *ref_pulse* relative to *cal_pulse* with sub-1ps accuracy despite pulse widths of hundreds of ps. Calibration error is not cumulative as the process moves through the VC since each *ref_pulse* is compared to the length of the zeroth-bit pulse.

Figure 3.5 shows the implementation of the proposed slack monitor in an Alpha processor fabricated in a 45nm process. The core interacts with the TDCs through memory mapped IO. In Figures 3.9 and 3.10 we analyze the accuracy of the RDC statistical sampling method. After setting the RDC to a fixed phase pulse width (min, middle, max), we swept a range of calibration counter values and plotted the standard deviation of the reference counter after the calibration counter reaches its target value; trials continued until the calibration counter reached a particular value and then stopped, resulting in a corresponding reference counter value. The stability of readings indicates a deviation below 1ps for a calibration counter of 500,000 samples. Additional accuracy can be gained with exponentially larger calibration counter values, or by using stronger statistical methods than averaging. In this implementation, the reference clock was implemented with a current starved ring oscillator; a crystal oscillator would provide further accuracy.

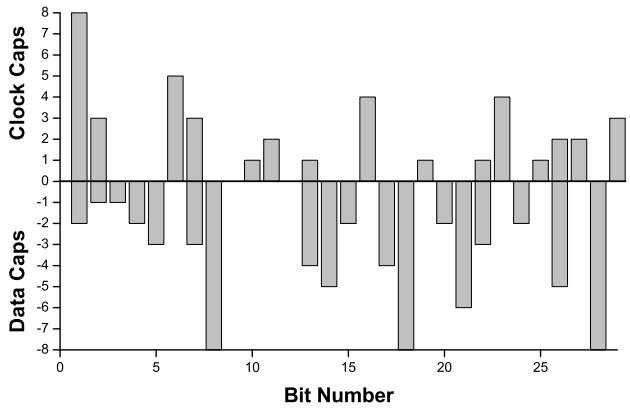


Figure 3.11: **Sample settings for a Vernier chain.** Shown are the calibration settings for a single Vernier chain. The caps are set using a hill-climbing methods resulting in caps being used on both sides of the Vernier chain in some cases.

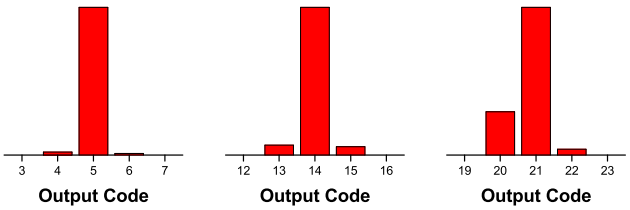


Figure 3.12: **Output code histograms.** Histograms are shown for a selection of output codes. Noise increases as the signals travel through the Vernier chain.

3.3 Measured Results

In Figures 3.11 and 3.13 we show the calibration settings of a TDC VC delay chain. The skew signal for the clock chain side is enabled to provide a baseline delay of 5ps. To address local variation, the 1ps calibration capacitors are enabled as needed on either chain to obtain equal probability of a zero or one reading when the reference delay chain is stepped with 5ps increments. In this experiment we ran 10,000 trials to check the 50% probability condition (corresponding to a count of 5000) for each bit in the VC chain. Each time the condition is checked we try to find an improvement by enabling another capacitor on one side of the chain. For instance, if a count of 8000 is recorded, then the data transition is coming before the clock transition too frequently, so we enable a data capacitor to try to reduce the count. Once there are no more capacitors available, the configuration that yielded the best 50% condition is restored and the next bit of the VC is calibrated.

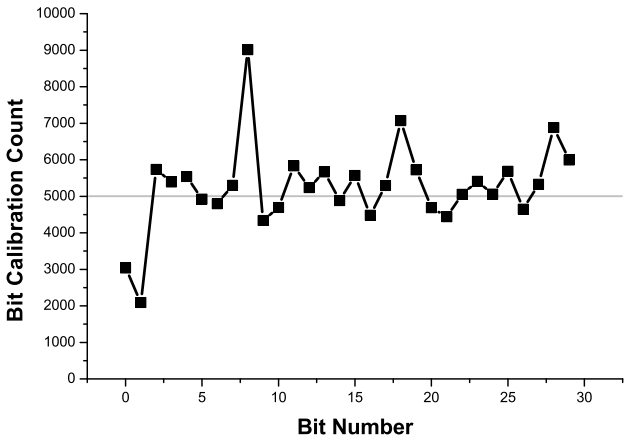


Figure 3.13: **Sample calibration count readings for a Vernier chain.** Shown are the calibration count readings for single Vernier chain. Ideally, each bit should read 5000 (*i.e.*, 50%) after calibration.

In Figures 3.14 and 3.12 we use the reference delay chain to sweep 1ps step inputs through the VC and show output codes. Monotonicity is guaranteed due to the calibration method and also shows excellent linearity.

Finally, the use of the TDC is illustrated by using it to measure a timing slack distribution of the most critical paths in the Alpha processor, as shown in Figure 3.15, illustrating the feasibility of Safety Razor.

3.4 Summary

In this chapter we described Safety Razor and its novel self-calibrating time-to-digital converter. Safety Razor is a hybrid approach that achieves many of the benefits of both canary- and Razor-based timing margin reduction approaches, while avoiding the most serious disadvantages of each. Safety Razor was fabricated in a 45nm process, with silicon measurements provided for the time-to-digital converter and timing distributions of the Alpha core. The time-to-digital converter achieved sub-picosecond calibration accuracy via statistical sampling methods, which enabled the accurate measurement of critical path delay slacks in the Alpha processor.

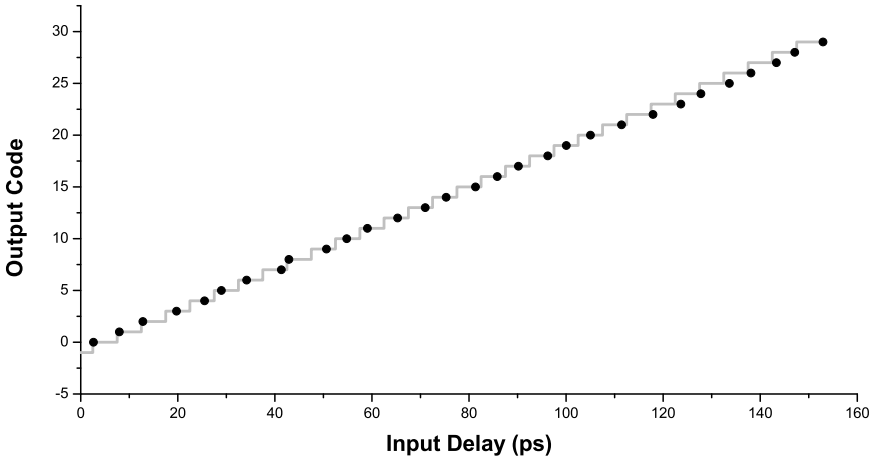


Figure 3.14: **Input output characteristics of the TDC post-calibration.** The output codes are shown for a range of input delays. Good linearity is achieved, as shown by the grey line representing ideal behavior.

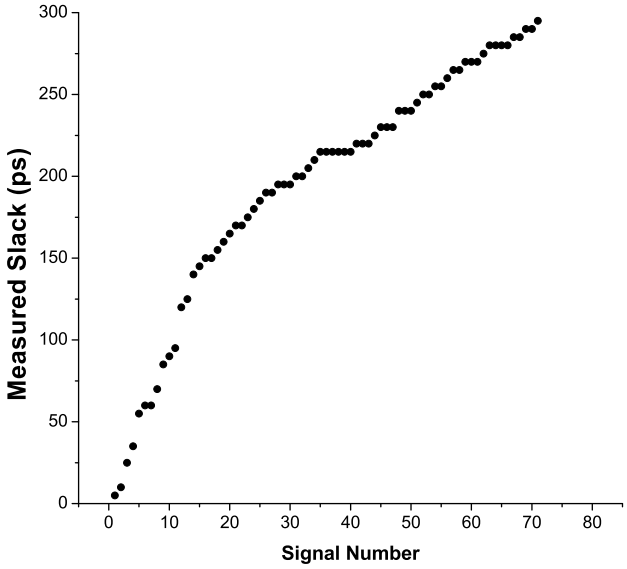


Figure 3.15: **Slack distribution measurement.** A distribution of delay slack was measured for the core under test.

CHAPTER 4

A Highly Resilient Routing Algorithm for Fault-Tolerant NoCs

4.1 Motivation

As the semiconductor industry moves further into the nanometer regime, two salient problems arise: efficiently connecting the increasing number of on-chip resources and effectively managing decreasing transistor reliability. Current methods of connecting on-chip resources, such as buses and crossbars, may soon be precluded by the delay of long wires and the large number of elements that must communicate with each other. Networks on Chip (NoC) help mitigate this problem by decentralizing and distributing communication across the chip using a lightweight networking protocol, resulting in a high throughput, scalable solution.

NoCs use a shared interconnect system to transmit packets of information between intellectual property components (IP, *e.g.*, processor cores, caches, ASICs, *etc.*) through a distributed system of routers connected by links. Data sent through the network is converted into a packet, a formatted block of data, by a network adapter. Each packet is then labeled and divided into a sequence of uniformly sized flow units (flits) and sent from the adapter to a router [64], from where it travels from router to router through the network.

While the NoC approach has been increasing in popularity with commercial chips such as Tile64 [65] and Polaris [66], it is threatened by the decreasing reliability of aggressively

scaled transistors. Transistors are approaching the fundamental limits of scaling, with gate widths nearing the molecular scale, resulting in break down and wear out in end products [22]. Permanent faults due to device wearout are caused by mechanisms such as negative bias temperature instability (NBTI), oxide breakdown, and electromigration. These device level failures have architecture level ramifications, as a single faulty link or router will cause an entire NoC to fail, halting all traffic. Future processor technology generations will require significant error tolerance to many simultaneous faults.

On-chip networks provide excellent opportunities for building a reliable system, as they provide redundant paths between IPs [67]. As the "glue" that holds a chip together, a NoC should be highly resilient to hardware failures and able to work around faulty routers and links. In contrast, a faulty IP with even little or no protection can be disabled and isolated by the network, thus promoting NoC as a reliable system platform. A disabled IP would not hinder network performance, as the associated router can still be used. An ideal network should be able to diagnose where faults are in its own components and then reconfigure to mitigate those faults, maintaining full connectivity when possible. To avoid single points of failure, the reconfiguration process should be distributed and performed within the network itself.

We present a distributed routing algorithm for networks on chip, allowing a network to reconfigure around faulty components. Our solution's novelty lies in its ability to overcome *large numbers of faults* in a fine-grained fault model without using virtual channels or adaptive routing, and in a *distributed nature*, which avoids single points of failure. Given only local information, the algorithm runs in lockstep at each network router to collectively reconfigure the network's routing tables. Using a hardware implementation is more reliable as it may be unknown what IPs are safe for routing computation, and a software implementation would still require some degree of hardware support. The algorithm is implemented with less than 300 gates at each router, thus making it a low-overhead approach for both simple designs (even those without virtual channels) as well as complex ones. We found that networks routed using this algorithm are 99.99% reliable when 10% of the interconnect links have failed.

4.2 Related Work

Our algorithm adopts the *turn model* originally proposed by Glass and Ni for adaptively routed networks in [68], which prevents network deadlock by disallowing various network *turns*. Later, in [69], they showed how the technique can be applied to n-mesh networks to tolerate (n-1) router failures (1 router failure for 2D-mesh). Their strategy uses no virtual channels but requires adaptive routing, whereas our algorithm requires neither. Additionally we are not limited to any particular number of failures.

Jie Wu uses the odd-even turn model to address convex, disjoint fault regions that do not lie on the mesh boundary [70]. Although his technique does not require virtual channels, the fault-requirements are strict. The algorithm that we present has no requirements on the fault patterns.

A series of other works present solutions to adaptively route around router *fault regions* while enduring various restrictions and requirements [71–80]. Packets are routed normally until encountering a fault region, and are then transmitted along the edge of that region. Initially, only rectangular, disconnected, non boundary regions could be tolerated while requiring a large number of virtual channels. Sometimes fully functional routers had to be disabled to meet the shape requirements. Recent work extends these techniques to router failures of arbitrary patterns using as few as two virtual channels. These additional virtual channels do not add to the performance of the system, however, as they are reserved for fault tolerance. Our algorithm requires no virtual channels reserved for fault tolerance, so any virtual channels added to the system add to performance. Moreover, our algorithm allows link-level failures and does not require adaptive routing, nor do we disable functional routers.

Ho and Stockmeyer present a strategy that disables the IPs connected to a few *sacrificial lamb routers*, which helps prevent deadlock [81]. They demonstrate high reliability for a small number of faulty routers (3%), which is suitable for their application in the Blue Gene supercomputer. Our algorithm targets reliability at higher failure rates while not disabling resources.

Other approaches use additional virtual channels combined with adaptive routing to provide fault tolerance. Pruente *et al.* [82] present a technique that tolerates link-level failures

using a relatively complicated algorithm and two virtual channels (one reserved for fault tolerance), while Gomez *et al.* [83] are able to tolerate up to five link failures with the addition of one virtual channel.

In contrast with previous work, our solution requires no virtual channels, no adaptive routing, no particular fault restrictions, and no disabled routers. Many these works provide 100% reliability when conditions are met - while our technique can not guarantee that, experimental results show over 99.99% reliability when up to 10% of interconnect links have failed. Moreover, we provide hardware implementation results proving the feasibility of our approach.

4.3 Routing Algorithm Overview

The routing algorithm presented in this chapter reconfigures network routing tables in an offline process. It consists of a basic routing step and a number of rule checking steps. The rules constrain the basic routing step depending on the network topology and existing faults in order to safely redirect traffic around failed network resources, while the checks determine which rules to use. The algorithm is specifically designed for deterministically routed on-chip networks in SoC and CMP applications. Each router in our network contains a routing table (Figure 4.1), which lists an output port for each destination in the network. Packets traversing the network have a header flit with a destination ID field that is used to look up what direction to go at each router. Virtual channels are not required, but could be used to provide additional performance. The algorithm is implemented as a small hardware module in each router that runs in a distributed lock-step fashion.

We model faults as link-level hard failures, thus each bidirectional link can be individually bypassed, allowing routers to be only partially functional. An entirely non-functional router is represented by marking all four of its links as faulty. No restriction on the number or location of faulty links is assumed, but the routers must know which of their adjacent links are faulty. Each router works with its neighbors to collectively reconfigure their routing tables based only on this information. In addition, it is assumed that the routers know when they need to invoke the algorithm and how to resume operation after reconfiguration finishes.

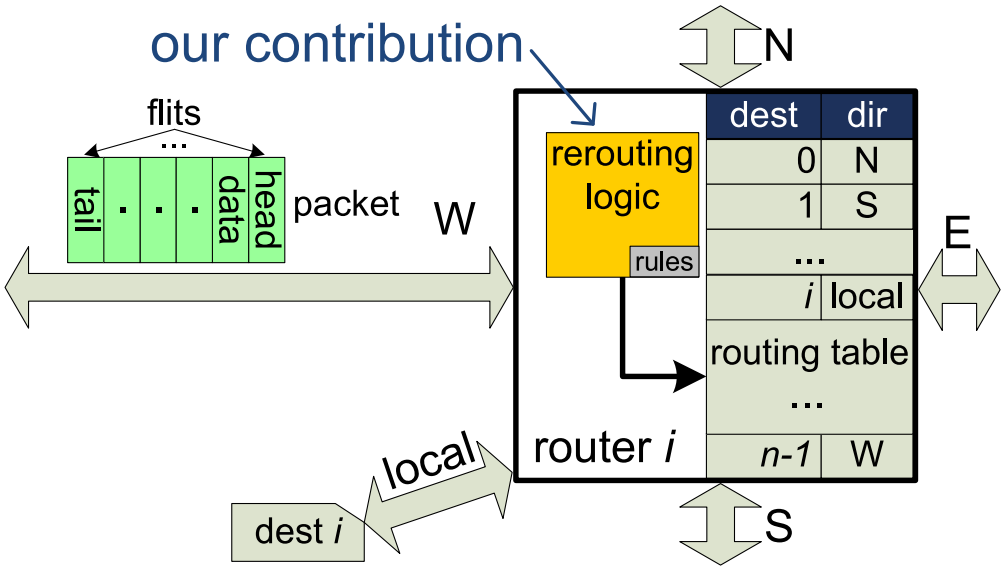


Figure 4.1: **NoC router.** When a packet enters the router, its header is decoded and the destination is looked up in the routing table. The packet is then transmitted in the direction specified in the routing table. Our algorithm rewrites this table when faults appear in the network.

Failure detection and diagnosis, computation checkpointing, *etc.*, are topics that have been extensively explored in other work [84] and are not discussed in this chapter. The algorithm is comprised of multiple iterations of a *basic routing step* – a procedure that updates an entry in all routing tables for a particular destination. In the remainder of this section we describe the basic routing step as well rules to constrain it. We later show how to use these rules for specific topologies, 2D-mesh and 2D-torus, in Sections 4.4 and 4.5 respectively.

4.3.1 Basic Routing Step

The basic routing step is shown in the second half of Figure 4.2. Each router selects the best direction to route a given destination based on information provided by its neighbors, then updates the corresponding *entry* in its routing table. Routers communicate through *flags* – one-bit signals transmitted to adjacent routers. At any point during the execution of the algorithm, entries in a routing table are marked either *valid*, if the corresponding destination is known to be reachable, or *invalid* otherwise. In the *valid* case the entry also contains the direction that packets must take to reach that destination.

```
 1: function main () {
 2:   foreach (rule) {
 2:     foreach (router) {
 3:       run_rule_check(rule, router)
 4:     }
 5:   }
 6:   foreach (dest) {
 7:     run_basic_routing_step(rules, dest)
 8:   }
 9: }
10:
11: function run_basic_routing_step (rules, dest) {
12:   if (dest == self) {
13:      write_entry(‘‘local’’)
14:   } else {
15:      write_entry(‘‘invalid’’)
16:   }
17:   for num_router cycles {
18:     if entry != ‘‘invalid’’ {
19:       transmit_flags(rules)
20:     } else {
21:       check_for_flags(rules)
22:       write_entry(priority_flag)
23:     }
24:   }
```

Figure 4.2: **Pseudocode for rerouting algorithm.** The function *run_basic_routing_step* will often be reused by *run_rule_check*.

To determine the best path to the destination, all routers start by initializing the corresponding entry to invalid, except for the router connected to the destination locally. This router marks the entry as valid, and the proper direction in the table (**Entry Initialization**). All routers then repeat the following two steps until every router has had an opportunity to update its entry:

1. **Flag transmission**. During the flag transmission step, all routers whose destination entry is valid will send a flag to all of their adjacent routers. The other routers are silent.

2. **Routing entry update**. In this step, all routers who have (i) an invalid routing entry for the destination under analysis and (ii) have received a flag through any of

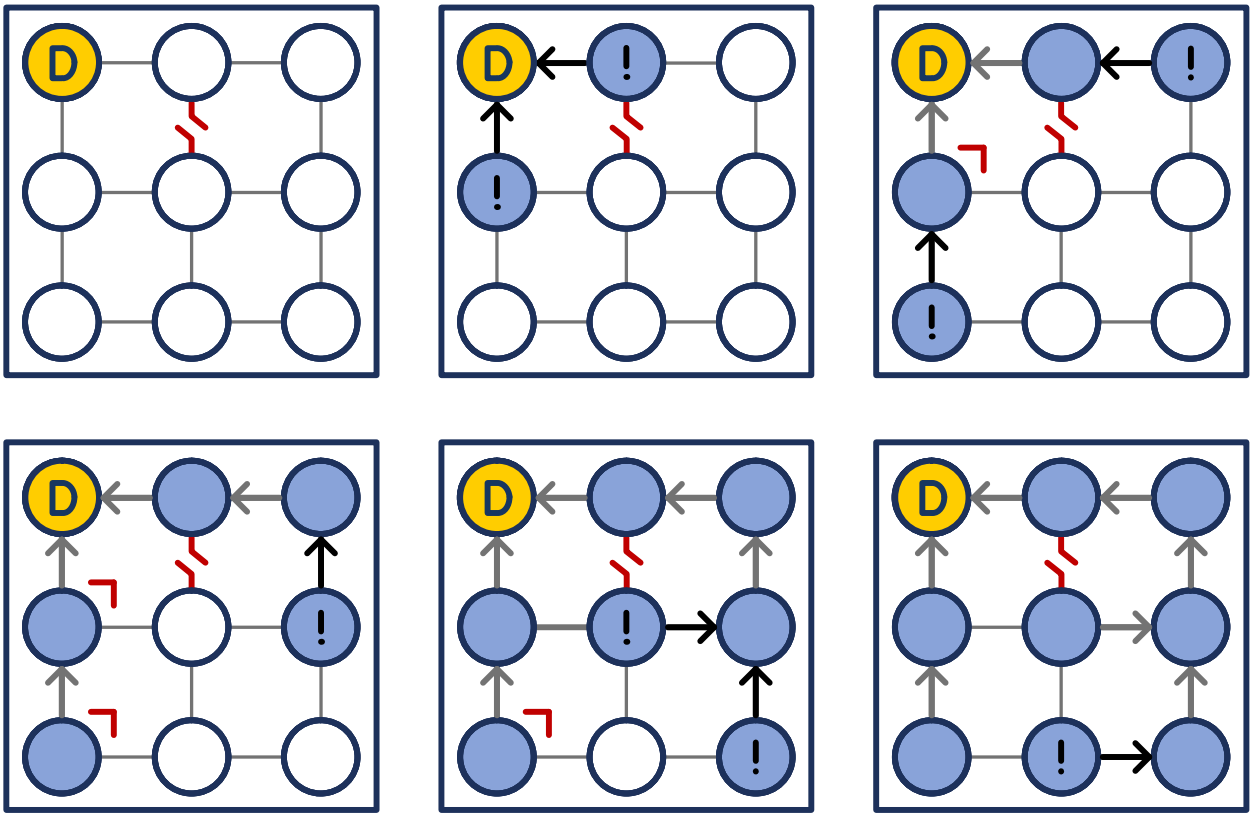


Figure 4.3: **Example of six iterations of the algorithm's basic step.** The only rule enforced disallows the Northeast corner turns, so the Southwest routers must go around the East side of the faulty link. "D" denotes the destination being routed, "!" denotes routers that wrote a valid entry that step, and "¬" denotes places where a turn rule prevented a flag from propagating.

their links in the previous step, update their entry to valid with the direction of the incoming flag. If a router has received a flag through multiple links, a priority selection procedure is used to select the preferred routing direction. If a router has not received any flag, or already has a valid entry, it does not take any action.

Each panel of Figure 4.3 shows a single iteration of the basic routing step for a 3x3 mesh network. For an N router network, the routing steps above must be repeated $N-1$ times to cover the worst case scenario, where the routers are connected as a long chain. At the end of this process, if the router still has an invalid entry for the destination under analysis, then that destination is unreachable from that router.

4.3.2 Rules for the Basic Routing Step

To avoid deadlock loops forming while performing the basic routing step, each router must also keep a set of rules – a list of disallowed links and turns.

Links can be disallowed in the basic routing step by not transmitting flags across them, or by not accepting transmitted flags. Therefore a link can be disallowed by either router that it connects.

Turns comprised of a path through two links connected by a router must be disallowed by their center router. The center router can disallow a turn between two links by not transmitting flags from one to the other, *i.e.*, if its valid entry points to the second link, it will not transmit flags to the first link.

These sets of rules are determined on a per-router basis depending on both the topology of the network and the present faults. In our subsequent examples of 2D-mesh and 2D-torus networks, we reuse the basic routing step to evaluate which rules are necessary to avoid deadlock. Each router will start with a set of default rules, removing or adjusting them based on the set of faulty links.

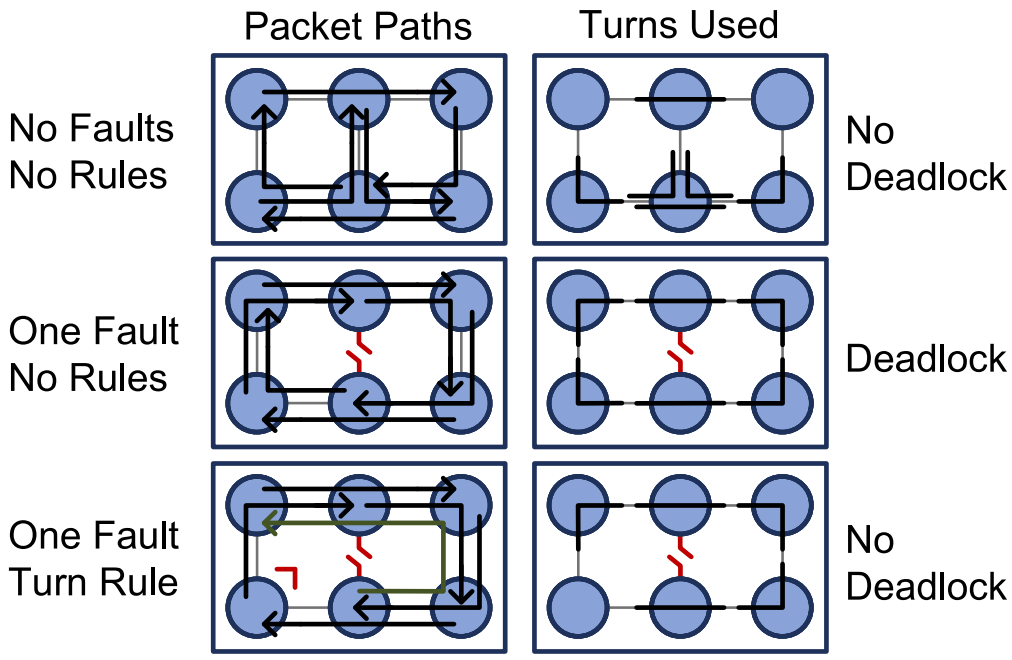


Figure 4.4: **Disallowing turns to avoid deadlock.** Three examples are shown routing the same set of packets on the left, and the turns that they use on the right. Top row shows how the basic routing step will perform with no rules and no faults - no deadlock loops appear in the turn graph. The center row shows how routing the same packets results forms a deadlock loop when a fault is present. All packets take the only other available 2-hop route, and use overlapping resources which will eventually conflict with each other indefinitely. The third row shows how we disallow a turn to make one packet to take a longer path around the fault (green), which prevents a deadlock loop. "¬" denotes places where a turn rule prevented a flag from propagating.

4.4 2D-Mesh Routing

2D-mesh is a common network topology for large scale chip multiprocessors due to its simple physical implementation. In this section we first discuss the natural loops that form in a mesh, and the rules required to prevent them. We then investigate situations when rules at individual routers need to be removed to maintain connectivity. Finally, we discuss a pathological case, as well as its solution. An example of mesh routing is provided in Figure 4.3.

4.4.1 2D-Mesh Rules

A fault-free 2D-mesh network routed according to our algorithm will be deadlock-free without enforcement of any rules since S-E-W-N priority leaves the Northwest and Northeast corners naturally unused (Figure 4.4, first row). However, a single fault in the network may cause a deadlock loop to form (Figure 4.4, second row). Deadlock occurs when a loop of utilized turns is formed. The basic routing step naturally uses minimal length paths, and every turn in the loop around the fault is the minimal length path between the two routers that it connects, thereby creating deadlock.

Glass and Ni show that disallowing a pair of turns in an adaptive routing network prevents deadlock situations [68]. One turn must be disallowed for the clockwise direction, and another for the counter-clockwise direction. In our experience, the best results were obtained when both the clockwise and counter-clockwise turns at the same corner were disallowed. An example of how disallowing a pair turns removes deadlock is shown in the bottom row of Figure 4.4.

In our examples we choose to disallow the Northeast corner turns, which are North→East and East→North.

4.4.2 Selectively Removing 2D-Mesh Rules

Strict adherence to disallowed the turn rule may produce an inconsistent network, where *consistency* means that if one router can reach another then a return path also exists. In Figure 4.5, a single faulty horizontal link on the North edge of the network prevents six of the routers from obtaining a valid path to the Northwest router. All of the routers on the West edge of the network can reach this router by directing traffic to the North, however, since the East→North turn is disallowed, these fringe routers never transmit a flag to the East, cutting off the rest of the routers.

In order to prevent this scenario we must identify routers where the turn rules should be removed to maintain a consistent network. To do this, we must check each corner by trying route from one end of the corner to the other using the basic routing step. If this cannot be successfully accomplished with the disallowed turn, then the corner turns must be allowed

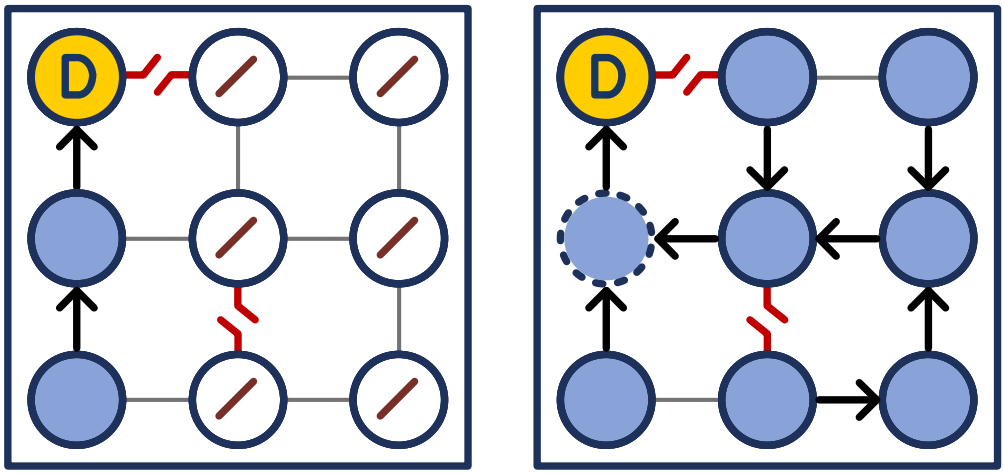


Figure 4.5: **Selectively removing rules to allow full network connectivity.** Strict adherence to the disallowed turn rule may result in a case where some routers are not able to reach the Northwest destination, although possible paths exist (left). This occurs because the routers on the western edge have the East→North turn disabled to avoid deadlock. Therefore, we remove the turn rule at the West router, restoring network connectivity (right). "D" denotes the destination being routed and "/" denotes routers unable to reach the destination.

to maintain network consistency and both turns for the corner are then allowed. The center router knows if the corner was successfully routed based on the flags that it receives from the two other routers. We check one corner router at a time, so the minimal number of rules are removed in sequence. If this were not done then both fringe routers would have their turns allowed and the second faulty link in the example would cause a deadlock.

4.4.3 Pathological Case

A pathological case infrequently arises in large networks with many faults. As shown in Figure 4.6, deadlock loops can form when two subnetworks are connected by a router with the turn rule removed. Effectively, the deadlock loop passes through the connecting router twice, as if folded over a point. We reduce this problem by using a routing priority that disfavors a different corner - in our example we use S-E-W-N priority to disfavor the Northwest corner. The pathological case can still appear if there is a single fault inside of each of the subnetworks, since the Northwest turns will still be used in that case, similar to the second row of Figure 4.4. We address this by formally disallowing a different corner for one of the subnetworks. Once we have changed which corner is being disallowed for this

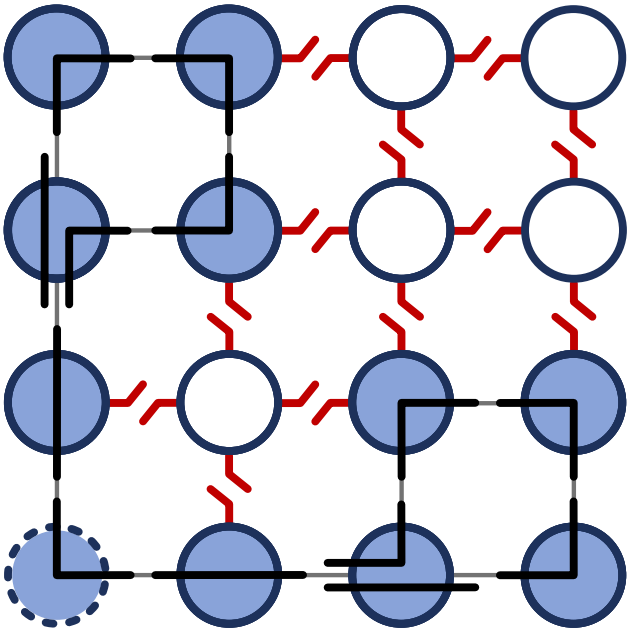


Figure 4.6: **Pathological case for large networks with many faults.** The corner rule for the Southwest router was removed as desired, but a deadlock loop forms by passing through that router twice. We adjust the rules for one side to break the loop.

part of the network via a directed broadcast, we must check each of these new rules.

The pattern can also repeat inside one side for a large network, so we allow the turn to be changed back to the original turn when new rules need to be removed. This part of the algorithm can loop by changing between these two corners as long as it needs. Any router that does not have its corner rule modified becomes "fixed", and is no longer changeable so that forward progress is made. In our implementation we switch between the Northeast and Northwest corners, broadcasting the corner-change signal to the East/North and North/West respectively.

4.5 2D-Torus Routing

Tori are very similar to meshes so we start with the same rules described in our 2D-mesh example. Unique challenges for the basic routing step (described in Section 4.3.1) are present, since a torus network automatically forms deadlock loops even in absence of faults.

4.5.1 2D-Torus Rules

Torus networks form loops around the outside of the network by continuing in the same direction until reaching the same router again. We address these loops with the addition of link rules. We first disallow all wrap-around links along the top edge of the network, then disallow one horizontal link in each row of the network. The horizontal links prevent a loop from forming in the same row. The vertical link rules along the North edge prevent a zigzagging pattern from looping around the network, as well as loops that would form in the same column. We choose a staggered pattern for the horizontal links in order maintain the performance provided by the torus topology.

Each of these new rules (vertical link and horizontal link) needs to be checked, as shown in the first part of Figure 4.2. Horizontal link rules can be checked first, and all in parallel. If there is a broken link in the same row as a horizontal link rule then it is not necessary to have the rule since the broken link provides the same benefits. We can accomplish this check with a horizontal broadcast, where knowledge of broken links propagates horizontally and lifts any horizontal link rules. Since this broadcast starts at the ends of each broken link it is guaranteed to reach every router in the row.

Vertical link rules can be checked similarly to the corner rules. One side of the link maintains the rule - we route the other side of the link and check if this side ever has a valid entry. If it does not then the link is needed for consistency and the rule is discarded.

The horizontal and vertical rules only need to be checked once, regardless of whether the corner rules are changed or not. They are put in place to create a network that can be successfully routed by manipulating the corner rules.

4.5.2 Corner Rule Consistency

The 2D-torus topology adds the possibility of corner evaluation being inconsistent. Each corner has two turns, so the corner could be evaluated by routing one turn or the other. Meshes are always consistent, *i.e.*, routing either turn yields the same result (routable or unroutable). Tori can be inconsistent in some cases since paths around the outside of the network may be blocked off when routing one turn versus another. This often happens when

one of the routers connected by the turn is reachable only through the center router. When the other connected router is the destination, the first router is not able to reach it since the center router always routes directly to the destination, which blocks off all possibilities for the first router. When the first router is the destination, however, the second router may find a path around the outside of the network, which is an option that would not be available in a 2D-mesh network.

Removing this turn rule would allow a deadlock path in the network, but not allowing it would create an inconsistent network. We resolve this by applying a fix-up to this turn in the form of a new link rule. Whichever connected router could not find the other is cut off by the center router, therefore disallowing the link to the other router will not cut off any part of the network. This forces both connected routers to take the path around the outside of the network to reach the other, and keeps the network consistent while not allowing deadlock. Since this rule is only added when it is needed, it doesn't need a rule check.

4.5.3 2D-Torus Optimization

The link rules imposed for tori disallows more turns than necessary since they in effect disallow two sets of turns, one set for each connected router. We can limit this rule to only one set by ignoring the rule when the destination is on either side of the link. If we cross a disallowed link to reach a destination then we must also ignore the turn rule for that router in order to maintain network consistency. This allows us to more efficiently use the 2D-torus topology without incurring deadlock or consistency problems.

4.6 Experimental Results

We evaluated our algorithm on a number of 2D mesh and torus network sizes, ranging from 4x4 to 12x12 (Table 4.1). These sizes are representative of current industrial designs such as Tile64 (8x8) [65] and Polaris (10x8) [66]. Using a cycle-accurate simulator written in C++, each router was modeled as a single cycle design with 16-flit input FIFOs. Stimulus was produced with a constrained-random traffic generator with each packets being 8 flits long. We modeled faults as link-level failures, which allowed for a high degree of flexibility,

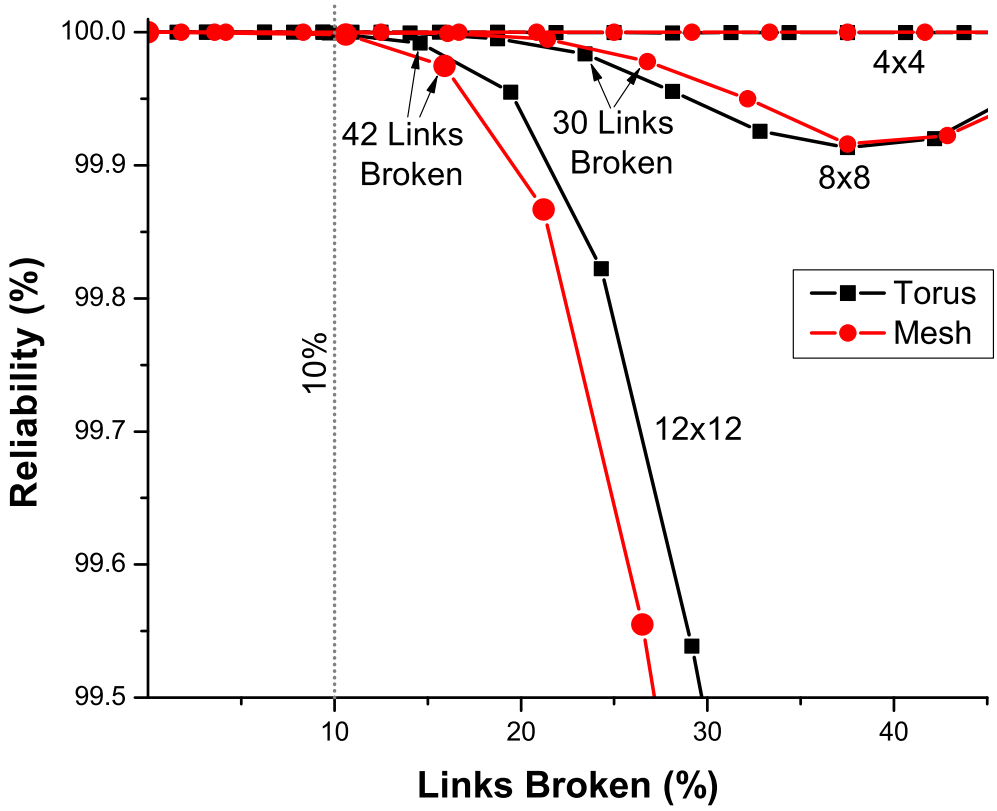


Figure 4.7: **Reliability with increasing faults.** All topologies were found to be at least 99.99% reliable at the 10% point.

since routers can retain partial functionality when experiencing faulty sub-components.

Table 4.1: **Networks topologies and sizes used for evaluation.**

2D-Mesh	4x4	24 links	2D-Torus	4x4	32 links
	8x8	112 links		8x8	128 links
	12x12	264 links		12x12	288 links

4.6.1 Reliability Analysis

In our first study, we explore the relationship between the number of faulty links and the reliability of the network. We randomly inject a number of unique link faults into various network topologies. After injecting the faults and allowing the network to reconfigure, we inspect the resulting tables to verify that the following properties hold true:

- No deadlock condition is present: no set of turns used in routing the network form a

ring (see Figure 4.4).

- Routing tables are consistent: if router A has a valid entry for router B, then A and B have the same set of valid entries.
- No routers are unnecessarily cut off: if adjacent routers A and B are connected by a non-faulty link, then each have a valid entry for the other.

We repeated the experiment one million times for each data point, obtaining the results shown in Figure 4.7. As shown in the chart, all network configurations exhibit a reliability over 99.99% when a tenth of the links are faulty. Smaller 4x4 networks had 100% reliability for 2D-mesh, and 99.99999% reliability for 2D-torus, regardless of the number of faulty links. With larger networks however, the probability of a faulty network configuration increased as the number of faults increased beyond this point, although reliability in real implementations would be much higher, since errors tend to exhibit spatial locality (temperature, utilization, clock distribution, *etc.*).

4.6.2 Performance Evaluation

In our second study, we investigated the effects of link failures on the performance of the network. Specifically, we measured the average latency of packets traversing the network as the density of randomly generated traffic increased. Density is shown as a percentage of the total injection bandwidth of the system, which is fixed and proportional to the number of routers in the network.

Figure 4.8 shows the result of this analysis for an 8x8 2D-torus network. For low traffic densities, the latency is kept under 20 cycles, however, as the density increases, network saturation occurs, resulting in a *latency wall*. With injected faults, the latency wall is reached at lower traffic densities, as indicated by the graph. In addition, random fault injection creates variation in the onset of network saturation, shown by the shaded region (5th to 95th percentile).

In our next experiment we further investigate the relationship between the number of broken links and the location of the latency wall, analyzing multiple network configurations. Here, we vary the number of broken links and record the traffic density at which the latency

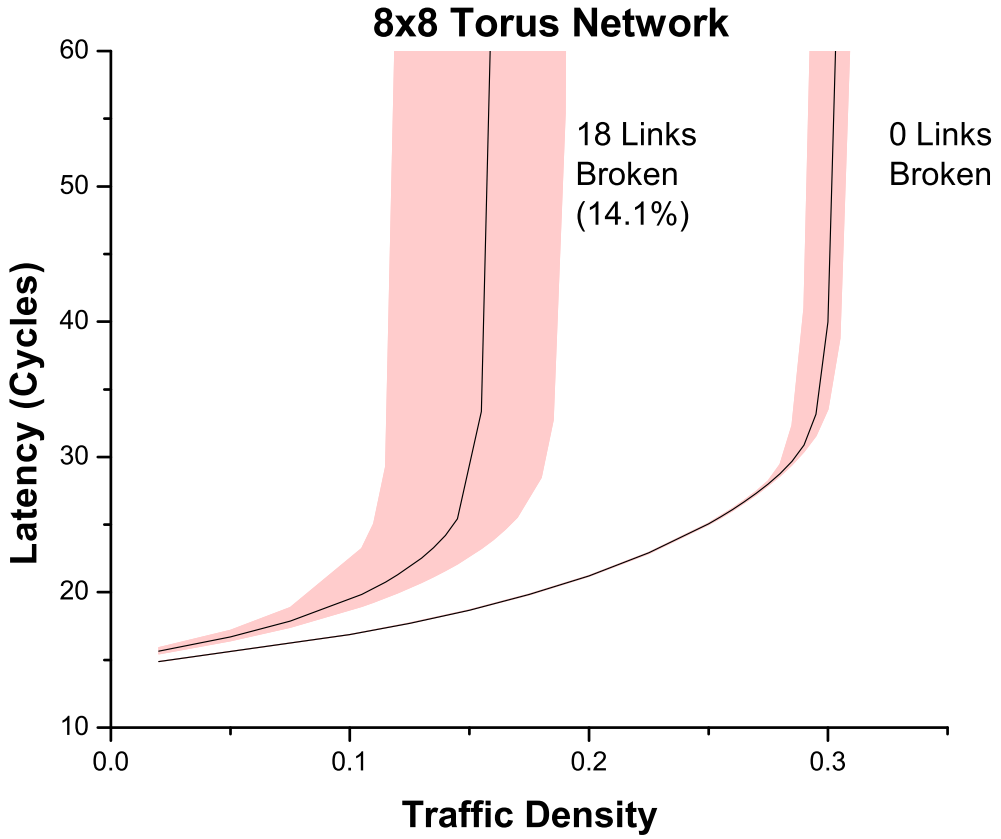


Figure 4.8: **Packet latency** for a given traffic density and number of faults. The highlighted region shows the range of 5th to 95th percentile, while the center line denotes the median. The network hits a *latency wall* at 0.30 and 0.15 traffic densities for 0 faults and 18 fault respectively.

wall (70-80 cycle latency) is encountered. Each data point represent 5000 tests of 50,000 packets. The results shown in Figure 4.9 demonstrate that with an increasing failure rate, network saturation occurs at progressively lower traffic densities. This can be attributed to fewer operational paths available for communication among network routers, as well as longer routes around failed network components.

4.6.3 Area Evaluation

Finally, we evaluated the area impact of implementing our rerouting solution in hardware. We implemented the 2D-torus variant of the algorithm as a Verilog module, and synthesized it with Synopsys Design Compiler and a state-of-the-art 45nm library. The overhead of implementing the algorithm in a 4x4 network was less than 300 gates per router. For com-

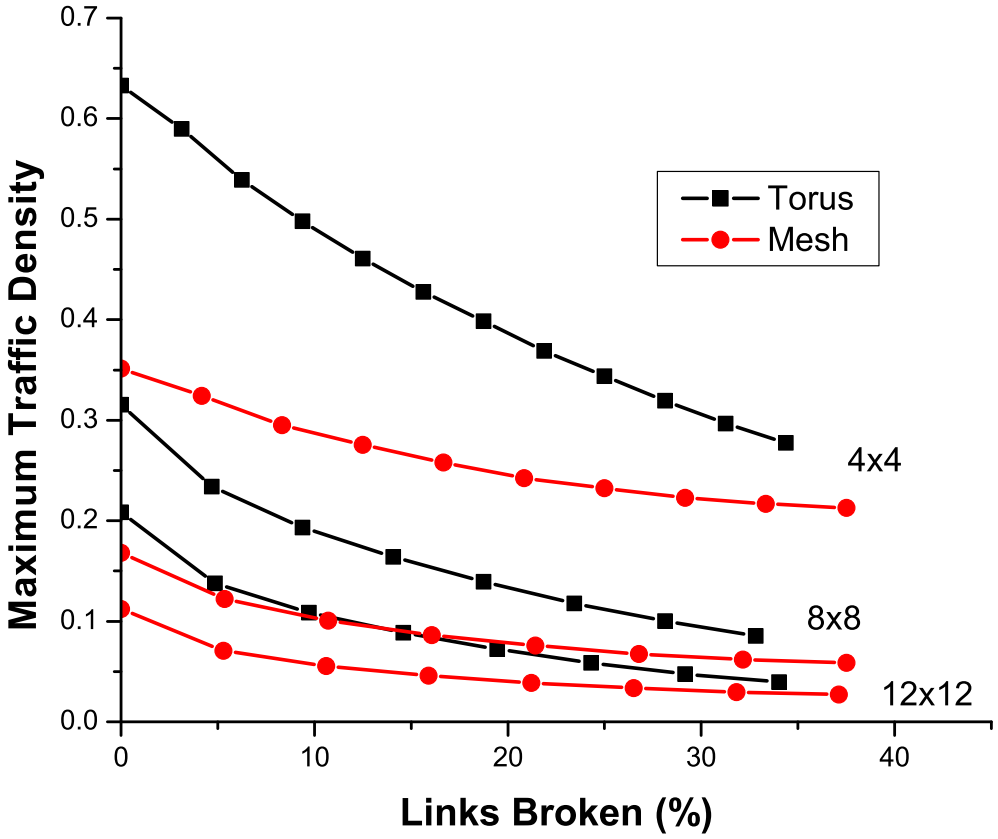


Figure 4.9: **Network traffic saturation.** The amount of traffic that a network can support before it reaches the *latency wall* for a given number of faults. The median traffic density is used for this calculation.

parison, a 12x12 network requires an additional 30 gates, attributed to slightly larger state machine counters.

4.7 Summary

In this chapter, we have presented a general fault tolerant routing algorithm targeting NoC designs implemented on unreliable silicon as foreseen for future technology nodes. We also discuss specific implementations of the algorithm for 2D-mesh and 2D-torus networks. Our solution routes around network failures by leveraging redundancy inherent in NoC topologies while not incurring virtual channel costs. At less than 300 gates per router, this approach is an efficient and effective addition to either simple or complex router designs. Experimental results showed an average reliability of over 99.99% when 10% of the network

links have failed across a variety of networks sizes. Our routing algorithm enables the deployment of NoC architectures and systems where network connectivity and correctness must be maintained possibly at a performance cost, enabling graceful performance degradation as network resources fail.

CHAPTER 5

Vicis: A Reliable Network for Unreliable Silicon

5.1 Motivation

Silicon processes have continued to improve in transistor density and speed due to aggressive technology scaling. However, each generation increases variability and susceptibility to wearout as silicon features approach the atomic scale. It has been predicted that future designs will consist of hundreds of billions of transistors, where upwards of 10% of them will be defective due to wear-out and variation [85]. At that point we must learn to design reliable systems from unreliable components, managing both design complexity and process uncertainty [86].

The ElastIC project approaches this challenge by having a large network of processing cores that are individually monitored for various wearout mechanisms and operating conditions [67]. Each processing core can be tested, repaired, powered down, or quarantined individually. With proper lifetime management, the performance of some cores can even be boosted for short periods of time when needed [22]. The interconnect architecture becomes a single point of failure as it connects all other components of the system together. A faulty processing element may be shut down entirely, but the interconnect architecture must be able to tolerate partial failure and operate with partial functionality. Network-on-chip provides opportunities to address this issue, as redundant paths exist from point to point, potentially allowing for reconfiguration around failed components.

Network-on-chip is a distributed, router-based interconnect architecture that manages

traffic between IPs (intellectual properties, *e.g.*, processing cores, caches, *etc.*) [64]. Traffic sent through the network is organized into packets, of arbitrary length, which are broken down into smaller, link-sized chunks called flits (FLow unITS). This style of interconnect scales bandwidth with network size, making it suitable for systems with many more communicators than what could be supported by bus-based architectures. Recent commercial designs include the Tilera TILE64 [65], Intel Polaris [66], and Ambric AM2045 [87,88]. NoCs do not inherently support fault tolerance - the first link or router failure in the network will cause the entire network to deadlock.

The reliability of NoC designs is threatened by transistor wear-out in aggressively scaled technology nodes. Wear-out mechanisms such as oxide breakdown and electromigration become more prominent in these nodes as oxides and wires become thinned to the physical limits. These breakdown mechanisms occur over time, so traditional post burn-in testing will not capture them. Additionally, so many faults are expected to occur that *in situ* fault management will be economically beneficial. In order to address these faults, we propose a framework with the following steps: 1) error detection, 2) error diagnosis, 3) system reconfiguration, and 4) system recovery. Error detection mechanisms notices new faults by invariance checks such as cyclic redundancy code checks on packets. Error diagnosis determines the location of the new fault via a built-in-self-test. System reconfiguration disables faulty components and configures functional components to work around the faulty ones. Lastly, system recovery restores the system to a previously known good state or checkpoint.

Network-on-chip provides inherent structural redundancy and interesting opportunities for fault diagnosis and reconfiguration. Each router is comprised of input ports, output ports, decoders and FIFOs, which are duplicated for each channel. By leveraging the redundancy that is naturally present in routers and the network, Vicis is able to provide robustness at a low area cost of 42% while exhibiting greater fault tolerance than the previously proposed N-modular redundancy based solutions. We experimentally show that Vicis is able to sustain as many as one stuck-at fault per every 2000 gates, and still maintain half of its routers.

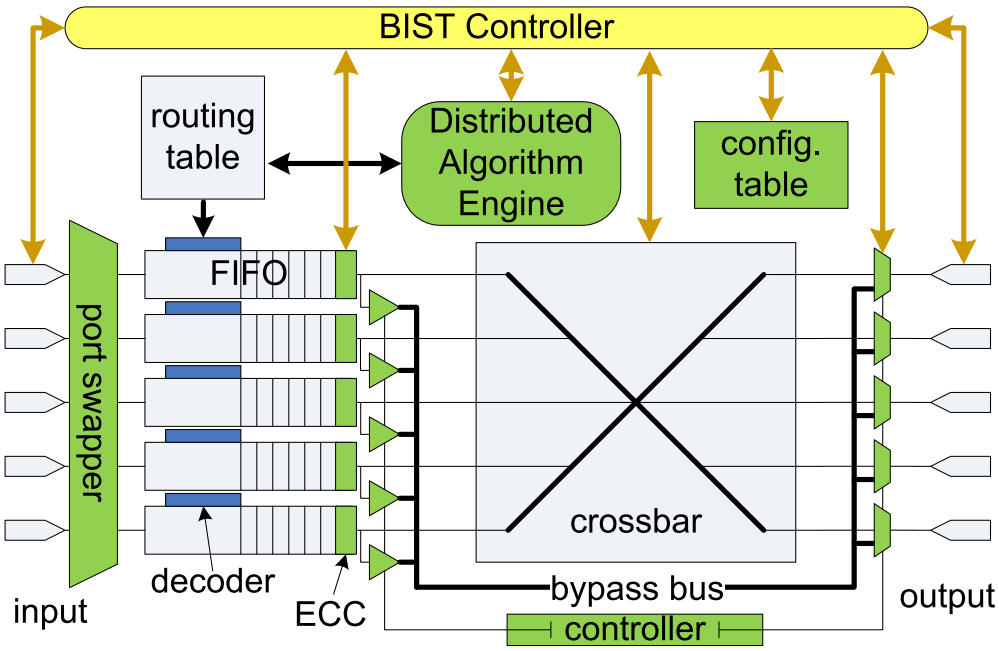


Figure 5.1: **Architecture of the Vicis router.** Enhancements include a port swapper, ECC, crossbar bypass bus, BIST, and distributed algorithm engine.

5.2 Related Work

Reliable network design was first introduced by Dally *et al.* with a reliable packet-switching network, focusing on protecting against faulty board traces due to physical and thermal stress [89]. This network used link-level monitoring and retransmission to allow for the loss of a single link or router anywhere in the network, without interruption of service.

Constantinides *et al.* demonstrated the BulletProof router, which efficiently used NMR techniques for router level reliability [90]. However, NMR approaches are expensive, as they require at least N times the silicon area to implement. Additionally, network level reliability needs to be considered since some logic is impossible or expensive to duplicate (*e.g.*, clock tree) or spares may run out, resulting in the loss of a router, and subsequently the network.

Pan *et al.* explored another strategy that exploits redundant components [91]. Instead of using NMR, inputs are sent through two copies of the same component and the outputs are then compared. If they do not match, then an error is detected and can be repaired. In addition, they included on-line testing and a built-in-self-test (BIST) for correction.

A recent architecture was proposed by Park *et al.* that focused on protecting the intra-

router connections against soft errors [92]. They explored a hop-by-hop, flit retransmission scheme in case of a soft error with a three-cycle latency overhead. Though each component has localized protection, there is no guarantee that the network will be operational if certain hard faults occur. Researchers have also explored efficient bus encoding schemes to protect against transient errors [93, 94].

Routing algorithms for fault-tolerant networks have been extensively explored for network level reconfiguration [69, 70, 75, 80–83, 95]. These algorithms direct traffic around failed network components in a way that avoids network deadlock. Although Vicis must also accomplish this, it additionally diagnoses the location of hard faults and reconfigures around them using other methods as well. Vicis adopts the routing algorithm described in Chapter 4 for part of its network level reconfiguration.

5.3 The Vicis Network

Vicis is able to maintain and prolong its useful lifetime by accurately locating faults, reconfiguring around them at the router level, and then reconfiguring at the network level. The following section is split into three parts, giving a top down view of Vicis. The first section discusses how Vicis reconfigures around failed routers and links at the network level. The second section looks at how Vicis uses ECC and a crossbar bypass bus to tolerate faults internally. The last section discusses the built-in-self-test and how it is able to accurately diagnose faults within the router with 100% coverage.

5.3.1 Network Level Reconfiguration

Network level reconfiguration addresses network visible problems by running distributed algorithms to optimize connectivity and redirect traffic. The input port swapping algorithm runs first in order to increase the number of available links in the network. The network routing algorithm then rewrites the static routing table (shown in Figure 5.1) to detour traffic around failed network links and routers. These algorithms use fault information provided by the router level reconfiguration discussed in Section 5.3.2.

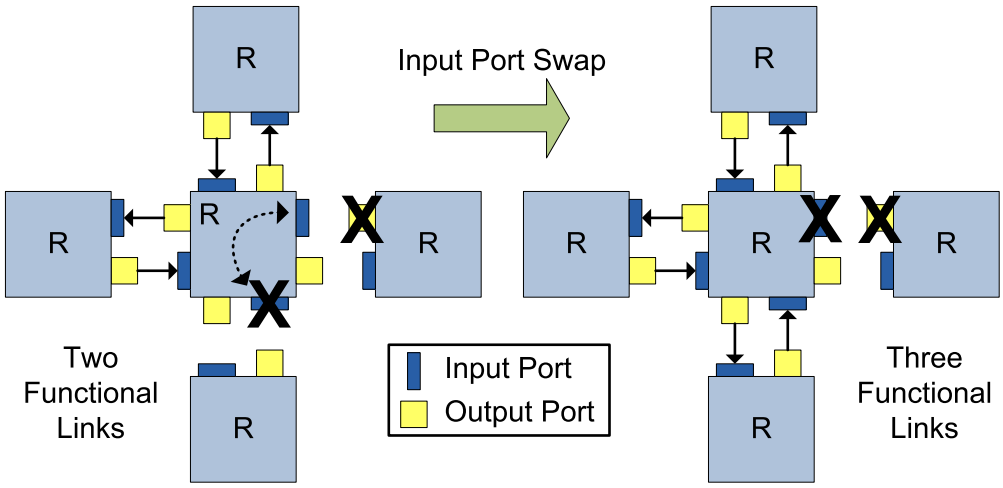


Figure 5.2: **An example of port swapping.** On the left side a failed input port and a failed output port are part of two different links. By swapping the failed input port to the link connected to the failed output port, Vicis increases the number of functional links from two to three.

Input Port Swapping

The routing algorithm needs fully functional bidirectional links in order to safely route through the network. Each link is comprised of two input ports and two output ports, all four of which must be functional to establish a link. If one of these four ports fail, then there are still three functional ports that might be used in other places (three spares).

As shown in Figure 5.1, the input ports are comprised of a FIFO and a decode unit which are identical for each lane of traffic. Vicis includes a port swapper to the input of the FIFOs in order to change which physical links are connected to each input port. A port swapper is not included for the output ports due to their relatively small area. The area of the input ports, however, is majority of the total area of the router and will therefore attract the most faults, so adding an input port swapper gives Vicis great ability to move faults around the router. Since the outputs of the input ports are connected to a crossbar, they do not need a second swapper.

Figure 5.2, shows an example of the input port swapper in use. On the left side there is a failed input port and a failed output port. Since these two failed ports are on different links, both links fail and become unusable by the network routing algorithm. One of the failed ports is a input port of the router in the center, so the physical channel that is connected

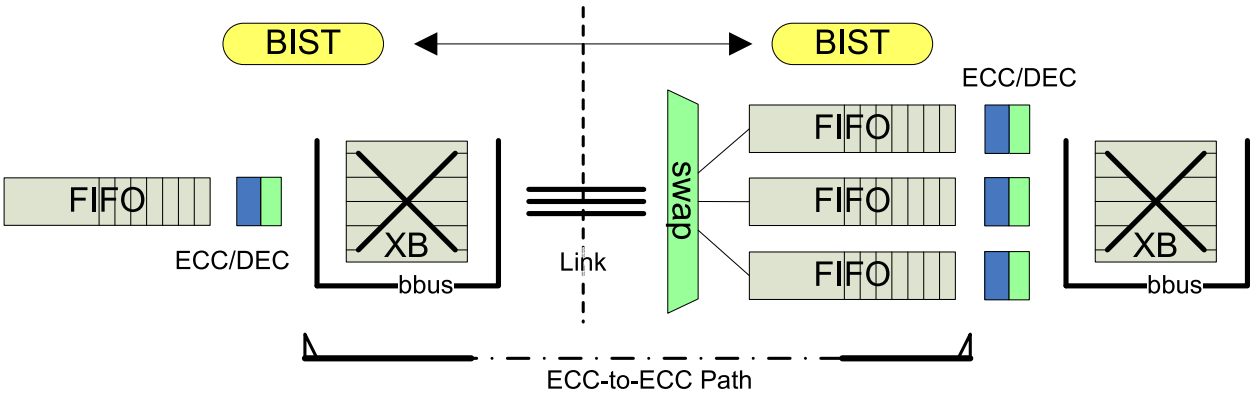


Figure 5.3: **Convergence path of ECC-mitigated faults.** Faults in the datapath can be corrected using ECC as long as no more than one is encountered between corrections. The bypass bus and port swapper provide alternate paths between routers to reduce the number of faults in each ECC-to-ECC path. There are six available paths in this example.

to can be changed. The port swapping algorithm reconfigures the port swapper so that the failed input port is instead connected to the neighboring failed output port, as shown on the right side. By doing this, Vicis takes advantage of the inherent redundancy in the router, and increase the number of functional links from two to three.

The port swapper does not need to be fully connected, that is, not every input port needs to be connected to every physical link. In this implementation of Vicis, the link to the local network adapter is able to connect to three different input ports, and the other links are able to connect to two input ports. The port swapping algorithm is a greedy algorithm that takes into account the ECC related information described in Section 5.3.2. It also gives priority to the local network adapter link, making sure that it is always connected when possible.

After the port swapping algorithm finishes running, each router knows which of its links are functional, and provides that information to the routing algorithm, which completes the network reconfiguration.

Network Rerouting

For a non-fault tolerant network, or a network that supports only router level reliability, the first link or router failure will cause the entire network to fail. Traffic that gets directed to that link or router gets lost, and eventually the network will deadlock when traffic flow tokens fail to be back propagated. In order to detour traffic around faulty links and routers,

Vicis uses a routing algorithm to rewrite the routing tables shown in Figure 5.1. We selected the algorithm described in Chapter 4, since it does not need virtual channels, it supports link-level faults, and it has a low overhead implementation. Since it is a distributed algorithm, the loss of any number of links or routers will not prevent it from running, and the remaining routers will have correctly written routing tables.

5.3.2 Router Level Reconfiguration

Router level reconfiguration is used to contain faults within the router so that they are not visible at network level in the form of failed links and routers. The Vicis architecture, shown in Figure 5.1, includes a crossbar bypass bus to protect against crossbar failures, and single error correct, single error detect error correction codes (SEC-SED ECC) to protect datapath elements. Information about the location of faults is provided by the hard fault diagnosis stage, described in Section 5.3.3.

Crossbar Bypass Bus

The bypass bus, parallel to the crossbar, allows flits to bypass a faulty crossbar (see Figure 5.1). The crossbar controller is configured by the BIST to direct traffic to either the crossbar or to the bypass bus on a packet by packet basis. If multiple flits needs to use the bypass bus (simultaneously), then one flit is chosen to proceed first while the others wait until a later cycle. This spare path allows Vicis to maintain operation when multiple faults appear in the crossbar, trading off performance to maintain correct operation.

Error Correcting Codes (ECC)

ECC-protected units allow each datapath component to tolerate a small number of faults while maintaining correct functionality at the cost of minimal overhead. Any fault that manifests along an ECC-guarded datapath can be corrected when the flit passes though an ECC unit. In order to take full advantage of ECC, the BIST tracks the location of every datapath fault and ensures that every ECC-to-ECC path has at most one hard fault.

In Figure 5.3 we show the ECC-to-ECC path in Vicis. After encoding, a flit travels

through the crossbar and bypass bus, the link between the two routers, the input port swapper, and finally the FIFO. At each mentioned unit along the path, the faults (if any) are diagnosed and cataloged by the two BISTs. If two faults accumulate in the same datapath, the bypass bus and port swapper provide alternative configurations to either avoid one of the faults or move one of the faults to a different datapath, respectively.

For example, one fault could be in the crossbar, another in a link, and a third in the default FIFO for the datapath. The ECC implementation in Vicis can only correct one of these faults, so the crossbar bypass bus and input port swapper need to mitigate two of them. The bypass bus will be used to avoid the crossbar fault, potentially resulting in a loss of performance. The input port swapper will be used to swap in a fault-free input port to the datapath, moving the single-fault input port to another physical link that does not have any faults. By doing this, full functionality is maintained, even with three faults originally in the same datapath.

The FIFOs were designed to take advantage of ECC as well. The FIFO is implemented as a circular queue, which results in a given flit only ever entering a single line of the FIFO. In this way, a FIFO could have upwards of one fault per entry while remaining operational, as it effectively adds only a single fault to the ECC-to-ECC path.

5.3.3 Hard Fault Diagnosis

The techniques in Vicis rely on knowing precisely where faults exist. A port cannot be swapped if it is not known what ports are faulty and which are functional. The use of ECC requires that Vicis knows precisely how many faults are in each part of the datapath. In this section we discuss the strategies that Vicis uses to locate faults with low cost, yet obtain 100% coverage.

Built-in-Self-Test (BIST)

The built-in-self-test (BIST) unit controls all of the router diagnosis and reconfiguration. It is power gated during normal operation in order to protect it from wearout. Because of this, the BIST controls other units through a configuration table, which stays powered on

```
 1: (*)  Propagate Error Status (Error Unit)
 2: ( )  Power up BIST
 3: (*)  Synchronize Network
 4: (D)  Test Router Configuration Table
 5: (P)  Test Error Propagation Unit
 6: (P)  Test Crossbar Controller
 7: (P)  Test Routing Table
 8: (P)  Test Decode/ECC Units
 9: (P)  Test Output Ports
10: (P)  Test FIFO Control Logic
11: (D)  Test FIFO Datapath
12: (D*) Test Links
13: (D*) Test Swapper
14: (D)  Test Crossbar
15: (D)  Test Bypass Bus
16: (*)  Communicate ECC-to-ECC Path Information
17: (*)  Run Port Swapping Algorithm
18: (*)  Run Network Rerouting Algorithm
19: ( )  Power Down BIST
20: ( )  Resume Operation
```

Figure 5.4: **Built In Self Test procedure.** Distributed algorithms are marked (*), datapath testing is marked (D), and pattern based testing is marked (P).

at all times. The configuration table is tested as well, as part of testing the units that it controls. Non-wear-out faults in the BIST need to be found after fabrication, and the router disabled by the manufacturer.

Figure 5.4 shows the built-in-self-test procedure that occurs when the BIST is activated. One of network adapters will detect a network failure by using a cyclic redundancy check or another invariance check. Once a network adapter detects an error, it will send an error status bit to its connected router, at which point the routine in Figure 5.4 begins.

The network first broadcasts an error status bit via a low-overhead error unit. The error unit also powers up the BIST for that router. Once the BIST has been powered up, it performs a distributed synchronization algorithm so that every BIST in the network runs the rest of the routines in lock-step. Once synchronization completes, each unit is diagnosed for faults. The diagnosis step does not use information from previous runs of the routine, or from the detection mechanism that started the routine, so all faults will be diagnosed regardless of whether they caused the error or not. Once all units have been

tested, distributed configuration algorithms run, and finally the BIST powers down and normal operation resumes.

Functional Unit Wrappers

Each functional unit has a test wrapper so that the built-in-self-test can control it during fault diagnosis. The wrapper consists of multiplexers for each of the functional unit inputs, which switch between the normal unit inputs, and a testing bus coming from the BIST. A signature bus takes a copy of each of the outputs back to the BIST for analysis. The wrappers are interlocked to guarantee fault-coverage is not lost between each of the units. Although a hard fault on a wrapper gate may be caught trivially, a hard fault may occur on the gate driving the multiplexer select bit, forcing it to always be in test mode. By reading the inputs after the multiplexer to the next unit, this case will also be caught. Additionally, synthesis might insert buffers between where the signature bus connects and the other units connecting to the output - those buffers would not be covered without interlocking wrappers. Cooperative testing between routers is needed to test links, the input port swapper, and their associated wrappers.

Datapath Testing

Lines marked with (D) in Figure 5.4 are part of a datapath test. Units in the datapath need an exact count of ECC errors for each unit so that the maximum number of errors is not exceeded for any ECC-to-ECC path. The tests each send all 1's or all 0's through the datapath, looking for bit-flips faults. A specially designed bit-flip count unit determines if the datapath has zero, one, or more bit flips so that multiplexers are not needed to inspect each bit individually. Units like the FIFO reuse the same test for each instance, which helps maintain a low overhead.

Pattern Based Testing

Lines marked with (P) in Figure 5.4 are part of a pattern based test. These units are implemented with synthesized logic, so a hand-written test would not have full coverage.

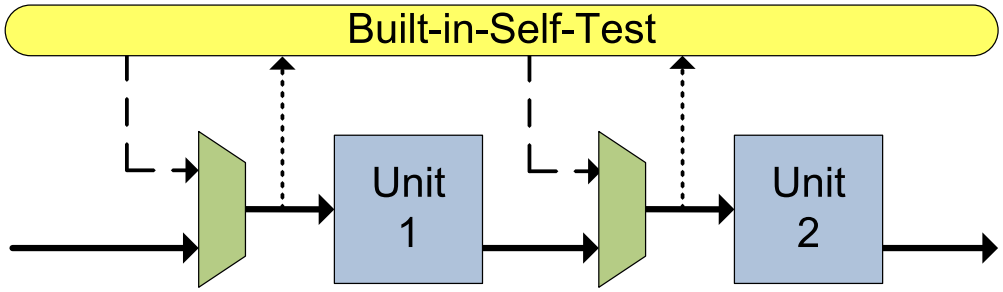


Figure 5.5: **Functional Unit Wrappers.** Vicis uses interlocking functional unit wrappers to provide the BIST with access to each unit for hard-fault diagnosis. Incoming patterns from the BIST are shown with long dash arrows, outgoing signatures are shown with short dash arrows, and normal operation paths are shown with solid arrows.

To test these units, Vicis uses a linear feedback shift register (LFSR) to generate a number of unique patterns, and a multiple input signature register (MISR) to generate a signature. Each unit tested with the pattern based test receives the same sequence of patterns from the LFSR, but each has its own signature. Identical units, such as all of the decoders, have the same signature. A signature mismatch will flag the corresponding unit as unusable, and the unit is then disabled by the BIST. Implementation of the pattern based test is lightweight due to the simplicity of the LFSR and MISR.

Pattern based testing dominates the total runtime of the BIST, taking approximately 200,000 cycles when 10,000 patterns are used per test. The time between faults should be measured in hours, however, so this is a relatively short amount of time.

5.4 Experimental Results

In this section, we discuss Vicis' effectiveness in improving network reliability. First we discuss the experimental setup used to evaluate Vicis. Next, we look at network reliability and compare Vicis to a TMR based implementation. Then we look at router reliability and its *Silicon Protection Factor*. Finally, we show results for network performance degradation.

5.4.1 Experimental Setup

We implemented Vicis in a 3x3 torus topology, as well as a baseline router design. Each design routes flits with a 32-bit data portion, and have 32-flit input port buffers, which is consistent with the Intel Polaris router [66]. The baseline router has the same functionality as Vicis, except for the reliability features. Both Vicis and the baseline NoC were written in Verilog and were synthesized using Synopsis Design Compiler in a 42nm technology. Compared against the baseline NoC, Vicis has an area overhead of 42%, which is much lower than the 100+% needed for NMR.

To test the reliability features of Vicis, we randomly injected stuck-at faults onto gate outputs in the synthesized netlist. Selection of the gates was weighted based the area of the gate, which is consistent with the breakdown patterns found experimentally in Keane *et al.* [96].

Test packets were generated at each network-adapter. It was verified that each packet made it to the correct destination with the correct data. For each error combination, we ran 10,000 packets of a random uniform traffic distribution, each packet having between 2 and 11 flits.

Network throughput was measured using a custom made, cycle accurate C++ simulator. Profiles of the network were generated with the Verilog simulations described above, which were then given to the C++ simulator to model longer simulation traces. Throughput was measured using random uniform traffic for this test as well.

5.4.2 Network Reliability

To test the reliability of the simple network, we randomly injected faults into the system and monitored the packet traffic. Although the full system is prone to wear-out, we primarily focused on injecting faults into the router - a total of 20,413 gates. Since we only consider wear-out induced faults, we power-gated the BIST, thus making it immune to fault injection.

To rigorously test the system, we considered eleven different cases with varying simultaneous faults: 1, 10, 20, 30, 40, 50, 60, 70, 80, 90, and 100, 100 being one fault for every 2000 gates. For each number of simultaneous faults, we considered 1200 different (random) error

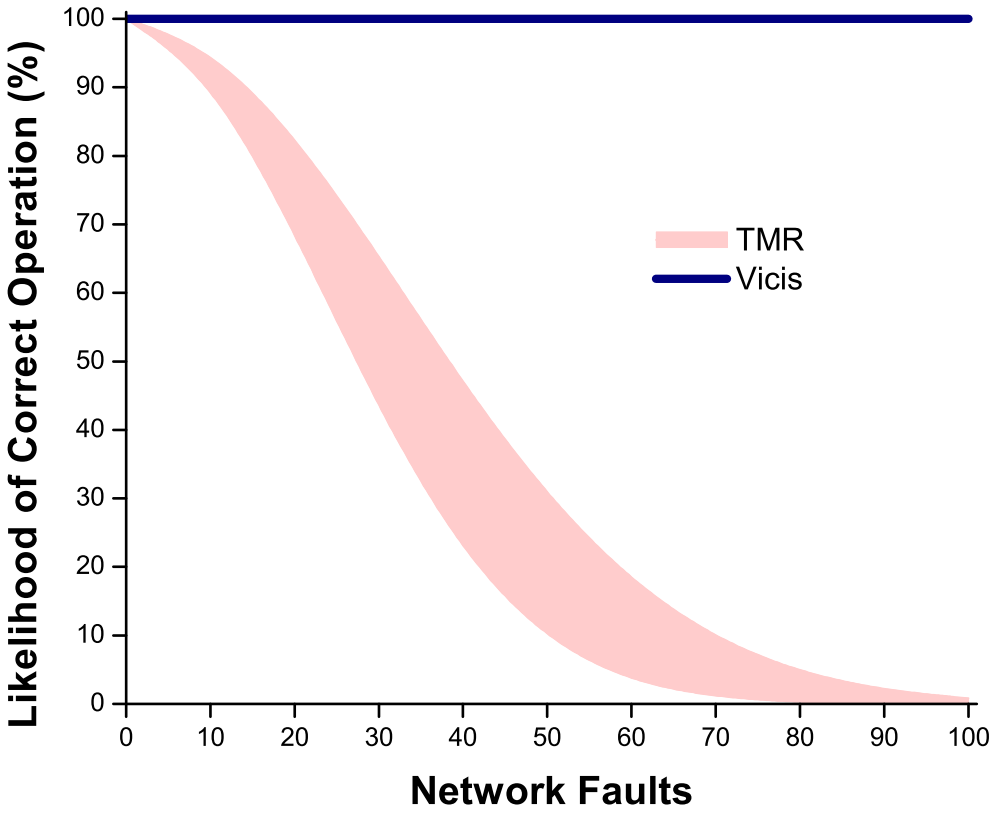


Figure 5.6: **Reliability Comparison with TMR.** TMR has probabilistic reliability while maintaining performance, where Vicis has probabilistic performance while maintaining reliability (see Figure 5.8.)

combinations.

In this series of tests there was a single failure that manifested at 30-40 faults, but disappeared at 50 faults. The disappearance of the failure is likely due to new faults being injected into the same unit as the fault(s) that were previously overlooked by the BIST. We expect the failure to be an implementation error, as opposed to a problem with the described techniques. Either way, the reliability of the network is dominated by the routing algorithm for larger networks, as described in Chapter 4.

We compare the reliability of our design to a TMR-based implementation. Note that TMR only provides probabilistic reliability - since the voter takes the most common signal of the three replicated units, two well-placed faults could cause the system to fail. In the worst case, a single fault could cause system failure if it occurred in a clock tree or another non-replicable cell. Unlike BulletProof [90] and other prior work that relies on maintaining

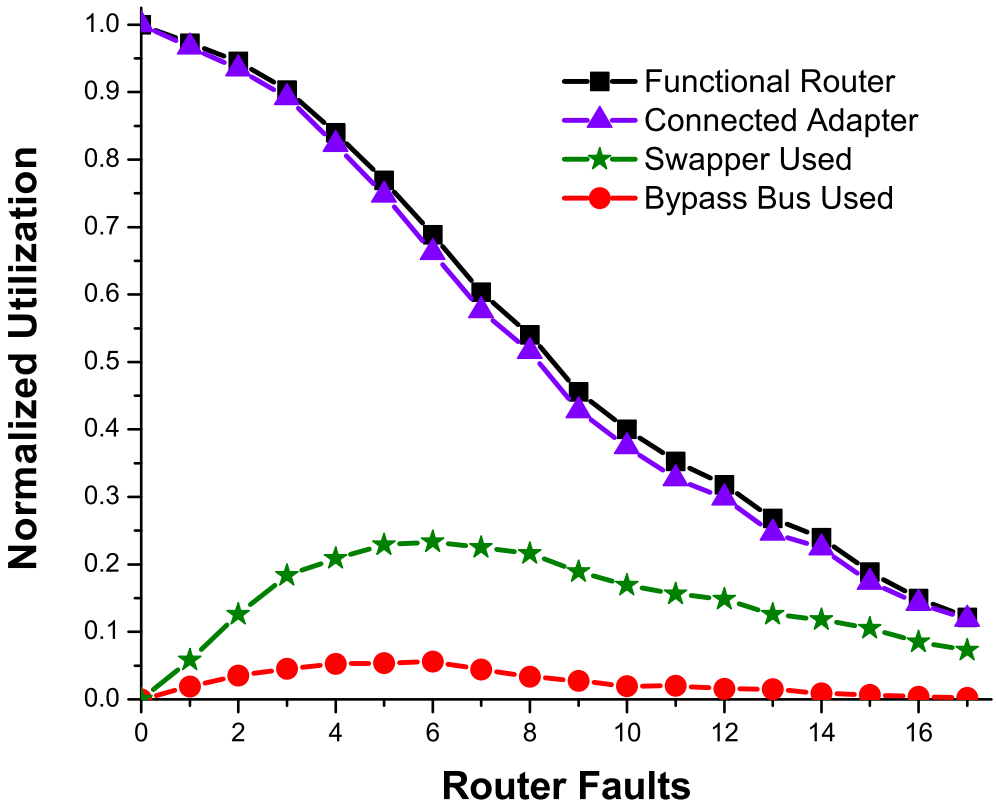


Figure 5.7: **Router reliability with increasing faults.** Reliability of an individual router as it receives stuck-at faults inside of a faulty network. Utilization of individual features is also shown. *Connected Adapter* shows how many functional routers also have a functional link to its adapter.

total functionality, Vicis is able to tolerate multiple, simultaneous faults, including ones that render entire routers useless. Thus, Vicis is able to maintain 100% reliability regardless of the number or occurrence of faults, although performance becomes degraded.

In Figure 5.6 we show the reliability of TMR, which is based on analysis from [90]. Where TMR provides 100% performance and degrading reliability for 200+% area overhead, Vicis provides 100% reliability and degrading performance for 42% overhead. Comparing TMR reliability degradation to Vicis performance degradation, TMR degrades much more quickly - at 100 network faults, TMR's reliability approaches zero whereas Vicis continues to operate with half of its routers still functional and communicating.

5.4.3 Router Reliability

In Figure 5.7, we present the different internal router statistics based on the tests described above. To fairly evaluate and collect results for each router component, we injected faults on standard-cell outputs in a network level netlist in order to consider the states of the surrounding routers. A router is considered operational if it has at least two functional bidirectional ports connected to either other routers or network adapters.

Input Port Swapper

We found that the input port swapper was very successful at keeping the cores (IPs) connected to the network. As shown in Figure 5.7, only eight percent of the available routers not have a functional network adapter. The swapper had a high utilization, being used nearly 24% of the time for routers with seven faults.

Crossbar Bypass Bus

At seven faults, the crossbar bypass bus was used less than six percent of the time. This is due to two reasons: 1) the crossbar is relatively small - less than five percent of the total area of the router, and 2) the crossbar is protected by both the input port swapper and ECC as well. We expect that an improved implementation of the bypass bus will have an greater effect, and are pursuing that in future work.

Silicon Protection Factor (SPF)

Constantinides *et al.* introduce the concept of *Silicon Protection Factor* for router level reliability, defined as the number of faults a router can tolerate before becoming inoperable, normalized by the area overhead of the technique [90]. The normalization step is to take into account the increase of gates since having more gates means that the design will also experience more faults.

From Figure 5.7 we can interpolate to get the median value, which is 9.3 faults before the system fails. Normalized to our area overhead of 42%, this gives Vicis an SPF of 6.55. In comparison, the best SPF provided by the Bulletproof router is 11.11, which incurs a

242% area overhead. The lowest area overhead Bulletproof configuration is 52% overhead, but provides an SPF of only 2.07.

Actual network reliability cannot be compared since a router failure in Bulletproof causes full network failure, whereas a router failure in Vicis only renders that single router inoperable. Additionally, Bulletproof does not give a breakdown of what faults were critical (first fault causes a failure) versus cumulative (multiple faults interact to cause a failure). The number of critical failure points in the Bulletproof router would place a limit on overall network reliability, where that is not the case in Vicis.

5.4.4 Network Performance

In Figure 5.8, we demonstrate how network level performance gracefully degrades as the number of faults in the network increases. The black line with squares shows the number of available IPs connected through the network to at least one other IP. At 90 faults, which is more than 10 faults per router, or one fault per 2000 gates, over 50% of the original IPs are still available. Since faults are injected at the gate level, there would not likely be any functional IPs to connect, as they would be experiencing the same fault levels.

In Figure 5.8, we also show the normalized network throughput across different number of simultaneous hard faults (data points as triangles). For the first 30-40 faults, the network experiences a loss in normalized throughput. This is due to link failures (2+ faults within links), forcing packets to take longer paths and avoid network obstructions. After 40 faults, however, the network loses enough routers to effectively create a smaller network that intrinsically supports higher normalized throughput. The light shading behind the data line shows 5th-95th percentiles of normalized throughput - the line itself is the median.

5.5 Summary

In this chapter we presented Vicis, a network-on-chip that is highly resilient to hard-faults. By using the inherent redundancy in the network and its routers, Vicis can maintain higher reliability than NMR based solutions while incurring only a 42% overhead. Each router uses a built-in-self-test (BIST) to diagnose the locations of hard faults and runs a number of

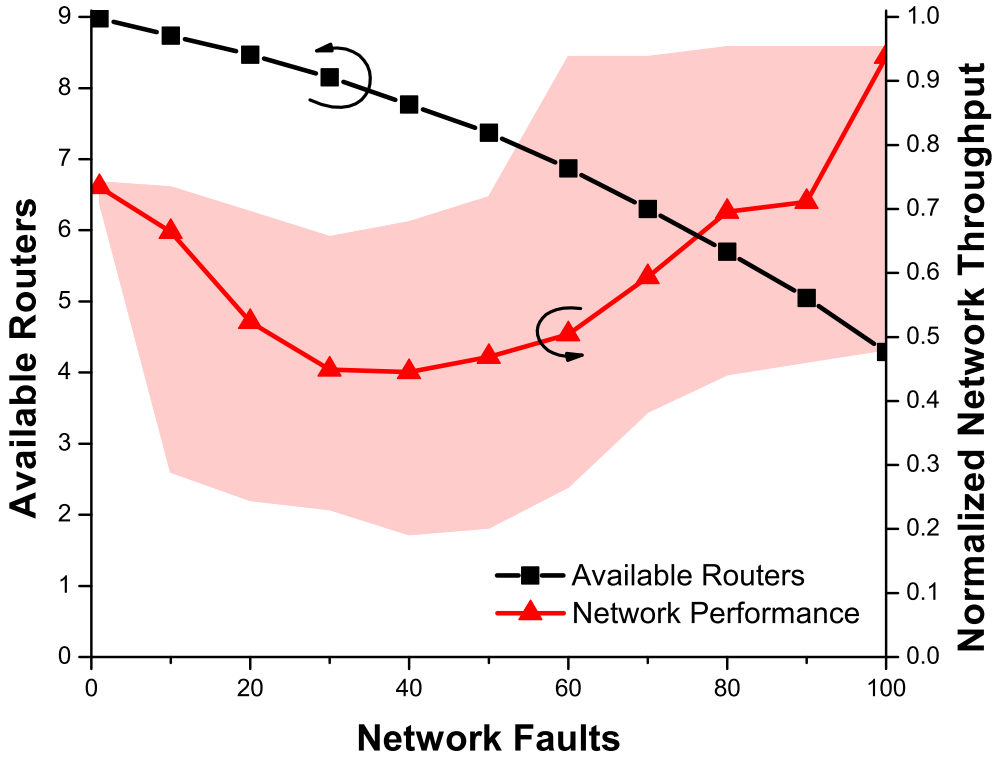


Figure 5.8: **Network performance with increasing faults.** Normalized network throughput is shown as the number of faults in the network increases. Throughput is normalized to the bandwidth of the remaining network adapter links. The shaded region show the 5th-95th percentiles, while the line is the median.

algorithms to best use ECC, port swapping, and a crossbar bypass bus to mitigate them. The routers work together to run distributed algorithms to solve network-wide problems as well, protecting the networking against critical failures in individual routers. Ultimately, we show that with stuck-at fault rates as high as 1 in 2000 gates, Vicis will continue to operate with approximately half of its routers still functional and communicating. Additionally, we provide results detailing the utilization of some of the architectural features, as well as a reliability comparison with triple modular redundancy and prior work.

CHAPTER 6

Stochastic Computing Enhanced Image Sensor

6.1 Motivation

Moore's law has been in effect since the 1970s, but exponential growth of information has been occurring since the beginning of the 20th century [97, 98]. Factors such as population growth, improvements in education, and technological innovation have contributed to this growth. Researchers estimate that the world had created more than 295 exabytes of information by 2007 [98], but "exascale computing" (machines that can achieve 10^{18} operations per second) have only recently been on the forefront of discussion [99, 100]. Because of this, researchers are looking for new ways to process this information.

A well-known efficient processor of large data sets is the mammalian brain and its neocortex. Operating on signaling frequencies of tens of hertz [101], it is able to handle complex tasks, such as image recognition, much faster and more accurately than any computer today. This has led to much research of "neural networks", which are aimed towards mimicking its functionality [102]. Being biologically inspired, these learning machines are comprised of neurons and synapses and have had some success at creating efficient classifiers.

Software implementations of neural networks usually execute on unnecessarily precise Turing-machine computers. This redundant precision costs energy, but it also reduces the quality of the result, *i.e.*, adding randomness is beneficial [102]. Some hardware implementations exploit this fact by using analog circuits for computations [103,104]. Another approach is "stochastic computing", which inherently uses randomness to complete calculations.

Stochastic computing represents values between 0 and 1 by using binary streams that are proportional to the value that they represent, *e.g.*, 75% ones and 25% zeros to represent 0.75. Computation on these streams is based on math of independent probabilities. A multiplication can be performed via an AND gate, and a weighted average can be performed by a multiplexer. The compactness of mathematical computation was a primary motivator for its original development in the 1950s [105–109], when computers were implemented with bulky vacuum tubes.

Additional benefits include inherent fault tolerance and arbitrary precision. Since each bit of the stream is random, flipping a single bit will have limited impact on the computation. This makes the platform resilient against soft-faults like single event upsets (SEUs) caused by radiation. Stochastic computing is still susceptible to hard-faults, although low-area implementations are much easier to protect with techniques such as triple modular redundancy (TMR) than traditional designs.

Precision is controlled with the length of the stream, which can be can be changed while leaving the underlying hardware unaffected. This means that arbitrary precision can be achieved with a sufficiently long stream, or precision can be reduced when it is not needed, saving time and energy. For example, a value may only need to be known to high precision if it is above 0.75. After a short time, confidence of that threshold could be sufficiently high enough to stop the computation. [105–109]

Higher precision calculations require exponentially longer computations. For instance, generation of a 10-bit number requires 1024 sub-computations, while a 12-bit number requires 4x as many. This makes stochastic computing a more natural fit for communication and media applications where the input data is noisy and environmentally driven, as opposed to finance and banking, where high precision is a necessity. A key example is low-density parity-check codes (LDPC codes), which allows data to be sent over noisy channels near the Shannon Limit, and can be efficiently implemented using stochastic computing [110].

In this work, we analyze the use of stochastic computing for image sensor applications. In Section 6.2 we discuss how stochastic computing can be used to enhance image sensors. In Section 6.3 we describe a novel architecture for this application in a 130nm process. Simulated results are analyzed in Section 6.4, with comparisons made to alternative imple-

mentations. This chapter is summarized in Section 6.5.

6.2 Stochastic Enabled Image Sensing

In this section we will describe how to insert stochastic computing into an image sensor, and how to use stochastic computing to perform useful operations within the image sensor itself.

6.2.1 Stochastic Analog-to-Digital Conversion

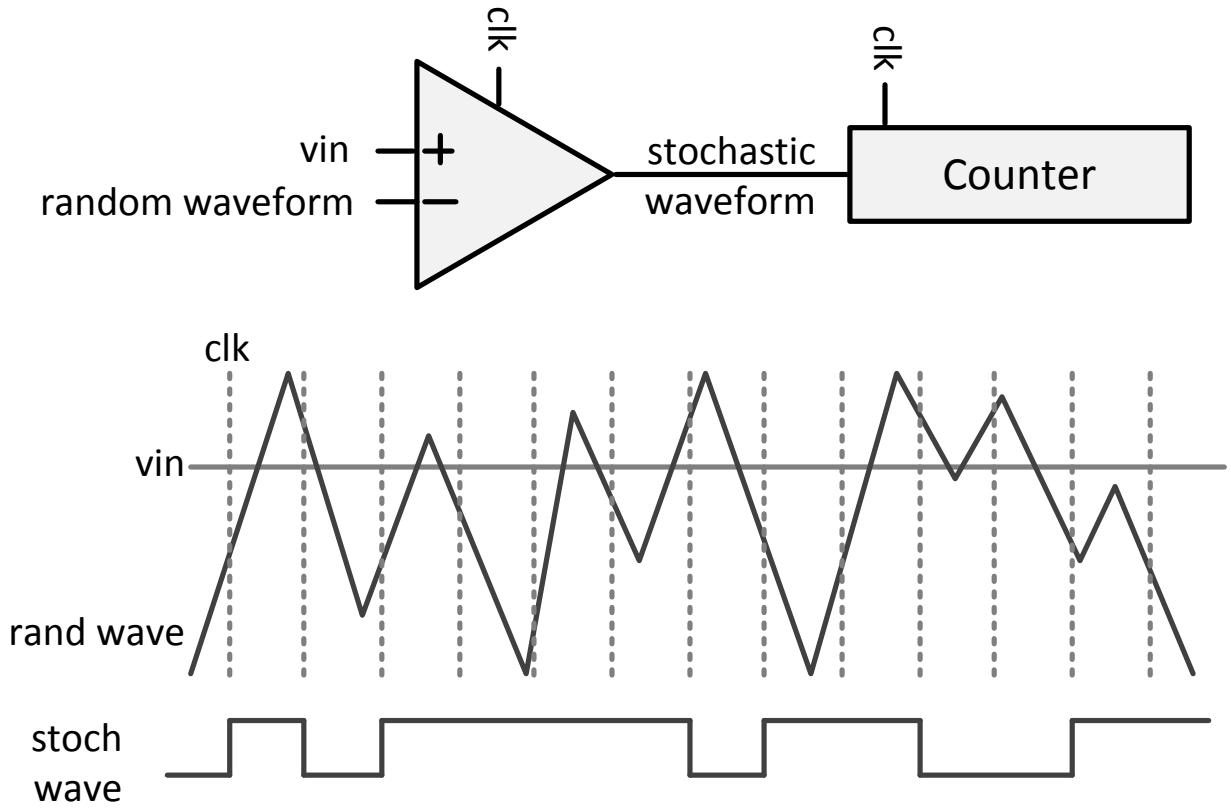


Figure 6.1: **Operation of the stochastic ADC.** Similar to an integrating ADC, the stochastic ADC contains a dynamic latched comparator and a counter. By comparing against a random waveform, a stochastic waveform is generated, which can be used for stochastic computation before aggregation.

Image sensors are created with large arrays of photo diodes and capacitors. At each pixel, a capacitor is first charged via a reset signal, then it is discharged via incident light and a

photo-active diode. The capacitor voltages are then read with analog-to-digital converters (ADCs) row-by-row, either by shifting the values through the pixels as for charge-coupled device (CCD) imagers, or by actively driving a bitline, as for CMOS images. Since CCD is a specialized process like DRAM, this work uses CMOS based image sensors. [111]

We can introduce stochastic computing to this system by replacing the traditional ADC with a stochastic-based equivalent. Instead of a highly-complex unit that converts a voltage to an 8-bit number, we instead use a much smaller unit that converts the voltage to a random stream of ones and zeros. We can achieve this by using a low-area dynamic latched comparator to compare against a uniform-random signal. To complete the analog-to-digital conversion, we aggregate the stream using a ripple counter. Since any ADC would need to store a result after generation, the area impact of this counter is negligible. With this architecture, it is now possible to insert stochastic logic between the stream generation and the aggregating counter. Figure 6.1 illustrates this design.

The stochastic ADC described here is related to a class of traditional ADCs called "integrating ADCs", which use a comparator to compare against a known ramp and a counter to count the ramp time [112]. The key observation in this work is that it is possible to use a uniform-random signal to compare against instead of a known ramp, which enables stochastic computing and further processing.

6.2.2 Stochastic Edge Detection

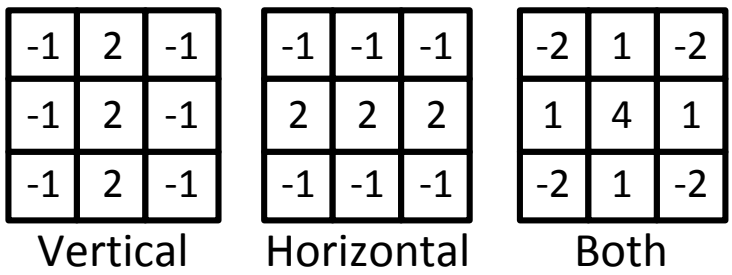


Figure 6.2: **3x3 matrices of weights to compute edge values.** The computation is a weighting of a pixel with its eight neighbors, and is repeated for each pixel in the image.

Edge detection is an image processing transformation that highlights edges within an image. This is accomplished by comparing neighboring pixels to find regions of large intensity.

The technique used in this work is called "second-order differentiation", where a 3x3 matrix is applied to the image, with weightings shown in Figure 6.2. The resulting image has both positive and negative edges values, and those closest (*i.e.*, the zero crossing pixels) are the edges. [113]

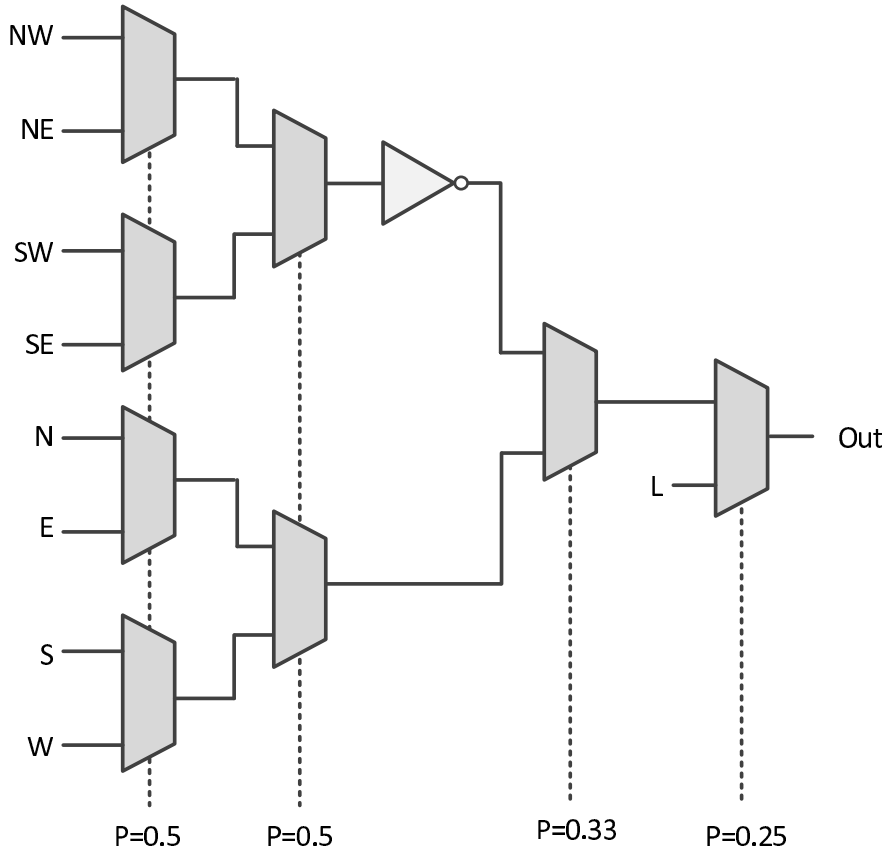


Figure 6.3: **A stochastic computing implementation of the filters from Figure 6.2.** The nine stochastic streams from the comparators enter on the left. Four additional weighting streams are needed for this implementation, which can be shared between the columns.

Generating this matrix in stochastic computing requires a random stream from each of the pixels in the matrix. In our stochastic image sensor, we replace the row of traditional ADCs with a row of stochastic ADCs, as described in Section 6.2.1. This exchange provides one of the three rows of streams. To generate streams for the other two rows, we use sample and hold (SAH) circuits to store those pixel values, and add comparators for those rows, for a total of three comparators per column.

With nine stochastic streams being generated, it is straightforward to construct the needed filter. Figure 6.3 depicts a basic stochastic implementation of the filter using a mul-

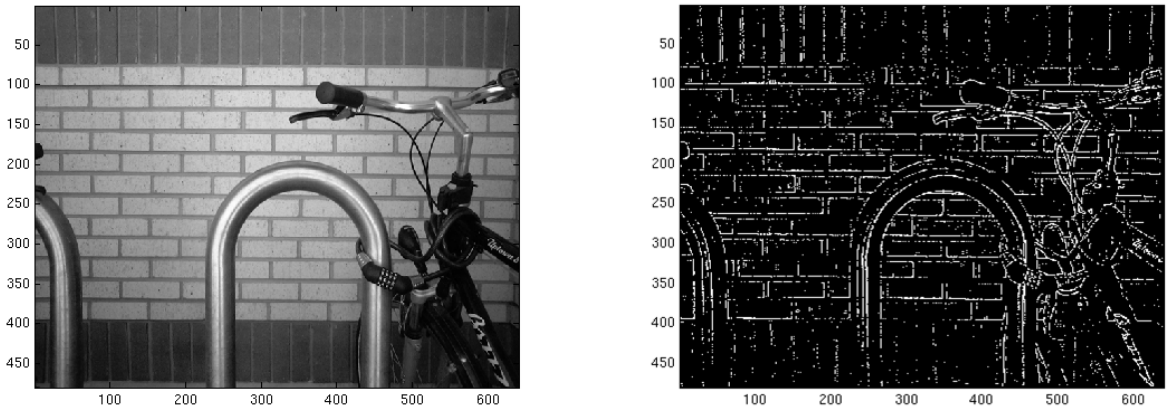


Figure 6.4: **Example image being filtered with stochastic implementation from Figure 6.3.** This example was generated using MATLAB with zero-crossings from the output highlighted.

tiplex network, where each of the nine streams enter on the left. Four additional stochastic streams need to be generated with values of 0.5, 0.5, 0.33, and 0.25. These streams can be shared among all of the columns since their computations are independent, and can be generated trivially with a linear-feedback shift register (LFSR). Figure 6.4 shows an example image generated with this technique in MATLAB, with zero crossings highlighted.

6.2.3 Stochastic Gaussian Blur

Gaussian blur is an image filter that averages pixels with its neighbors with weightings that have a normal distribution. This type of filter is often used to reduce noise in an image, which is particularly helpful for edge detection. In a sense, the edge detection algorithm described in Section 6.2.2 is a *noise amplifier*. For instance, a random black pixel added to an otherwise white image will cause both it and its neighbors to be highlighted after the edge detection transformation. Reducing this noise is important to generating meaningful edge data. Additionally, Gaussian blur filters can be used to control the sensitivity of the edge detection transformation, where the most edges are highlighted in an unfiltered image, and fewer edges are highlighted the stronger the blurring effect (higher sigma) [113].

A stochastic implementation of the Gaussian blur is similar to Figure 6.3. The Gaussian

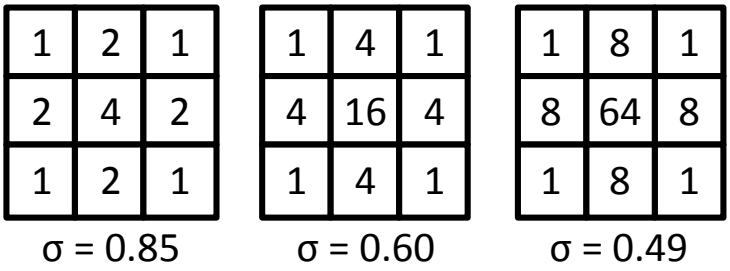


Figure 6.5: **3x3 matrices of weights to compute a Gaussian blur.** The computation is a weighting of a pixel with its eight neighbors, and is repeated for each pixel in the image. The three sigmas used for this work are shown.

blur needs to be calculated for each pixel before the edge value, and requires its own 3x3 matrix. Because of this, three Gaussian blur calculations are needed per column, to generate three rows for the edge detection algorithm. Two additional SAH circuits are needed to store additional rows to generate these blurred rows, for a total of five. Figure 6.5 shows Gaussian blur weights used in this work; three sigmas are used for configurability, for a total of four levels of edge detection sensitivity.

6.2.4 Architectural Enhancements

Enhancements can be made upon the architecture described above. First, both the edge detection and Gaussian blur operations are accomplished by using weighted averages via multiplexer networks. Instead of a network of multiplexers, it is possible to build a single-stage multiplexer that accepts nine inputs and a 9-bit one-hot select signal. This does not complicate the weight generation, since it is straightforward to construct once the original architecture is known, *i.e.*, it is reversing the selection process. This enhancement improves the speed and reduces the area of the design since the weight generation only needs to be done once per array, while the multiplexing is on a per-column basis.

Another enhancement reduces the number of comparators. In the original stochastic architecture, the SAH circuits each have a corresponding dynamic latched comparator, and the comparator converts these SAH voltages to a stochastic waveform which is then multiplexed. However, it is possible to multiplex the SAH voltages instead and pass a single selected value to the comparator. Since all of the columns are selecting the same relative SAH circuit, no

SAH needs to drive more than one comparator at a time. This enhancement reduces the number of needed comparators from five to one.

6.2.5 Stochastic Image Sensor

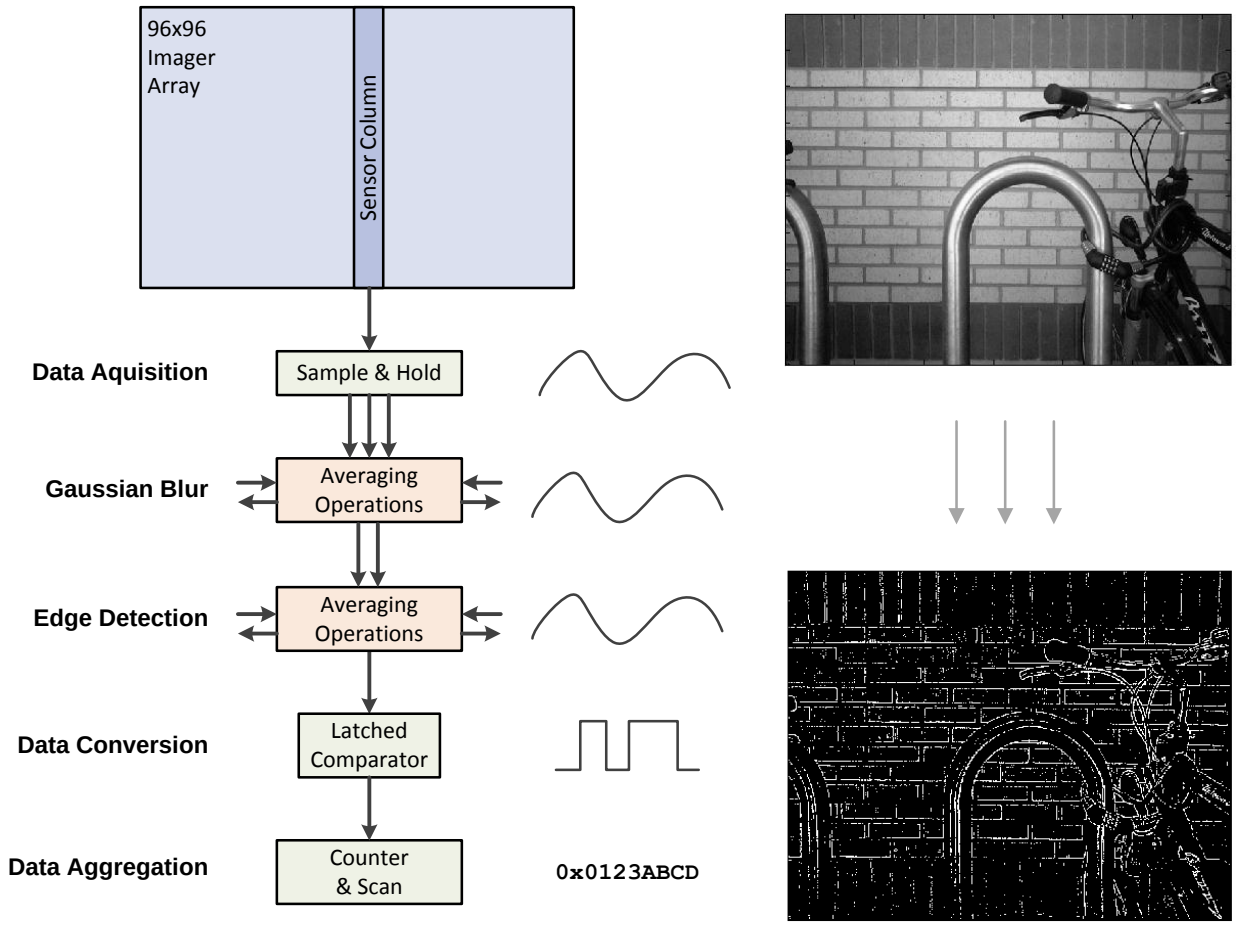


Figure 6.6: **High-level diagram of the stochastic image sensor architecture.** Each image sensor column includes five SAH circuits, switching networks for Gaussian blur and edge detection, a clocked comparator, and a counter.

A traditional image sensor has an array of photodiodes, organized into rows and columns. Each column has an ADC so that the rows of pixel values may be read out in parallel. In the stochastic image sensor architecture studied in this work, the traditional ADC is replaced with a stochastic ADC and a number of additional components, which can be seen in Figure 6.6. Included are five SAH circuits, to use as a scratch pad for computation. Next are two switching networks, one for Gaussian blur and one for edge detection, which perform

the bulk of the computation. Finally is the stochastic ADC, which aggregates the result from the computation and converts it to a digital value.

Figure 6.6 provides a high-level understanding of this architecture, while Section 6.3 explains in detail each of the components used. The advantage of this architecture is its ability to produce computation within the image sensor itself for minimum area and energy overheads. The weighted averages performed here require a large number of multiply-accumulate (MAC) operations for a normal processor: 30/pixel/frame, which is 8.3M/sec for a 30fps rate on the 96x96 image sensor analyzed in this design, or 900M/sec for a modest 1-megapixel image sensor. Implementing this in a traditional system would require a substantial co-processor, which this technique replaces.

6.2.6 Related Work

Researchers at the University of Waterloo [114] and NEC [115] created CMOS image sensors that accomplished a unidirectional, first-order differentiation by comparing adjacent rows using the built-in correlated double-sampling mechanism used in many image sensors. This work uses an omnidirectional second-order differentiation, and provides tunable noise filtering.

Other works include sensors with attached co-processors or ASICs comprised of digital or mixed-signal processing [116–118] While these systems are more capable than the correlated double-sampling works, their area and power consumption are orders of magnitude greater than the techniques proposed in this work.

6.3 Proposed Architecture

This section contains a detailed description of the proposed architecture, including design requirements, circuit diagrams, and sizings. Figure 6.7 depicts a block diagram of the column.

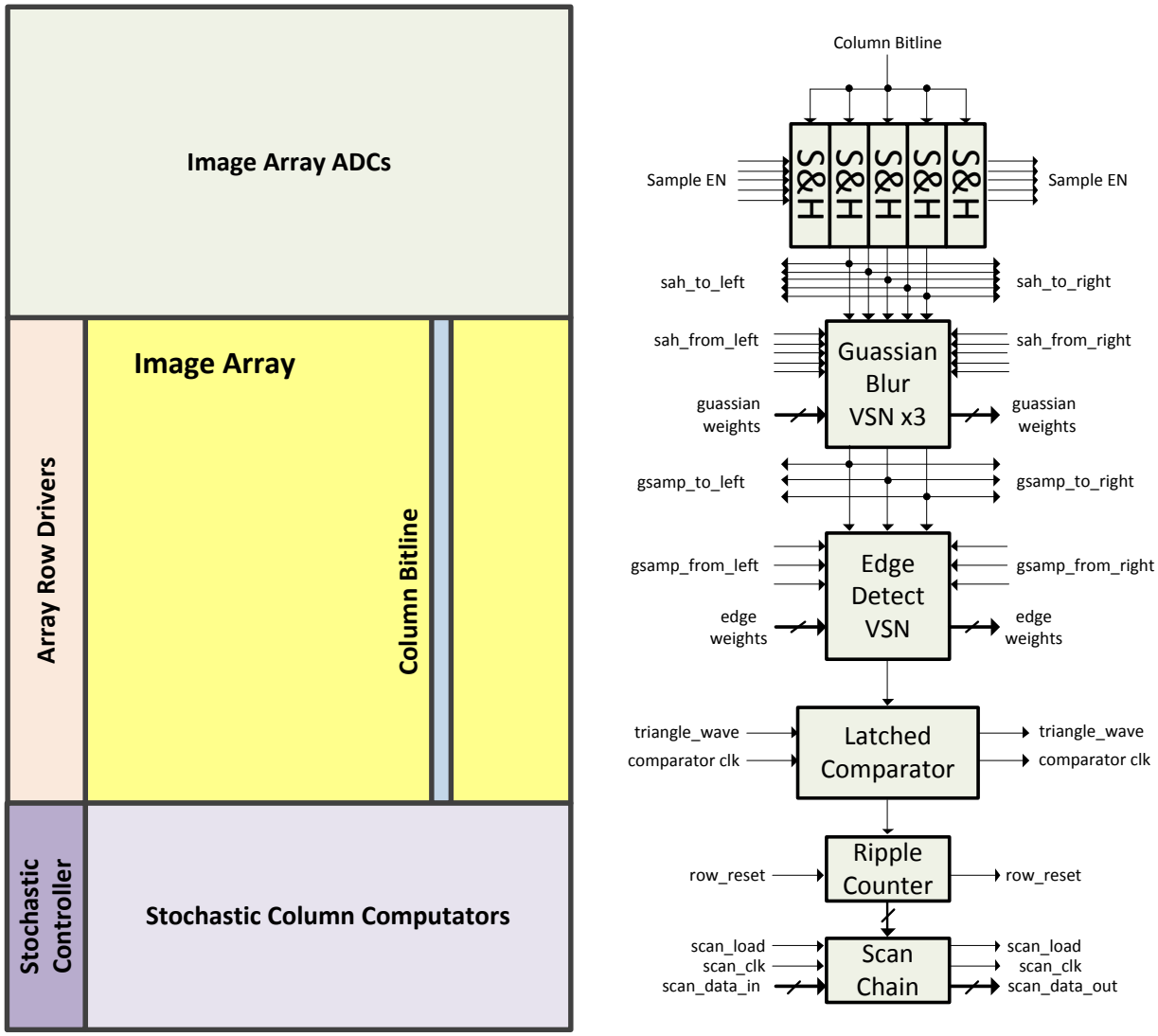


Figure 6.7: **Detailed block diagram of the proposed architecture.** A traditional ADC readout method is included for comparison.

6.3.1 Sample and Hold

The sample and hold circuit used in this design is shown in Figure 6.8. To reduce gate leakage, thick oxide devices were used except for the output source follower. To prevent subthreshold leakage, a feedback circuit is used to drive the output voltage onto the input of the sampling node. This technique greatly reduces worst case V_{GS}, of which subthreshold leakage is exponentially dependent, thereby drastically reducing worst-case leakage. In simulation gate-leakage through the source follower gate is dominate, but under monte-carlo, leaks away only 0.34mV in 1ms, with a standard deviation of 0.3mV. For nominal operation

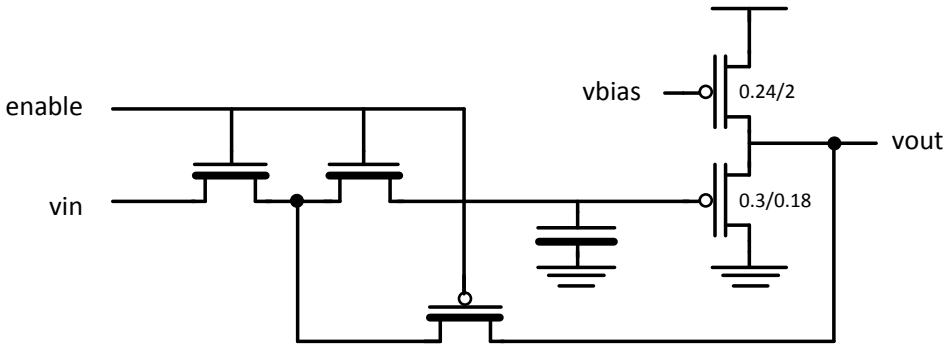


Figure 6.8: **Sample and hold circuit.**

of 30fps, the SAH would need to hold its value for 1.8ms

The output of the CMOS array sensor source followers range from 0.1-0.9V. Since a 1.5V signal is used for enabling the input to the SAH, V_t cutoff is not an issue with the pass-transistor. Due to a relatively long sample time, linearity of the capacitor is not a concern, so a moscap is used.

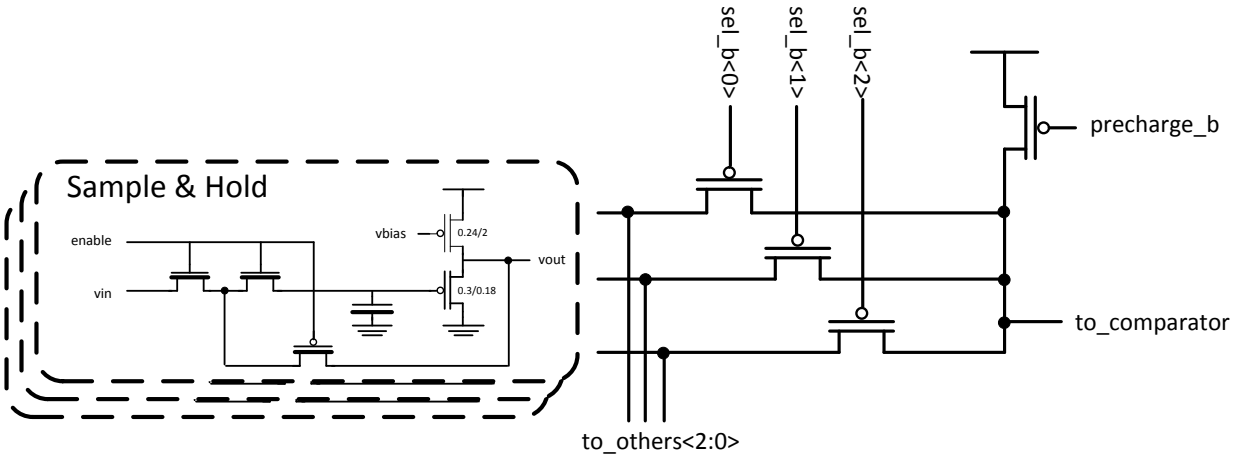


Figure 6.9: **Sample and hold voltage switching architecture** By precharging to V_{DD}, an arbitrarily weak pull-up device can be used for the SAH source follower. This prevents the tail current from dominating the power of the design.

The tail current through the output source follower is controlled by a bias voltage. By tuning the bias to be exceptionally weak (1.0V in simulation for this 1.2V process) the tail current can be reduced to below 10nA. Normally this intractably reduces bandwidth,

however, in our voltage switch network (VSN) we first precharge before reading, thereby requiring the source follower to function in only a single direction. This functionality is depicted in Figure 6.9. By setting the bias voltage to 1.2V, the source follower can be disabled entirely if it is unnecessary.

6.3.2 Gaussian and Edge Detection Voltage-Switch Networks

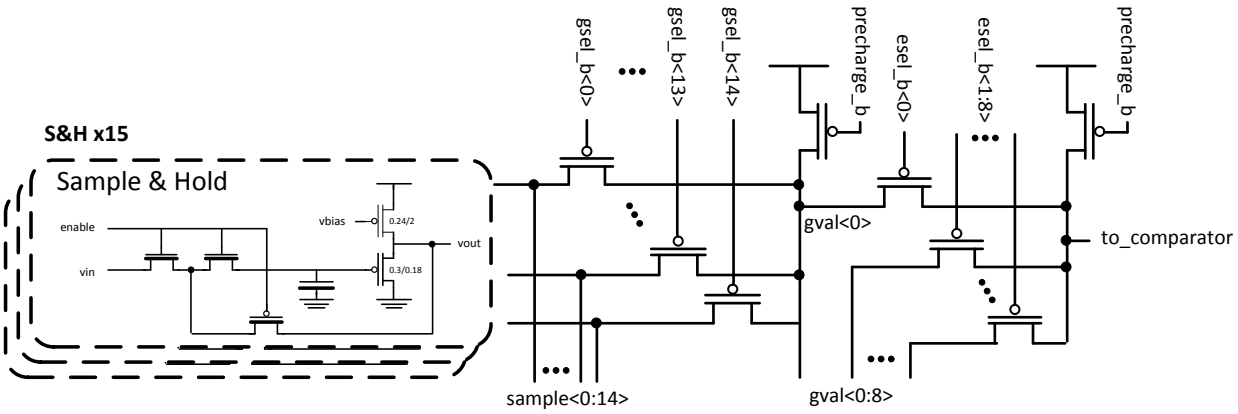


Figure 6.10: **Voltage switch network architecture.** The network has two stages: the first for Gaussian blur, and the second for edge detection. Both stages are precharged to assist the SAH circuit.

The Gaussian and edge detection VSNs form two stages as shown in Figure 6.10. In the first stage, three rows of Gaussian VSNs sample the five SAH rows to form the three rows of stochastic data needed for edge detection. These new rows are sampled by a single row of edge detection VSNs, which passes its value to the comparator. The outputs of the three Gaussian VSNs and the edge VSN are independently precharged each cycle.

Since a different SAH circuit samples each row, the weighting pattern for the Gaussian VSN needs to shift as well. This shift is performed at the controller to reduce the complexity in the array. However, each Gaussian VSN needs to select from all five SAH circuits from their own columns and neighboring columns, for a total of 15 inputs, even though for a given row they will only select from 9 of those inputs. The same 15 weightings are sent to all three Gaussian VSNs but are shifted for each of the three to create the three different rows. The relationships of the outputs for the three Gaussian VSN rows are fixed, so the edge detection

weights do not need to similarly shift.

The controller generating the weightings can disable either of the VSNs by selecting only a single input, so it is possible to have just a Gaussian blur effect, just edge detection, both, or neither. Similarly, the controller selects the sigma for the Gaussian blur and the types of edges that are detected.

6.3.3 Triangle Wave Generation

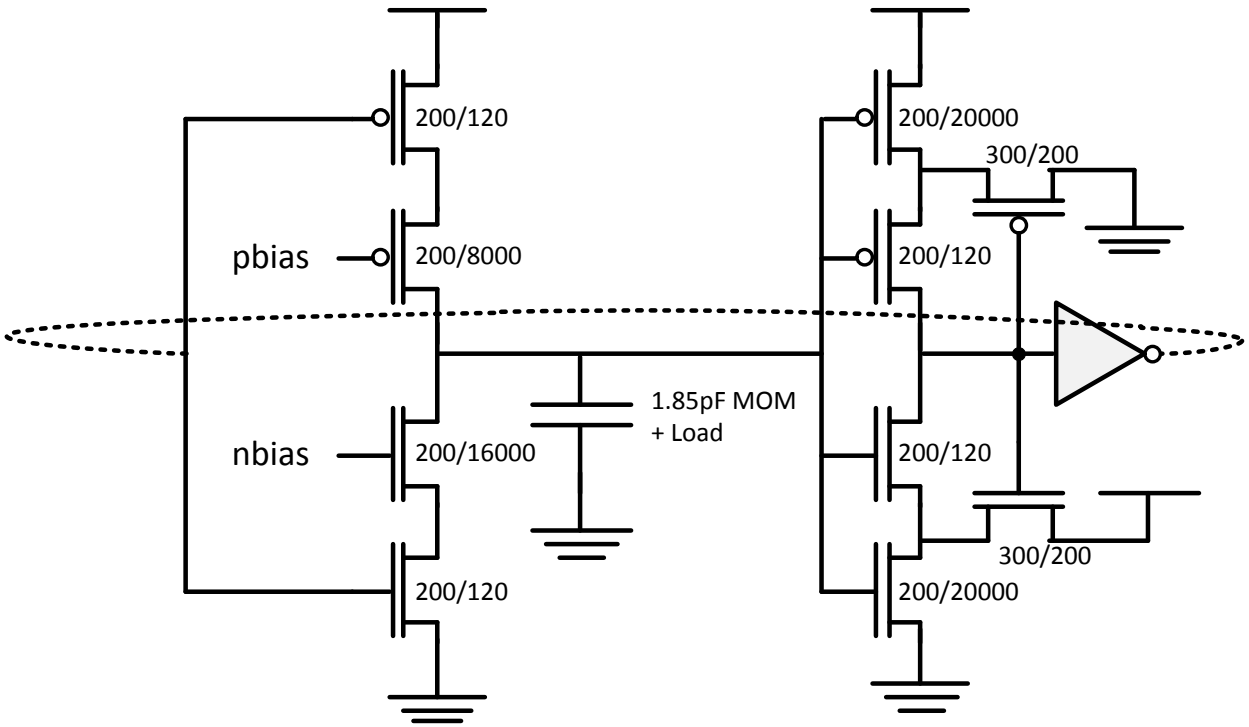


Figure 6.11: **Triangle wave generator** An asynchronous triangle wave is used as the uniform-random signal used for comparison in the stochastic ADC. Only one generator is needed for the array.

An asynchronous triangle wave is used as the uniform-random signal for comparison in the stochastic ADC. The outputs of the comparators are aggregated separately, so only one triangle wave generator is needed for this design.

The triangle wave generator is implemented with a three stage ring oscillator comprised of specialized components, as seen in Figure 6.11. The first stage is a current source inverter,

with a large metal-insulator-metal (MOM) capacitive load on its output. Bias voltages are used to control the current sources, and they are tuned to provide a slow, even ramp.

Following the current source inverter is a schmitt trigger and an inverter. Once the ramp reaches a sufficiently high or low point, the schmitt trigger and inverter fire, which reverses the direction of the current source inverter. Combined, these components create a tunable triangle wave.

Schmitt trigger designs intrinsically have short-circuit current. They are designed such that the output state feeds back to fight the input, which gives them their unique functionality. To reduce this current, the header and footer transistors in the stack have $20\mu m$ lengths. This makes the short circuit power roughly equal to the power of charging and discharging the capacitor and load. The current source power is similarly reduced with long lengths. This is possible since the triangle wave does not need to be high frequency to provide randomness; it only needs to be uncorrelated to the comparator clock.

Separate supplies are used to operate the triangle wave generator in order to provide tunability of its peak voltages. An isolated p-substrate is not used, so a body-effect V_t shift occurs for some NMOS devices. This is not an issue, however, since tunability of the current sources and the supplies are provided. The MOM cap is distributed among the columns with thick metal routing to reduce local kickback from the latched comparators.

6.3.4 Dynamic Latched Comparator

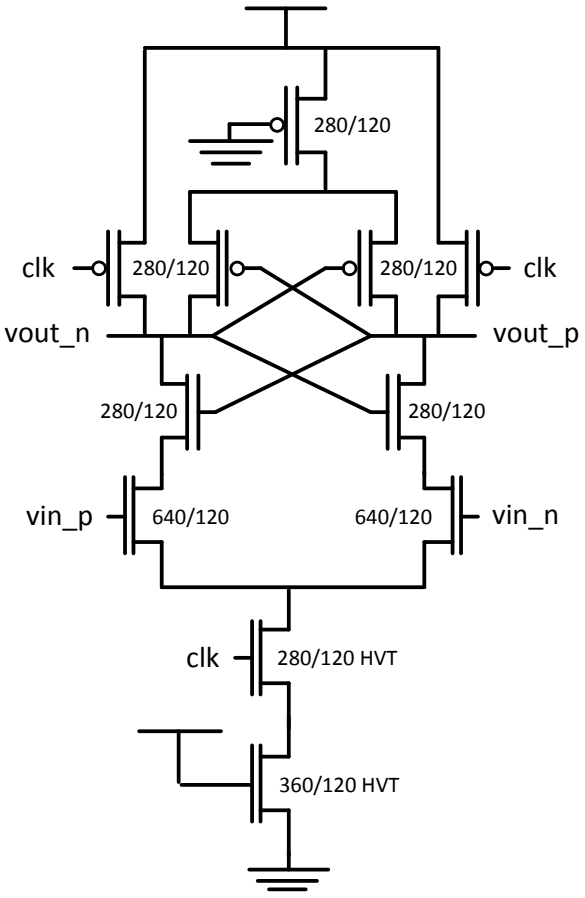


Figure 6.12: **Dynamic latched comparator.**

The dynamic latched comparator is designed to reduce energy consumption per comparison, since it is the most active unit in the column. Unlike the aggregating counter that follows it, it cannot be easily voltage scaled without greatly increasing offset variation. The majority of the devices are sized to be minimum width and length, since increasing their sizes further has a minimal effect on variation. Indeed, random dopant fluctuation (RDF) has a square root relationship to total device area [119]. To improve gain, a header is added to create a current steering effect, and the footer has a starving transistor in series.

Along with its low dynamic energy, this design was also chosen since it precharges with the negative phase of the clock, thereby creating a pulse on one of the outputs each cycle. This form of output is easily captured by a ripple counter without adding additional clock

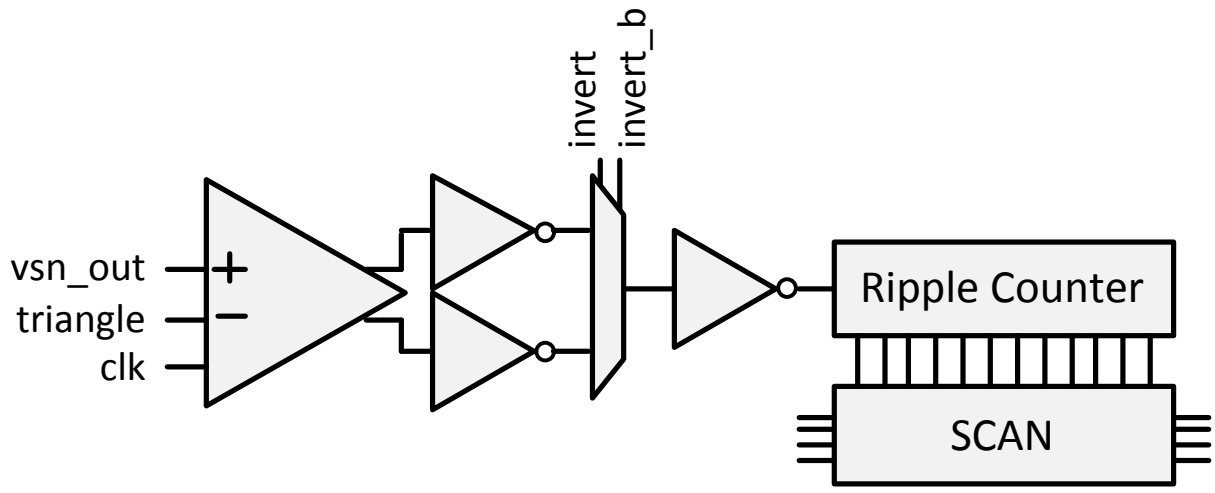


Figure 6.13: **Comparator and buffering ripple counter.** The multiplexer restores the inversion from Figure 6.3 that was lost by switching to a VSN topology.

loading. As seen in Figure 6.13, each output of the comparator is buffered with an inverter. The outputs are then multiplexed depending if inversion is needed, which is determined by the controller. This inversion is to restore the functionality of the inverter of Figure 6.3 that was lost when converting to a VSN topology.

6.3.5 Counter and Scan

The ripple counter is implemented with a transmission gate toggle flip-flop (TGTFF) design, which is depicted in Figure 6.14. The output clocks of one bit are the clocks to the next. To reduce power, local clock buffering is not used, and all devices are minimum size high-V_t devices. This design is robust under variation and voltage scaling, and can operate below $600mV$ for the frequencies needed in this application.

For read-out, the output of the ripple counter is latched using a scan chain. Immediately after scan latches the output the ripple counter can be reset to start the next row while scan operates in parallel.

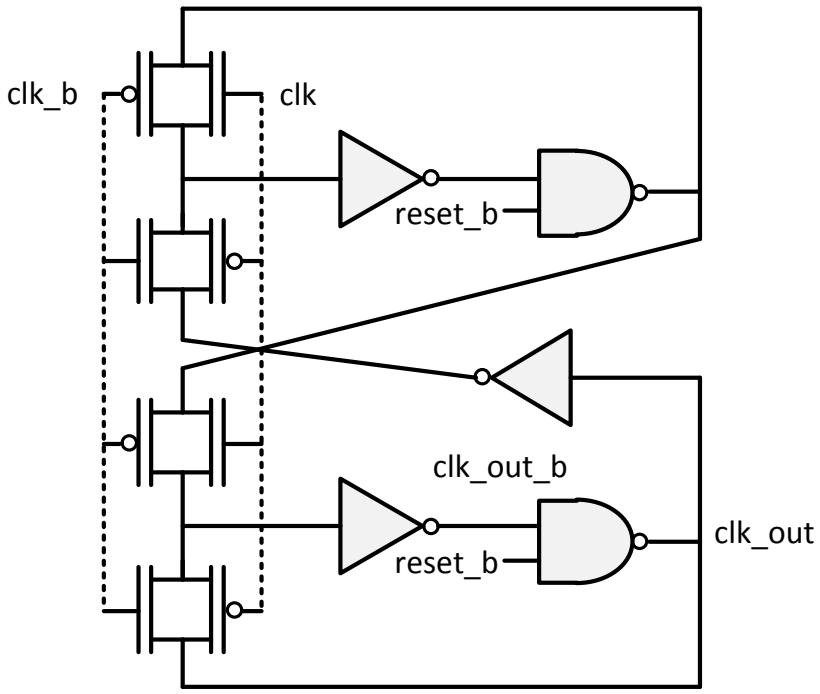


Figure 6.14: **Transmission gate toggle flip-flop.**

6.3.6 Stochastic image sensor controller

The block diagram for the stochastic image sensor controller is depicted in Figure 6.15. Its operation is broken down into five major sections: timing, Gaussian weight generation, edge weight generation, sample enable control, and scan control.

The Gaussian and edge weight generators each contain an LFSR and stochastic computing elements. The LFSR in the Gaussian generator is ten bits, while it is nine bits in the edge weight generator, in order to prevent direct correlation between the two units. The LFSR outputs are multiplied to create stochastic stream values of 0.5, 0.25, and 0.125, and a 3-bit, 1-hot register with a randomized barrel shifter generates factors of 0.33 and 0.66. These factors are then combined to generate the weightings needed for the VSN. Multiplexers in the units control the weightings to allow for different sigmas for the Gaussian weight generator, and for different edges for the edge weight generator. Each unit also contains an override so that the same bit may be selected each cycle.

The sample enable generates a configurable length pulse to be sent to the SAH circuits

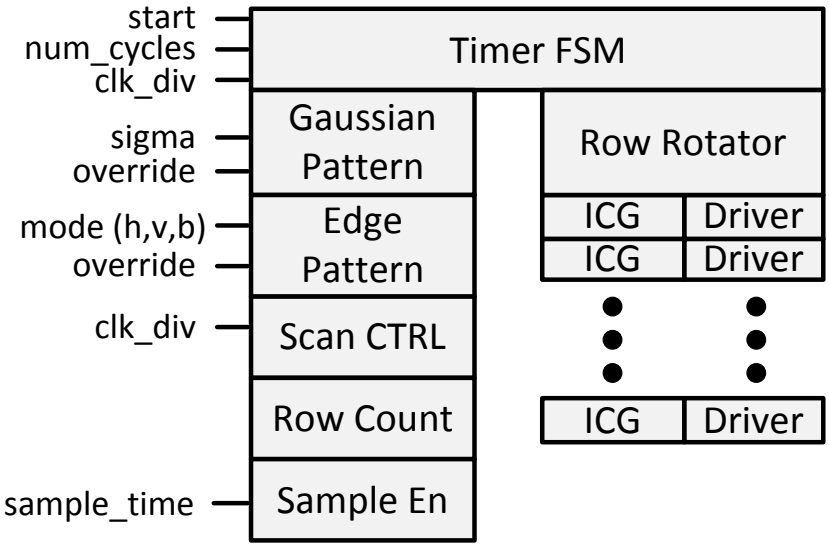


Figure 6.15: **Block diagram of the controller for the stochastic image sensor.**

at the beginning of each row. A 5-bit, 1-hot register with a sequential barrel shifter selects which SAH row is enabled. For the first row, all five SAH rows are enabled. The 1-hot register is also used to control the rotation of the Gaussian weights. The input of each of their output drivers contains a five-to-one multiplexer that is selected with this register.

The scan controller is a separate state machine that starts at the end of aggregation. It first loads the scan chain, then scans out each of the bits. The scan controller contains a clock divider so that it can operate at a much lower frequency than the rest of the controller, if needed.

The timing unit contains a sychronizer for the incoming start signal, which comes from the CMOS image array controller. Since it is from a clock domain with an unknown phase relationship, it is registered twice before being used. The timing unit looks for a rising edge, then resets all counters and other registered units. Next it counts up to a preset stopping time. While it is counting up, an integrated clock gating cell (ICG cell) passes the clock to the other units of the controller, which causes the weights to be generated.

The output drivers of the controller are registered, either directly or with an ICG, in order to prevent glitching which reduces energy consumption and protects correct operation. Dependencies between signals are enforced directly by clock ICGs with preceding dependencies. A tunable delay buffer is included to configure the delay between the comparator clock

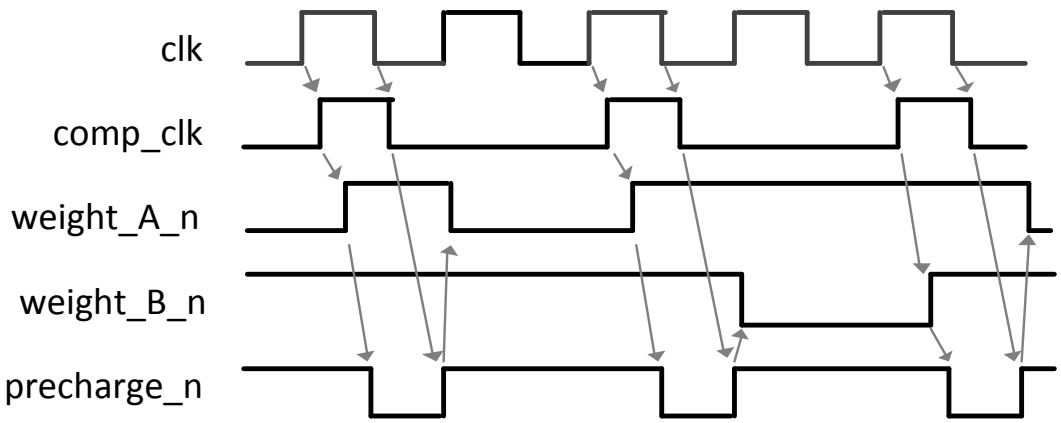


Figure 6.16: **Waveforms generated by the stochastic image sensor controller and their relationships.** The comparator clock, *comp_clk* is active-high, while the VSN weight and precharge signals are active-low. Dependencies are highlighted with arrows.

and the precharge for the next cycle.

A waveform of operation is shown in Figure 6.16. The comparator clock, *comp_clk* is active-high, while the VSN weight and precharge signals are active-low. First, precharge asserts, which prepares the VSN for discharging. Next, a weight signal is asserted, and is given almost three phases of the source clock (*clk*) to discharge. Finally, the clock is asserted, and the dynamic latched comparators evaluate.

The dependencies in the waveform are highlighted with arrows. The comparator clock triggers the deassertion of the active weight signal, which in turn starts the precharge for the next cycle. Since the dynamic latched comparator needs some time to evaluate, a tunable delay buffer is included to control this timing. The precharge ends once the comparators go back into reset, which enables the weights to assert again.

The controller was designed using a custom standard cell process in Cadence Virtuoso, which allowed for the design of low-energy gates and logic styles. Additionally, custom design allowed straightforward verification of low-voltage operation to further reduce energy consumption.

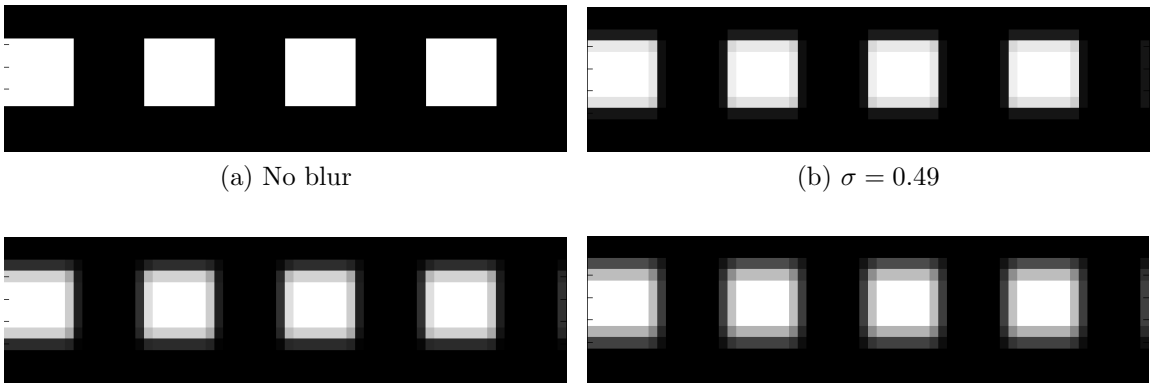
6.4 Simulated Results

To validate the architecture of this design, it was fully implemented in Cadence Virtuoso, including layout, and simulated using Spectre models via Synopsys FineSim.

For a baseline comparison, an ASIC implementation was designed and synthesized using Synopsys Design Compiler. The ASIC implementation has the same functionality as the described system, including configurable Gaussian blur and edge detection.

6.4.1 Simulated Output

Example output from the stochastic image sensor ADC is depicted in Figures 6.17, 6.18, and 6.19. A set of squares is drawn in each of the images with different filters applied. In these images, 96 columns are simulated, with 64 columns of data shown here. Each bit is computed in 128 cycles to reduce simulation time, as opposed to 1024 cycles.



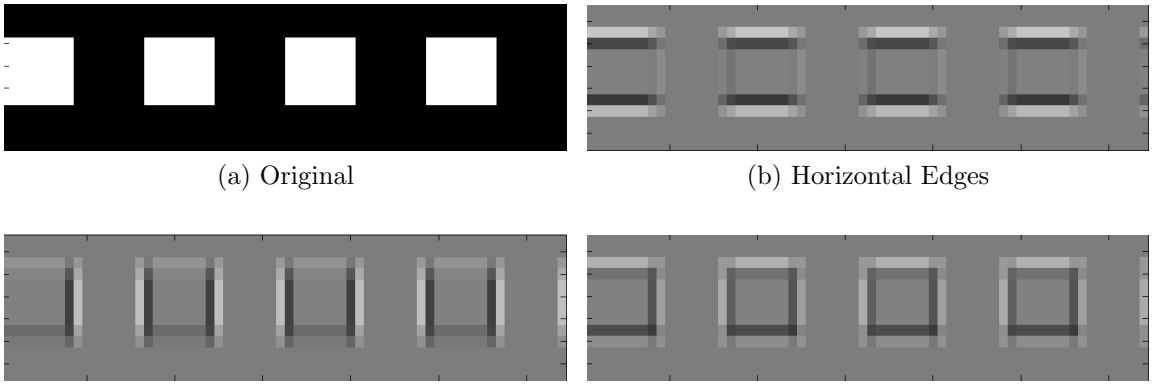
(a) No blur (b) $\sigma = 0.49$

(c) $\sigma = 0.60$ (d) $\sigma = 0.85$

Figure 6.17: **Example output of squares with Gaussian blur.** The range of sigma settings are shown.

In Figure 6.17, a Gaussian blur is applied with increasing sigma. As sigma increases, the amount of bluring effect also increases, which aides noise rejection, and guides the sensitivity of the edge detection stage.

In Figure 6.18, edge detection is to the pattern of squares. The edge detection unit is configurable, which allows the detection of horizontal edges, vertical edges, none, or both. By selecting only one type of edge, selectivity for this edge is increased, making those edges

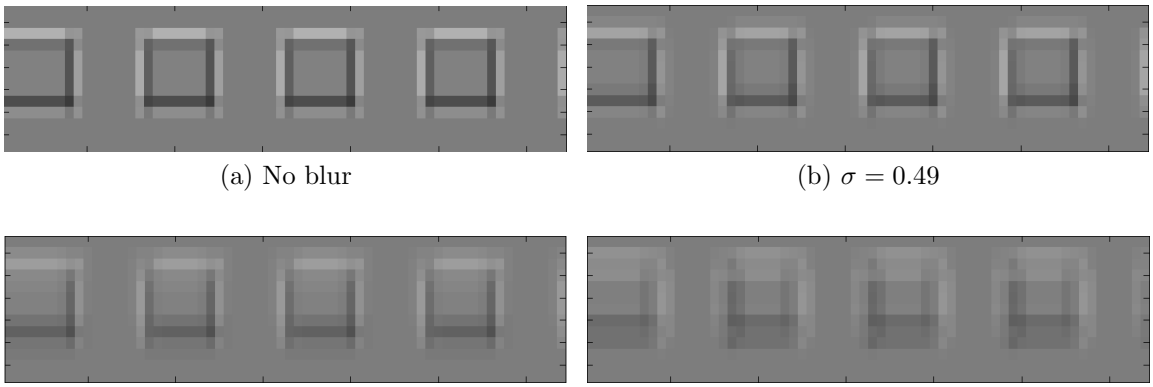


(a) Original (b) Horizontal Edges

(c) Vertical Edges (d) Both Edges

Figure 6.18: **Example output of squares with edge detection.**

more identifiable. This can be particularly useful for applications like object avoidance, where this selectivity can be used to find either walls (vertical edges) or ledges (horizontal edges).



(a) No blur (b) $\sigma = 0.49$

(c) $\sigma = 0.60$ (d) $\sigma = 0.85$

Figure 6.19: **Example output of squares with both edges detected and Gaussian blur.** The range of sigma settings for the Gaussian blur are shown.

In Figure 6.19 the two stages are combined. Both edges are detected and a Gaussian blur with increasing sigma is applied. Although the increasing sigma visually makes the transition blurrier to the human eye, the zero crossing in the images remain the same.

6.4.2 Area Analysis

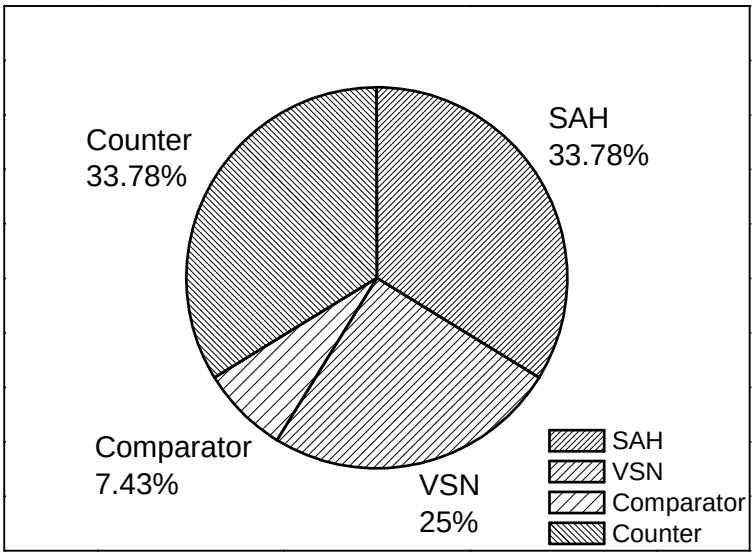


Figure 6.20: **Area breakdown of the stochastic ADC with VSN.**

An area breakdown is depicted in Figure 6.20. The total area of the stochastic ADC is $64,000\mu m^2$, which is 38% less than the area of the original ADC, which was $103,000\mu m^2$. The area of the ASIC implementation (not including the ADC) is $322,000\mu m^2$, more than 5x as large as the stochastic ADC. This area assumes 100% utilization for standard cell placement.

6.4.3 Power Analysis

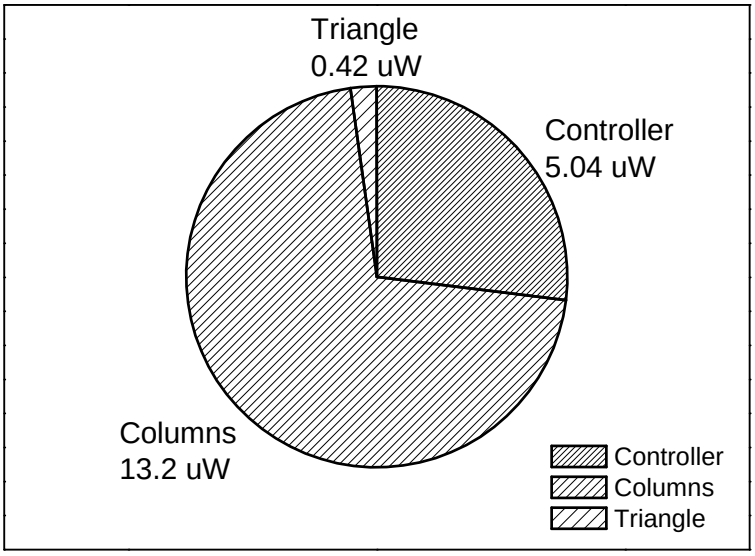


Figure 6.21: **Power breakdown of stochastic image sensor ADC (system).**

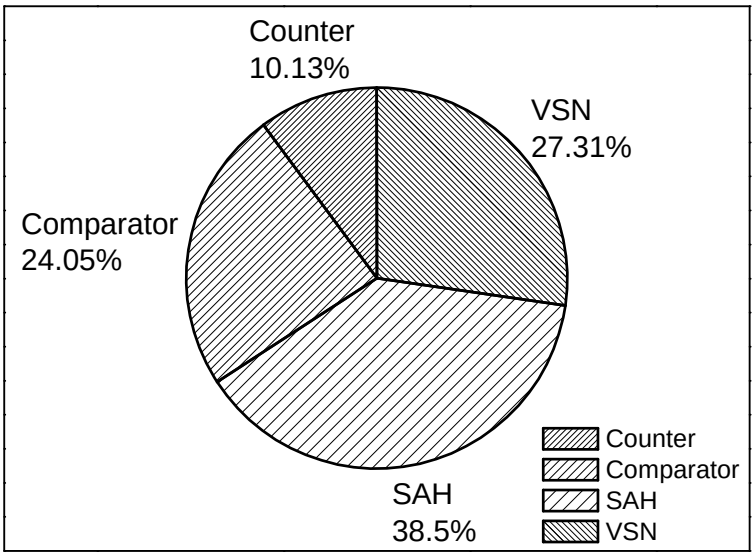


Figure 6.22: **Power breakdown of stochastic image sensor ADC (column).**

Figures 6.21 and 6.21 illustrate the power breakdown of the design. The total stochastic ADC power consumption is $18.7\mu W$ when running with a 10-bit counter at 30 frames/second. If accuracy is reduced to an 7-bit counter, this can be reduced by nearly 70%. The power for the ASIC implementation is $35.8\mu W$, not including ADC power. Assuming 100fJ/conversion-step, the ADC power for the ASIC implementation is estimated to be $7.1\mu W$.

6.5 Summary

In this work, we analyzed the use of stochastic computing for image sensor applications. The stochastic enabled image sensor is able to perform *in situ* edge detection and noise filtering. The area of the stochastic enabled image sensor is 5x smaller than an ASIC implementation, while being comparable in area to a typical image sensor implementation. The power consumption of the stochastic enabled image sensor is $18.66\mu W$, less than half of an ASIC implementation.

CHAPTER 7

Conclusion and Future Directions

While consumer computing devices have moved to increasingly smaller form factors, large, datacenter-size machines are continuing to be built to solve complex problems far outside of the capabilities of consumer devices. A recent machine, the "K computer", is a canonical example which draws 10 megawatts of power to do global weather simulation and detailed chemical reaction simulations, with 705,024 processing cores completing more than 10 petaflops of calculations per second.

Although advances in computing technology have greatly improved system capabilities and form factors, they have introduced problems in heat dissipation, reliability, and yield. Large scale systems are greatly affected by this, where their power usage is measured in megawatts, their reliability goal is running a mere 30 hours without system failure, and their processor count numbers in the hundreds of thousands.

This dissertation addresses the issues of energy efficiency, interconnect scalability, and system reliability within these large scale computers. Section 7.1 summarizes the contributions of this dissertation and Section 7.2 discusses future directions of this area of research.

7.1 Contributions

The key contributions of this dissertation are summarized as follows:

- Chapter 2 discussed Centip3De, a large scale near-threshold computing (NTC) system implemented in a 3D process technology. With a novel low-voltage cluster-based archi-

tecture, Centip3De achieved a 75x decrease in processor power and a 5.1x improvement in energy efficiency. This approach enables designs with a large number of cores, with 64 ARM Cortex-M3 cores demonstrated in this design, making it one of the largest academic projects to date. Centip3De tackled issues of power density, interconnect scaling, and multicore design.

- Chapter 3 described "Safety Razor" along with its novel, time-to-digital converter with sub-picosecond calibration. Safety Razor uses a timing estimation technique that recovers margins missed by canary-based approaches while avoiding the design complexity introduced by traditional Razor techniques. This project addresses yield and energy efficiency issues in large scale system design.

- Chapters 4 and 5 discussed interconnect reliability for large scale systems its efficient implementation. In particular, Chapter 4 discussed how network-on-chip can reroute traffic around broken network components, and analyzed an algorithm to safely reconfigure without the aide of an outside processing element. Chapter 5 expanded on this work and showed how network routers can be designed with a BIST that analyzes the router's own functionality, and then re-organizes its components to reduce the number of communication links in the network. The proposed failure-tolerant interconnect architecture "Vicis", allows for fault rates of over 1 in 2,000 gates while maintaining reliability. These projects address interconnect architecture and reliability in large scale systems where the inclusion of hundreds of thousands of cores makes failure tolerance a necessity.

- Chapter 6 discussed the use of stochastic computing as a method of processing multimedia information. By using neural-inspired processing elements, multimedia information can be efficiently processed with natural soft-fault tolerance. In this chapter, a stochastic computing enhanced image sensor with *in situ* edge detection and noise filtering was described and analyzed, with an achieved area improvement of 5x and energy efficiency improvement of 2x over an ASIC implementation. This project addresses reliability and energy efficiency in large scale systems.

7.2 Future Directions

The projects and ideas presented in this dissertation may be extended, this section briefly describes future directions for these future works.

3D Stacked Near-threshold Computing

Centip3De (described in Chapter 2) leveraged 3D stacking by building vertically expanding clusters. By doing this the 2D footprint of the design was reduced which improved the performance of the interconnect architecture. Centip3De used only two layers for a cluster, but this can be expanded significantly to further reduce the footprint. Taller stacks have the benefit of improving inter-unit routing within the cluster, *e.g.*, routing one pipeline stage to the next or routing the clock tree architecture, and allow all power-intensive layers to be placed near the heatsink where they will have better cooling capabilities. Strategies for how to partition and floorplan such an architecture will require significant research.

3D stacked NTC architectures will have a significantly larger die area than traditional designs. Since power can be reduced as much as 80x, this allows for 80x more silicon per socket, which makes yield a salient issue. Although "known good die" (KGD) techniques [18] can improve yield, it will still be challenging to manufacture entirely fault-free layers. Instead, designs should identify faults and work-around them. Having a large number of two-dimensionally "thin" clusters enables redundancy. Faulty cores within a layer could be identified with KGD techniques and disabled, leaving those clusters in a permanently boosted or semi-boosted mode. Further research includes improving KGD testing for a large number of cores, but design strategies will eventually need to include online testing for wearout, which can be accomplished efficiently with *in situ* sensors [120–123]. Current memories use spare columns for redundancy; this technique could be expanded in a 3D memory to sub-column sparing. The 3D stacked NTC architecture lends itself well to this redundancy strategy, which is also explored in Vicis network from Chapter 5.

Reliable Network-on-Chip

The Vicis network described in Chapter 5 used a "let fail and spare" strategy for reliability, where network reliability is maintained by disabling faulty components and working around them. The main source of area overhead (42%) in this design was from the diagnosis mechanism which would attempt to determine which components were faulty, while the re-routing algorithm was a mere 1.5% of the area of the design. Ideally, *in situ* degradation sensors should be used to reduce the overhead down to near that 1.5% [120–123].

High performance network-on-chip products often have simple router designs (*i.e.*, no adaptive routing, no virtual channels, *etc.*) with more networks or bandwidth. A canonical example is the Tilera TILE64, which has five simple networks in parallel to provide the bandwidth that it needs. Reducing reliability overhead is critical for these network designs since the original router areas are small. However, the architecture provides a higher degree of redundancy which can be leveraged. The network re-routing algorithm developed for Vicis could be expanded to support these higher dimensional networks, which can be combined with *in situ* degradation sensors to create an optimal solution.

The network re-routing algorithm for Vicis currently only finds a functional solution, however, it could be expanded to also balance network load so that there are not bottlenecks around network obstructions. The algorithm works by using a breadth-first search, this form of search can be extended to include path weight information. By doing this, network performance could be improved for light-wearout scenarios.

Logic Timing Speculation

The Safety Razor project used *in situ* delay slack measurements to estimate the critical path delay of the design. This strategy relies on speculation that is vulnerable to forms of attack. A key example is a program could run a no-op program in a tight loop for a long period of time, then switch to a "power virus" program that simultaneously causes super-ordinary power consumption while exercising critical paths. The first stage of this program may cause Safety Razor to adjust its DVFS operation aggressively such that the second stage causes a timing failure. Even without the no-op stage, a power virus has been

shown to cause significant IR and Ldi/dt droops well outside of bounds of normal programs, thereby necessitating a V_{DD} margin of 10% or more [53].

Protection against attacks can take two forms. First, algorithms and methodologies could be developed to protect Safety Razor's operation. This could include adding on-chip test vectors to simulate a power virus for training purposes, intelligently classifying running programs to account for their contribution during training, sensing shifts in IR or Ldi/dt droops to trigger operation of a "safe" mode, or performing memory based analysis (perhaps on cache misses) to determine when program sections are changing. Proving that one or more of these strategies is completely resilient would be challenging.

The second approach is to include a coarse-grain rollback mechanism in the design. One of the key advantages of Safety Razor was its architectural non-invasiveness, which could be lost depending on the approach taken here. The strategy would be to provide a very coarse-grain mechanism that is easy to implement, or to piggyback on an existing mechanism, such as transactional memory [124]. Transactional memory is now seeing adoption in some commercial products, such as the BlueGene/Q processor used in the IBM Sequoia supercomputer [125], making it a good fit for this approach.

An additional issue for the rollback approach the ability to guarantee that timing violations have been witnessed. A brief IR droop may cause timing violation at different registers than the one being currently viewed by the TDC. This guarantee can be accomplished by using the *in situ* timing measurements to tune a canary. The additional benefit of the rollback approach is the ability to recover margins lost by that power virus. A 10% voltage margin for power viruses induces a 21% increase in power, which can be recovered by this strategy.

Stochastic Computing

The stochastic computing enhanced image sensor described in Chapter 6 was a demonstration of the viability of this technology, but significant enhancements can still be made. First, the zero crossing edge detection algorithm used in this technique identifies transitions from positive to negative values and vice versa to determine edges. Once a value of a pixel has been identified as sufficiently positive, it is possible to discontinue the calculation at that

point, thereby saving energy.

Next, the voltage switching variant of this design used only multiplexers to accomplish its computation. Using sample and hold circuits, it is possible to build feedback mechanisms to accomplish more complex operations such as division and square root [106]. This can expand the capabilities of the stochastic image sensor beyond linear filters.

Third, the stochastic computing process used in this design requires exponentially longer sequences to achieve higher resolution. Although analog voltages are being switched, the output of the comparator is binary, which means that the analog information of the input was effectively lost. By using low-area RC filters for intermediate stages computation stages, more information can be saved, which will reduce the length of the needed computation and ultimately save time and energy.

Finally, the stochastic outputs of the image sensor can be combined to make decisions in some applications. For instance, instead of an output image from the sensor a tilt value could be generated which would help a mobile system level itself. The stochastic nature of this design lends itself well to higher-level, bio-inspired decision making processes.

BIBLIOGRAPHY

[1] Herman H. Goldstine. *The Computer from Pascal to Von Neumann*. Princeton University Press, Princeton, NJ, USA, 1972.

[2] TOP500 Supercomputing Sites. http://www.top500.org.

[3] M. Yokokawa, F. Shoji, A. Uno, M. Kurokawa, and T. Watanabe. The k computer: Japanese next-generation supercomputer development project. In *International Symposium on Low Power Electronics and Design*, pages 371 –372, Aug. 2011.

[4] T. Maruyama, T. Yoshida, R. Kan, I. Yamazaki, S. Yamamura, N. Takahashi, M. Hondou, and H. Okano. Sparc64 viiifx: A new-generation octocore processor for petascale computing. *Micro, IEEE*, 30(2):30 –40, Mar. 2010.

[5] Ian Foster. *Designing and building parallel programs: Concepts and tools for parallel software engineering*. Addison-Wesley, Reading, Massachusetts, 1995.

[6] United States Environmental Protection Agency (EPA). Enterprise server and data center efficiency initiatives. 2007.

[7] Moore's Law Timeline. http://www.intel.com/pressroom/kits/events/moores_law_40th/.

[8] Intel ARK. http://ark.intel.com.

[9] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54:67–77, May 2011.

[10] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38, Apr. 1965.

[11] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th edition, 2010.

[12] M.T. Bohr. Interconnect scaling-the real limiter to high performance ulsi. In *Electron Devices Meeting, 1995., International*, Dec. 1995.

[13] Tezzaron Semiconductor FaStack®Technology. http://tezzaron.com/technology/FaStack.htm.

[14] Deepak Sekar, Calvin King, Bing Dang, Todd Spencer, Hiren Thacker, Paul Joseph, Muhannad Bakir, and James Meindl. A 3d-ic technology with integrated microchannel cooling. In *Interconnect Technology Conference, 2008. IITC 2008. International*, pages 13 –15, June 2008.

[15] Tezzaron Semiconductor Octopus DRAM. http://www.tezzaron.com/memory/Octopus.html.

[16] Gabriel H. Loh. 3D-Stacked Memory Architectures for Multi-core Processors. *SIGARCH Comput. Archit. News*, 36(3):453–464, 2008.

[17] Taeho Kgil, Ali Saidi, Nathan Binkert, Steve Reinhardt, Krisztian Flautner, and Trevor Mudge. PicoServer: Using 3D stacking technology to build energy efficient servers. *J. Emerg. Technol. Comput. Syst.*, 4(4), 2008.

[18] J. U. Knickerbocker, P. S. Andry, B. Dang, R. R. Horton, M. J. Interrante, C. S. Patel, R. J. Polastre, K. Sakuma, R. Sirdeshmukh, E. J. Sprogis, S. M. Sri-Jayantha, A. M. Stephens, A. W. Topol, C. K. Tsang, B. C. Webb, and S. L. Wright. Three-dimensional silicon integration. *IBM Journal of Research and Development*, 52(6):553 –569, Nov. 2008.

[19] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, and Trevor Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. *Proc. Micro*, page 7, 2003.

[20] T. Kehl. Hardware self-tuning and circuit performance monitoring. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 188 –192, Oct. 1993.

[21] D. Blaauw, S. Kalaiselvan, K. Lai, Wei-Hsiang Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pages 400 –622, Feb. 2008.

[22] Eric Karl, David Blaauw, Dennis Sylvester, and Trevor Mudge. Reliability modeling and management in dynamic microprocessor-based systems. *Proc. Design Automation Conference*, pages 1057–1060, 2006.

[23] Tae-Hyoung Kim, R. Persaud, and C.H. Kim. Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits. *IEEE Journal of Solid-State Circuits*, 43(4):874 –880, Apr. 2008.

[24] T.M. Austin. Diva: a reliable substrate for deep submicron microarchitecture design. In *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on*, pages 196 –207, 1999.

[25] Shantanu Gupta, Shuguang Feng, Amin Ansari, and Scott Mahlke. Stagenet: A reconfigurable fabric for constructing dependable cmps. *IEEE Transactions on Computers*, 60:5–19, 2011.

[26] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, and S. Kottapalli. A 45nm 8-core enterprise Xeon® processor. In *Proc. International Solid-State Circuits Conference*, pages 56–57, Feb. 2009.

[27] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan. 2008.

[28] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, Liewei Bao, J. Brown, M. Mattina, Chyi-Chang Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook.

Tile64 - processor: A 64-core soc with mesh interconnect. In *Proc. International Solid-State Circuits Conference*, pages 88 –598, Feb. 2008.

[29] R.S. Patti. Three-dimensional integrated circuits and the future of system-on-chip designs. *Proceedings of the IEEE*, 94(6):1214 –1224, June 2006.

[30] T. Karnik, D. Somasekhar, and S. Borkar. Microprocessor system applications and challenges for through-silicon-via-based three-dimensional integration. *Computers Digital Techniques, IET*, 5(3):205 –212, May 2011.

[31] Matt Wordeman, Joel Silberman, Gary Maier, and Michael Scheuermann. A 3d system prototype of an edram cache stacked over processor-like logic using through-silicon vias. In *Proc. International Solid-State Circuits Conference*, pages 186 –187, Feb. 2012.

[32] S. Borkar. 3d integration for energy efficient system design. In *Proc. Design Automation Conference*, pages 214–219, June 2011.

[33] Dae Hyun Kim, K. Athikulwongse, M. Healy, M. Hossain, Moongon Jung, I. Khorosh, G. Kumar, Young-Joon Lee, D. Lewis, Tzu-Wei Lin, Chang Liu, S. Panth, M. Pathak, Minzhen Ren, Guanhao Shen, Taigon Song, Dong Hyuk Woo, Xin Zhao, Joungho Kim, Ho Choi, G. Loh, Hsien-Hsin Lee, and Sung Kyu Lim. 3d-maps: 3d massively parallel processor with stacked memory. In *Proc. International Solid-State Circuits Conference*, pages 188–190, Feb. 2012.

[34] David Fick, Ronald G. Dreslinski, Bharan Giridhar, Gyouho Kim, Sangwon Seo, Matthew Fojtik, Sudhir Satpathy, Yoonmyung Lee, Daeyeon Kim, Nurrachman Liu, Michael Wieckowski, Gregory Chen, Trevor Mudge, Dennis Sylvester, and David Blaauw. Centip3de: A 3930dmips/w configurable near-threshold 3d stacked system with 64 arm cortex-m3 cores. In *Proc. International Solid-State Circuits Conference*, pages 190–192, Feb. 2012.

[35] M. Horowitz, E. Alon, D. Patil, S. Naffziger, Rajesh Kumar, and K. Bernstein. Scaling, power, and the future of CMOS. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 7–15, Dec. 2005.

[36] Bo Zhai, R.G. Dreslinski, D. Blaauw, T. Mudge, and D. Sylvester. Energy efficient near-threshold chip multi-processing. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 32–37, Aug. 2007.

[37] R.G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits. *Proceedings of the IEEE*, 98(2):253–266, Feb. 2010.

[38] G. Chen, M. Fojtik, Daeyeon Kim, D. Fick, Junsun Park, Mingoo Seok, Mao-Ter Chen, Zhiyoong Foo, D. Sylvester, and D. Blaauw. Millimeter-scale nearly perpetual sensor system with stacked battery and solar cells. In *Proc. International Solid-State Circuits Conference*, pages 288–289, Feb. 2010.

[39] J. Kwong, Y.K. Ramadass, N. Verma, and A.P. Chandrakasan. A 65 nm sub-v_t microcontroller with integrated sram and switched capacitor dc-dc converter. *IEEE Journal of Solid-State Circuits*, 44(1):115–126, Jan. 2009.

[40] G. Chen, H. Ghaed, R. Haque, M. Wieckowski, Yejoong Kim, Gyouho Kim, D. Fick, Daeyeon Kim, Mingoo Seok, K. Wise, D. Blaauw, and D. Sylvester. A cubic-millimeter energy-autonomous wireless intraocular pressure monitor. In *Proc. International Solid-State Circuits Conference*, pages 310–312, Feb. 2011.

[41] Ronald G. Dreslinkski, Bo Zhai, Trevor Mudge, David Blaauw, and Dennis Sylvester. An Energy Efficient Parallel Architecture Using Near Threshold Operation. *Proceedings Parallel Architecture and Compilation Techniques*, pages 175–188, 2007.

[42] Krisztián Flautner, Nam Sung Kim, Steve Martin, David Blaauw, and Trevor Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the 29th annual international symposium on Computer architecture*, pages 148–157, 2002.

[43] M. Qazi, K. Stawiasz, L. Chang, and A.P. Chandrakasan. A 512kb 8t sram macro operating down to 0.57 μv with an ac-coupled sense amplifier and embedded data-retention-voltage sensor in 45 nm soi cmos. *IEEE Journal of Solid-State Circuits*, 46(1):85 –96, Jan. 2011.

[44] Bo Zhai, D. Blaauw, D. Sylvester, and S. Hanson. A sub-200mv 6t sram in 0.13 μm cmos. In *Proc. International Solid-State Circuits Conference*, pages 332 –606, Feb. 2007.

[45] J. Kulkarni, B. Geuskens, T. Karnik, M. Khellah, J. Tschanz, and V. De. Capacitive-coupling wordline boosting with self-induced vcc collapse for write vmin reduction in 22-nm 8t sram. In *Proc. International Solid-State Circuits Conference*, pages 234 –236, Feb. 2012.

[46] L. Chang, R.K. Montoye, Y. Nakamura, K.A. Batson, R.J. Eickemeyer, R.H. Dennard, W. Haensch, and D. Jamsek. An 8t-sram for variability tolerance and low-voltage operation in high-performance caches. *IEEE Journal of Solid-State Circuits*, 43(4):956 –963, April 2008.

[47] ARM Cortex-M3. `http://www.arm.com/products/CPUs/ARM_Cortex-M3.html`.

[48] OpenCores: Scan Based Serial Communication. `http://opencores.org/project,scan_based_serial_communication`.

[49] ARM Cortex-A9. `http://www.arm.com/products/processors/cortex-a/cortex-a9.php`.

[50] S. Das, D. Roberts, Seokwoo Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning DVS processor using delay-error detection and correction. *IEEE Journal of Solid-State Circuits*, 41(4):792–804, Apr. 2006.

[51] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D.M. Bull, and D.T. Blaauw. RazorII: In situ error detection and correction for pvt and ser tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, Jan. 2009.

[52] K.A. Bowman, J.W. Tschanz, Nam Sung Kim, J.C. Lee, C.B. Wilkerson, S.-L.L. Lu, T. Karnik, and V.K. De. Energy-efficient and metastability-immune resilient circuits for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):49–63, Jan. 2009.

[53] D. Bull, S. Das, K. Shivashankar, G.S. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation. *IEEE Journal of Solid-State Circuits*, 46(1):18–31, Jan. 2011.

[54] K.A. Bowman, J.W. Tschanz, S.L. Lu, P.A. Aseron, M.M. Khellah, A. Raychowdhury, B.M. Geuskens, C. Tokunaga, C.B. Wilkerson, T. Karnik, and V.K. De. A 45nm resilient microprocessor core for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 46(1):194–208, Jan. 2011.

[55] T.D. Burd, T.A. Pering, A.J. Stratakos, and R.W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35(11):1571–1580, Nov. 2000.

[56] K.J. Nowka, G.D. Carpenter, E.W. MacDonald, H.C. Ngo, B.C. Brock, K.I. Ishii, T.Y. Nguyen, and J.L. Burns. A 32-bit PowerPC system-on-a-chip with support for dynamic voltage scaling and dynamic frequency scaling. *IEEE Journal of Solid-State Circuits*, 37(11):1441–1447, Nov. 2002.

[57] M. Nakai, S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura. Dynamic voltage and frequency management for a low-power embedded microprocessor. *IEEE Journal of Solid-State Circuits*, 40(1):28–35, Jan. 2005.

[58] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *Proc. International Solid-State Circuits Conference*, pages 398–399, Feb. 2007.

[59] Kyoungho Woo, S. Meninger, T. Xanthopoulos, E. Crain, Dongwan Ha, and Donhee Ham. Dual-dll-based cmos all-digital temperature sensor for microprocessor thermal monitoring. In *Proc. International Solid-State Circuits Conference*, pages 68 –69,69a, Feb. 2009.

[60] Jong Sung Lee, K. Skadron, and Sung Woo Chung. Predictive temperature-aware DVFS. *Computers, IEEE Transactions on*, 59(1):127–133, Jan. 2010.

[61] N. Mehta and B. Amrutur. Dynamic supply and threshold voltage scaling for cmos digital circuits using in-situ power monitor. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(5):892–901, May 2012.

[62] B. Greskamp, Lu Wan, U.R. Karpuzcu, J.J. Cook, J. Torrellas, Deming Chen, and C. Zilles. Blueshift: Designing processors for timing speculation from the ground up. In *High Performance Computer Architecture, IEEE Symposium on*, pages 213–224, Feb. 2009.

[63] A.B. Kahng, Seokhyeong Kang, R. Kumar, and J. Sartori. Recovery-driven design: Exploiting error resilience in design of energy-efficient processors. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 31(3):404–417, Mar. 2012.

[64] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computer Survey*, 38(1):1, 2006.

[65] S. Bell et al. TILE64 Processor: A 64-Core SoC with Mesh Interconnect. *Proc. International Solid-State Circuits Conference*, pages 88–89, Feb. 2008.

[66] S. R. Vangal et al. An 80-tile sub-100w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.

[67] D. Sylvester, D. Blaauw, and E. Karl. ElastIC: An Adaptive Self-Healing Architecture for Unpredictable Silicon. *IEEE Design & Test*, 23(6):484–490, June 2006.

[68] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *ACM SIGARCH Computer Architecture News*, 20(2):278–287, 1992.

[69] Christopher J. Glass and Lionel M. Ni. Fault-tolerant wormhole routing in meshes without virtual channels. *IEEE Trans. Parallel and Distributed Systems*, 7(6):620–636, 1996.

[70] Jie Wu. A fault-tolerant and deadlock-free routing protocol in 2D meshes based on odd-even turn model. *IEEE Trans. Computers*, 52(9):1154–1169, 2003.

[71] Suresh Chalasani and Rajendra V. Boppana. Fault-tolerant wormhole routing algorithms for mesh networks. *IEEE Trans. Computers*, 44(7):848–864, 1995.

[72] A.A. Chien and Jae H. Kim. Planar-adaptive routing: Low-cost adaptive networks for multiprocessors. *Proc. ISCA*, pages 268–277, 1992.

[73] José Duato. A theory of fault-tolerant routing in wormhole networks. *IEEE Trans. Parallel and Distributed Systems*, 8(8):790–802, 1997.

[74] Seong-Pyo Kim and Taisook Han. Fault-tolerant wormhole routing in mesh with overlapped solid fault regions. *Parallel Computing*, 23(13):1937–1962, 1997.

[75] Samuel Rodrigo, José Flich, José Duato, and Mark Hummel. Efficient unicast and multicast support for CMPs. *Proc. Micro*, 2008.

[76] Jau-Der Shih. A fault-tolerant wormhole routing scheme for torus networks with nonconvex faults. *Information Processing Letters*, 88(6):271–278, 2003.

[77] Chien-Chun Su and Kang G. Shin. Adaptive fault-tolerant deadlock-free routing in meshes and hypercubes. *IEEE Trans. Computers*, 45(6):666–683, 1996.

[78] P.-H. Sui and S.-D. Wang. Fault-tolerant wormhole routing algorithm for mesh networks. *IEEE Computers and Digital Techniques*, 147(1):9–14, Jan. 2000.

[79] Jipeng Zhou and F.C.M. Lau. Adaptive fault-tolerant wormhole routing in 2d meshes. *Proc. IPDPS*, pages 8 pp.–, Apr. 2001.

[80] Jipeng Zhou and Francis C. M. Lau. Multi-phase minimal fault-tolerant wormhole routing in meshes. *Parallel Computing*, 30(3):423–442, 2004.

[81] Ching-Tien Ho and Larry Stockmeyer. A new approach to fault-tolerant wormhole routing for mesh-connected parallel computers. *IEEE Trans. Computers*, 53(4):427–439, 2004.

[82] V. Puente, J. A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. *ACM SIGARCH Computer Architecture News*, 32(2):198, 2004.

[83] M. E. Gomez et al. An efficient fault-tolerant routing methodology for meshes and tori. *IEEE Computer Architecture Letters*, 3(1):3, 2004.

[84] Daniel P. Siewiorek and Robert S. Swarz. *Reliable computer systems (3rd ed.): design and evaluation*. 1998.

[85] Shekhar Borkar. Microarchitecture and design challenges for gigascale integration. *Proc. Micro, keynote address*, 2004.

[86] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Proc. Micro*, 25(6):10–16, Nov. 2005.

[87] Tom R. Halfhill. Ambric's New Parallel Processor: Globally Asynchronous Architecture Eases Parallel Programming. *Microprocessor Report*, Oct. 2006.

[88] Massively Parallel Processing Arrays Techhnology Overview. *Ambric Technology Overview*, 2008.

[89] William J. Dally, Larry R. Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. *Proc. PCRCW*, pages 241–255, 1994.

[90] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky. BulletProof: a defect-tolerant CMP switch architecture. *Proc. HPCA*, pages 5–16, Feb. 2006.

[91] Sung-Jui Pan and Kwang-Ting Cheng. A framework for system reliability analysis considering both system error tolerance and component test quality. *Proc. Design, Automation, & Test in Europe*, pages 1581–1586, 2007.

[92] Dongkook Park, C. Nicopoulos, and Jongman Kimand N. Vijaykrishnanand C.R. Das. Exploring fault-tolerant network-on-chip architectures. *Proc. DSN*, pages 93–104, 2006.

[93] D. Bertozzi, L. Benini, and G. De Micheli. Low power error resilient encoding for on-chip data buses. *Proc. Design, Automation, & Test in Europe*, pages 102–109, 2002.

[94] Heiko Zimmer and Axel Jantsch. A fault model notation and error-control scheme for switch-to-switch buses in a network-on-chip. *Proc. CODES+ISSS*, pages 188–193, 2003.

[95] David Fick, Andrew DeOrio, Gregory Chen, Valeria Bertacco, Dennis Sylvester, and David Blaauw. A highly resilient routing algorithm for fault-tolerant NoCs. *Proc. Design, Automation, & Test in Europe*, 2009.

[96] John Keane, Shrinivas Venkatraman, Paulo Butzen, and Chris H. Kim. An array-based test circuit for fully automated gate dielectric breakdown characterization. *Proc. CICC*, pages 121–124, Sept. 2008.

[97] How Much Information? 2003. http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/.

[98] Martin Hilbert and Priscila Lpez. The worlds technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[99] Tilak Agerwala. Challenges on the road to exascale computing. In *Proc. International Conference on Supercomputing*, pages 2–2, 2008.

[100] Donald Grice. The roadrunner project and the importance of energy efficiency on the road to exascale computing. In *Proc. International Conference on Supercomputing*, pages 2–2, 2009.

[101] R Frackowiak, KJ Friston, C Frith, R Dolan, and JC Mazziotta. *Human Brain Function*. Academic Press, San Diego, California, 2nd edition, 2004.

[102] SS Haykin. *Neural Networks and Learning Machines*. Pearson Prentice Hall, Upper Saddle River, New Jersey, 3rd edition, 2009.

[103] Jinwook Oh, Junyoung Park, Gyeonghoon Kim, Seungjin Lee, and Hoi-Jun Yoo. A 57mw embedded mixed-mode neuro-fuzzy accelerator for intelligent multi-core processor. In *Proc. International Solid-State Circuits Conference*, pages 130 –132, Feb. 2011.

[104] Yongwoo Jeong and A.H. Titus. Biological inspired cmos foveated sensor: For neural network training. In *Proc. Sensors Applications Symposium*, pages 183 –187, Feb. 2011.

[105] B. R. Gaines. Stochastic computing. In *Proc. of spring joint computer conference*, pages 149–156, 1967.

[106] B. R. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, pages 37–172, 1969.

[107] P. Mars and H.R. Mclean. Implementation of linear programming with a digital stochastic computer. *Electronics Letters*, 12:516–517, 1976.

[108] W. J. Poppelbaum, Apostolos Dollas, J. B. Glickman, and C. O'Toole. Unary processing. *Advances in Computers*, 26:47–92, 1987.

[109] A. Alaghi and J.P. Hayes. Stochastic computing. In *ACM Trans. Embedded Computing Systems*, 2012.

[110] W.J. Gross, V.C. Gaudet, and A. Milner. Stochastic implementation of LDPC decoders. In *Signals, Systems and Computers*, pages 713–717, Nov. 2005.

[111] E.R. Fossum. CMOS image sensors: electronic camera-on-a-chip. *IEEE Transactions on Electron Devices*, 44(10):1689–1698, Oct. 1997.

[112] R.J. Baker. *CMOS: Circuit Design, Layout, and Simulation*. IEEE Press, 2nd edition, 2008.

[113] M. Petrou and C. Petrou. *Image Processing: The Fundamentals*. John Wiley & Sons, 2010.

[114] M. Tabet and R. Hornsey. Cmos image sensor camera with focal plane edge detection. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 1129–1133, 2001.

[115] Y. Muramatsu, S. Kurosawa, M. Furumiya, H. Ohkubo, and Y. Nakashiba. A signal-processing CMOS image sensor using a simple analog operation. *IEEE Journal of Solid-State Circuits*, 38(1):101–106, Jan. 2003.

[116] Pei-Yung Hsiao, Chia-Hao Lee, and Chia-Chun Tsai. A novel CMOS imager with 2-dimensional binarization and edge detection for highly integrated imaging systems. In *International Conference on Consumer Electronics*, pages 71–72, Jan. 2006.

[117] N. Takahashi and T. Shibata. A row-parallel cyclic-line-access edge detection CMOS image sensor employing global thresholding operation. In *Proc. International Symposium of Circuits and Systems*, pages 625 –628, June 2010.

[118] V.B. Baru and C.S. Deokar. VLSI implementation of real-time image segmentation. In *International Conference on Industrial Electronics*, pages 211–214, Dec. 2010.

[119] Y. Taur and T.H. Ning. *Fundamentals of Modern VLSI Devices*. Cambridge University Press, 2009.

[120] P. Singh, Cheng Zhuo, E. Karl, D. Blaauw, and D. Sylvester. Sensor-driven reliability and wearout management. *IEEE Design Test of Computers*, 26(6):40 –49, Nov. 2009.

[121] P. Singh, Zhiyoong Foo, M. Wieckowski, S. Hanson, M. Fojtik, D. Blaauw, and D. Sylvester. Early detection of oxide breakdown through in situ degradation sensing. In *Proc. International Solid-State Circuits Conference*, pages 190 –191, Feb. 2010.

[122] P. Singh, D. Sylvester, and D. Blaauw. Adaptive sensing and design for reliability. In *IEEE International Reliability Physics Symposium*, pages 676–682, May 2010.

[123] P. Singh, E. Karl, D. Sylvester, and D. Blaauw. Dynamic NBTI management using a 45nm multi-degradation sensor. *IEEE Transactions on Circuits and Systems*, 58(9):2026 –2037, Sept. 2011.

[124] T. Harris, A. Cristal, O.S. Unsal, E. Ayguade, F. Gagliardi, B. Smith, and M. Valero. Transactional memory: An overview. *Proc. MICRO*, 27(3):8–29, May 2007.

[125] H. Ruud. The BlueGene/Q compute chip. *Hot Chips*, Aug. 2011.