# Interconnect and Memory Design for Intelligent Mobile System

by

Jingcheng Wang

A dissertation submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical and Computer Engineering)
in The University of Michigan
2020

Doctoral Committee:

Professor Dennis Sylvester, Chair
Professor David Blaauw
Professor Reetuparna Das
Professor Hun-Seok Kim

Jingcheng Wang

jiwang@umich.edu

ORCID iD: 0000-0001-5831-9063

To my family and friends

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure**

# LIST OF TABLES

**Table**

# ABSTRACT

Technology scaling has driven the transistor to a smaller area, higher performance and lower power consuming which leads us into the mobile and edge computing era. However, the benefits of technology scaling are diminishing today, as the wire delay and energy scales far behind that of the logics, which makes communication more expensive than computation. Moreover, emerging data centric algorithms like deep learning have a growing demand on SRAM capacity and bandwidth. High access energy and huge leakage of the large on-chip SRAM have become the main limiter of realizing an energy efficient low power smart sensor platform.

This thesis presents several architecture and circuit solutions to enable intelligent mobile systems, including voltage scalable interconnect scheme, Compute-In-Memory (CIM), low power memory system from edge deep learning processor and an ultra-low leakage stacked voltage domain SRAM for low power smart image signal processor (ISP).

Four prototypes are implemented for demonstration and verification. The first two seek the solutions to the slow and high energy global on-chip interconnect: the first prototype proposes a reconfigurable self-timed regenerator based global interconnect scheme to achieve higher performance and energy-efficiency in wide voltage range, while the second one presents a non Von Neumann architecture, a hybrid in-/near-memory Compute SRAM (CRAM), to address the locality issue. The next two works focus on low-power low-leakage SRAM design for Intelligent sensors. The third prototype is a low power memory design for a deep learning processor with 270KB

custom SRAM and Non-Uniform Memory Access architecture. The fourth prototype is an ultra-low leakage SRAM for motion-triggered low power smart imager sensor system with voltage domain stacking and a novel array swapping mechanism. The work presented in this dissertation exploits various optimizations in both architecture level (exploiting temporal and spatial locality) and circuit customization to overcome the main challenges in making extremely energy-efficient battery-powered intelligent mobile devices. The impact of the work is significant in the era of Internet-of-Things (IoT) and the age of AI when the mobile computing systems get ubiquitous, intelligent and longer battery life, powered by these proposed solutions.

# CHAPTER I

# Introduction

## 1.1 Technology Scaling and Intelligent Mobile System

Guided by the Moore's Law [1], the minimum feature size of transistor keeps shrinking and the number of transistors on chip increases exponentially. As predicted by the Bell's Law [2], in the first two decades of the 21st century, we have witnessed a shifted from the PC era to the smart phone and ubiquitous computer era (Figure 1.1). Now we all get used to the convenience of fast wireless communication and excellent computing power brought by these battery-powered mobile systems. Tablet is as powerful as a PC with a touch-screen which allows us to continue the work even on the feet. Smart phone has become a indispensable part of our life, a personal assistant and a second brain. Smart watch and other wearable device monitor our health data and offload many frequently used functions to a smaller screen. Besides the ever growing demand in the handheld devices, many foresee the coming of Internet-of-Things (IoT) driven by the faster 5G communication and the advances in low power techniques. Wireless sensor nodes will get a large-scale deployment in many areas of life. For example, an interocular pressure sensor can be implanted in human eyes to prevent glaucoma [3], and infrastructure monitoring sensor can measure vibrations and material conditions in buildings, bridges and historical monuments to prevent accidents. The number of smart sensors is estimated to increase by 3X and reaches 1

trillion by 2023 [4].

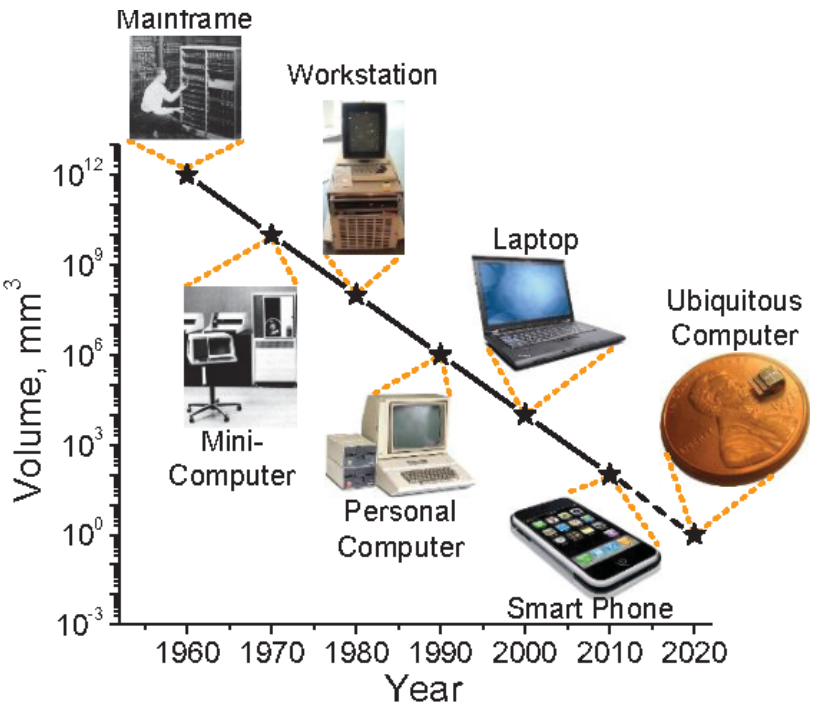



Figure 1.1: Bell's Law on scaling of computing platforms [5].

At the same time, benefiting from the high performance computing and "Big Data" driven by the technology scaling, artificial intelligence has embraced the "Second Wave" as described by DARPA's John Launchbury [6]: a shift from Handcrafted Knowledge to Statistical Learning. The recent advance in deep learning has led to many revolutionary improvements in various application domains, including computer vision, speech recognition, and nature language processing. Nowadays, AI has become the most promising and popular applications to both consumers and enterprises. With the explosive growth in demand, AI empowered intelligent system has become the hottest research and development topic (Figure 1.2), from the 8-Core neural engine in Apple's high performance System on Chip (SoC) [7] to the Edge TPU in Google's purpose-built ASIC [8], and the heterogeneous integrations of image sensor and object recognition processor [9].

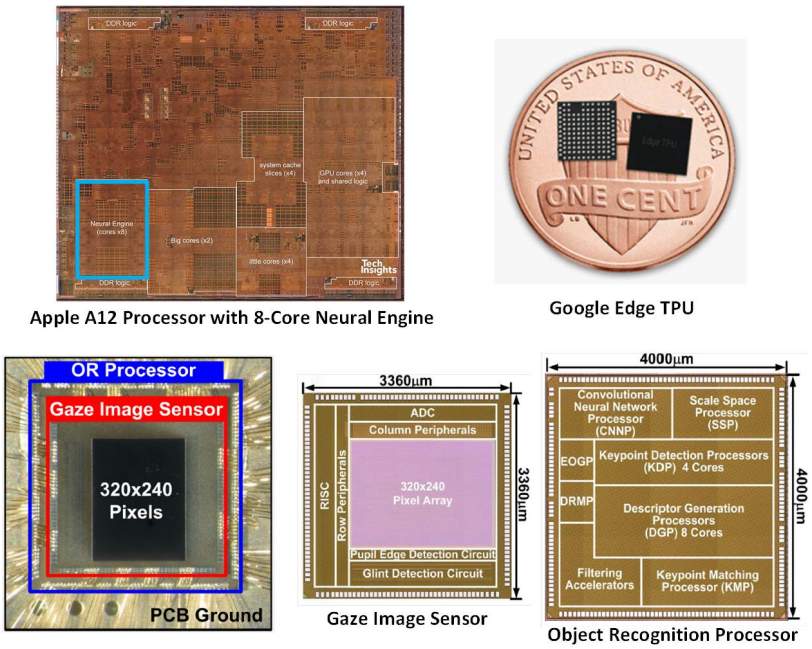



Figure 1.2: Apple A12 SoC with 8-Core Neural Engine (top-left), Google Edge TPU ASIC (top-right), KAIST 3D-stacked gaze-activated object-recognition system (bottom).

## 1.2 Challenges for Intelligent Mobile System

Thanks to technology scaling, more transistors can be integrated into a smaller chip area with increased performance, which helps the development of a more powerful and smaller battery-operated intelligent system. However, even though the minimum feature size of the transistor continues decreasing, today a number of factors have made the benefits of technology scaling diminishing. The first barrier we hit is the "Power Wall". To combat the sub-threshold leakage, the scaling of threshold voltage and supply voltage has been greatly slowed down, but the number of transistors continue to scale exponentially with a constant die size, resulting in an exponential increase in both active and leakage power [10], as shown in Figure 1.3. The small form factor of the mobile devices limits the size of the battery, but many AI applications like keyword spotting and face ID requires part of the system to be always-on. To extend the battery life, ultra-low power and energy-efficient circuits needs to be specially

optimized for the intelligent mobile system.

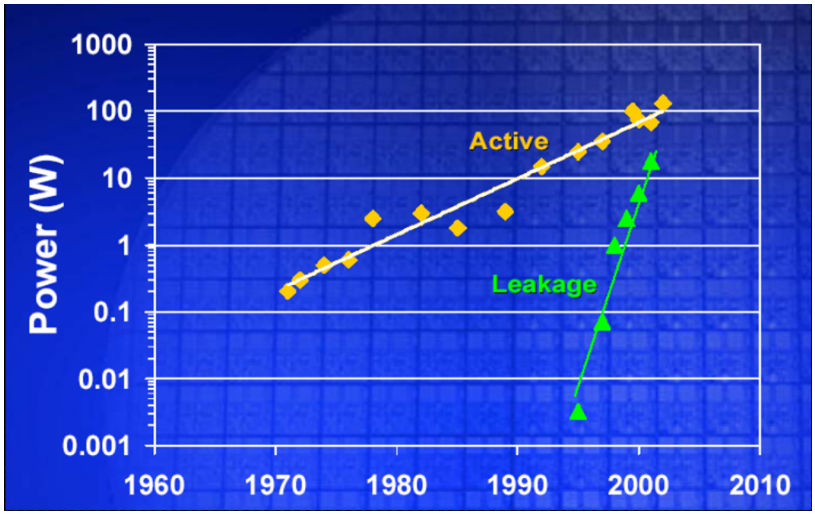



Figure 1.3: Processor power scales exponentially (Moore, ISSCC Keynote, 2003) [10].

### 1.2.1 Slow and High Energy Interconnect

The first challenge comes from the slow and high energy global interconnect. Even though technology scaling improves the driving strength and reduces the parasitic capacitance of the transistor, which generally results in higher performance and lower energy cost, the delay and energy scaling of interconnect fall far behind that of the logic. The reducing minimum width and pitches increase the wire resistance and inter-wire capacitance. Even with the deployment of low-K dielectric materials, unit length resistance and capacitance of wire doesn't scale much with the technology nodes. Therefore, the RC delay dominates the logic delay and becomes the main bottleneck of high-speed circuit, as shown in Figure 1.4. Technology scaling also non-uniformly improves the energy efficiency of computation and communication. As shown in Figure 1.5, the energy of a standard-cell-based double-precision fused-multiply-and-add (DFMA) is reduced from 50pJ in 40nm to 8.7pJ in 10nm, while the energy of 10mm 256-bit bus only scales from 310pJ to 200pJ. The cost of accessing 256 bits of operands from a distant memory is 6 times greater than the cost of com-

puting in 40nm. This ratio goes up to 23 times in 10nm. Scaling makes locality even more important, since fetching the operands is getting much more expensive than computing it. One solution to this challenge is to find a interconnect scheme that improves both the delay and energy-efficiency of communication, which will be introduced in Chapter II. Another solution is to exploit locality and reduce the amount of data movement per operation as much as possible, which leads to the proposal of a non Von Neumann Architecture, Compute-in-Memory (CIM), introduced in Chapter III.

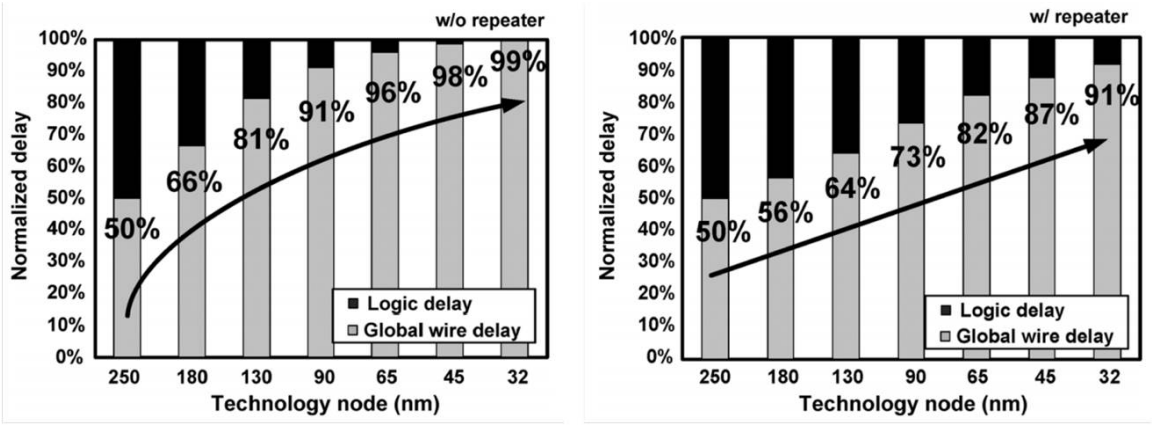



Figure 1.4: Delay scaling trend of logic and interconnect without (left) and with repeater (right) [11].

| | 2010 | 2017 | |
|---|---|---|---|
| **Process technology** | **40 nm** | **10 nm, high frequency** | **10 nm, low voltage** |
| $V_{DD}$ (nominal) | 0.9 V | 0.75 V | 0.65 V |
| Frequency target | 1.6 GHz | 2.5 GHz | 2 GHz |
| Double-precision fused-multiply add (DFMA) energy | 50 picojoules (pJ) | 8.7 pJ | 6.5 pJ |
| 64-bit read from an 8-Kbyte static RAM (SRAM) | 14 pJ | 2.4 pJ | 1.8 pJ |
| Wire energy (per transition) | 240 femtojoules (fJ) per bit per mm | 150 fJ/bit/mm | 115 fJ/bit/mm |
| Wire energy (256 bits, 10 mm) | 310 pJ | 200 pJ | 150 pJ |

Figure 1.5: Energy scaling trend of logic, SRAM, and interconnect [12].

### 1.2.2 Large On-Chip Memory

The growing size and power of on-chip memory has become one of the biggest challenges for low power Machine Learning and modern DSP chip design. Static random access memory (SRAM) is an indispensable part of Very-Large-Scale Integration (VLSI) system since it dominates most chip area and power consumption. Figure 1.6 shows that in the 45nm 8-core Enterprise Xeon Processor, more than 60% of the chip area is occupied by L1/2/3 cache [13]. And the power breakdown of a recent 40nm, 8-core server processor shows that over 50% of the active energy is dissipated in the caches and register files. What's more, the leakage power of a modern last-level cache (LLC) can be comparable to the active power of one simple core running in full speed [14].

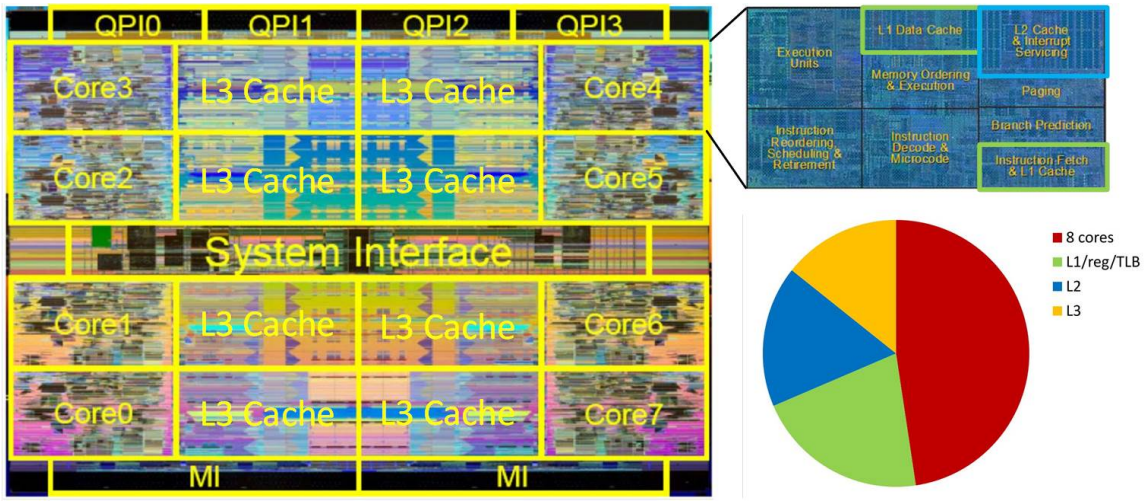



Figure 1.6: Die photo of 45nm 8-core Enterprise Xeon Processor (left and top-right) and power breakdown of an 8 core server chip (bottom-right) [13, 14].

This problem is exacerbated by deep learning algorithm. Since the complex training part of the neural network is usually done off-line in the cloud, hardware accelerator in the mobile SoC is only responsible for real-time inference. Compared to many traditional DSP algorithms, the core arithmetic computation of deep learning inference is very simple, mostly just matrix multiplication and accumulation (MAC) operation on 8-bit precision data. In contrast to the low requirement of computa-

tion, deep neural networks requires huge amount of parameters/weights storage. For example, AlexNet, that won the ImageNet competition in 2012, has 60 Million parameters, around 240MB of memory storage [15]. Even a keyword spotting network has 2.1 Million parameters, around 8.4MB of memory [16]. Many studies have shown that most of the energy and latency in the neural network accelerator is consumed by the data transfer. For example, in a conventional fully connected neural network (FCNN) accelerator, 62% of the energy and 97% of the latency is due to weight transfer, as shown in Figure 1.7 [17]. One of the reason is that memory access energy is way higher than arithmetic operation energy. In 45nm, DRAM access energy is 5000 times higher than a 8-bit MAC operation and SRAM access energy of a small 8KB bank is still 40 times higher (Figure 1.8).

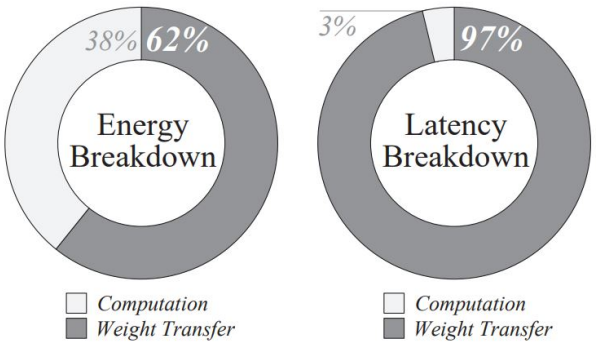



Figure 1.7: Energy and latency breakdown of a conventional FCNN accelerator [17].

| Integer | |
|---|---|
| Add | |
| 8 bit | 0.03pJ |
| 32 bit | 0.1pJ |
| Mult | |
| 8 bit | 0.2pJ |
| 32 bit | 3.1pJ |

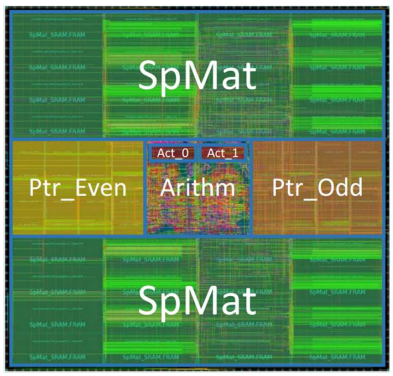
| FP | |
|---|---|
| FAdd | |
| 16 bit | 0.4pJ |
| 32 bit | 0.9pJ |
| FMult | |
| 16 bit | 1.1pJ |
| 32 bit | 3.7pJ |

| Memory | |
|---|---|
| Cache | (64bit) |
| 8KB | 10pJ |
| 32KB | 20pJ |
| 1MB | 100pJ |
| DRAM | 1.3-2.6nJ |

Figure 1.8: Energy costs for various operations in 45nm at 0.9V [14].

Since edge devices can't afford to have high power off-chip DRAM, people try

to squeeze all the weights into on-chip SRAM by pruning the network and compress the weights. Still a deep learning accelerator requires hundreds of Kilo-bytes of memory. A good example is Standford's Efficient Inference Engine (EIE) for "Deep Compressed" network. The layout photo in Figure 1.9 shows that 162KB SRAM takes 93% of the total area [18].




| | Power (mW) | (%) | Area ($\mu m^2$) | (%) |
|---|---|---|---|---|
| Total | 9.157 | | 638,024 | |
| memory | 5.416 | (59.15%) | 594,786 | (93.22%) |
| clock network | 1.874 | (20.46%) | 866 | (0.14%) |
| register | 1.026 | (11.20%) | 9,465 | (1.48%) |
| combinational | 0.841 | (9.18%) | 8,946 | (1.40%) |
| filler cell | | | 23,961 | (3.76%) |

Figure 1.9: SRAM takes 93% of total area of the Efficient Inference Engine (EIE) for Deep Compressed Network [18].

However, huge amount of on-chip SRAM brings another two challenges: low leakage and high yield. Customized low leakage SRAM become more crucial in neural network based intelligent sensor. Usually a battery-powered sensor node can frequently shut down the supply during sleep mode to extend battery life. However, since neural network requires all the weights to be retained after sleep, SRAM, now the largest portion of the system, has to stay on in standby mode. Due to the extremely low activity ration of a sensor-based system, the leakage power can even be higher than active power. Low voltage operation is one of the most effective ways to reduce both active power, due to its quadratic relationship with supply voltage, and leakage power, due to the Drain-Induced-Barrier-Lowering (DIBL) effect. However, low supply voltage greatly compromises the stability of SRAM operations. What's more, technology scaling makes it even harder to design robust SRAM, since process variation like Random Dopant Fluctuation (RDF) [19] and Line Edge Roughness (LER) [20] gets worse with reduced bitcell size. In Chapter IV and V, we are going

to focus on the issues in low power SRAM design for intelligent sensor nodes.

## 1.3 Contribution of This Work

This work contributes some solutions to the interconnect and memory challenges in development of intelligent mobile and edge devices. This proposal presents four works in detail.

Dynamic Voltage and Frequency Scaling (DVFS) is a frequently used low power technique in many mobile systems. The supply voltage may drop to near-threshold when the chip is in low workload and suddenly rise to full VDD when a burst of high workload appears. To make the slow RC delay of global interconnect go linear instead of quadratic with the wire length, repeater is usually inserted with carefully designed number and size based on the wire and repeater delay properties. However, the relative delay between wire and repeater is greatly affected by the voltage causing the optimal number and size of the repeaters at one voltage become sub-optimal at another voltage. In Chapter II, this proposal presents a reconfigurable self-timed regenerator based global interconnect scheme which enables graceful degradation of performance and power in wide range dynamic voltage/frequency scaled systems. A test chip demonstrates up to 40% and 25% better performance scaling than a traditional repeater based interconnect at 1V and 0.5V, respectively, in 45nm SOI CMOS. This work resulted in publications in ASSCC'15[21].

Conventional Von Neumann architecture involves frequently data transfer between memory and computation unit incurring significant energy and latency cost. This problem is amplified by technology scaling and "Data Centric" application like deep learning. In Chapter III, this proposal presents a non Von Neumann architecture —a hybrid in-/near-memory Compute SRAM (CRAM) that uses 8T transposable bit-cell and vector-based, bit-serial arithmetic to accomplish a wide range of operations with flexible bit-width. The proposed design was implemented in a small IoT processor

in 28-nm CMOS consisting of a Cortex-M0 CPU and 8 CRAM banks of 16 kB each (128 kB total). The system achieves 475 MHz operation at 1.1 V and, with all CRAMs active, produces 30 GOPS or 1.4 GFLOPS on 32-bit operands. It achieves the energy efficiency of 0.56 TOPS/W for 8-bit multiplication and 5.27 TOPS/W for 8-bit addition at 0.6 V and 114 MHz. This work resulted in publications in ISCA'18[22], ISSCC'19[23], MICRO'19[24] and JSSC'19 (to appear).

In Chapter IV, this proposal presents a Deep Learning Accelerator (DLA) with all weights stored in 270KB custom low power SRAM and non-uniform memory architecture for intelligent edge computing, like keyword spotting and face detection. Implemented in 40nm CMOS, the DLA achieves 288µW power consumption of and 374 GOPS/W energy efficiency with the following techniques: 1) Flexible and compact memory storage for highly truncated fixed-point network weights ranging from 6–32 bit precision via programmable control; 2) All weights stored in 270KB on-chip SRAMs with four processing elements (PEs) located amidst them, minimizing data movement energy; 3) A non-uniform memory architecture provides optimal energy-density trade-off between small, low power memory banks for frequently used data (e.g., input neurons) and large, high density banks for the large amount of infrequently accessed data (e.g., synaptic weights); 4) A 0.6V custom 8T SRAM with both active power reduction techniques like low-swing bit-line and sequential decoder, and leakage reduction techniques like peripheral power-gating, array voltage clamping and bank-by-bank drowsy mode. This work resulted in publications in SiPS'15[25], ISSCC'17[26] and JSPS'18[27].

In Chapter V, this proposal presents an ultra-low leakage SRAM in a smart image signal processor (ISP) for an energy-efficient low-noise CMOS image sensor [28]. The system is designed for motion-triggered IoT applications empowered by change detection and three dedicated neural networks to do human detection, face detection, and face recognition respectively. Including main memory, frame buffer, and memory

for neural network weights, the system requires 6.4 Mbit of on chip SRAM in total. With a special designed differential 8T bitcell, we are able to bring down total leakage power of SRAM array to 2uW by retaining the data at 0.3V, and still achieve a good stability. What's more, 0.3V supply is generated directly on chip using a novel stacking array technique instead of a DC-DC converter. The proposed design is taped-out using TSMC 40nm Low Power technology in April 2019.

# CHAPTER II

# Reconfigurable Self-Timed Regenerators for Wide-Range Voltage Scaled Interconnect

## 2.1 Introduction

Near-threshold (NT) operation has been shown to provide a reasonable balance between energy efficiency and performance demands for a wide range of applications [29, 30], particularly in the mobile space. However, even with the recent focus on energy efficiency, high single-thread performance demands still dictate nominal voltage operation at times. Wide-range dynamic voltage and frequency scaling (DVFS) enables operation across the energy/performance design space, but requires underlying circuits to scale across voltage in a robust and predictable manner. Without this, the ability to adapt to dynamic runtime constraints will be limited.

Recent work has shown how to optimize logic [31, 32] and memory [33] across both near-threshold and full voltage regimes. However, little work has addressed interconnect optimization across this wide voltage range. Unlike logic delay, which changes dramatically with supply voltage, interconnect RC delay is insensitive to voltage scaling. This leads to different optimization approaches in comparison to logic and memory. As designs are limited by their critical path, interconnections that are poorly optimized for certain voltage modes cause the entire design to suffer.

Optimal repeater insertion for a long interconnect differs significantly at full and near-threshold (NT) voltages. The optimal repeater count $N_{opt}$ and size $w_{opt}$ are given by the well-known equations [34] in Fig. 2.1. As supply voltage reduces, the effective repeater driver resistance $R_d$ increases relative to the interconnect resistance $r_w$, which remains constant. Wire capacitance $c_w$ and gate capacitance $C_g$ also remain constant as voltage scales. Therefore, $N_{opt} \propto 1/\sqrt{R_d(1+\gamma)}$ and $w_{opt} \propto \sqrt{R_d}$, such that at low VDD fewer, yet larger, repeaters are optimal.

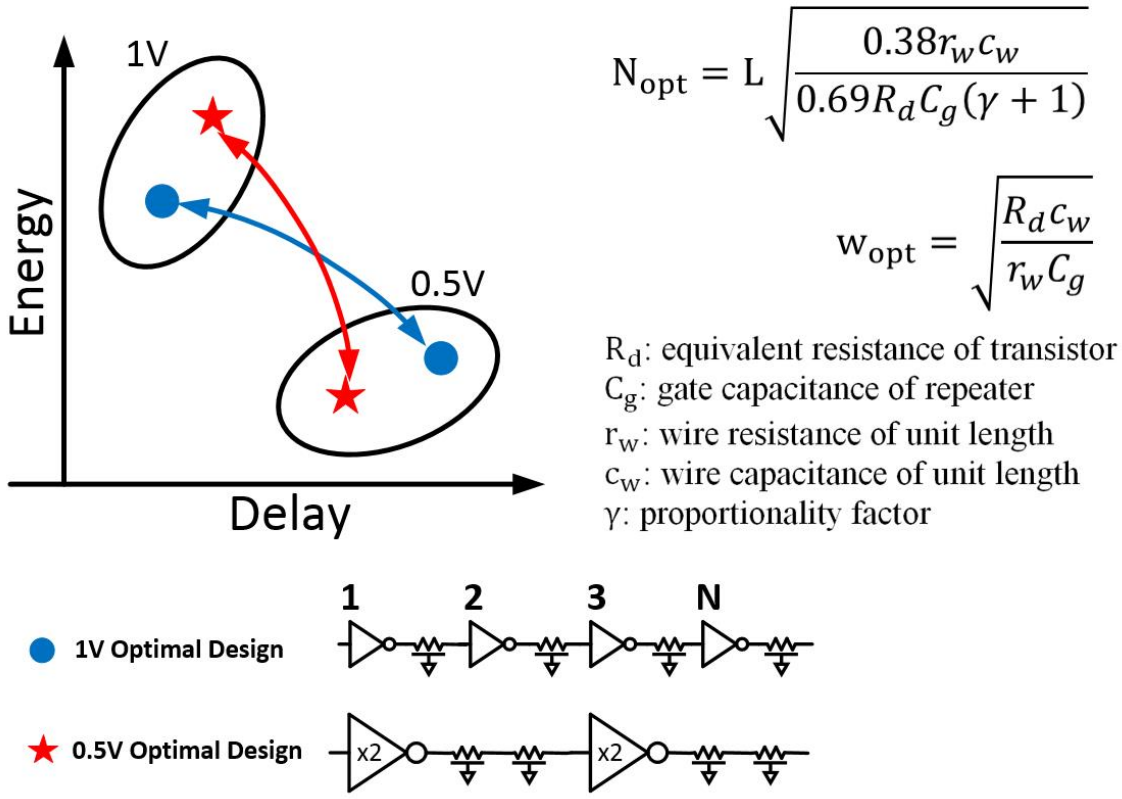



Figure 2.1: Differing optimal repeater designs for high and low supply voltages lead to sub-optimality in wide-range voltage scaled systems.

For the 45nm SOI technology used in this work, nominal voltage is 1V while 0.5V can be considered near threshold, hence we consider this range during optimization. In this technology, $R_d$ increases by roughly 4× from 1V to 0.5V, therefore an optimized interconnect at 0.5V uses half as many repeaters of twice the size as an in-

terconnect optimized for 1V. Operating repeated interconnects at a voltage they were not targeted for leads to large sub-optimality in energy and delay, shown conceptually in Fig. 2.1.

On-chip interconnect has been studied in-depth by the circuit community with many specialized designs, such as low-swing transceivers, being proposed to save energy and increase throughput. However, within circuit blocks, long wires are repeated with inverters and buffers by commercial place and route tools. While specialized transceivers are desirable for well-defined interconnections spanning long distances, we propose using regenerators for shorter, within-block, wired interconnects in voltage scaled systems when simplicity, low overhead, and ease of integration into a design is valued over absolute performance and energy improvements. This proposed technique does not replace specialized interconnect techniques, but instead is meant to replace repeaters for general purpose use.

## 2.2 Proposed Approach

The poor voltage scalability of repeater-based interconnect currently forces the designer to choose between a design that is optimal at either full or NT voltages, but not both. Furthermore, the interconnect delay does not track the fanout-of-4 (FO4) inverter delay, characteristic of how digital circuits scale with voltage, and hence the interconnect will become performance-limiting for the entire design during either full or NT operation if traditional design methodologies are followed. SPECTRE simulations of industrial wire and device models provided by a 45nm foundry are shown in Table 2.1 with results matching the analytical predictions of Figure 2.1. The baseline repeaters were inverters in this simulation. As expected, NT favored fewer, larger repeaters as compared to nominal voltage.

An obvious approach to overcome the $N_{opt}$ discrepancy between VDD and NT operation is to selectively disable repeaters along an interconnect. However, this

| VDD (V) | Optimal Delay (ps/mm) | Optimal Size $w_{opt}$ (µm) | Optimal # $N_{opt}$ |
|---|---|---|---|
| **0.5 (NT)** | 1680 | 12.6 | 35 |
| **1.0 (Nom.)** | 740 | 6.3 | 49 |

*Interconnect configuration: 10mm length with minimum width and spacing.

Table 2.1: Simulated optimal repeater design.

only shifts the problem from drivability of the repeater to drivability of the bypass devices, amounting to a zero sum game. For instance, if transmission gates are used to bypass repeaters then they suffer similar Rd degradation to that of the repeater, unless driven by a separate nominal voltage supply, which incurs considerable level shifting and power delivery overheads.

We propose using single-ended regenerators based on [35, 36] which, unlike conventional repeaters, are single-ended gates attached along a wire. Instead of discrete input and output pins, regenerators rely on detection circuits to sense partial transitions along the wire, triggering a temporary regenerative drive of the wire until it has fully transitioned to a new value. Regenerators have the unique property of not partitioning a long interconnect into separate wire segments. If a regenerator is enabled, it acts as a repeater passively monitoring the interconnect and then actively driving it to transition. Disabling the regenerator in effect extends the repeated distance, as the inactive regenerator does not change the characteristics of the wire other than added parasitic capacitance. Using regenerators addresses the scalability of the number of inserted repeaters, but to address repeater size we also add regenerators in parallel and selectively enable them.

Fig. 2.2 shows a circuit schematic for our proposed regenerator, named Reconfigurable Self-Timed Regenerator (RSTR), which is based on [35] but with extensions for reconfiguration. The new reconfigurable components are highlighted in red.

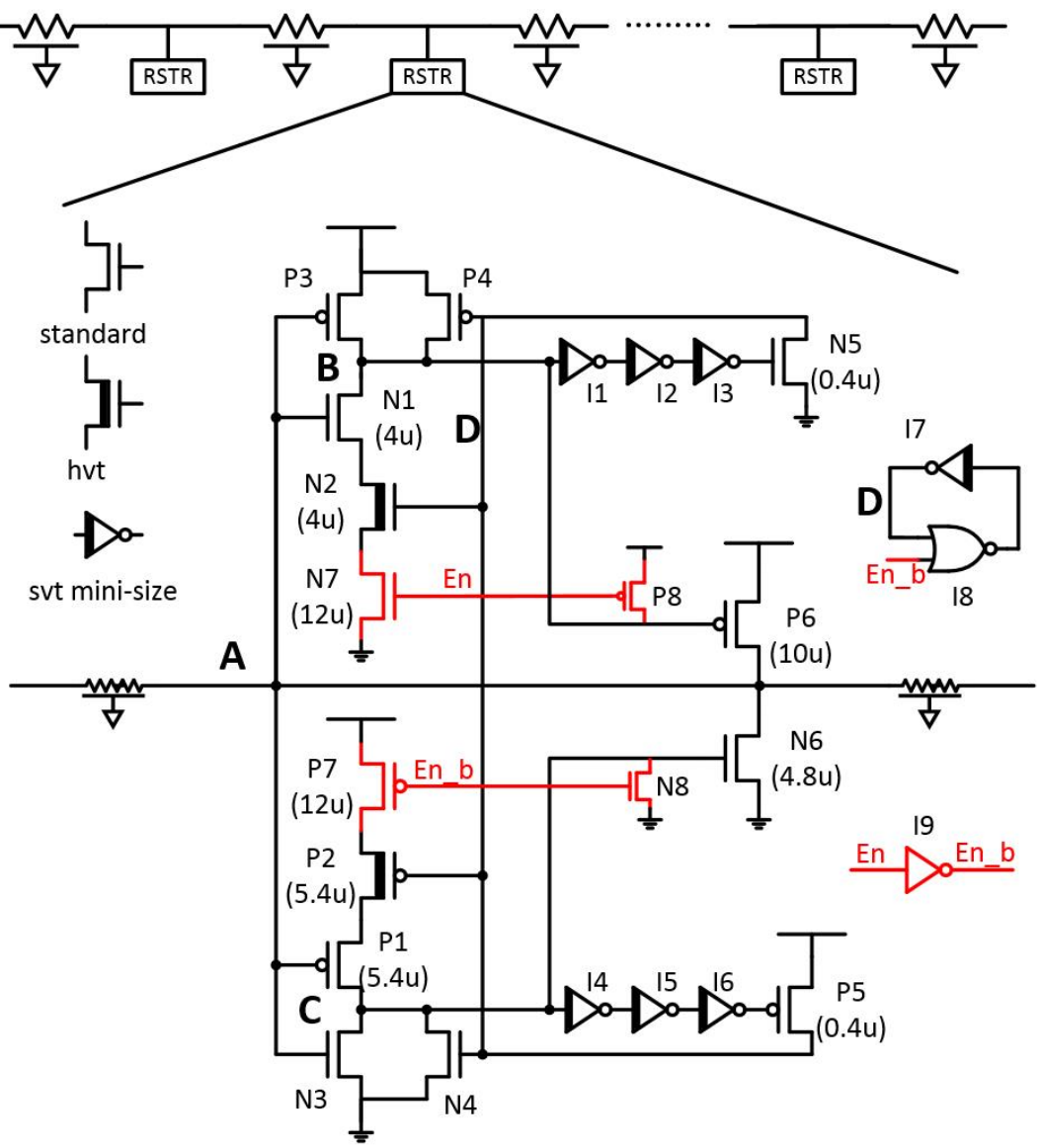



Figure 2.2: RSTR schematic with transistor sizing. Transistors with unlabeled sizes are minimum width (152nm). Enable signal and header/footer transistors provide reconfigurability.

The circuit operates by early detection of a transition along the interconnect wire at point A. The transition is then aided by turning on either the PMOS or NMOS driving transistor, P6 and N6, to supply additional current in driving the wire. To avoid global control signals a self-timed delay chain (I1-3 and I4-6) turns off the driving transistors and awaits the next transition. The regenerator is enabled through the En signal that, when asserted, activates N1-2 and P3-4 forming a NAND structure to sense the low-to-high transition and turn on driver P6, while remaining insensitive to high-to-low transitions. Similarly, high-to-low transitions are detected by a NOR (P1-2, N3-4) that controls N6. To allow for this hysteresis, I7 and I8 form a latch to store the previous value on the wire. Lastly, N7 and P7 in the NAND/NOR detection circuits disable the sensing of transitions while P8 and N8 disable the output drivers.

Because of the internal delay chain, RSTR controls its own pulse width, namely the duration of the pull-up/pull-down time, hence careful delay selection is needed to ensure that the wire transitions substantially before the RSTR resets itself across a range of Vdd. Also the delay should not be so long that it interferes with the next signal transition. The delay chain consists of three SVT minimum-sized stacked inverters; simulation across design corners and process variation ensures all these requirements are met.

Fig. 2.3 shows the energy-delay curve for repeaters and RSTR at 0.5V and 1V, simulated with the industrial 45nm SOI CMOS models. The driven interconnect is a 7.5mm intermediate (2× thickness) wire with 140nm spacing (1× min.) and 280nm width (2× min.), chosen to represent a reasonably long within-block interconnect. At both voltages, the size and number of repeaters are swept to find the optimal energy/delay points, marked as the Pareto frontier curve in Fig. 2.3. On the 1V frontier, we chose "INV #23" to represent the 1V-optimized design containing $N_{opt}$=23 inverter repeaters, each with size $w_{opt}$ = 12$\mu$mPMOS and 6$\mu$mNMOS. On the 0.5V frontier, design "INV #9" is selected with $N_{opt}$=9 inverters ($w_{opt}$ = 24$\mu$mPMOS,

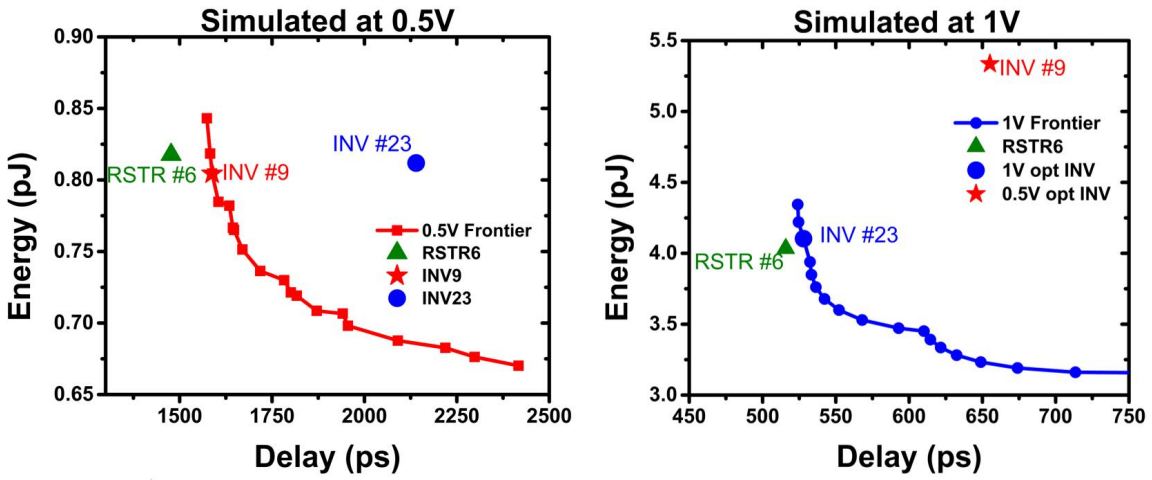



Figure 2.3: Simulated energy versus delay curves for RSTR. Optimal inverter and RSTR designs are chosen from the frontier curves at each voltage.

12$\mu$mNMOS). All repeaters are placed evenly along the interconnect.

The RSTR design space is similarly swept and we observed that some configurations on its 1V frontier also appeared on the 0.5V frontier. One such design "RSTR #6" uses $N_{opt}$=6 RSTRs which are evenly distributed along the 7.5mm wire with device sizes given in Fig. 2.2. This RSTR design is labeled on both plots of Fig. 2.3 for comparison. Unlike traditional repeated interconnects, RSTR can achieve better performance and energy characteristics over a wide voltage range, such as 0.5V to 1V as demonstrated in this simulation.

Despite the simplicity of the proposed RSTR scheme (the regenerator topology adds only small overhead beyond the design in [35]) it provides the following important benefits over traditional repeated interconnects:

1) RSTR remains optimal (in energy/delay space) across the full VDD range.

2) RSTR reconfigurability provides a new knob for adaptive designs to compensate for variability at NT operation. This is achieved by selectively turning on/off RSTRs along a wire to trade performance for power (e.g., 24% performance loss for 40% lower energy).

3) RSTR is faster than an optimal repeater design at both full and NT supply

while maintaining energy efficiency.

4) RSTR does not partition the wire, allowing for bi-directionality.

## 2.3 Measurements and Results

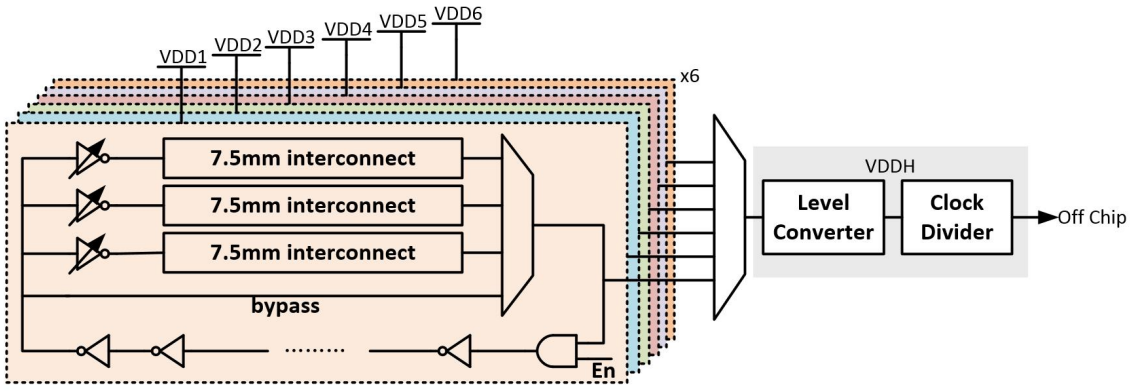



Figure 2.4: Reported delays are measured based on the frequency of a ring oscillator structure. Each interconnect design is in a separate voltage domain to measure energy. Each interconnect under test has adjacent neighbors with 140nm spacing (1× min.).

A test chip was fabricated in 45nm SOI CMOS to evaluate the efficacy of RTSRs in silicon and validate simulation predictions. A total of four inverter repeater (INV) designs and two proposed RSTR designs were included on the test chip, which measured 1×1mm (Fig. 2.4). Fig. 2.5 shows the test harness; the interconnect matches the structure simulated above and is implemented as a bypassable delay chain within a ring oscillator. After level conversion and a clock divider, frequency is measured off chip both with and without interconnect to assess delay.

Fig. 2.6 shows measured results confirming the relatively poor voltage scalability of repeater-based designs. A 1V optimal design is 31% slower than the 0.5V optimal design when operating at 0.5V. Conversely, a 0.5V optimal design is 18% slower with 29% higher energy than a 1V optimal design when both operate at 1V. In contrast the RSTR design shows good voltage scalability. Specifically at 1V it is 28% faster than the 1V optimal INV design while consuming 5% less energy. At 0.5V, the "RSTR

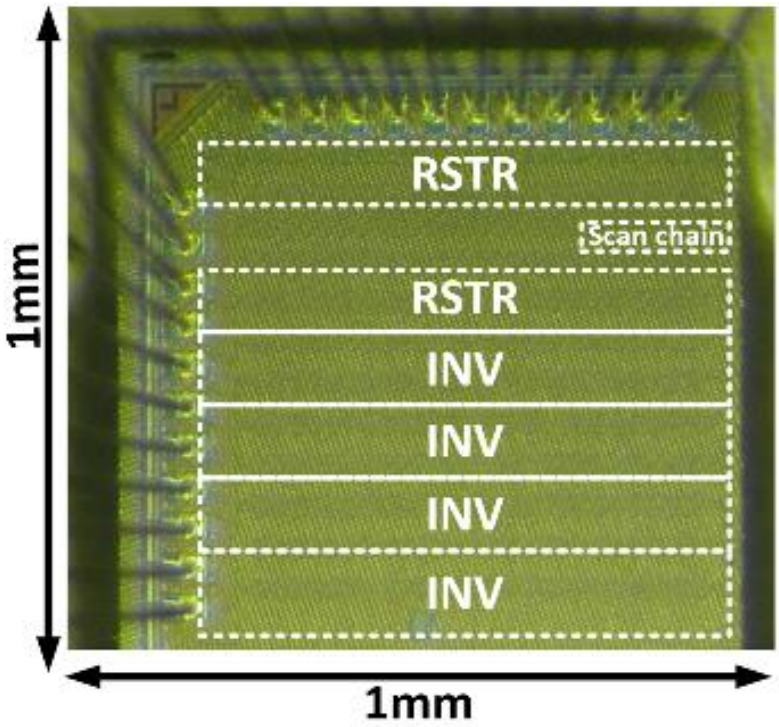



Figure 2.5: Die photo of 45nm SOI test chip. The 7.5mm interconnect is folded ten times.

#6" energy and delay essentially match the 0.5V optimal INV design. In addition to being superior to INV-based designs, recall that "RSTR #6" appears along the Pareto optimal frontier at both supply voltages. This indicates that excellent performance can be obtained across voltage scaling, relative to other RSTR designs.

Green triangles in Fig. 2.6 represent RSTR energy-delay points with varying number of RSTR enabled, representing dynamic reconfiguration options depending on real-time energy-performance priorities. This allows the RSTR design to also operate at lower energy with faster delay than "INV #23" at 0.5V, if desired. Also, if interconnect was performance limiting for the design at full VDD (1V), turning on six additional RSTR along the wire (reconfiguring RSTR #6 into RSTR #12) offers 10% faster performance, potentially rebalancing the overall design. In NT mode (0.5V), regenerators can then be turned off to achieve a minimal energy of 0.6pJ in this example.

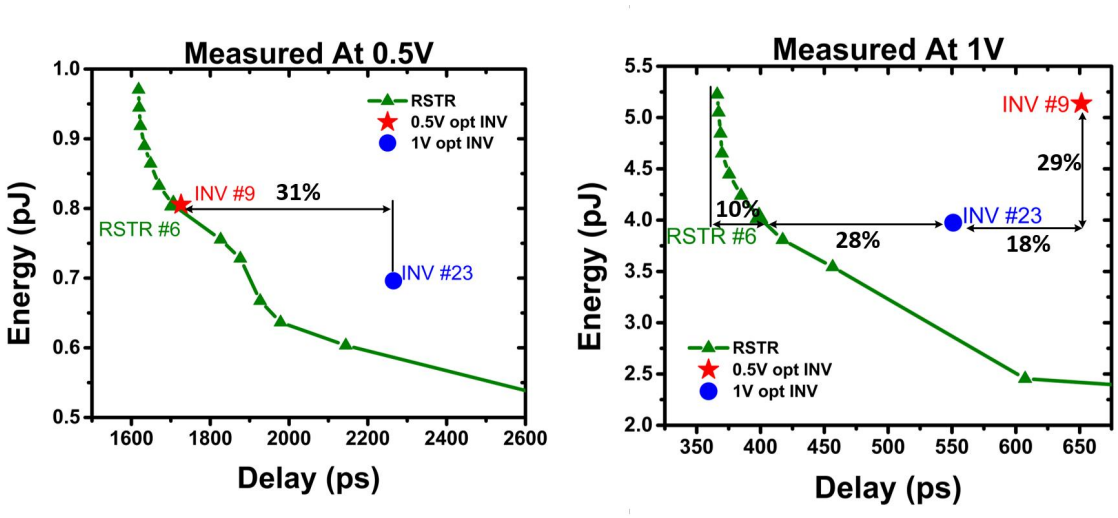



Figure 2.6: Measured energy versus delay curves showing RSTR and repeater performance. Green triangles represents different RSTR configurations (i.e., different number of RSTR enabled).

Fig. 2.7 shows measured delay scaling of repeater and RSTR designs across VDD, indicating the sub-optimality of using a single inverter-based repeater design in wide-range voltage scaling. RSTR is able to achieve better performance across the entire 0.5V to 1V range. Fig. 2.8 plots this measured data normalized to inverter FO4 delay across a range of voltages. Ideally an interconnect scales identical to circuit delay, which would be shown as a fixed line at 1.0 of FO4 in Fig. 2.8. Again, this supports the more graceful scaling of delay offered by an RSTR design over a conventional repeater-based approach.

## 2.4 Summary

Today's emerging mobile applications require high energy efficiency, which is often provided by scaling supply voltage across a wide range according to real-time workload variation. We present a reconfigurable, self-timed, regenerator-based interconnect scheme that remains optimal in terms of energy-delay efficiency at both full and near-threshold voltages. RSTR interconnect delay tracks FO4 logic delay more closely than repeated wires. In addition, RSTR offers higher speed and better energy efficiency

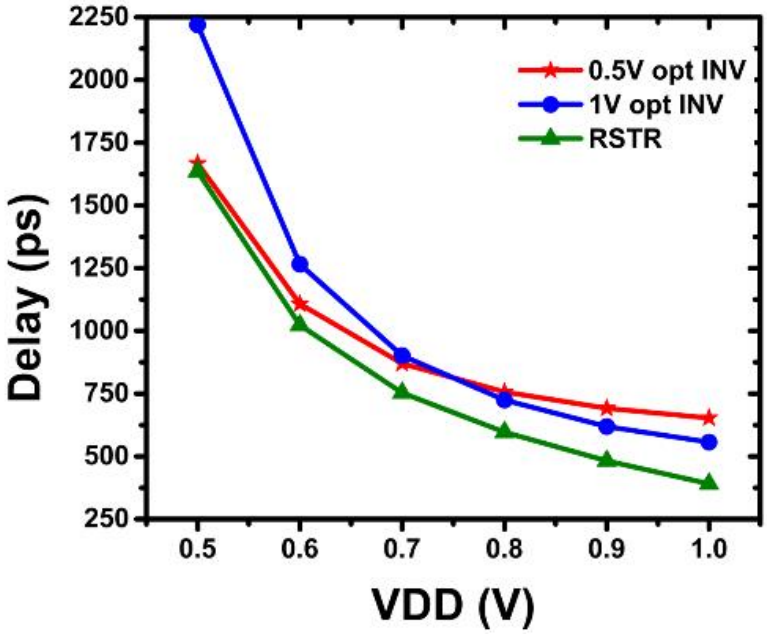



Figure 2.7: Simulated energy versus delay curves for RSTR. Optimal inverter and RSTR designs are chosen from the frontier curves at each voltage.

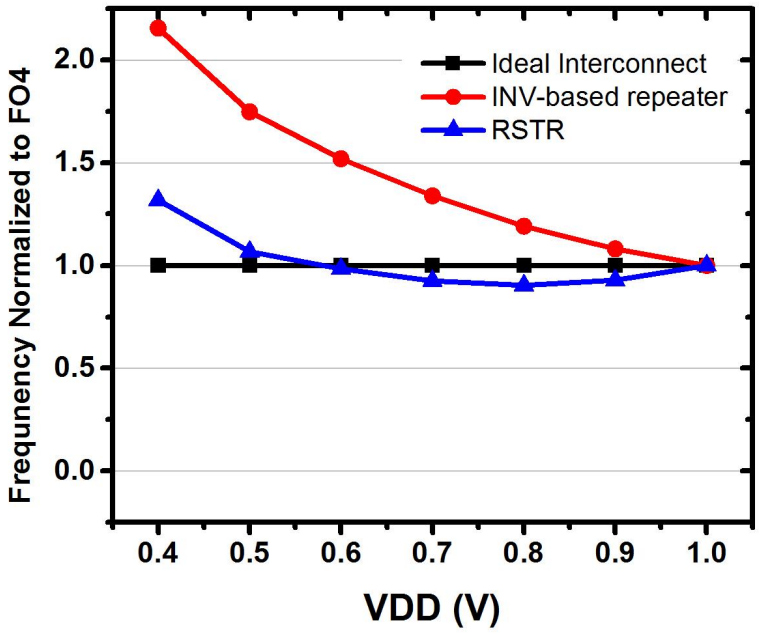



Figure 2.8: RSTR speed scales more similarly to digital logic than inverter-based repeated wires.

overall compared to traditional repeater approaches.

# CHAPTER III

# A 28-nm Compute SRAM with Bit-Serial Arithmetic Operations for Programmable In-Memory Vector Acceleration

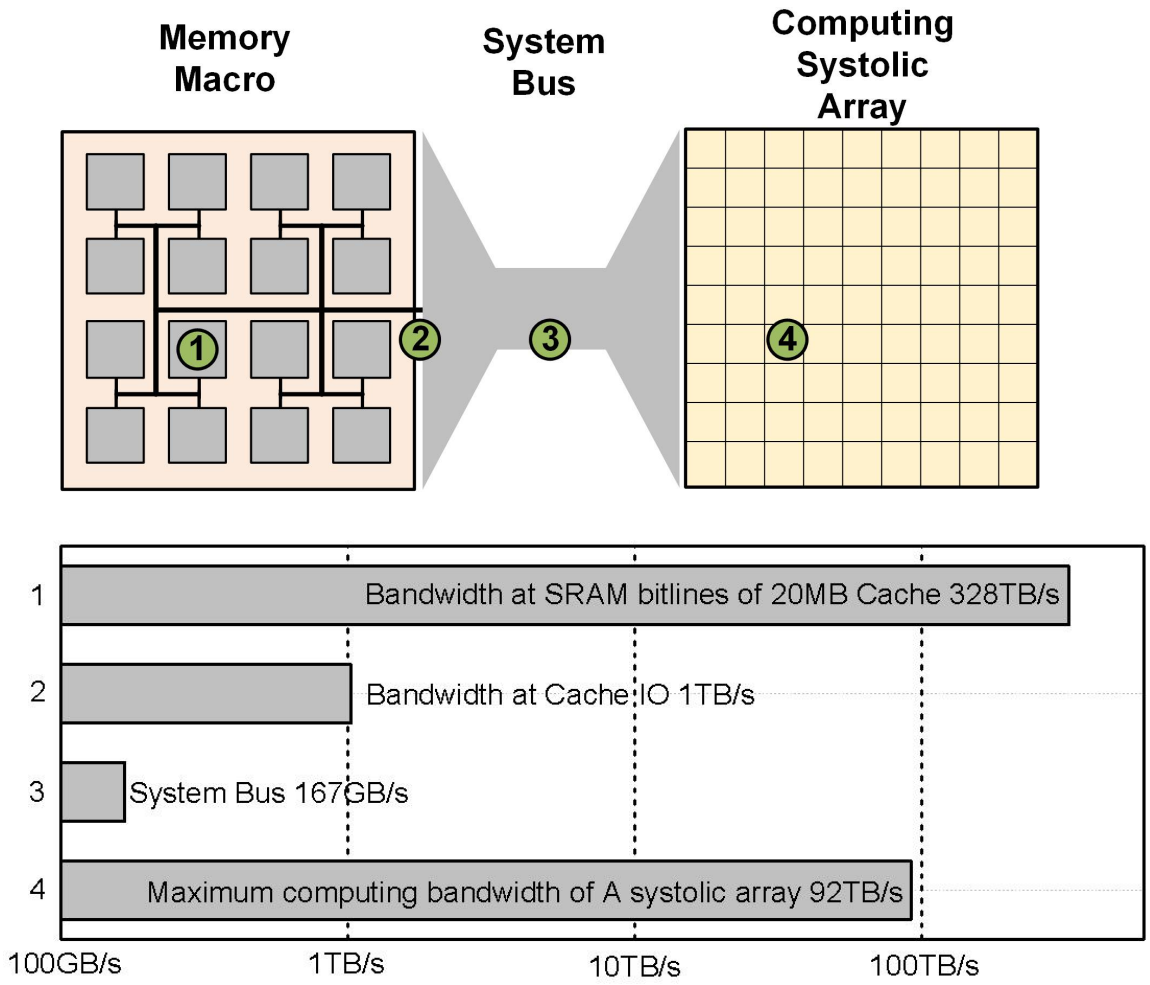
## 3.1 Introduction

In the conventional von Neumann architecture, a clear gap lies between data storage and processing: memories store data, while processors compute on data. Thanks to Moore's Law, in the past few decades, the computing power of integrated circuits has rapidly scaled as logic gates became faster and faster and the number of processing cores increased steadily until we hit the "Memory Wall" [37]. But the on-chip global interconnects latency and energy cannot keep up with the scaling of logic gates. Thus, the computation throughput and energy have become dominated by the memory bandwidth and data movement energy. As shown in Figure 3.1a, the bandwidth at the I/Os of all SRAM banks inside a big memory macro such as a 20 MB L3 cache is over a hundred TB per second [38, 39], which is comparable to the theoretical maximum computation bandwidth of the state-of-the-art systolic processing array [40]. Hence, the bottleneck is the local data network inside the memory macro and the global data bus on chip. Furthermore, a large fraction of energy consumption today is spent on moving data back and forth between memory

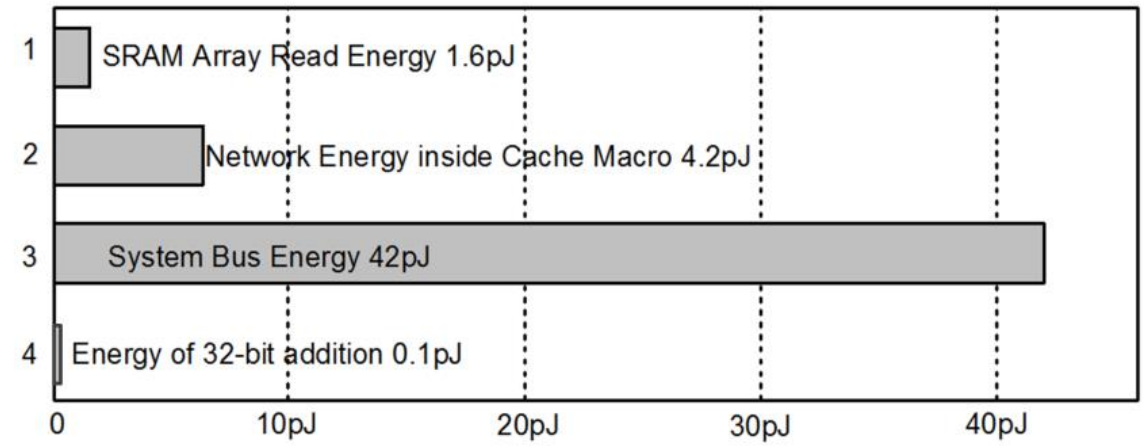
and compute units [14]. As shown in Figure 3.1b, it only takes sub-pico joules of energy to do a 32-bit addition while tens of pico joules are spent on retrieving data from far away memory banks.

Previously, people tried to overcome the "Memory Wall" by introducing more memory hierarchies, in an effort to bring the data closer to the computation. However, the memory problem is further exacerbated by the advent of data-intensive applications such as neural networks [41, 42], computer vision [43] and steam processing [44]. The need to shift from computation-centric to data-centric architecture has led to extensive research focused on the area of in-/near-memory computing, which moves computation to where the data is located. Recently, we have seen many studies that try to bring computation to different levels of memory hierarchies, including DRAM [45] and non-volatile memories like STT-MRAM [46], ReRAM [47], and Flash [48]. This paper focuses on designing computational SRAM banks. Most SRAM in today's chips is located in the caches of CPUs or GPUs. These large CPU and GPU SRAMs present an opportunity for extensive in-memory computing and have, to date, remained largely untapped.

There are two main types of emerging in-memory computing architectures for SRAM. The first is analog in-memory computing. In this case, one of the operands is pre-stored in the SRAM array. A multi-bit operand will have its bits spread into different word-lines, while the other operand is usually modulated into the analog voltage level in the word-lines [49] or pulse width of the word-line enable signal [50, 51]. The multiplication result of the two operands is then represented by the various discharge currents of the bit cell. Often multiple word-lines are activated simultaneously, and the multiplication results are accumulated on the bit-line as the total bit-line discharge current is the sum of the each individual bit-cell current. The final multiply-accumulate result is naturally represented by the analog bit-line voltage, which can be sensed by an analog-to-digital converter. This approach can achieve very high energy




(a) (a) On-chip bandwidth comparison.




(b) (b) On-chip energy comparison.

Figure 3.1: Bottlenecks in conventional von Neumann architecture: (a) low on-chip network bandwidth and (b) high data movement energy.

efficiency and performance, but it requires expensive analog-to-digital and digital-to-analog conversions at the array boundary. Also, the computation accuracy is highly susceptible to noise and PVT variations, and therefore its functionality is limited to low precision addition or multiplication. The second type is digital in-memory computing, which usually activates two word-lines with full-rail voltage in the same cycle and employs a sense amplifier on each bit-line to give a binary result [52, 53, 54, 55]. This type of approach offers better accuracy and robustness than analog approaches and can achieve a moderately high energy efficiency and performance. However, its functionality is limited to only bit-wise logic operation or low precision arithmetic in Binary Neural Networks.

Although traditional computing architectures such as CPU and GPU show limitations in energy efficiency and memory bandwidth, their appeal lies in their general functionality. They can perform a wide range of operations from bit-wise logic operation to Integer/Floating-Point Arithmetic. Not only are these computations accurate and robust since the designs are fully digital, but they are highly flexible and can implement many algorithms and neural network types and sizes. In this respect, both current in-memory approaches suffer from the same major limitation: they accelerate only one type of algorithm and are inherently restricted to a very specific application domain due to their limited bit-width precision and non-programmable architecture. On the other hand, software algorithms continue to evolve rapidly, especially in novel application domains such as neural networks, vision and graph processing, which makes rigid accelerators of limited use.

To address these limitations, we present a general purpose hybrid in-/near-memory Compute SRAM (CRAM) [56] that combines the efficiency of in-memory computation with the flexibility and programmability necessary for evolving software algorithms. It does part of the logic operations in SRAM bit-lines and most arithmetic operations in pitch-matched, near-memory peripherals at the end the each bit-line. It can

accommodate a wide range of bit-widths, from single to 32 or 64 bits, and operation types, including integer and floating point addition, multiplication and division, with a small amount of hardware overhead. Its high-throughput computation is accurate and robust, and the design offers good energy efficiency. CRAM tries to repurpose the large existing on-chip memory storage by augmenting a conventional SRAM bank in a cache with vector-based, bit-serial in-memory/near-memory arithmetic. To maintain compatibility with current CPU/GPU architecture, CRAM writes/reads operands conventionally with horizontal word-lines and vertical bit-lines, which is made possible by the 8T transposable bit cell.

The remainder of this paper is organized as follows. Section II generally introduces the bit-serial operation and the architecture of the proposed Computational SRAM. Section III describes the 8T transposable bit cell and the computing peripheral in detail. Section IV presents the algorithm of multi-bit arithmetic operations. Section V discusses the measurement results of the proposed design, and finally, the conclusions are presented in Section VI.

## 3.2 Overview of Bit-serial Arithmetic and CRAM Architecture

### 3.2.1 Bit-serial Arithmetic

Several previous digital in-memory computing works [53, 54, 55] supported some simple bit-parallel operations such as bit-wise logic and copy. However, these are carry-less operations that do not require interaction between bit-lines. In order to make in-memory computing as general purpose as the ALU in a CPU, support is needed for more complex arithmetic operations such as addition, multiplication, and even floating point operation. The critical challenge in supporting these complex computing primitives is facilitating carry propagation between bit-lines. We propose

bit-serial implementation with a transposable bit cell to address this challenge.

Since the 1980s, bit-serial computing architectures have been widely used for digital signal processing because it can usually provide the most area-efficient design in the presence of a massive bit-level parallelism [57, 58]. The key idea is to process one bit of multiple data elements every cycle. This model is particularly useful in scenarios where the same operation is applied to the same bit of all data elements in a vector, like in SIMD architectures. For example, in order to compute the element-wise sum of two arrays with 512 32-bit elements, a conventional processor would take at least 512 cycles to get the operands element-by-element from the SRAM and then perform the operation. A bit-serial processor, on the other hand, would complete the operation in 32 steps as it processes the arrays bit-slice by bit-slice instead of element-by-element. Note that a bit-slice is composed of bits from the same bit position but corresponding to different elements of the array. Since the number of elements in arrays is typically much greater than the bit-precision for each element stored in them, bit-serial computing architectures can provide much higher throughput than bit-parallel arithmetic. Note also that bit-serial operation allows for flexible operand bit-width, which can be especially advantageous in DNN hardware designs where the required bit width can vary from layer to layer [59, 60].

### 3.2.2 CRAM Architecture

Figure 3.2 shows the overall architecture of one 16-KB CRAM bank. Each CRAM bank consists of 4 128x256 arrays that load or store data conventionally using horizontal word-lines and vertical bit-lines. The normal SRAM peripherals, such as a row decoder, column mux, and sense amp, are shown in blue. In this diagram, the array has been preloaded with two vectors of data, vectors A and B. Data elements from the same vector are placed into different rows and aligned by the column, while the corresponding elements from the two vectors that are going to be operated must be

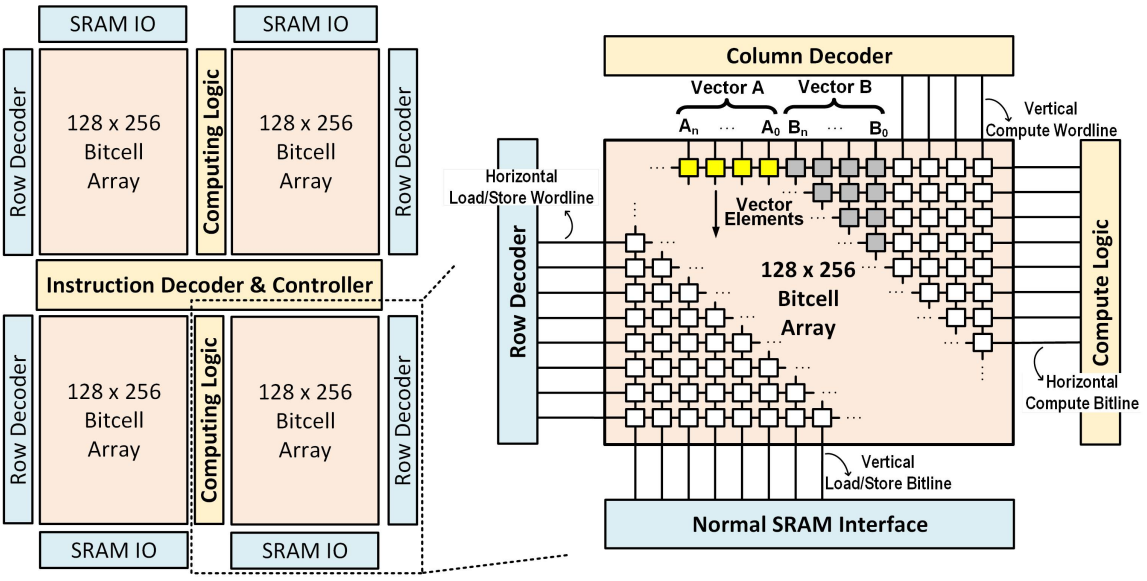



Figure 3.2: Proposed CRAM Architecture.

aligned on the same word-line. To perform bit-serial operation, we need to activate the same bit position from two vectors. Therefore, column decoder and pitch-matched compute logic are added so that in-memory computing can be performed using vertical compute word-lines and horizontal compute bit-lines. For example, in the first cycle, we simultaneously activate the vertical word-lines of the Least Significant Bits (LSB) from the two vectors. Then the computation is performed in both horizontal bit-lines and the compute logics at the end of the bit-lines. Near the end of the cycle, the result is then stored back in the array at some destination bit location selected by a third vertical word-line. In the next cycle, other bits of each operand are activated to continue the computation. Again, the result is stored back at the designated position at the end of the cycle. By repeating single bit operations cycle-by-cycle, we can perform any complex multi-bit arithmetic with carry-propagation. For example, a 32-bit adder will take 32 cycles to finish. Note that although bit-serial computation is expected to have high latency per operation, it gains significantly in terms of throughput. A 16-KB SRAM bank contains 256 vertical compute bit-lines in total, and a 35-MB Last Level Cache (LLC) in the Haswell server processor can accom-

modate 2240 such 16-KB banks [2], which means a total of 573,440 bit-lines can do computations in parallel. In this case, maximum throughput would be equivalent to 17,920 32-bit adders or 71,680 8-bit adders. The computing logic is shared between the arrays on the left and right and takes 4.5% of the CRAM bank area. The instruction decoder and controller in the middle of the bank, shared by all 4 arrays, take 32-bit instruction and generate control signals for the computing logic. They occupy 5.2% of the bank area. The details of the controller instructions will be presented in Section III.

## 3.3 CRAM Circuitry

### 3.3.1 8T transposable bit cell

Many previous in-memory computing works [51, 55, 61] choose to store each word unconventionally by spreading bits into different rows of the same vertical bit-line. This approach makes the computation much easier and can directly use 6T bit cell for minimizing area. But the normal SRAM read/write operation gets much more complicated and becomes incompatible with current computer architecture since in one cycle, we can't read out a complete word but only the same bit position from multiple different words. Therefore, we propose to use an 8T transposable bit cell. Figure 3.3 shows the schematic and the layout of the bit cell [62]. Four of the transistors form the cross-coupled inverter pair to hold the data, and there are two pairs of access transistors for read/write. The structure is similar to the conventional 8T dual port SRAM bit cell except that it provides bidirectional access: the bit cell can be read or written from either vertical bit-line or horizontal bit-line. Therefore, CRAM can operate directly on the stored operands in memory by enabling the same bit position from two vector elements with vertical word-lines and perform the computation on horizontal bit-lines. Furthermore, it can also directly read a complete word by en-

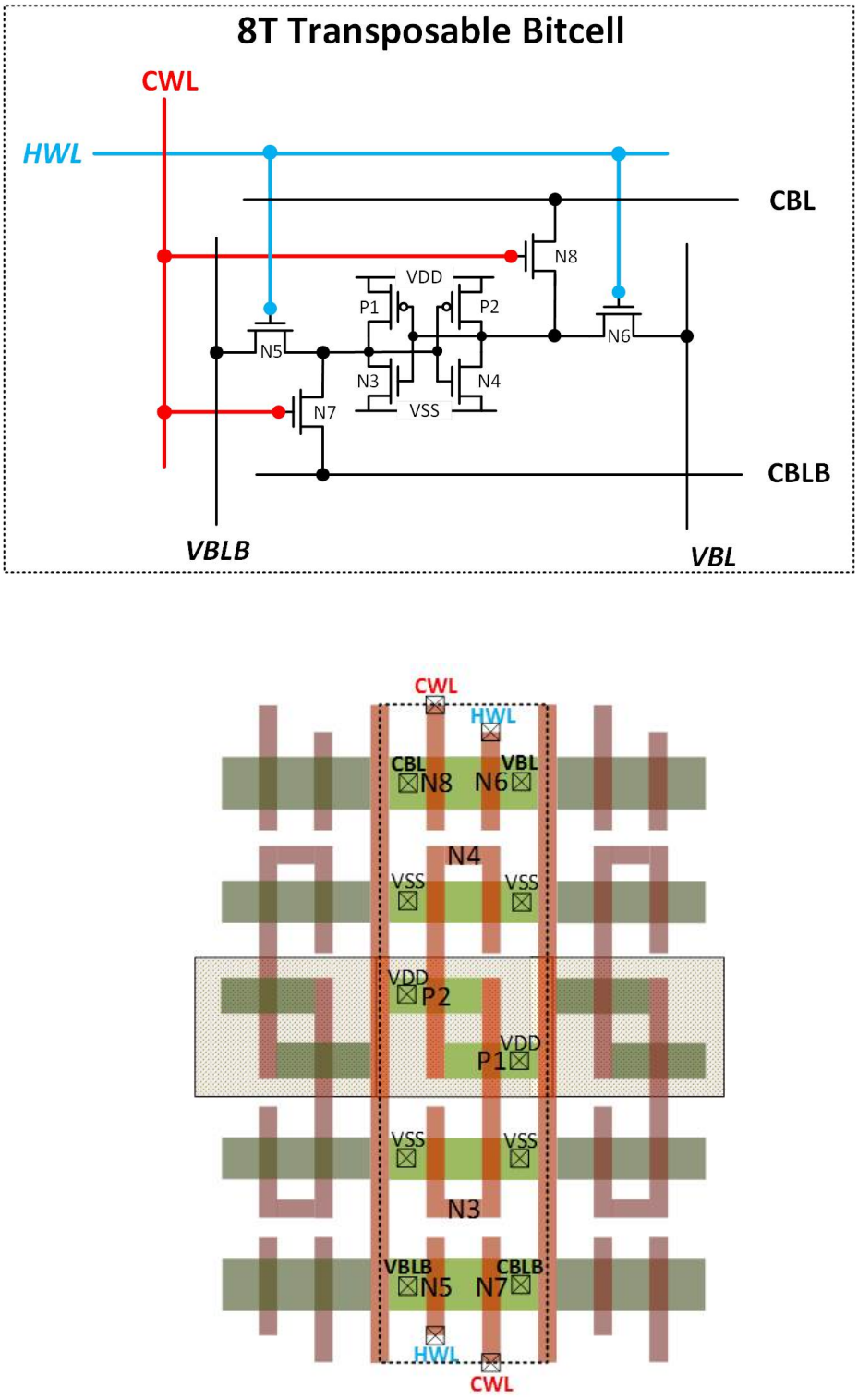



Figure 3.3: Schematic and layout of 8T transposable bit cell.

abling the horizontal word-line and sense the result from vertical bit-lines. With the logic rule transistor in 28-nm CMOS, the bit cell size is 0.405 um by 1.93 um, which is $638F^2$ when normalized to technology node feature size (F).

### 3.3.2 Computing Peripherals

Figure 3.4 gives a detailed view of one row in the bit cell array. Logic operations are performed on the bit-line (in-memory), while small additional in-row logic (near-memory) enables carry-propagation between successive bit-serial calculations. An example of 1-bit addition will be used to illustrate the CRAM single cycle operation and computing peripherals. Here we add the second bit of vector A ($A_1$) and vector B ($B_1$) with carry-in ($C_{in}$) from the previous cycle and store the sum back to the second bit of vector D ($D_1$) and latch the carry-out ($C_{out}$) for the next cycle. First, the CRAM instruction decoder receives the ADD instruction with the 3 column addresses for bits $A_1$, $B_1$ and $D_1$. After pre-charging the compute bit-line (CBL) and compute bit-line bar (CBLB), we activate the vertical compute word-lines (CWL) of $A_1$ and $B_1$ simultaneously to generate 'A AND B' on CBL and '$\overline{A}$ AND $\overline{B}$' on CBLB. We use a separate voltage rail for the driver of $CWL_{A/B}$, so that we can lower the word-line voltage to prevent the read disturbance issue when necessary. This is the in-memory part of the computation. Next, after the dual sense amps are enabled, the in-memory logic operation results propagate into the near-memory region located at the end of each CBL. The NOR gate generates 'A XOR B,' which combined with $C_{in}$ from the C latch produces Sum and $C_{out}$. Then $CWLD_D$ is activated, and the sum is written back to destination bit $D_1$. Finally, near the end of the cycle, Cout updates the C latch, which provides $C_{in}$ for the next cycle.

When we activate the CWL, all 256 CBLs in the 16-KB CRAM banks are performing the same single bit instruction in a SIMD fashion. In order to support complex multibit arithmetic, CRAM has to be able to execute instructions only on certain

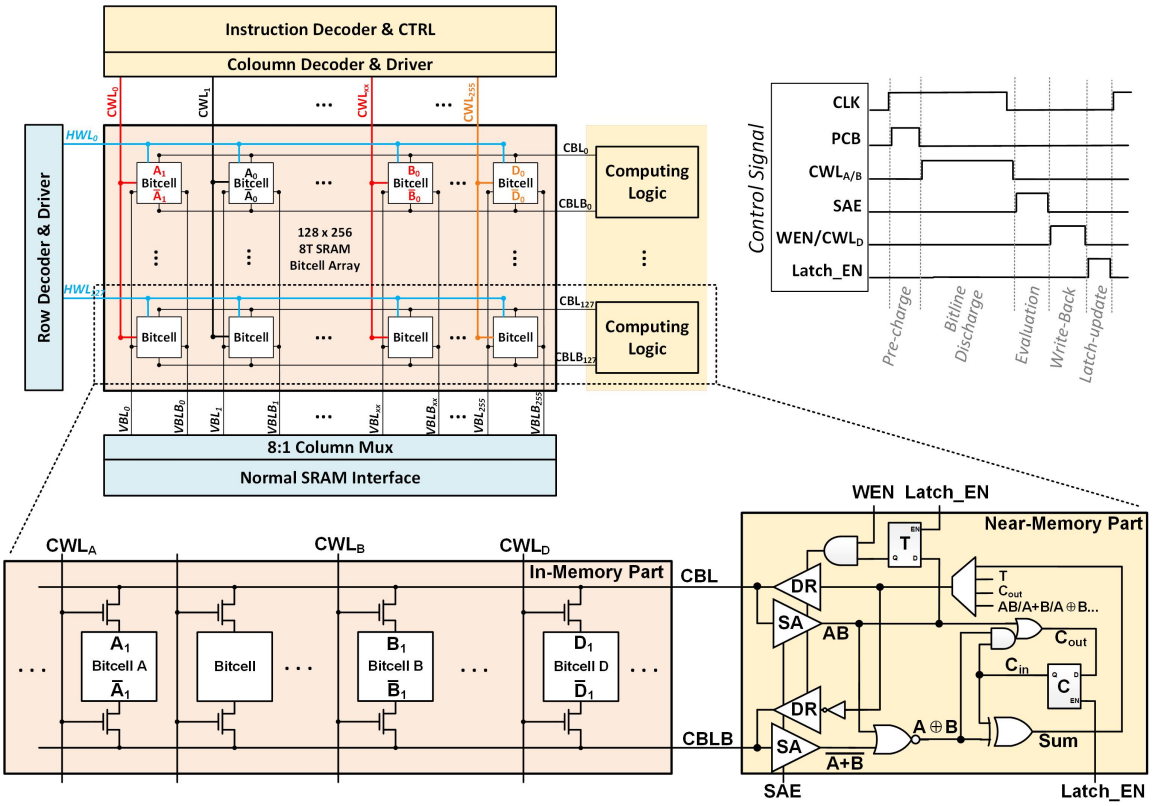
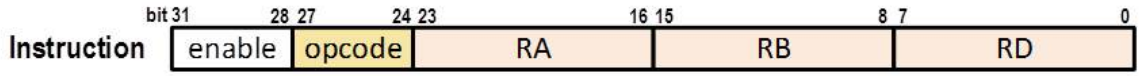



Figure 3.4: CRAM Array Architecture (Top-left), computation control signal timing diagram (Top-right), and in/near-memory computing peripherals (Bottom).

selected CBLs; and therefore, we add the Tag (T) latch to enable conditional operation. Tag latch is used as the enable signal of the write-back driver. Therefore, for the CBL whose Tag latch stores 0, the computation result will not be written-back to the memory, as if the instruction is not executed at all. The content of the Tag latch can be loaded from or written into the memory array. In addition to the logics introduced before, we also add a multiplexer to allow for the write-back of signals besides the Sum, such as A AND B, A OR B, $C_{out}$, or Tag.

With the computing peripherals shown in Figure 3.4, the CRAM controller can support up to 16 single-cycle instructions, shown in Table 3.1. Besides the logic and add operation, it includes copy, inversion, load/store of carry or tag, comparison, and set/reset carry. The CRAM controller takes 32-bit instruction. Four bits ([31:28]) are used for various enable signals for different features. Four bits ([27:24]) are used

for the opcode for the 16 instructions. Eight bits are used for the address since every memory array contains 256 compute word-lines. Bits [23:16], [15:8], and [7:0] represent the address of operand A, the address of operand B and the destination location D, respectively. Using these single-cycle micro instructions, we can build complex multi-cycle macro instructions, including search, multiplication, division, and floating point arithmetic.




| Single-Cycle Primitives | | | | | |
|---|---|---|---|---|---|
| **Type** | **Opcode** | **RA** | **RB** | **RD** | **Comments** |
| **Logic** | AND/OR/XOR/ NAND/NOR/XNOR | ✓ | ✓ | ✓ | Perform logic operation on RA and RB, and store the result back to RD |
| **Arithmetic** | ADD | ✓ | ✓ | ✓ | Add RA and RB, write back to RD |
| **Shift** | Copy | ✓ | | ✓ | Copy RA to RD |
| | INV | ✓ | | ✓ | INV RA and write back to RD |
| **Comparison** | Equal | ✓ | | | Write "RA == AddrRB[0]" to Tag latch |
| **Utility** | LOAD T | ✓ | | | Load RA to Tag latch |
| | STORE C/T | | | ✓ | Write Carry/Tag latch back to RD |
| | Set C | | | | Set Carry Latch to 1 |
| | Reset C | | | | Reset Carry Latch to 0 |
| | C to T | | | | Write Carry Latch to Tag Latch |

Table 3.1: CRAM Instruction Set.

## 3.4 Multi-cycle Arithmetic

Users can program CRAM to achieve many complex computations. Table 3.2 shows a sample list of the supported multi-cycle operations and the number of single-cycle instructions each takes. Next, we will introduce some commonly used arithmetic operations and the way to program them in CRAM.

| Sample Multi-Cycle Operations | | |
|---|---|---|
| **Type** | Operation | **# Cycles** |
| **Logic** | AND | N |
| | NOR | N |
| | XOR | N |
| | NAND | N |
| | OR | N |
| | XNOR | N |
| **Integer** | Add | N+1 |
| | Sub | 2N+1 |
| | Mult | $N^2 + 5N - 2$ |
| | UDiv | $1.5N^2 + 5.5N$ |
| **32-bit Float Point** | Add/Sub | 4978 |
| | Mult | 679 |
| | Div | 697 |
| **Comparison** | Equal | 2N+1 |
| | Greater/Less | 2N+1 |
| | Search | N |
| N is the bit-width of data | | |

Table 3.2: Sample of supported operations and cycle counts.

### 3.4.1 Integer Addition and Subtraction

We use the addition of two vectors of 3-bit numbers (A and B) to explain how the addition algorithm is carried out bit-by-bit starting from the least significant bit (LSB) (Figure 3.5). The two vectors each occupying 3 columns need to be placed in the same array with their corresponding elements aligned on the same row but not necessarily abutted. In cycle 0, we first initialize the entire carry latch to 0 by using instruction 'Reset C.' In cycle 1, we apply instruction 'ADD' and provide the column address of the LSBs for RA and RB. We can either write the sum to an empty column of the array or one of the operand LSBs can be directly overwritten by the result depending on the destination address, RD, we give in the instruction. Carry

latch is automatically updated with Cout at the end of the cycle. In cycles 2 and 3, we add the second and third bit location the same way as we did in cycle 1. Thus, an N-bit addition takes N+1 cycles. Subtraction can be performed by first inverting vector B and then adding to A with carry latch initialized to 1.

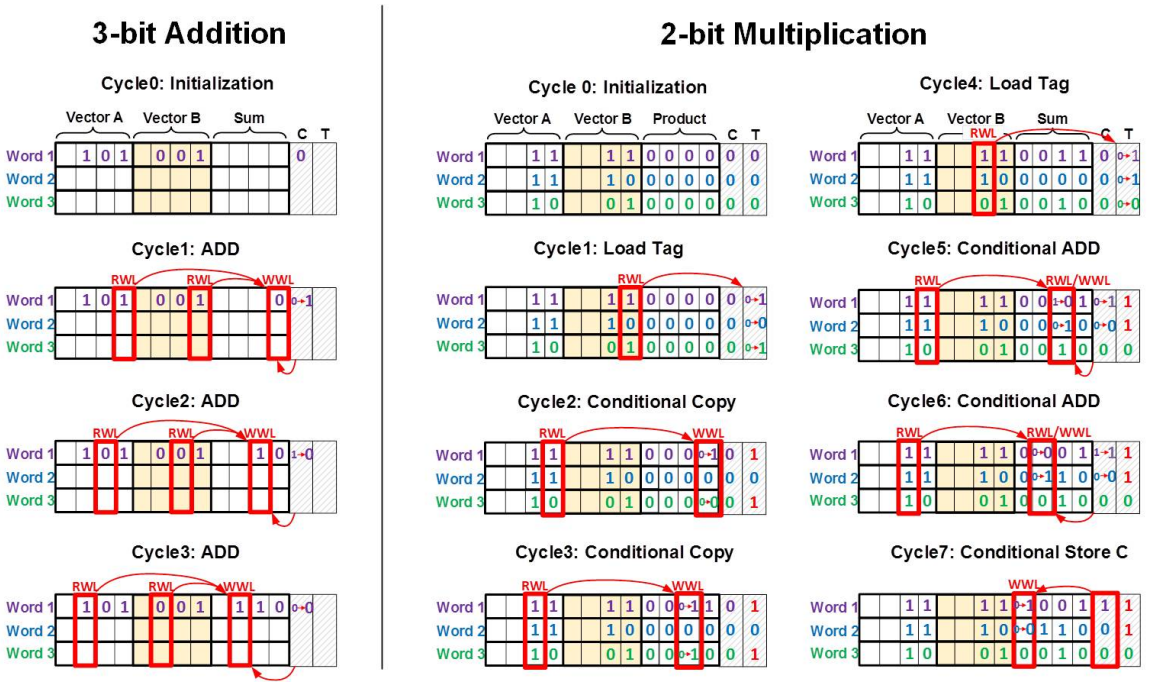



Figure 3.5: 3-bit Addition Cycle-by-Cycle Demonstration (left), 2-bit Multiplication Cycle-by-Cycle Demonstration (right).

### 3.4.2 Unsigned Integer Multiplication

One way to perform multiplication is using shift and add. It requires the conditional copy and addition instruction enabled by the tag latch. As explained in Section 3.3.2, if we enable the conditional execution feature, the tag latch becomes the local write bit-line enable signal of the row, and the result of any instruction will only be written back into the destination bit RD if the tag latch stores 1. Figure 3.5 demonstrates the example of a 2-bit multiplication. Suppose that vector A is the multiplicands and vector B is the multipliers. Initially, four columns in the array are reserved for the product and initialized to zero by setting all carry latches to 0 first

using 'Reset C' and then writing the carry latch back to product columns in 4 cycles using 'Store C.' In the first computing cycle, the LSB of the multiplier is loaded to the tag latch using 'Load T' instruction. In cycles 2 and 3, the multiplicands are copied to product columns only if the tag latch in that row equals 1. In cycle 4, the second bit of the multiplier is loaded to the tag latch. In the next 2 cycles, for rows with tag equals 1, the multiplicands are added to the second and third bits of the product, shifting the multiplicands by 1 to account for the multiplier bit position. Finally, we store Cout in the most significant bit (MSB) of the product to complete the multiplication. Note that partial products are implicitly shifted as they are added using appropriate bit addressing in the bit-serial operation, and no explicit shift is performed.

### 3.4.3 Unsigned Integer Division

Division is conducted similarly by implicit shifting and subtracting from a partial result. The pseudo-code for CRAM is shown in Table 3.3. The quotient is computed starting from the MSB. First, we copy the MSB of dividend to the partial result (remainder). Then, we subtract the divisor from the partial result, put the result into a temporary location and check whether the result is positive or negative by looking at the overflow bit Cout in the carry latch. A positive result from subtraction means the partial result is greater than the divisor, and the tag latch of that row will be set to 1. We conditionally update the corresponding bit in the quotient and remainder if the tag is 1. We repeat the previous steps N times until all the bits of the quotient are computed.

### 3.4.4 Comparison and Search

Comparison operations like "greater/less than" or "equal to" can be performed by using subtraction or XOR logic operation. CRAM also provides a multi-bit search

**Input: Divisor A[N-1:0], Dividend B[N-1:0]**
**Output: Quotient Q[N-1:0], Remainder R[N-1:0]**

[Note: extra N columns (TEMP) is used for temporary result]

0: initialize Q and R to 0
1: **for** i = 0 to N-1 **do**
2: copy B[N-1-i] → R[N-1-i]
3: R[N-1:N-1-i] - A[N-1:0] → {$C_{out}$, TEMP[N-1:0]}
4: if {$C_{out}$, TEMP[N-1:0]} is positive, update Tag to 1
5: (if Tag = 1) write 1 into bit Q[N-1-i]
6: (if Tag = 1) copy TEMP[i:0] to R[N-1:N-1-i]
7: **end for**
8: **return** Q, R

Table 3.3: Pseudo-code: Unsigned Integer Division.

operation like those in content addressable memory (CAM) by repeatedly using the CRAM single-cycle instruction 'Equal.' A given pattern is compared with the memory content within a specified range of columns, and the matched memory row will have its Tag latch stored as 1. The pattern is given cycle by cycle into the memory as the 8th bit of CRAM instruction (the LSB of address RB filed) and is compared to all the bits in the column specified by address RA of the instruction. Therefore, N-bit search operation takes N cycles.

### 3.4.5 Floating-point Arithmetic

Taking 32-bit IEEE-754 floating point as an example, we will demonstrate one way to implement floating point arithmetic on the CRAM using repeated conditional integer addition, subtraction, multiplication, division and search operation. A 32-bit floating number is represented by one sign bit in the MSB followed by 8-bit exponent and 23-bit mantissa. During computation, we always use one extra memory column of all 1s to represent the implicit 24th bit of mantissa. Floating point multiplication and division is relatively simple. First, the result sign bit can be determined by XOR the operand sign bits. Then an eight-bit addition between the two exponents is

**Input: A[31:0], B[31:0]**
**Output: S[31:0]**

[Note: extra 32 columns (TEMP) is used for temporary result]
[Note: $S_{A/B/S}$ : sign bit, $E_{A/B/S}$: exponent, $M_{A/B/S}$: Mantissa]

I. Equalize exponent: (first consider the case $E_A \geqslant E_B$)
0: $E_A - E_B \rightarrow E_{TEMP}$
1: compare & if($E_A \geqslant E_B$) copy $E_A \rightarrow E_S$
2: **for** i = 1 to 24 **do**
3: Search for row with $E_{TEMP}$ = i, right shift $M_B$ by i $\rightarrow M_{TEMP}$
4: **end for**
5: compare & if($E_{TEMP} \geqslant 24$), clear $M_{TEMP}$ to all 0
II: Add Mantissa
6: XOR $S_A$, $S_B \rightarrow$ Tag
7: if(Tag = 0) $M_A + M_{TEMP} \rightarrow M_S$, $S_A \rightarrow S_S$
8: if(Tag = 1 & $M_A > M_{TEMP}$) $M_A - M_{TEMP} \rightarrow M_S$, $S_A \rightarrow S_S$
9: if(Tag = 1 & $M_A < M_{TEMP}$) $M_{TEMP} - M_A \rightarrow M_S$, $S_B \rightarrow S_S$
III: Normalize result
10: **for** i = 1 to 24 **do**
11: Search $M_S$ with #i leading 0, left shift $M_S$ by i, $E_S - i \rightarrow E_S$
12: **end for**
(Repeat previous steps again for $E_B \geqslant E_A$ case)
13: $E_B - E_A \rightarrow E_{TEMP}$
14: …

Table 3.4: Pseudo-code: Floating Point Addition.

performed if it is multiplication or eight-bit subtraction if it is division. Then a 24-bit multiplication or division between the mantissa is performed. However, floating point addition and subtraction is much more complicated. Table 3.4 shows the pseudo-code for floating point addition. First, we equalize the exponents of the operands by shifting the one of the mantissa. If the operand A has a larger exponent, we right-shift the mantissa of operand B by the difference of the two exponents. Since the mantissa has at most 24 bits, we shift at most 24 times. Next, we add the mantissa if the signs of A and B are the same. Otherwise, we subtract B from A if A has a larger mantissa or subtract A from B if mantissa B is larger. Finally, we need to normalize the result by left-shifting the result until the 24th bit of mantissa is 1.

## 3.5 Measurements and Results

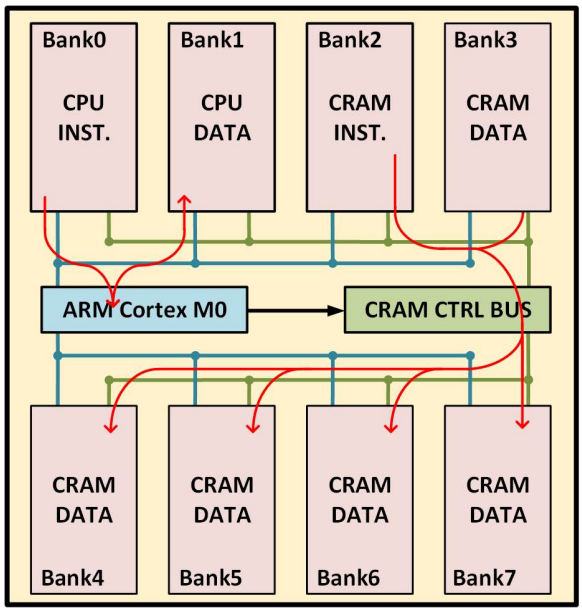



Figure 3.6: Test chip architecture with sample memory bank configuration.

To test the proposed in-/near-memory concept, we incorporate CRAM into an IoT processor. The chip consists of a Cortex-M0 CPU [63], a separate CRAM control bus, and eight 16-KB Compute SRAM banks (in total 128 kB memory with 2048 computing rows). These memories can function either as traditional or compute memories. Both the ARM core and CRAM control bus can access all eight memory banks, load or store data using standard memory IO, and perform computation in memory by sending 32-bit CRAM instruction to the CRAM controller IO in each bank. The diagram in Figure 3.6 shows an example memory bank configuration: two memory banks are used as CPU instruction and data memory while the rest are used for CRAM computation. Complex multi-cycle instructions are stored in one of the 6 banks and streamed or broadcasted to other 5 compute-configured banks by the CRAM control bus with all five banks performing CRAM operations in parallel. At the same time, the M0 can perform other processing with the remaining two memory banks through the standard AHB bus.

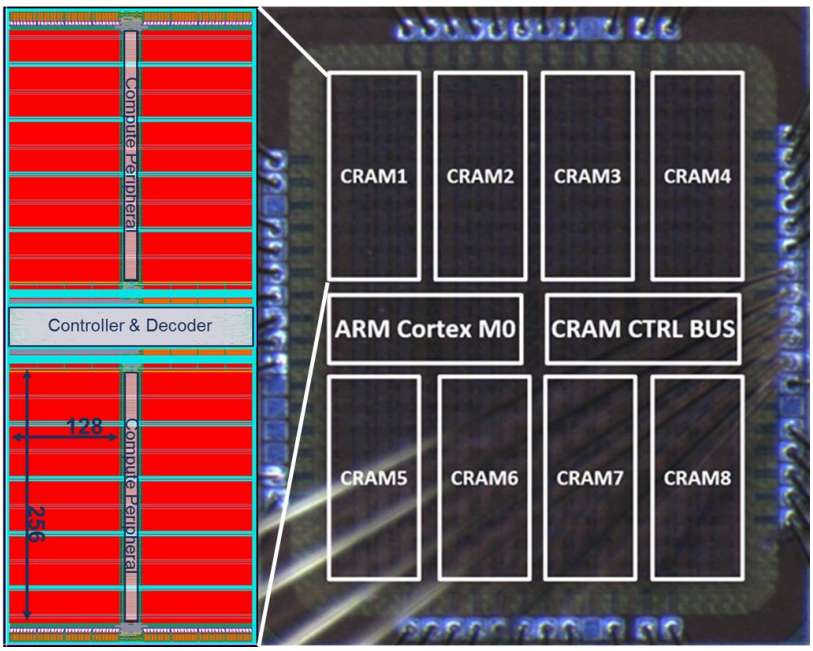



Figure 3.7: Layout of CRAM bank and die photo.

Figure 3.7 shows the layout of the CRAM bank and die photo of the prototype chip fabricated in 28-nm CMOS. A single memory bank is 245 x 625 um with 70% array efficiency. The chip size is 1.5 mm by 1.7 mm. Figure 3.8 shows the measured frequency and energy efficiency of 8-bit addition and multiplication across the supply voltage. The best energy efficiency is achieved at 0.6 V and 114 MHz, resulting in 0.56 TOPS/W for 8-bit multiplication and 5.27 TOPS/W for 8-bit addition. At 1.1 V, the frequency of 475 MHz results in 122 GOPS for 8-bit addition and 9.4 GOPS for 8-bit multiplication. If the memory size is scaled to 35MB, which is a similar capacity to an L3 cache in a modern server-class processor, CRAM is estimated to provide 34.2 TOPS of 8-bit additions while consuming 51.2 W. Figure 3.9 gives measured frequency and leakage power distributions for 21 measured dies. The performance of different multi-cycle operations is summarized in Table 3.5.

Figure 3.10 shows the performance of the test chip for diverse computationally intensive tasks ranging from neural networks to graph and signal processing. The total latency in cycles is compared with a baseline operation where CRAMs are only used

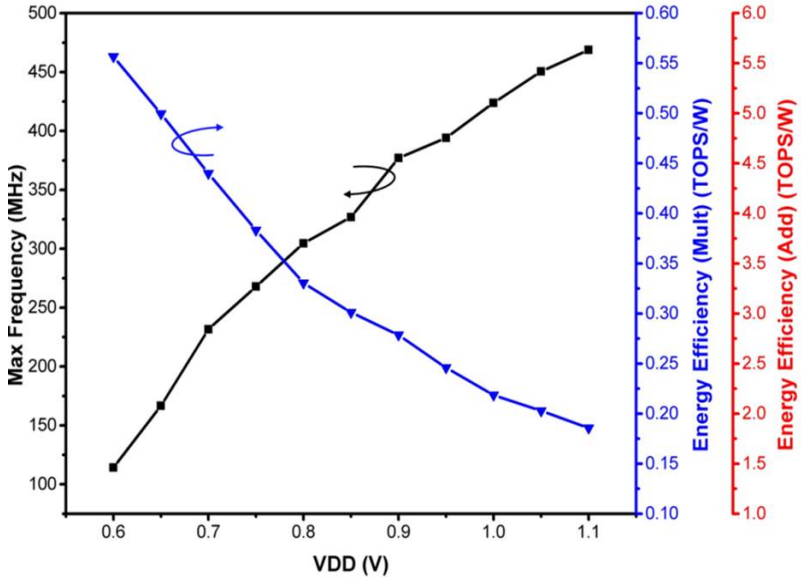



Figure 3.8: Frequency and energy efficiency of 8-bit multiplication and addition at different VDD.

as data memories and the computation is entirely performed on the ARM CPU. The first benchmark is the 1st convolutional layer from Cuda-convnet [64], and the second is the last fully connected layer from Alex-net [15]. Due to their size, these layers must be executed in multiple smaller sub-sections. The third application consists of 512 simultaneous 32-tap FIR filters and the fourth application performs traversal of a directed graph represented by a 192×192 adjacency matrix. The workload breakdown shows the percentage of time spent on input loading and output storing vs. in-memory computation. Speedup, compared to executing the same workload with the ARM Cortex-M0, varies from 7.2 to 114× with the greatest gains obtained when the operation is compute-heavy and low on input/output movement.

In Table 3.6, we compare the proposed approach with other state-of-the-art in-memory accelerators. We have by far the largest computing memory size. Furthermore our proposed work is the only solution to provide a wide range of instructions and flexible bit-widths. It repurposes the memory storage already available in processors, thereby accelerating computation while maintaining programmability.

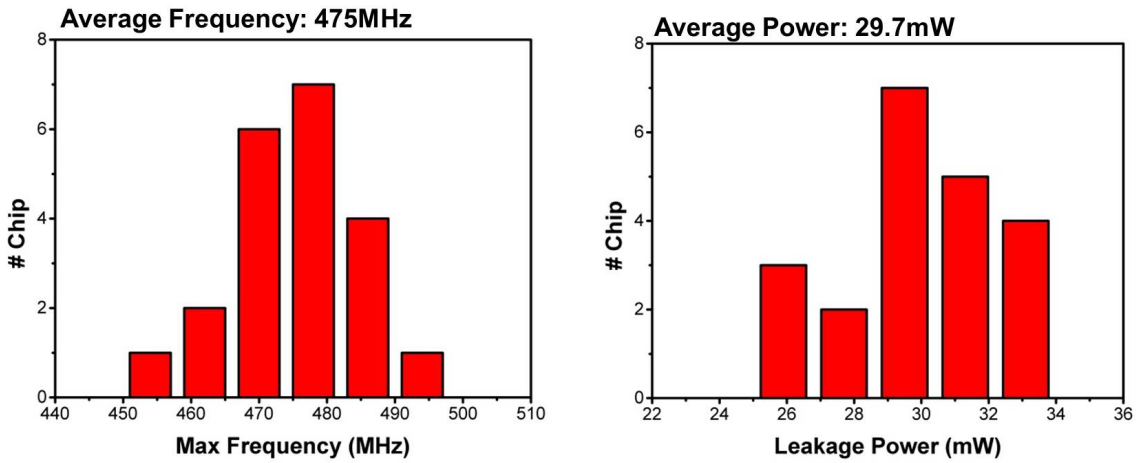



Figure 3.9: Maximum frequency and leakage power distribution of 21 dies at 1.1V.

## 3.6 Summary

To summarize, we proposes a general purpose hybrid in-/near-memory compute SRAM (CRAM) that combines an 8T transposable bit cell with vector-based, bit-serial in-memory arithmetic to accommodate a wide range of bit-widths, from single to 32 or 64 bits, as well as a complete set of operation types, including integer and floating point addition, multiplication and division. This approach provides the flexibility and programmability necessary for evolving software algorithms ranging from neural networks to graph and signal processing. CRAM is an area-efficient and low invasive technique that exploits vector-based bit-serial in-/near-memory arithmetic. It achieves both high throughputs by exploiting the massive bandwidth inside SRAM banks and good energy efficiency by suppressing data movement energy. The proposed design was implemented in a small IoT processor in 28-nm CMOS consisting of a Cortex-M0 CPU and 8 CRAM banks of 16 kB each (128 kB total). The system achieves 475 MHz operation at 1.1 V and, with all CRAMs active, produces 30 GOPS or 1.4 GFLOPS on 32-bit operands. It achieves the energy efficiency of 0.56 TOPS/W for 8-bit multiplication and 5.27 TOPS/W for 8-bit addition at 0.6 V and 114 MHz.

| Type | Operation | 32-bit Performance[1] | 8-bit Performance[1] |
|---|---|---|---|
| **Logic** | AND | 30.4GOPS | 122 GOPS |
| | NOR | | |
| | XOR | | |
| | NAND | | |
| | OR | | |
| | XNOR | | |
| **Integer** | Add | 30.4 GOPS | 122 GOPS |
| | Sub | 15.2 GOPS | 60.8 GOPS |
| | Mult | 0.83 GOPS | 9.40 GOPS |
| | UDiv | 0.57 GOPS | 6.97 GOPS |
| **32-bit Float Point** | Add/Sub | 0.20 GFLOPS | / |
| | Mult | 1.43 GFLOPS | / |
| | Div | 1.40 GFLOPS | / |
| **Comparison** | Equal | 14.9 GOPS | 57 GOPS |
| | Greater/Less | | |
| | Search | 30.4 GOPS | 122 GOPS |
| 1. Measured performance on 128KB CRAM test chip at 475MHz | | | |

Table 3.5: Performance of Test Chip at 475MHz.

| | CONV | | FC | | FIR | | GRAPH | |
|---|---|---|---|---|---|---|---|---|
| **Testbench** | Cuda-Convnet 1st layer | | AlexNet last layer | | 512 32-Tap Filter | | Nearest Neighbor Traversal | |
| **Input Size** | 24x24x3 | | 24x1 | | 32x10 | | 192x192 | |
| **Parameter Size** | 5x5x3x64 | | 1000x24 | | 512x32 | | 0 | |
| **Output Size** | 1x1x64 | | 1000x1 | | 512x10 | | 192x192 | |
| **Bit Precision** | 8 | | 8 | | 4 | | 1 | |
| **# Array used for compute** | 3 | | 6 | | 2 | | 2 | |
| **# Wordline for computation** | 375 | | 1000 | | 512 | | 192 | |
| | **cycle #** | **percentage** | **cycle #** | **percentage** | **cycle #** | **percentage** | **cycle #** | **percentage** |
| **Total Latency** | 39,628 | 100 | 33,434 | 100 | 251,290 | 100 | 1,572,628 | 100 |
| **Input loading** | 18,323 | 46.2 | 24 | 0.07 | 320 | 0.13 | 1,152 | 0.07 |
| **CRAM Compute** | 3,459 | 8.7 | 21,267 | 63.6 | 184,020 | 73.2 | 1,556,458 | 99.0 |
| **output readout** | 17,846 | 45.0 | 12,143 | 36.3 | 66,950 | 26.6 | 15,018 | 0.95 |
| | **cycle#** | **speedup** | **cycle #** | **speedup** | **cycle #** | **speedup** | **cycle #** | **speedup** |
| **Baseline** | 287,073 | 7.24x | 1,174,032 | 35.1x | 8,120,415 | 32.3x | 164,456,448 | 114x |

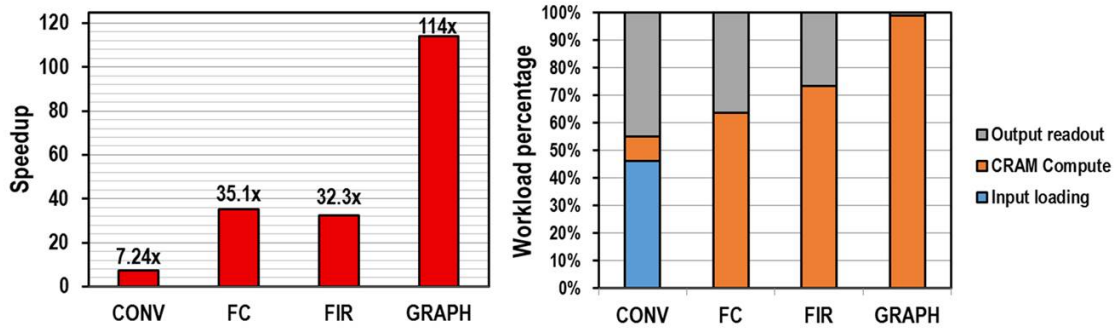



Figure 3.10: Performance comparison between CRAM and baseline scenario (top), workload breakdown (bottom).

| | This Work | JSSC2017 [1] | ISSCC2018 [2] | ISSCC2018 [3] | ISSCC2018 [4] | VLSI2017 [5] |
|---|---|---|---|---|---|---|
| **Technology** | 28nm | 130nm | 65nm | 65nm | 65nm | 40nm |
| **Supply Voltage** | 0.6~1.1V | 1.2V | 0.9~1.2V | 0.65~1V | 0.6~1V | 0.65~0.9V |
| **SRAM size** | 128KB(8x16) | 2KB | 2KB | 16KB | 1KB (2x0.5) | 24KB(3x8) |
| **SRAM bitcell** | 8T | 6T | 10T | 6T | 6T | 10T |
| **Method of Computing** | Digital | Analog | Analog | Analog | Digital | Digital |
| **Type of Supported Functions** | Logic/Add/Sub/Mult/Div/ FP | Add/Mult | Add/Mult | Add/Mult | Add/Mult | Logic |
| **Bit precision** | Arbitrary | 5b (input) 1b (weight) | 7b (input) 1b (weight) | 16b (train) 8b (infer) | 1b (input) 1b (weight) | Arbitrary |
| **Die Area (mm2)** | 2.7 | 0.36 | 0.067 | 1.44 | - | 1.28 |
| **Max Freqency (MHz)** | 475 | 50 | 6.7 | 1000 | 435 | 90 |
| **Normalized Performance (GOPS)*** | 32.7 | 40.5 | 1.17 | - | - | - |
| **Performance per unit Area (GOPS/mm$^2$)**** | 27.3 | 114 | 17.5 | - | - | - |
| **Normalized Energy Efficiency (TOPS/W) *** | 0.55 (mult) 5.27 (add) | 0.16 (mult) 4.58 (add) | 3.07 | 3.12 | 0.87 | - |
| *All normalized to 8-by-8 bit Multiply and Add Operation | | | | | | |
| ** Using SRAM area only | | | | | | |

Table 3.6: Comparison with Previous In-memory Computing Work.

# CHAPTER IV

# 288µW Deep-Learning Accelerator with 270KB Custom Low Power SRAM and Non-Uniform Memory Hierarchy for Mobile Intelligence

## 4.1 Introduction

Deep learning has proven to be a powerful tool for a wide range of applications such as speech recognition and object detection, among others. Right now, many try to deploy deep learning applications to mobile phone, wearable device, and even Internet-of-Things (IoT) sensor node to enable "mobile intelligence". Typically, these mobile devices just send data (e.g. image or sound) to the server, and the server executes the deep learning algorithm; then, the server sends results back. This way, even simple computation can result in latency, and energy overhead due to communication. Recently there has been increased interest in designing deep learning accelerator for mobile IoT [65] to enable intelligence at the edge and shield the cloud from a deluge of data by only forwarding meaningful events.

Therefore, some people propose a hierarchical deep neural network (DNN). As shown in Figure 4.1 different-scale DNNs is computed at different hardware platform. Small DNNs with computation power less than tens of mW, like voice activity detection (VAD), keyword spotting (KWS), face detection should be processed locally

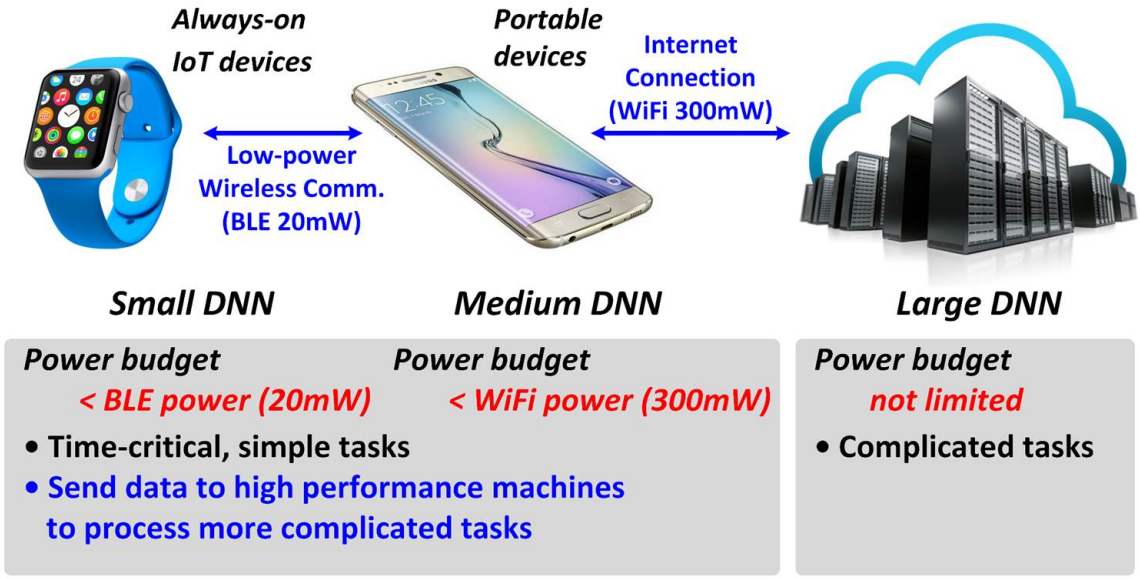



Figure 4.1: Hierachical deep neural network.

on always-on IoT and wearable devices, because the communication power of these device (usually bluetooth) costs around 20mW. Likewise, medium DNNs with computation power less than hundreds of mW can be processed on portable devices to handle time-critical tasks. And, only for complicated tasks, data is sent to servers. This hierarchical intelligence thereby enhances both radio bandwidth and power efficiency by trading-off computation and communication at edge devices. In this way, we extend battery life time of mobile and edge device by saving communication energy.

In this work, we focus on building a low-power programmable deep learning accelerator (DLA) to run "always-on" applications (e.g., voice commands or face detection) in IoT platform like [66, 67] with power budget less than tens of mW. These applications are crucial to the battery-powered device in that the chip can sleep most of the time and wake up by the always-on DLA only when meaningful activity is detected. Therefore, low power is a critical design constraint for this type of DLA. However, prior works [68, 69] have focused on high performance reconfigurable processors optimized for large-scale deep neural networks that consume >50mW. Off-chip weight storage in DRAM is also common in prior works [68, 69], which implies significant

additional power consumption due to intensive off-chip data movement. Therefore, we need a new design for low power DLA that can run small DNNs efficiently in edge devices.

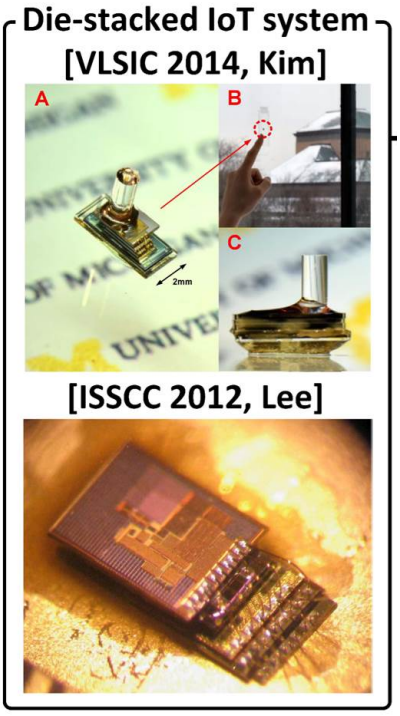



Figure 4.2: $1mm^3$ die-stacked sensing platform.

In summary, we propose a low-power, programmable deep learning accelerator with all weights stored in 270KB on-chip SRAM for mobile intelligence. Low power (less than 300µW) is achieved through the following 4 techniques:

1) Highly flexible and compact memory storage is realized via independent control of reconfigurable fixed-point bit precision ranging from 6–32 bits for neurons and weights.

2) Four processing elements (PEs) are located amidst the weight storage memory of 270kB, minimizing data movement energy;

3) A non-uniform memory hierarchy provides a trade-off between small, low power memory banks for frequently used data (e.g., input neurons) and larger, high density banks with higher power for the large amount of infrequently accessed data (e.g.,

synaptic weights).

4) A 0.6V 8T custom memory is specifically designed for DNNs with low-swing and replica bit-line, a sequential access mode, bank-by-bank drowsy mode control, power-gating for peripheral circuits, and voltage clamping for data retention;

These techniques were implemented into a complete deep learning processor in 40nm CMOS, including the DLA, an ARM Cortex-M0 processor, and MBus [66] interface to enable integration into a complete sensor system (Figure 4.2). The DLA consumes 288µW and achieves 374 GOPS/W efficiency. We demonstrate full system operation for two mobile-oriented applications, keyword spotting and face detection.

## 4.2 Deep Learning Algorithm and Processor

As mentioned before, edge devices can't affort to have high power off-chip DRAM, and we have to try fitting all the neural network weights on chip. However, even a state-of-art keyword spotting network [16] requires 2.1M parameters, 8.4MB of SRAM storage using 32-bit floating points number, which is still impossible for many small IoT sensors. Therefore, we first need to optimize these algorithms for better energy and area efficiency. We find that the only thing software designers care about is the accuracy, and they usually trade a lot storage and computation cost for only small gain in accuracy, which means in reverse, we can reduce the network size a lot with only mild degradation in accuracy [25, 27]. First, the bit-precison in DNN inference engine doesn't not have to be floating points or 32-bit fix-point. It may vary across different layers and networks from 3 bits to 16 bits [70, 71]. Figure 4.3 the minimum bit-width required for different layers in the well-known networks and corresponding the error tolerance. With variable precision fix-point representation, we can save at least 2-6× weight storage.

To take fully advantage of the variable bit precision and make memory storage more efficient, the processor hardware supports 4 different short data representations

| Tolerance | Bits per layer (I+F) |
|---|---|
| **AlexNet** (F=0) | |
| 1% | 10-8-8-8-8-8-6-4 |
| 2% | 10-8-8-8-8-8-5-4 |
| 5% | 10-8-8-8-7-7-5-3 |
| 10% | 9-8-8-8-7-7-5-3 |
| **NiN** (F=0) | |
| 1% | 10-10-9-12-12-11-11-11-10-10-10-9 |
| 2% | 10-10-9-12-12-11-11-11-10-10-10-9 |
| 5% | 10-10-10-11-10-11-11-11-10-9-9-8 |
| 10% | 9-10-9-11-11-10-10-10-9-9-9-8 |
| **GoogLeNet** (F=2) | |
| 1% | 14-10-12-12-12-12-11-11-11-10-9 |
| 2% | 13-11-11-10-12-11-11-11-11-10-9 |
| 5% | 12-11-11-11-11-11-10-10-10-9-9 |
| 10% | 12-9-11-11-11-10-10-10-10-9-9 |

Figure 4.3: Minimum bit-width in different layers and networks for error tolerance between 1% and 10%. (I: integer bits; F: fractional bits) [70].

for weight, input, output and temporary output. As shown in Figure 4.4, we can choose 6/8/12/16 bits for weight/input/output. Since temporary output is the intermediate accumulation result and has a higher dynamic range, they can be 16/24/32 bits. And we choose one word of SRAM to be 96 bits so that we can group integer numbers of weights/inputs/outputs/temporary outputs into one word without wasting the precious memory capacity. What's more, long memory word can also reduce the number memory accesses to save expensive data movement energy. Inside each processing elements (PE), we have programmable ping-pong buffer to unpack incoming 96-bit data to selected precision and pack out-going data till 96-bit before storing it back.

Second, we find that many weights in the neural networks are close to zero, which means network size can be greatly reduced by pruning zero-like weights and re-training the network to gain accuracy [18]. For example, we successfully reduced the keyword spotting network size from 2.1M to 300K, about 200KB of memory storage with average 6-bit precision [25], which is now affordable to a sensor node. In contrast

| Data type | Available precisions |
|---|---|
| Weight | 6b, 8b, 12b, 16b |
| Input | 6b, 8b, 12b, 16b |
| Output | 6b, 8b, 12b, 16b |
| Temporary Output (TO) | 16b, 24b, 32b |

| Precision | Number of elements per word |
|---|---|
| 6b | 16 |
| 8b | 12 |
| 12b | 8 |
| 16b | 6 |
| 24b | 4 |
| 32b | 3 |

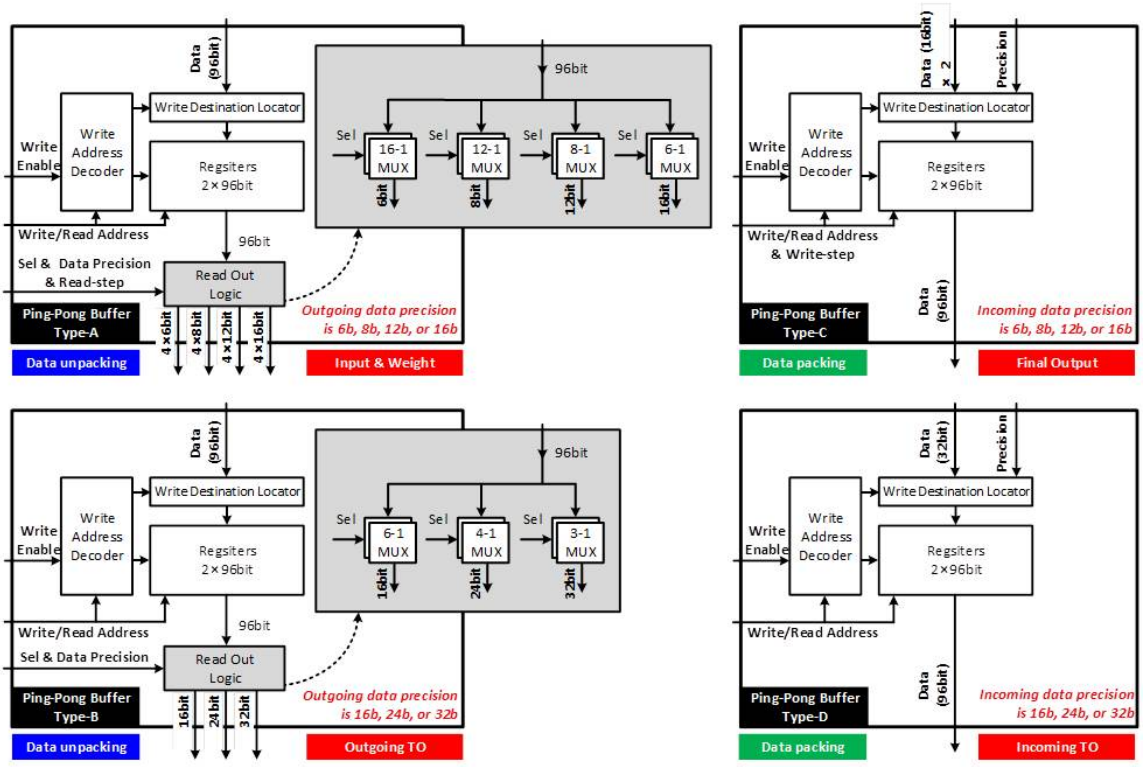



Figure 4.4: Available precisions for different data types (top) and programmable ping-pong buffer to unpack and pack data (bottom).

to the big storage, the throughput requirement of such small DNN is extremely low —only 300K multiply-and-accumulate operations (MAC) per 10ms. A systolic array of Multiply-and-Accumulate (MAC) unites like [72, 73] is completely unnecessary. Just 16 multipliers running at 2MHz rate is sufficient for the job like keyword spotting. High memory and small processing elements (PE) will make data movement very inefficiency. Since communication energy is more expensive than computation, we break one big PE into 4 smaller ones and surrounding each one by one-fourth of the

total memory to shrink the average distance between data and computation as shown in Figure 4.5. Though each PE can still access all 270KB memory, we will try to minimize data sharing and put most data one PE needs in its own memory sector to full exploit spatial locality.

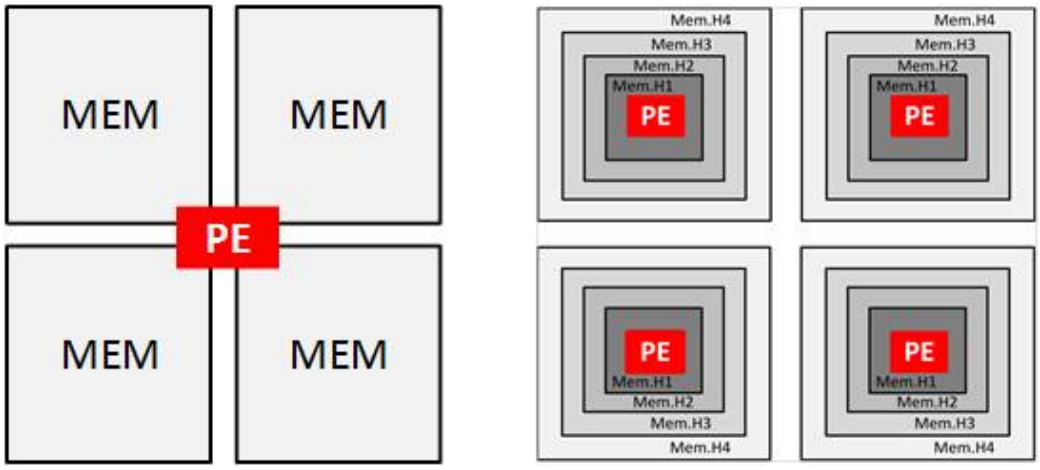



Figure 4.5: One big PE and memory (left), four PE surrounded by its own memory sector to exploit spatial locality (right).

Figure 4.6 shows the overall DLA architecture. The DLA has four PEs surrounded by their memory. Each PE has an ALU, instruction buffer, status register, data buffers, controller, memory address mapping unit, and memory arbitration unit. The ALU contains 4 8-bit multipliers, 4 16-bit multipliers, and 10 adders. The PE is programmed by two ping-pong CISC instruction buffer, which are 192b long including start address, size, precision, and operation-specific flags. The reconfigurable PE CISC operations are: 1) Fully Connected Layer (FCL) processing, 2) Fast Fourier Transfer (FFT), 3) data-block move, and 4) Look-Up Table(LUT)-based non-linear activation function. The memory address mapping unit and memory arbitration unit in each PE governs prioritized memory access arbitration, enabling PEs to access other PEs memory space. PEs can be programmed via offline scheduling optimization to avoid memory access collisions and contamination. The DLA operation sequence is controlled by the Cortex-M0, which loads data and instructions into PE memory.

As a PE instruction can take many cycles to complete, the Cortex-M0 supports clock-gating and it wakes upon PE completion. An external host processor can program the Cortex-M0 and DLA using a serial bus interface.

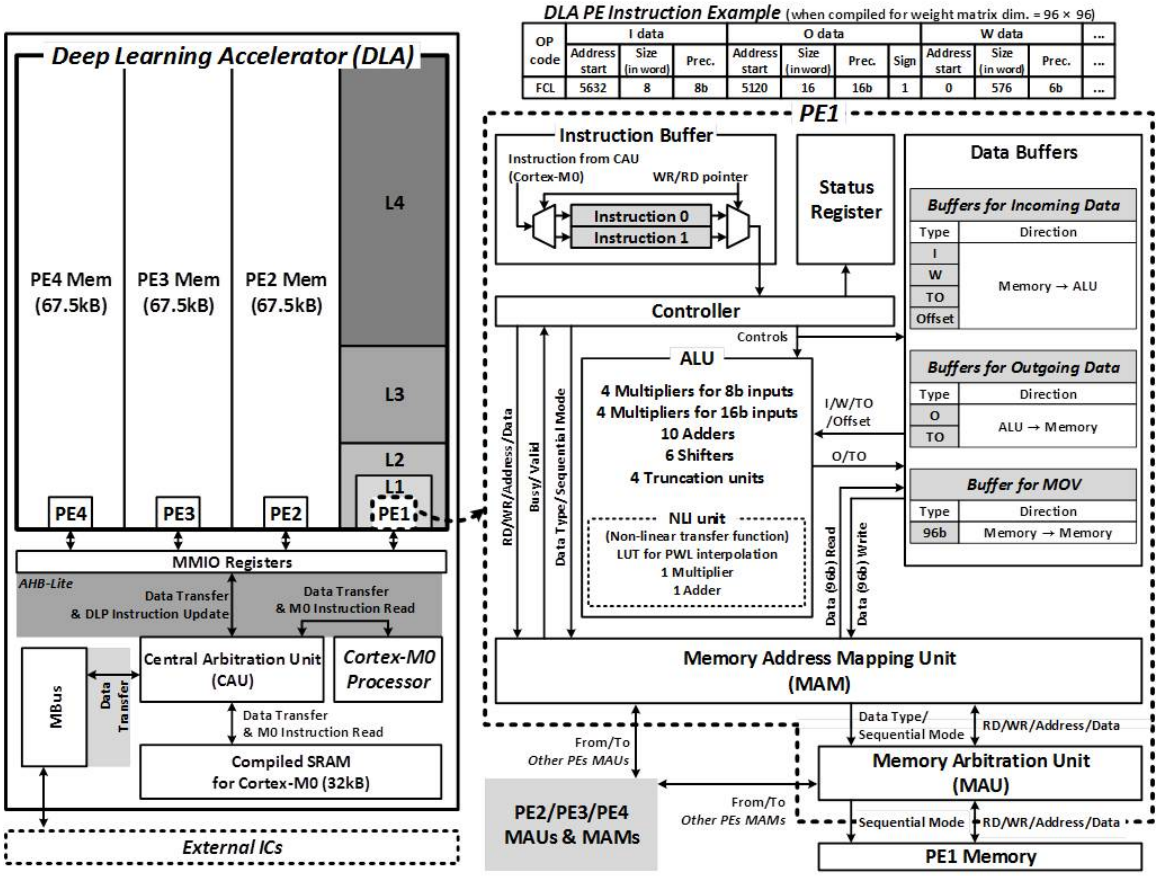



Figure 4.6: Top-level diagram of proposed deep learning accelerator (DLA) (left). DLA PE instruction example (top). DLA PE block diagram (right).

## 4.3 Non-Uniform Memory Access Architecture

We exploit temporal and spatial locality by using a cache-like hierarchical non-uniform memory access (NUMA) architecture. Since the DLA is optimized for implementing fully-connected layer (FCL) in deep neural networks and the FCL mainly performs matrix-vector multiplication, we observe that small inputs vector needs to be assessed multiple times per inference while large weight matrix has no data reuse

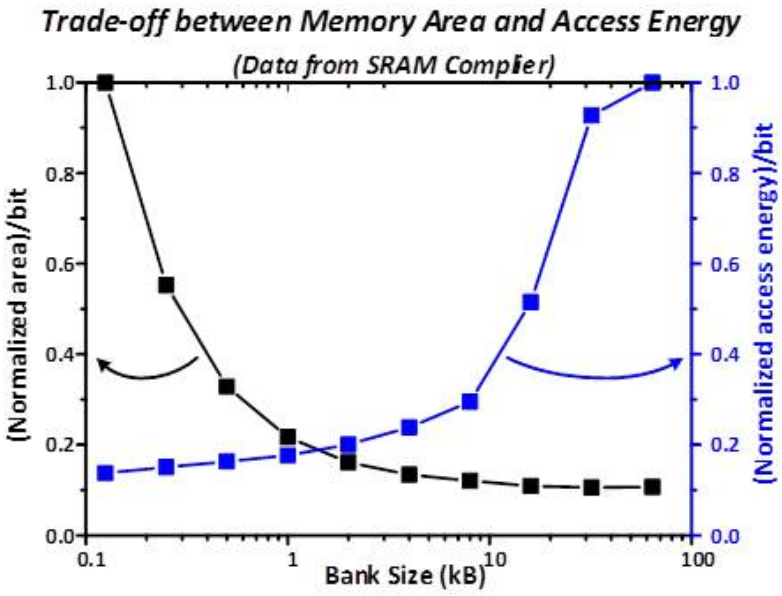



Figure 4.7: Trends in SRAM density and access energy with different bank size.

at all. Besides, we observe that smaller SRAM banks have lower access energy with relatively worse area density, while the opposite is true for larger banks as shown in Figure 4.7. Therefore, we can strategically map the input vector to the nearest local memory like L1 cache so that the DLA can reuse it as many times as possible once loaded, while the infrequently accessed weight matrix is loaded from dense (but higher access energy) upper hierarchy memory like L3 cache. However, different from caches, we don't need to pay significant power/area overhead for the content addressable memory and complicated controller. Instead, we just need SRAM banks will small ones closest to the PE and large banks in the distance. Because deep learning algorithms can be deterministically scheduled at compilation time, predetermining optimal memory assignments.

NUMA is carefully designed to strike a balance between memory area and access energy. The number of NUMA hierarchical levels and the memory size of each hierarchy in Figure 4.8 were determined via extensive simulations that analyzed NUMA configurations for various DNN topologies. In the proposed architecture, NUMA memory has 67.5kB in total with four banks in each level of hierarchy. Unit bank

sizes are 0.375, 1.5, 3, and 12kB. What's more, PE memory uses gating circuits to prevent unnecessary signal switching in hierarchical memory accesses. That is, lower level memory access signals do not propagate to higher levels. Simulations show that combining NUMA with the tiling strategy for 4 PEs leads to >40% energy saving with 2% area overhead compared to UMA (unit bank = 16kB) for the same tasks and total memory capacity (Figure 4.9).

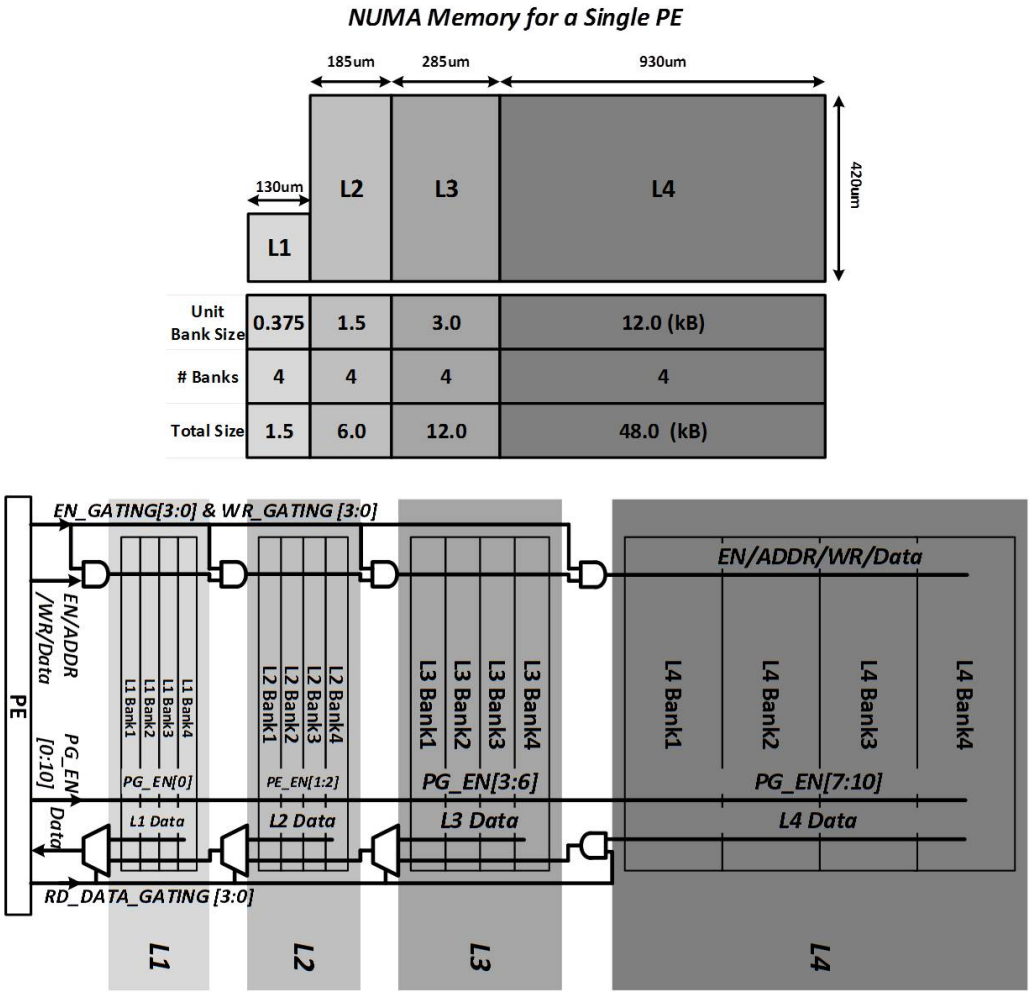



Figure 4.8: The number of NUMA hierarchical levels and the memory size of each hierarchy (top), and signal gating circuit to unnecessary signal switching (bottom).

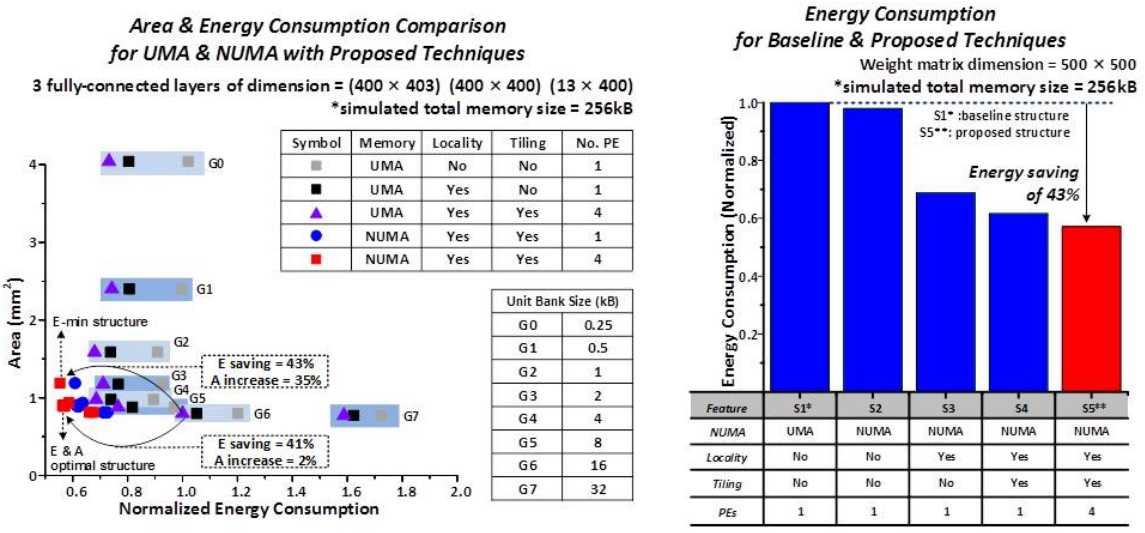



Figure 4.9: Area and energy comparison with UMA & NUMA and proposed techniques.

## 4.4 Low Power Custom SRAM

As mentioned before, in such a DNN computation unit area and power is not comparable to the memory storage. If all 16 multipliers on chip work at the same time, the computation power is about 31uW. If using the SRAM compiler provided by TSMC, the read active power of the compiled SRAM is 528uW and the total leakage power of 270KB memory is 8.27mW. Therefore, to reduce both system active and leakage power, it's crucial to have a custom low power SRAM.

### 4.4.1 8T HVT Bit-cell and Noise Margin

The compiled SRAM use push-rule 6T bitcell and requires a pretty high minimum operating voltage ($V_{min}$) to ensure functional correctness. Traditional 6T SRAM bitcell is good for density, but bad for low voltage operation because of the contention between the read static noise margin (SNM) and write noise margin (WNM) [74]. One of the most effective way to reduce both active and leakage power is lowering the supply voltage. To ensure SRAM robustness under low voltage (0.6V), we choose to scarifies some area density and use 8T bitcell [75, 76] whose read and write operation

can be separately optimized at low voltage. But half-select issue still remains if the bank has column-mux or is bit-interleaved, which may limit our freedom to optimize WNM. Since one word of our SRAM has 96 bits, considering the aspect ratio of the array layout , we choose not to have column-mux or bit-interleaving. Besides, we choose high threshold (HVT) transistor for 6 out of the 8 transistors, which reduce the leakage power of the array by an order of magnitude. The rest two transistors in the read port is still normal threshold (SVT) transistor for a faster read speed. The bitcell schematic and layout is shown in Figure 4.10.

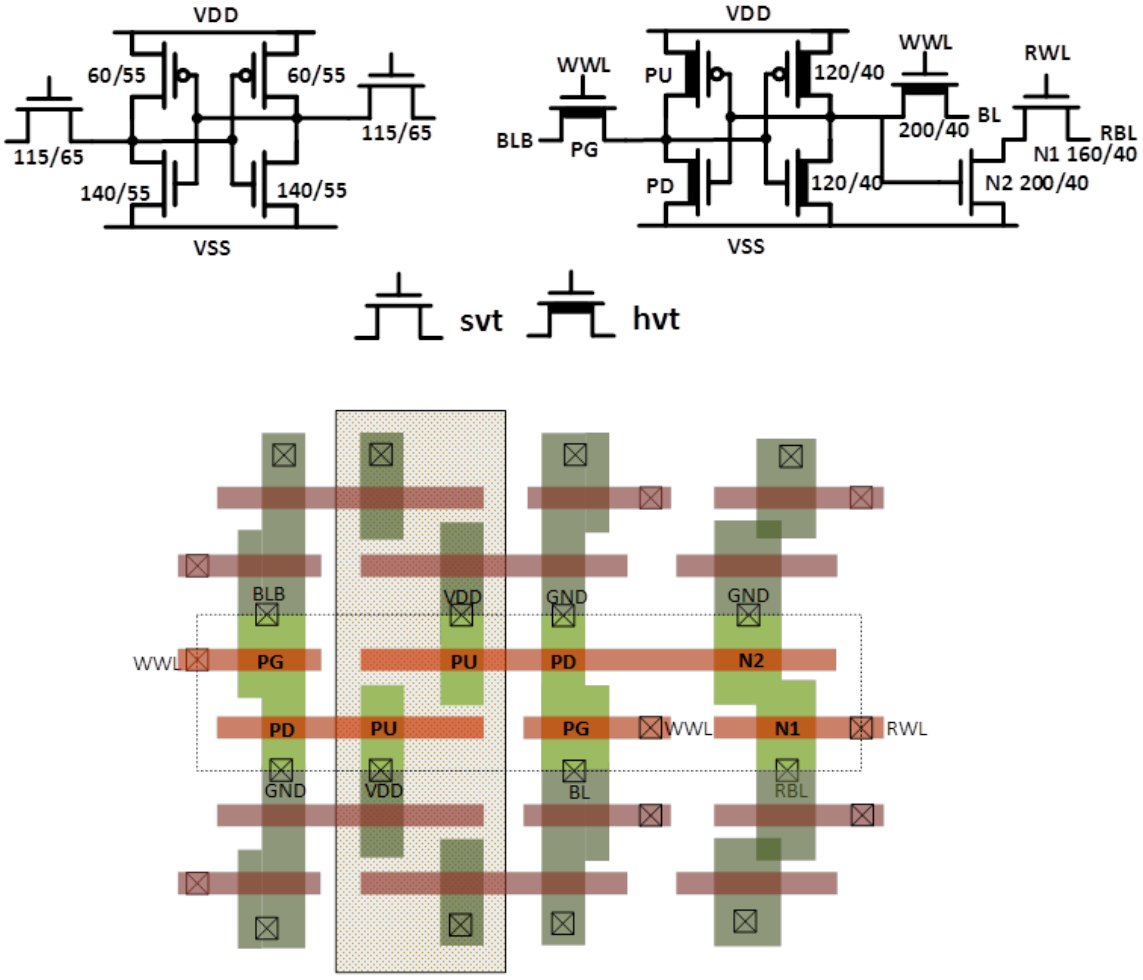



Figure 4.10: Compiler 6T push-rule bit-cell (top-left), 8T HVT bit-cell schematic (top-right) and layout (bottom).

### 4.4.2 Active Power Reduction Techniques

We proposed several techniques to reduce the SRAM access energy to 5-6fJ/bit. Through the spice simulation of a medium size SRAM bank with 128 word-lines and 32 bit-lines, we find that over 80% of the access energy is consumed by the bit-line charge and discharge, and peripheral takes over 70% of the total leakage as shown in Figure 4.11.

| | | | Array | Decoder | WL driver | BL & SA | **Total** |
|---|---|---|---|---|---|---|---|
| 128x32 | Read | Energy/bit (fJ) | 0.00005 | 0.2 | 0.413 | 3.26/0.87 | **4.75** |
| | | Percent | 0% | 4.2% | 8.7% | 87.1% | |
| | Write | Energy/bit (fJ) | 0.0873 | 0.162 | 0.419 | 5.55 | **6.22** |
| | | Percent | 1.4% | 2.6% | 6.7% | 89.2% | |
| | Leakage | Power(pw) | 3.573 | 3.433 | 3.705 | 1.83 | **12.54** |
| | | Percent | 28.5% | 27.3% | 29.5% | 14.6% | |

Figure 4.11: Active and leakage power breakdown of a SRAM array.

#### 4.4.2.1 Low-Swing and Replica Bit-line

The high bit-line discharge energy is due to the large bit-line capacitance and voltage swing. We reduce the bit-line capacitance by using a qual-array bank structure. Each bank consists of 4 (instead of 2) sub-arrays to share address decoder and readout circuits so that the bit-line length and capacitance is halved as shown in Figure 4.12.

Since 8T bitcell has only one read bit-line, most SRAM designs use a single-ended sense amplifier, skewed inverter, for large signal / full swing sensing. To reduce the bit-line voltage swing, we use the differential sense amplifier for small signal sensing. And we use replica bit-cell and bit-line to generate the reference voltage for the differential sense amplifier. Our normal 8T bit-cell use SVT transistors for the read port. The replica bitcell has only the read port, consisting of one HVT transistor and one SVT transistor with a smaller size. And a replica bit-line made up with a column of the replica bit-cells is placed in the middle of each sram array to generate the reference voltage for all other columns in the array. The area overhead of the

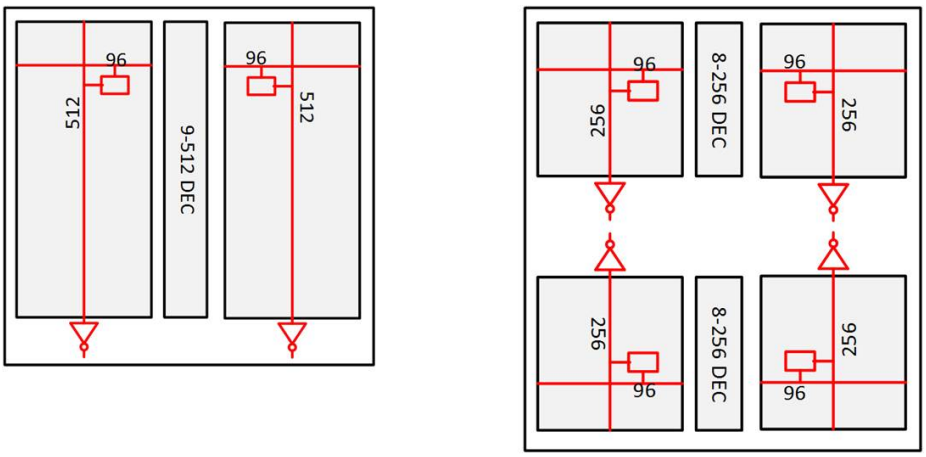



Figure 4.12: Long bit-line length of a dual-array bank (left) versus short bit-line length of a qual-array bank (right).

replica bitline is amortized to only 0.5%. Figure 4.13 and 4.14 shows the replica bit-cell/bit-line and the differential sense amplifier.

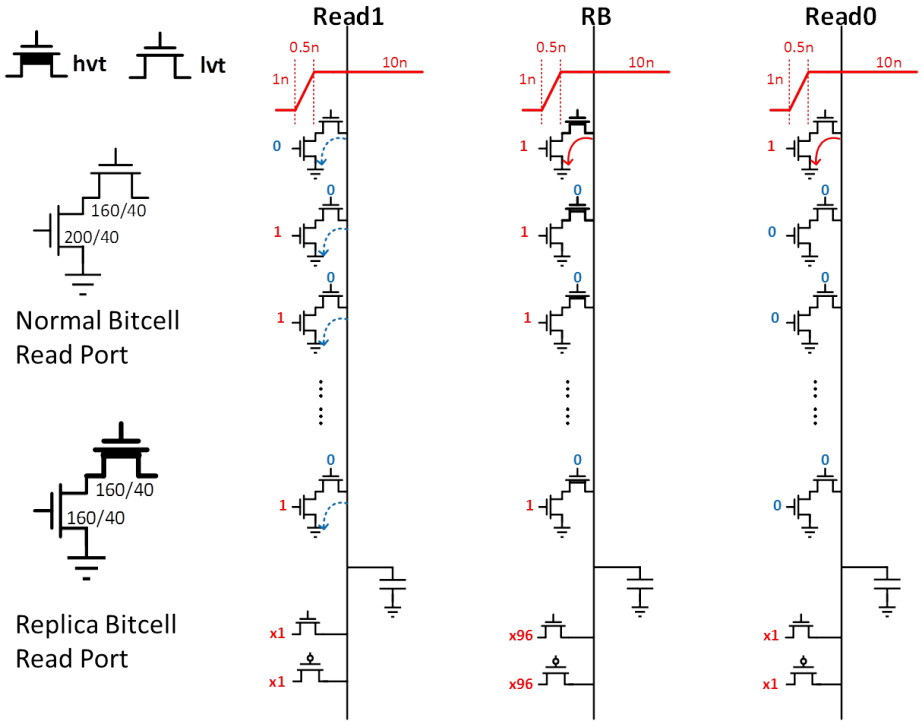



Figure 4.13: Replica bit-cell and bit-line to provide the reference voltage for differential sense amplifier.

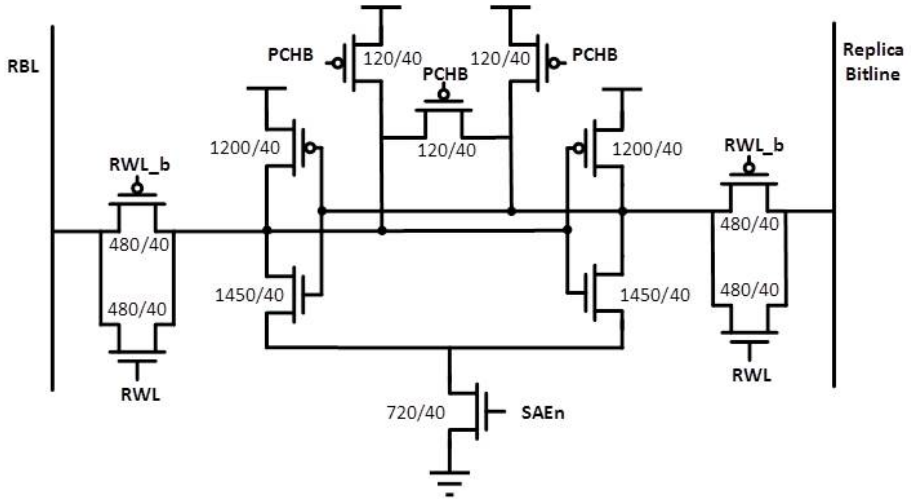



Figure 4.14: Differential Sense Amplifier Schematic.

#### 4.4.2.2 SR-Latch Gated Sequential Address Decoder

From the DNN algorithm study, we found that memory access pattern is predictable and most of the time is sequential. Therefore, we propose a shift-register based sequential decoder with SR-latch clock gating to save the energy in local address generation. First, each SRAM bank still has the traditional address decoder to generate the one-hot code and then use it to initialize the shift-register. Later, if the central memory address controller find the new address is just the previous address incremented by one, it will only send 1 bit sequential enable signal instead of 15-bit address signal. To save the energy in clock tree, sixteen shift registers are grouped into one clock group and gated by a SR-latch like Figure 4.15. Therefore, each time only one register group may see the clock signal. From the energy break down in Figure 4.15, sequential decoder costs only half power than random decoder. What's more, the main energy saving of sequential decoder comes from the reduced switching activity in the long address bus.

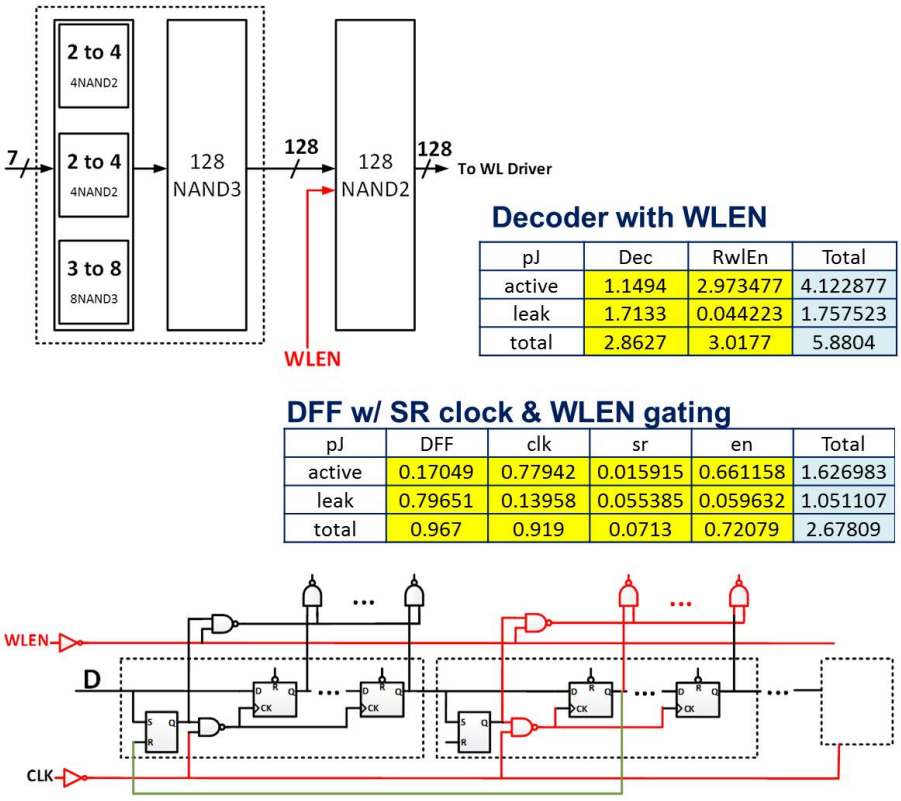



Figure 4.15: Energy comparison between Random Decoder (top-left) and Sequential decoder (bottom).

### 4.4.3 Leakage Power Reduction Techniques

Memory access of DNN algorithm can be deterministically scheduled. Given that only a few banks are actively accessed in a specific PE while the others stay idle during the majority of processing time, we employ a dynamic drowsy mode for SRAM leakage reduction. Each PE dynamically controls power gating and clamping headers of SRAM peripheral circuits and arrays, bank-by-bank based on the schedule. During drowsy mode, peripherals are power-gated using large HVT PMOS header, while array voltage is clamped with an small LVT NMOS source follower (Figure 4.16). The reference voltage to the gate of NMOS clamping header is generated on-chip by a diode stack and a programmable analog-mux array to ensure data retention (Figure 4.17).

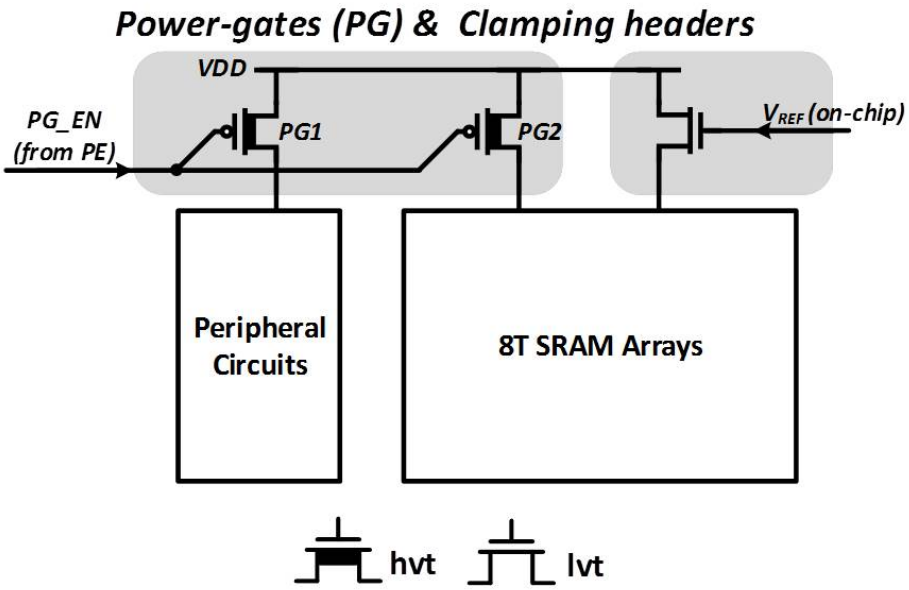



Figure 4.16: PMOS power header and NMOS clamping headers.

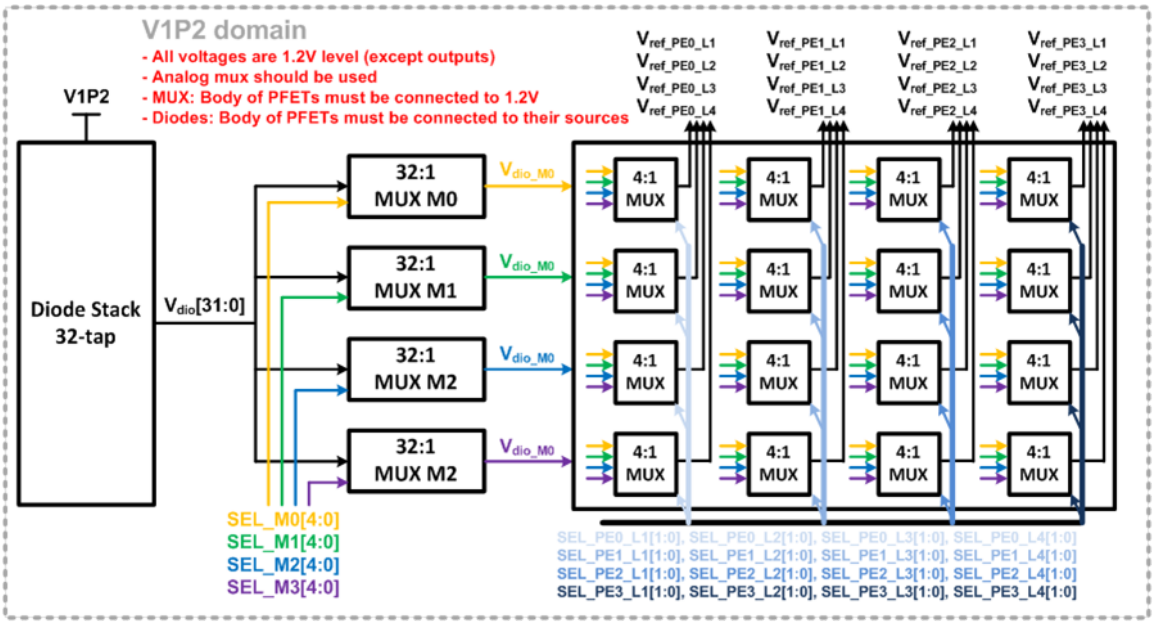



Figure 4.17: Diode stack for on-chip reference voltage generation.

## 4.5 Measurements and Results

The test chip is fabricated in TSMC 40nm Low Power CMOS technology. Figure 4.18 shows the die photo. As expected, the memory takes over 70% of total chip area. Measurement results confirm effectiveness of the proposed NUMA and drowsy

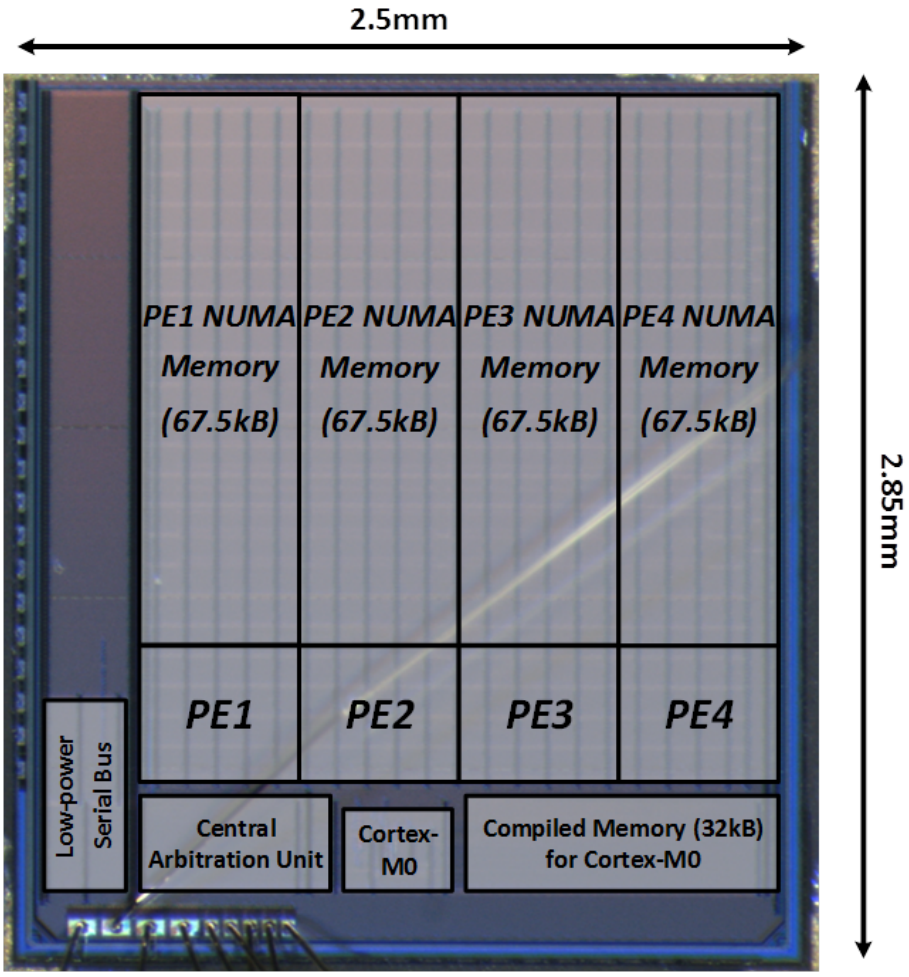



Figure 4.18: Die photo of Deep Learning Processor.

mode operation (Figure 4.19). Measured data access power consumption in L1 is 60% less than in L4. Memory drowsy mode operation reduces leakage by 54%, which is mainly attributed to peripheral circuits as the bit-cell is inherently low leakage. The test chip achieves peak efficiency of 374GOPS/W while consuming 288µW at 0.65V and 3.9MHz. Keyword spotting (10 keywords) and face detection (binary decision) DNNs are successfully ported onto the proposed DLA with layer dimensions and precision mapping specified in Figure 4.20. Both DNN classifications fit into the 270kB on-chip memory and exhibit $<$7ms latency, allowing for real-time operation. Figure 4.20 compares against state-of-the-art prior work.

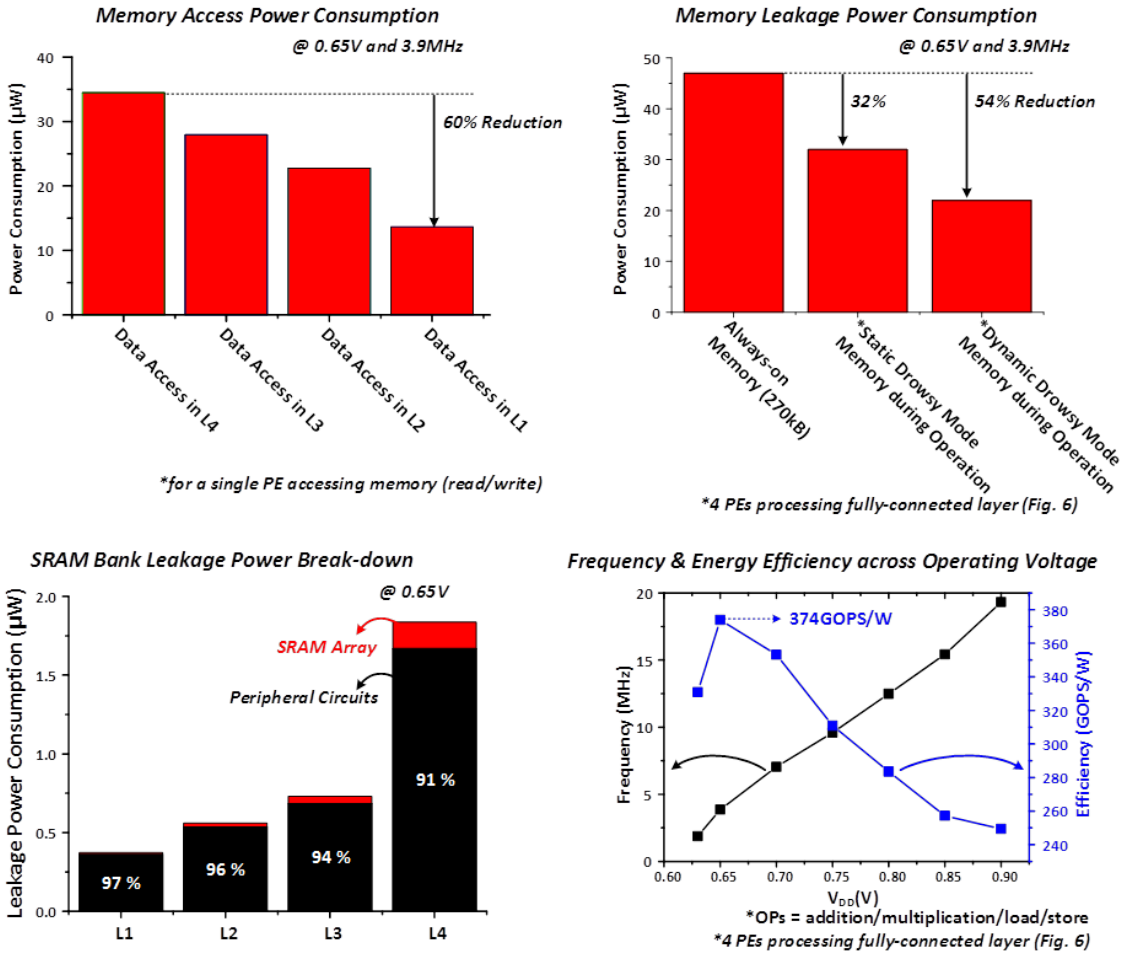



Figure 4.19: Memory access power consumption (top left). Memory leakage power comparison (top right). SRAM bank leakage break-down (bottom left). Performance and efficiency across voltage (bottom right).

## 4.6 Summary

To summarize, we proposes a a low-power, programmable deep learning accelerator with all weights stored in 270KB on-chip SRAM for mobile intelligence. Less than 300µW power is achieved through: 1) highly flexible and compact memory storage realized via reconfigurable fixed-point bit precision ranging from 6–32 bits; 2) minimizing data movement energy by locating four PEs amidst the 270kB memory; 3) NUMA architecture fully exploiting temporal and spatial locality; 4) custom low power memory specially designed for DNNs with 8T HVT bit-cell, low-swing bit-line,

*Performance Summary*

| App. | Layer* | #PE in use | Weight matrix | Precision (bit) | | | | Power (μW) | Latency (msec) | GOPS/W |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | W | I | O | TO | | | |
| *General* | 1st | 4 | 384 × 408 | 6 | 12 | 12 | 32 | 288 | 2.97 | 374 |
| App. | Layer* | #PE in use | Weight matrix | Precision (bit) | | | | Power (μW) | Latency (msec) | GOPS/W |
| | | | | W | I | O | TO | | | |
| *Keyword spotting* | 1st | 4 | 384 × 408 | 6 | 12 | 12 | 32 | 360 | 2.98 | 316 |
| | 2nd | 4 | 384 × 384 | 6 | 12 | 12 | 32 | 312 | 2.80 | 365 |
| | 3rd | 1 | 24 × 384 | 6 | 12 | 12 | 32 | 189 | 0.72 | 140 |
| Total | | | | | | | | 321 | 6.50 | 318 |
| *Face detection* | 1st | 1 | 16 × 1032 | 8 | 8 | 16 | 32 | 208 | 1.70 | 113 |
| | 2nd | 1 | 2 × 16 | 8 | 16 | 16 | 32 | 195 | 0.02 | 112 |
| Total | | | | | | | | 208 | 1.72 | 113 |

@ 0.65V and 3.9MHz
*Fully-connected layer

*Comparison Table*

| | This work | [2] | [3] |
|---|---|---|---|
| Technology | 40nm | 65nm | 40nm |
| Chip area | 7.1mm$^2$ | 16mm$^2$ | 2.4mm$^2$ |
| Design target | Fully-connected layer & FFT | Convolutional layer | Convolutional layer |
| Operating voltage | 0.63 – 0.9 V | 0.82 – 1.17 V | 0.55 – 1.1 V |
| Operating frequency | 1.9 – 19.3 MHz | 100 – 250 MHz | 12 – 204 MHz |
| Efficiency | 374 GOPS/W$^{*,\dagger}$ | 9.6 GOPS/W$^{\dagger,\#}$ | 300 – 2600 GOPS/W$^{\ddagger}$ |
| Operating power | 0.288 mW$^{\dagger}$ | 278 mW$^{\ddagger}$ | 76 mW$^{\ddagger}$ |
| On-chip core SRAM size | 270 kB | 181.5 kB | 144 kB |
| Off-chip memory requirement | No | Yes | Yes |
| Variable precision | 6/8/12/16/24/32-bit | No | 1 – 16-bit |

*OPs = addition/multiplication/load/store. †Operating at 0.65V and 3.9MHz.
‡DRAM access power not included. #Calculated based on reported GOPS and power.

Figure 4.20: Performance summary for neural networks with a variety of layer specification (top). Comparison table (bottom).

sequential decoder, peripheral power-gating, voltage clamping for data retention, and bank-by-bank drowsy mode control. The proposed design was implemented into a complete deep learning IoT processor in 40nm CMOS, including the DLA, an ARM Cortex-M0 processor, and MBus interface to enable integration into a complete sensor system. The DLA consumes 288μW at at 0.65V and 3.9MHz, and achieves 374 GOPS/W peak energy efficiency. We demonstrate full system operation for two mobile-oriented applications, keyword spotting and face detection.

# CHAPTER V

# 1.03pW/bit Ultra-low Leakage Voltage-Stacked SRAM for Intelligent Edge Processors

## 5.1 Introduction

As is discussed in chapter IV, SRAM area and leakage power have now dominated the smart sensor. In the previous project, we designed a deep learning processor targeting at small neural network to do simple tasks like keyword spotting. But in this project, we try to design a image signal processor (ISP) for an energy-efficient low-noise CMOS image sensor. Figure 5.1 shows the expected 3D-stacked system with a lens, imager, radio, flash storage, and image processor. The ISP is designed to run at least three different neural networks for human detection, face detection, and face recognition. Including main memory, image frame buffer, and data memory for neural network weights, ISP requires at least 6.4 Mbit of on-chip SRAM, which takes around 90% of the chip area as shown in Figure 5.2. Therefore, a custom ultra low leakage SRAM becomes crucial to the low power ISP.

## 5.2 Ultra Low Leakage SRAM for Low Power ISP

The sensor system is target for smart surveillance camera type of application and the imager is motion-triggered to save power. It means that the system has a very low

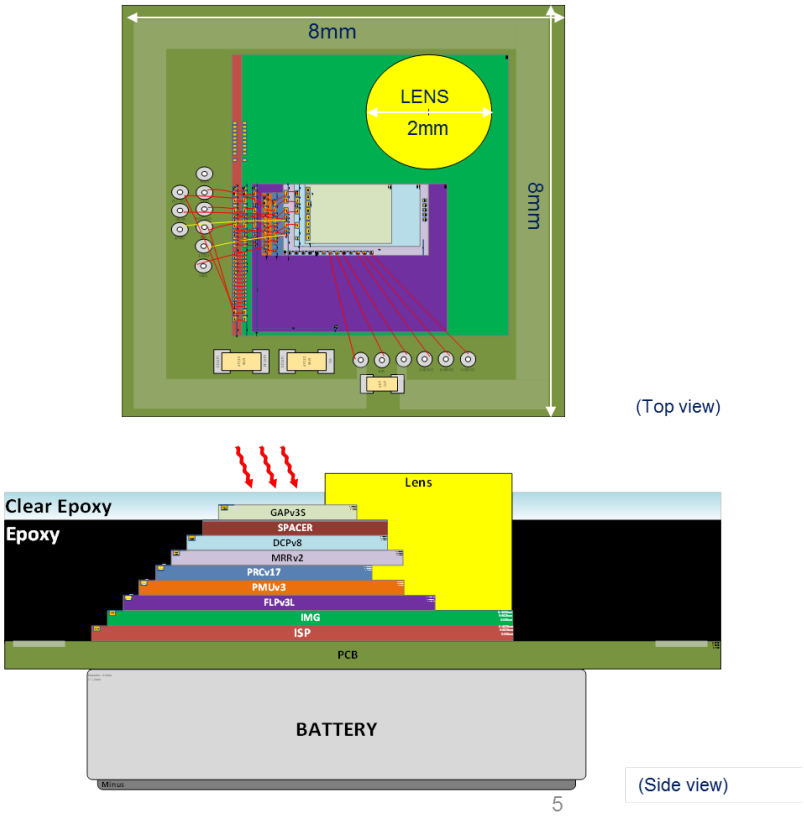



Figure 5.1: 3D-stacked smart sensor system with low power imager, radio, flash, and ISP.

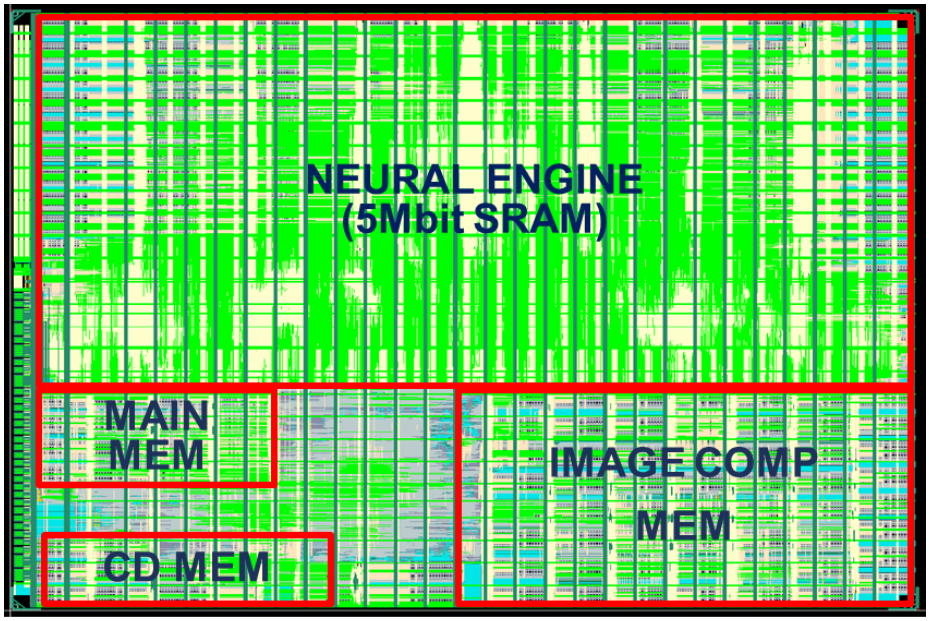



Figure 5.2: ISP Chip Layout shows that 90% area is memory.

activity, and most of the time, it will stay in sleep. However, over 5Mbit of SRAM in the neural engine of ISP cannot go to sleep even in idle mode since they retains the parameters of three neural networks. Leakage power of the on-chip SRAM is going to be a big burden for such a low activity sensor system. Therefore, we propose an ultra low leakage SRAM that is specially optimized to retain the data with the lowest power possible by sacrificing some area and active power.

### 5.2.1 Differential 8T Bit-cell

After increasing the channel length and using HVT transistor to bring us 18% and 8× leakage reduction respectively, in order to further reduce the leakage power, we found nothing is more effective than drop the supply voltage. If the array voltage drop from 0.6V to 0.3V, the leakage power can be reduced by another 11×. However, low supply voltage will greatly compromise the stability of SRAM bit-cell. And technology scaling has made it even harder to design robust SRAM, since process variation like Random Dopant Fluctuation (RDF) [19] and Line Edge Roughness (LER) [20] gets worse with reduced minimum feature size. Also the large on-chip SRAM requirement poses a bit challenge on maintaining a high yield even at low voltage. Therefore, we decide to trade the area for robustness and leakage reduction. By simulation, we find all channel width and array voltage pairs that has a hold noise margin of 8 mean-over-sigma as shown in Figure 5.3. Then we pick 220nm and 0.3V as the bit-cell size and array supply voltage, since it has both very low leakage and a reasonable size. However, we still use 0.6V for the peripheral and bit-line voltage to faster read/write speed and higher read/write margin. For low voltage operation, we still want to decouple the read and write ports, and therefore choose a differential 8T bit-cell design like [77]. It is very similar to a traditional 7T bit-cell except that one extra read transistor enables low swing differential read. Since we use 0.3V for array VDD and 0.6V for peripheral/bit-line voltage, during the read operation, bit-

cells in the unselected rows will have their read transistors in the super-cutoff region which can perfectly avoid the sneaking current issue in the traditional 7T SRAM [33]. Figure 5.4 shows the schematic of the bit-cell. We use HVT transistor for PU/PD/PG transistors for low leakage and LVT transistor for read access transistor for read speed.

| Width/VDD | 0.2 | 0.21 | 0.22 | 0.23 | 0.24 | 0.25 | 0.26 | 0.27 | 0.275 | 0.28 | 0.29 | 0.3 | 0.31 | 0.32 | 0.33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 140 | 4.14 | 4.44 | 4.74 | 5.04 | 5.33 | 5.63 | 5.93 | 6.24 | 6.39 | 6.54 | 6.84 | 7.14 | 7.45 | 7.75 | **8.06** |
| 150 | 4.27 | 4.58 | 4.88 | 5.19 | 5.50 | 5.81 | 6.12 | 6.43 | 6.58 | 6.74 | 7.05 | 7.36 | 7.68 | **7.99** | 8.31 |
| 160 | 4.40 | 4.71 | 5.03 | 5.35 | 5.67 | 5.98 | 6.30 | 6.62 | 6.78 | 6.94 | 7.26 | 7.59 | **7.91** | 8.23 | 8.56 |
| 180 | 4.64 | 4.97 | 5.31 | 5.64 | 5.98 | 6.31 | 6.65 | 6.99 | 7.16 | 7.33 | 7.66 | **8.00** | 8.35 | 8.69 | 9.03 |
| 190 | 4.75 | 5.09 | 5.44 | 5.78 | 6.12 | 6.47 | 6.81 | 7.16 | 7.33 | 7.51 | 7.85 | 8.20 | 8.55 | 8.90 | 9.25 |
| 200 | 4.87 | 5.22 | 5.57 | 5.92 | 6.27 | 6.62 | 6.98 | 7.33 | 7.51 | 7.69 | **8.04** | 8.40 | 8.76 | 9.12 | 9.48 |
| 220 | 5.08 | 5.45 | 5.82 | 6.18 | 6.55 | 6.92 | 7.29 | 7.66 | 7.84 | **8.03** | 8.40 | 8.78 | 9.15 | 9.53 | 9.90 |
| 240 | 5.27 | 5.66 | 6.04 | 6.42 | 6.80 | 7.18 | 7.57 | 7.95 | **8.15** | 8.34 | 8.73 | 9.11 | 9.50 | 9.89 | 10.28 |
| 260 | 5.47 | 5.86 | 6.26 | 6.66 | 7.05 | 7.45 | 7.85 | **8.25** | 8.45 | 8.65 | 9.05 | 9.45 | 9.86 | 10.26 | 10.67 |
| 280 | 5.66 | 6.07 | 6.48 | 6.89 | 7.30 | 7.71 | **8.13** | 8.54 | 8.75 | 8.96 | 9.37 | 9.79 | 10.21 | 10.63 | 11.05 |
| 300 | 5.86 | 6.28 | 6.70 | 7.13 | 7.55 | **7.98** | 8.41 | 8.83 | 9.05 | 9.26 | 9.70 | 10.13 | 10.56 | 10.99 | 11.43 |
| 320 | 6.01 | 6.45 | 6.89 | 7.32 | 7.76 | 8.20 | 8.64 | 9.08 | 9.30 | 9.52 | 9.96 | 10.41 | 10.85 | 11.30 | 11.75 |
| 340 | 6.17 | 6.62 | 7.07 | 7.52 | **7.97** | 8.42 | 8.87 | 9.32 | 9.54 | 9.76 | 10.23 | 10.69 | 11.15 | 11.61 | 12.06 |
| 360 | 6.33 | 6.79 | 7.25 | 7.71 | 8.17 | 8.64 | 9.10 | 9.57 | 9.79 | 10.01 | 10.50 | 10.97 | 11.44 | 11.91 | 12.38 |
| 380 | 6.49 | 6.96 | 7.44 | 7.91 | 8.38 | 8.85 | 9.33 | 9.81 | 10.03 | 10.25 | 10.77 | 11.25 | 11.73 | 12.22 | 12.70 |
| 400 | 6.65 | 7.13 | 7.62 | **8.10** | 8.59 | 9.07 | 9.56 | 10.05 | 10.30 | 10.54 | 11.04 | 11.53 | 12.03 | 12.52 | 13.02 |
| 420 | 6.79 | 7.28 | 7.78 | 8.27 | 8.77 | 9.27 | 9.77 | 10.27 | 10.52 | 10.77 | 11.27 | 11.78 | 12.28 | 12.79 | 13.30 |
| 440 | 6.93 | 7.43 | 7.94 | 8.44 | 8.95 | 9.46 | 9.97 | 10.48 | 10.73 | 10.99 | 11.50 | 12.02 | 12.54 | 13.05 | 13.57 |
| 460 | 7.07 | 7.58 | **8.10** | 8.62 | 9.13 | 9.65 | 10.17 | 10.69 | 10.95 | 11.21 | 11.74 | 12.27 | 12.79 | 13.32 | 13.85 |
| 480 | 7.21 | 7.73 | 8.26 | 8.79 | 9.31 | 9.84 | 10.37 | 10.90 | 11.17 | 11.44 | 11.97 | 12.51 | 13.05 | 13.59 | 14.13 |
| 500 | 7.34 | 7.88 | 8.42 | 8.96 | 9.49 | 10.03 | 10.57 | 11.12 | 11.39 | 11.66 | 12.21 | 12.75 | 13.30 | 13.85 | 14.40 |
| 520 | 7.47 | **8.02** | 8.56 | 9.11 | 9.66 | 10.20 | 10.75 | 11.31 | 11.58 | 11.86 | 12.42 | 12.97 | 13.53 | 14.09 | 14.65 |
| 540 | 7.59 | 8.15 | 8.71 | 9.26 | 9.82 | 10.37 | 10.94 | 11.50 | 11.78 | 12.06 | 12.63 | 13.19 | 13.76 | 14.33 | 14.90 |
| 560 | 7.72 | 8.28 | 8.85 | 9.42 | 9.98 | 10.55 | 11.12 | 11.69 | 11.98 | 12.26 | 12.84 | 13.41 | 13.99 | 14.57 | 15.15 |
| 580 | 7.84 | 8.42 | 8.99 | 9.57 | 10.14 | 10.72 | 11.30 | 11.88 | 12.17 | 12.46 | 13.05 | 13.63 | 14.22 | 14.81 | 15.40 |
| 600 | **7.97** | 8.55 | 9.14 | 9.72 | 10.31 | 10.89 | 11.48 | 12.07 | 12.37 | 12.66 | 13.26 | 13.85 | 14.45 | 15.05 | 15.65 |

Figure 5.3: Bit-cell hold noise margin at different channel width and supply voltage.

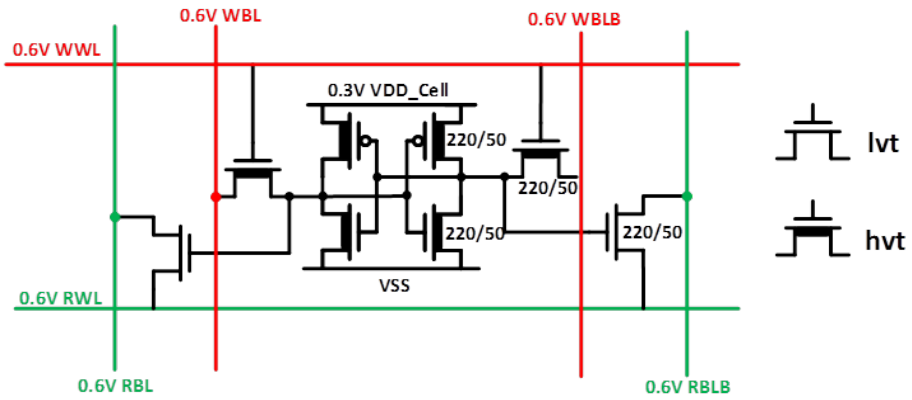



Figure 5.4: Schematic of the differential 8T bit-cell.

### 5.2.2 Stacked SRAM Array

There is substantial recent interest in implementing deep learning techniques within IoT devices to enable intelligence in edge devices and avoid the need for expensive wireless communication to the cloud. In addition, off-chip DRAM accesses are costly for highly miniaturized and power-constrained devices. As a result, it is beneficial to fit complete neural network models into on-chip memories, most commonly SRAM; given their relatively low density these memories can easily consume >80% of total chip area [18]. As a result, standby power of these battery-powered devices becomes dominated by SRAM leakage. For example, in the low-power, motion-triggered smart image sensor considered in this work, the firmware, reference frame, and neural network weights require an 8.9Mb SRAM that consumes up to 90% of the chip's standby power, dictating battery life. In this paper, we propose a stacked voltage domain SRAM where arrays are split into two sets (top and bottom) with their supplies connected in series. As a result, the system supply current is reused by top and bottom sets, and supply voltage is split between the two sets of arrays. This enables seamless integration of very low voltage SRAM retention in a larger system with a nominal supply, without resorting to a low efficiency LDO. A new array swapping approach (from top to bottom) provides stable access to arbitrary banks within one system clock cycle. We also employ a comprehensive sizing strategy (W and L) to optimally balance hold stability and bitcell size. Integrated in an imager IoT system in 40nm CMOS, the proposed 8.9Mb SRAM achieves 1.03pW leakage per bit marking over 100× reduction over conventional SRAM in the same technology.

Prior work has focused on reducing SRAM leakage via various techniques such as HVT/thick-oxide device [78, 79], reverse body bias [80, 81], floating bitline [82], raising VSS [83, 84], and lowering VDD [85, 86]. Apart from the use of HVT device, which enables an order of magnitude leakage reduction and is readily deployed, supply voltage lowering is one of the most effective approaches to reduce leakage due to the

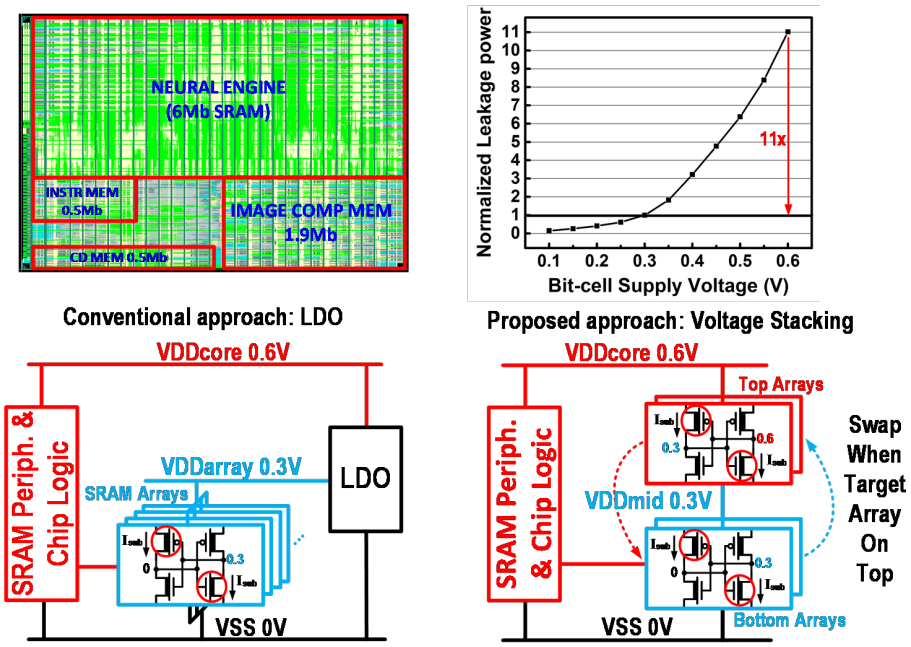



Figure 5.5: SRAMs on an image processing IoT chip (top-left), leakage power across voltage (top-right), proposed array stacking and swapping technique (bottom).

DIBL effect. Figure 5.5 shows that lowering the array voltage to 0.3V, which is half of our system VDD, reduces leakage by 11×. However, this raises two issues: 1) Commercial bitcell sizing is not optimized for holding data at very low voltages (e.g., subthreshold regime) and requires a careful hold margin / density tradeoff analysis; 2) Additional voltage regulation and level conversion is required to generate and interface with the separate voltage level of the SRAM array. Conventionally an LDO is used, incurring area overhead and extra power consumption due to efficiency loss. Voltage stacking is an alternative way to generate an intermediate voltage level by placing voltage domains in series, which has been shown to provide substantial benefits in power delivery for microprocessors [87] and high bandwidth data buses [88]. The biggest challenge in voltage stacking is balancing the active current between top and bottom levels and maintaining a stable mid-rail voltage level. This often requires an additional Voltage Regulator, negating some of the benefits under different top/bottom load conditions. However, we observe that SRAM arrays present a near-

ideal load for voltage stacking in that they draw mainly leakage current, and hence total current drawn does not change dramatically with circuit activity (writing a bit draws 10s of pA average current, which is negligible compared to µA-level background leakage current).

To allow access to arbitrary arrays during operation while avoiding insertion of complex level conversion, we propose a novel array swap mechanism. As shown in Figure 5.5, the SRAM peripherals are not stacked and therefore wordline and bitline voltages remain at VDDcore ( 2VDDmid) for faster operation speed, inherent write/read noise margin enhancement, and removing the need for level converters. Only bottom arrays are read/written directly. When an access is required to an array located in the top voltage domain, the memory controller first swaps a bottom array in the same quad-array SRAM bank with the desired top array (Figure 5.7). This swap mechanism ensures the leakage current remains balanced and can be completed in one system clock cycle due to the relatively low IoT processer clock frequency. In addition to leakage reduction from reduced supply voltage, the approach offers an additional 2× leakage reduction in top arrays due to their inherent reverse body bias and reduced bitline leakage effects. As a result, total leakage is minimized by increasing the % of top arrays to greater than half (i.e., to 75%); this is analyzed in measurement later and the optimal ratio can be set by a memory controller.

Figure 5.6 shows the bitcell schematic and layout. The cross-coupled 4T uses HVT devices to minimize hold leakage while LVT devices in the read port provide faster sensing speed. The bitcell is upsized for improved hold noise margin (HNM). Channel length is increased to the point where leakage is minimum, also improving HNM while incurring 8% cell density loss. Channel width is increased, initially improving HNM faster than leakage power, providing a favorable tradeoff. The final sizes are chosen to balance among density, HNM, and leakage. To decouple the read/write operation, we use a Z8T structure [77] instead of a traditional 8T, as differential sensing provides

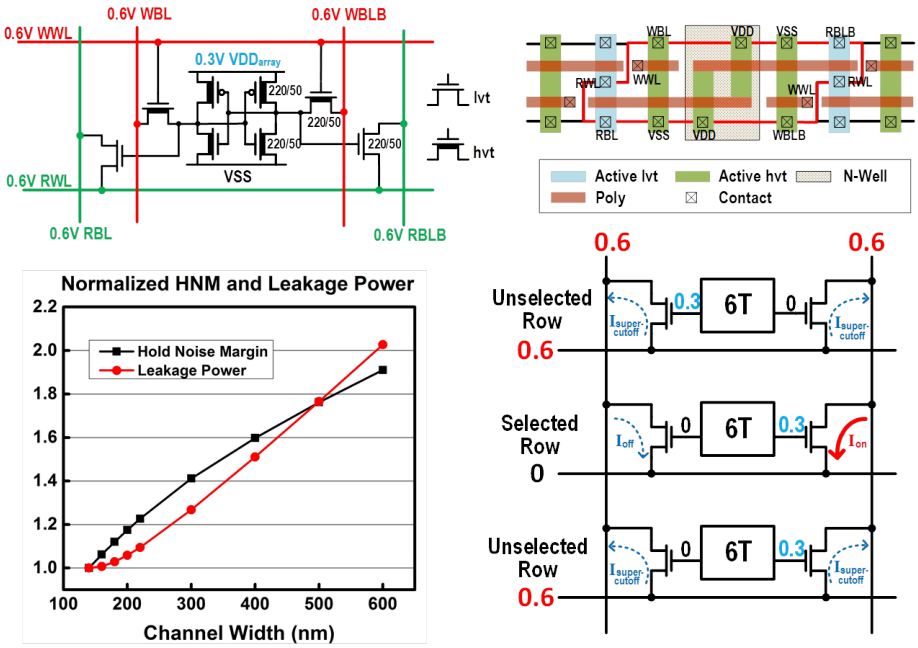



Figure 5.6: Bitcell schematic and layout (top), hold noise margin (HNM) and leakage versus bitcell sizing (bottom-left), currents on the bitline during read operation (bottom-right).

faster read speed and larger sensing margin. Since in our stacked SRAM the array voltage is around 1/2 the bitline voltage, it inherently avoids the clamping current problem in the original Z8T as all unselected cells are super-cutoff with negative VGS. Further, the write noise margin is greatly increased due to the word-line overdrive of the stacked configuration.

Each SRAM bank has 4 arrays with power switches that connect an array to either top or bottom voltage domains (Figure 5.7). The power switch settings are retained in latches under an always-on voltage domain. Each bank can have 0-3 top arrays but at least one array must be in the bottom domain. When accessing a top array, the SRAM controller swaps this array with a bottom array in the same bank in two steps: First, the two arrays (target and swapping) are expanded to full voltage (0:0.6V), after which they are collapsed to the appropriate half range. Since the two arrays are physically close, local charge sharing minimizes the disturbance to the mid-rail. All on-chip SRAM arrays in the system are connected to the same power/ground/mid-

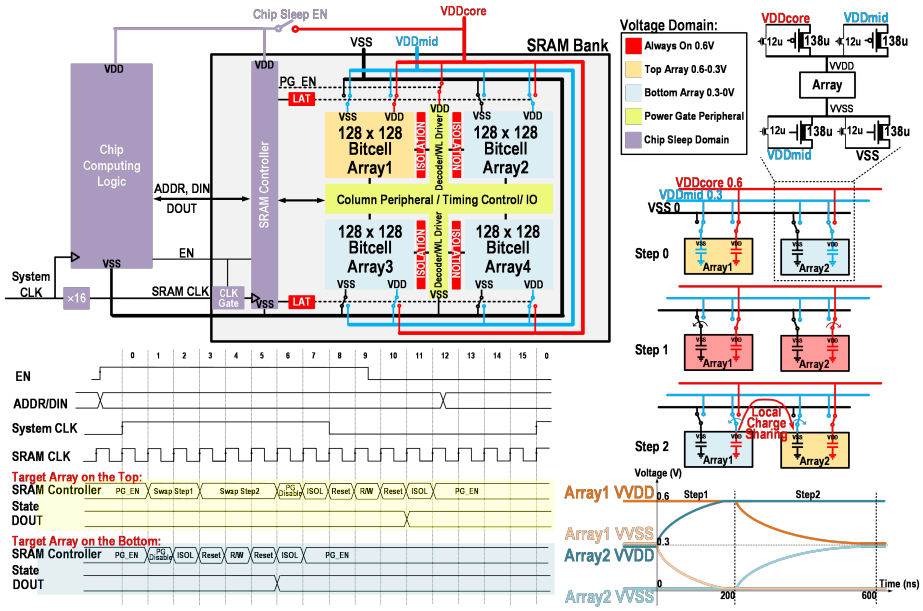



Figure 5.7: SRAM bank architecture and timing diagram (left), array swapping algorithm (right).

rail, resulting in a large amount of innate decoupling capacitance and background load current to suppress transient noise. To smooth transitions and reduce coupling noise, each power switch consists of both small and large headers/footers that are turned on in sequence. Each swap consumes around 8pJ, which is comparable to a single 128-bit read. To minimize the frequency of swaps, instruction memories (exhibiting mainly random accesses) are placed in the bottom domain, whereas neural engine memories with mostly sequential access patterns are primarily placed in the top domain. SRAM peripherals are power gated immediately after each access to reduce leakage.

The proposed stack SRAM approach was implemented in a 40nm CMOS image processing IoT chip with 8.9Mb memories. Figure 5.8 shows measured leakage across voltage and temperature. As the number of top arrays increases, the mid-rail voltage raises while the leakage keeps decreasing. Figure 5.9 shows excellent mid-rail voltage stability; VDDmid varies only ±16mV across 100°C, drops at most 1.74mV when arrays swap every 11 cycles, and is unaffected by read/write every cycle at full speed. It achieves 438kHz frequency at 0.7V (enabling 14fps in the supported image process-

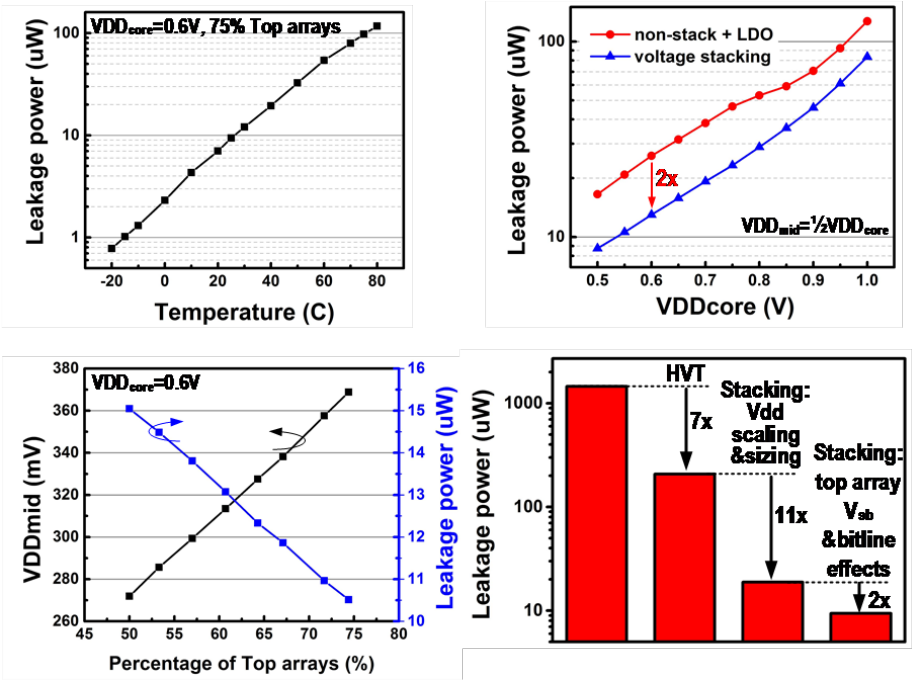



Figure 5.8: Leakage across temperature and voltage (top), Mid-rail voltage and leakage with temperature (bottom-left), leakage reduction effects. (bottom-right).

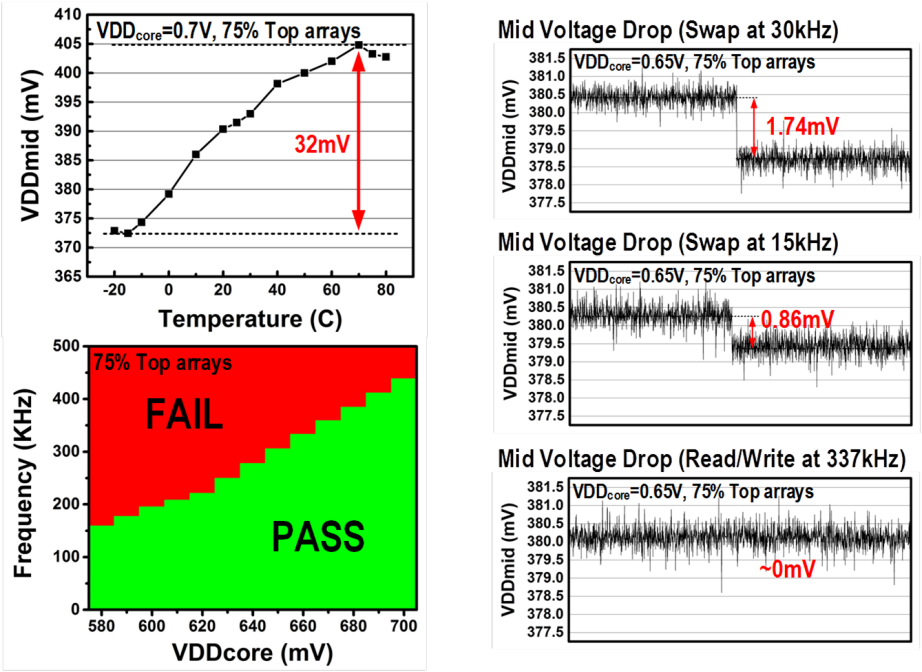



Figure 5.9: Mid-rail variation with temperature (top-left), voltage drop due to various memory activities (right), shmoo plot (bottom-left).

ing system) and 67fJ/bit access energy at 0.6V. Figure 5.10 compares this work to other state-of-the-art low leakage SRAMs. The proposed work achieves low leakage of 1.03pW/bit at 0.58V without extra supply levels or body bias voltage generation.

| | This work | ISSCC10 [2] | ISSCC14 [3] | VLSI13 [4] | VLSI17 [5] | JSSC09 [8] | JSSC08 [10] |
|---|---|---|---|---|---|---|---|
| Process | 40nm | 180nm | 65nm Thick Oxide | 28nm FDSOI | 65nm SOTB | 130nm | 65nm |
| Cell type | 8T | 10T | 6T | 6T | 6T | 8T | 8T |
| Cell Area ($um^2$) | 0.82* | 17.48 | 2.159 | 0.12 | 0.5408 | 0.61 | - |
| On-chip SRAM Capacity | 8.91Mb | 24kb | 128kb | 1Mb | 8Mb | 64kb | 256kb |
| Leakage | 1.03pW/bit (0.58V) | 3.3fW/bit (0.4V) | 32.4fW/bit (1.2V) | 300fW/bit (0.6V) | 13.7fW/bit (0.5V) | 70pW/bit (0.23V) | 6.45pW/bit (0.3V) |
| Extra Supply Level Required | No | Yes | No | Yes | No | Yes | No |
| Body Bias Voltage Required | No | No | No | -1.5V | -2V | No | No |
| Access time | 143ns (0.7V) | 13700ns (0.4V) | 7ns (1.2V) | - | 31.4ns (0.75V) | 66.7ns (0.6V) | 1250ns (0.6V) |
| Access Energy | 67fJ/bit (0.6V) | - | 195fJ/bit (1.2V) | - | 224fJ/bit (0.75V) | 25fJ/bit (0.4V) | - |
| *Logic design rule | | | | | | | |
| | JSSCC11 [13] | VLSI17 [14] | JSSC11 [15] | JSSC13 [16] | ISSCC13 [17] | ISSCC14 [18] | VLSI13 [19] |
| Process | 90nm | 55nm | 40nm | 28nm | 28nm | 28nm | 20nm |
| Cell type | 8T | 6T | 9T | 6T | 6T | 6T | 6T |
| Cell Area ($um^2$) | - | 0.803 | 1.058 | - | 0.12 | - | - |
| On-chip SRAM Capacity | 64kb | 16kb | 8kb | 512kb | 2Mb | 32kb | 128kb |
| Leakage | 305pW/bit (0.23V) | 97.6pW/bit (0.2V) | 6.25pW/bit (0.4V) | 10.6pW/bit (0.8V) | 3.5pW/bit (0.7V) | 109pW/bit (0.33V) | 33.6pW/bit (0.6V) |
| Extra Supply Level Required | No | Yes | No | No | Yes | No | No |
| Body Bias Voltage Required | No | No | No | No | No | No | No |
| Access time | 111ns (0.3V) | 250ns (0.25V) | 1000ns (0.4V) | 0.42ns (1V) | - | - | 1.16ns (0.9V) |
| Access Energy | 1.15pJ/bit (0.5V) | 5fJ/bit (0.25V) | 11.3fJ/bit (0.4V) | - | - | 93fJ/bit (0.6V) | 69fJ/bit (0.6V) |

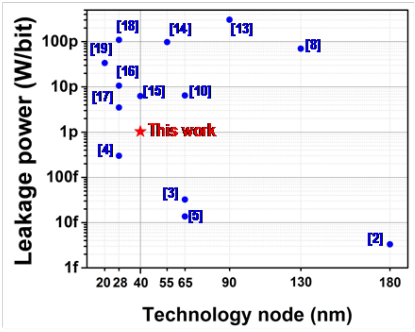
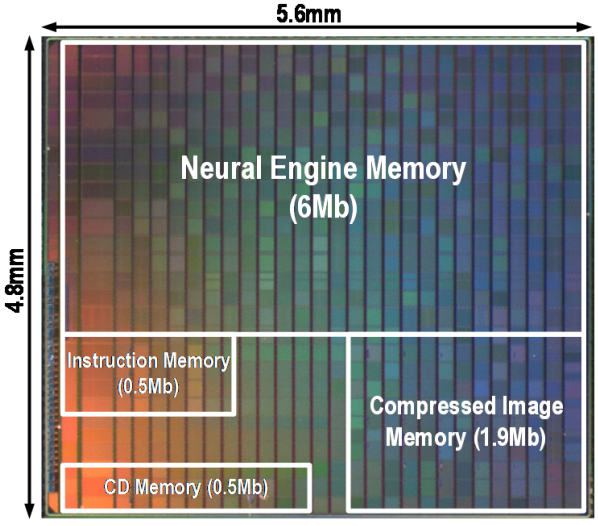



Figure 5.10: Comparison table and design space landscape.




| Technology | 40nm |
|---|---|
| Chip size | 4.8mm x 5.6mm |
| On-chip SRAM | 8.9Mbit |
| Array Size | 128 x 128 x 4 |
| Bank Area | 163um x 618um |
| Bitcell Size | 0.82um² |
| VDDcore | 0.58V ~ 0.7V |
| Leakage | 1.03pW/bit (0.58V) |
| Access time | 143ns (0.7V) |
| Access Energy | 67fJ/bit (0.6V) |

Figure 5.11: Die photograph in 40nm CMOS.

# BIBLIOGRAPHY

# BIBLIOGRAPHY


[1] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[2] Gordon Bell. Bell's law for the birth and death of computer classes. *Commun. ACM*, 51(1):86–94, January 2008.

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

[4] ITRS. 2015 international technology roadmap for semiconductors. http://www.itrs2.net/itrs-reports.html. Accessed: July 1, 2019.

[5] Y. Lee, D. Sylvester, and D. Blaauw. Circuits for ultra-low power millimeter-scale sensor nodes. In *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 752–756, Nov 2012.

[6] John Launchbury. A darpa perspective on artificial intelligence. ://www.darpa.mil/attachments/AIFull.pdf.

[7] TechInsights. Apple iphone xs max teardown. `https://www.techinsights.com/blog/apple-iphone-xs-max-teardown`. Accessed: July 1, 2019.

[8] Google. Edge tpu google's purpose-built asic designed to run inference at the edge. `https://cloud.google.com/edge-tpu/`. Accessed: July 1, 2019.

[9] I. Hong, K. Bong, D. Shin, S. Park, K. Lee, Y. Kim, and H. Yoo. 18.1 a 2.71nj/pixel 3d-stacked gaze-activated object-recognition system for low-power mobile hmd applications. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, pages 1–3, Feb 2015.

[10] G. E. Moore. No exponential is forever: but "forever" can be delayed! [semiconductor industry]. In *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*, pages 20–23 vol.1, Feb 2003.

[11] J. Kil, J. Gu, and C. H. Kim. A high-speed variation-tolerant interconnect technique for sub-threshold circuits using capacitive boosting. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(4):456–465, April 2008.

[12] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco. Gpus and the future of parallel computing. *IEEE Micro*, 31(5):7–17, Sep. 2011.

[13] S. Rusu, S. Tam, H. Muljono, J. Stinson, D. Ayers, J. Chang, R. Varada, M. Ratta, S. Kottapalli, and S. Vora. A 45 nm 8-core enterprise xeon¯ processor. *IEEE Journal of Solid-State Circuits*, 45(1):7–14, Jan 2010.

[14] M. Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14, Feb 2014.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[16] G. Chen, C. Parada, and G. Heigold. Small-footprint keyword spotting using deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4087–4091, May 2014.

[17] F. Su, W. Chen, L. Xia, C. Lo, T. Tang, Z. Wang, K. Hsu, M. Cheng, J. Li, Y. Xie, Y. Wang, M. Chang, H. Yang, and Y. Liu. A 462gops/j rram-based nonvolatile intelligent processor for energy harvesting ioe system featuring nonvolatile logics and processing-in-memory. In *2017 Symposium on VLSI Technology*, pages T260–T261, June 2017.

[18] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: Efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254, June 2016.

[19] T. Mizuno, J. Okumtura, and A. Toriumi. Experimental study of threshold voltage fluctuation due to statistical variation of channel dopant number in mosfet's. *IEEE Transactions on Electron Devices*, 41(11):2216–2221, Nov 1994.

[20] M. Hane, T. Ikezawa, and T. Ezaki. Atomistic 3d process/device simulation considering gate line-edge roughness and poly-si random crystal orientation effects [mosfets]. In *IEEE International Electron Devices Meeting 2003*, pages 9.5.1–9.5.4, Dec 2003.

[21] J. Wang, N. Pinckney, D. Blaauw, and D. Sylvester. Reconfigurable self-timed regenerators for wide-range voltage scaled interconnect. In *2015 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pages 1–4, Nov 2015.

[22] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 383–396, June 2018.

[23] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester. 14.2 a compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 224–226, Feb 2019.

[24] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, D. Sylvester, D. Blaauw, R. Das, and R. Iyer. Neural cache: Bit-serial in-cache acceleration of deep neural networks. *IEEE Micro*, 39(3):11–19, May 2019.

[25] M. Shah, J. Wang, D. Blaauw, D. Sylvester, H. Kim, and C. Chakrabarti. A fixed-point neural network for keyword detection on resource constrained hardware. In *2015 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Oct 2015.

[26] S. Bang, J. Wang, Z. Li, C. Gao, Y. Kim, Q. Dong, Y. Chen, L. Fick, X. Sun, R. Dreslinski, T. Mudge, H. S. Kim, D. Blaauw, and D. Sylvester. 14.7 a 288µw programmable deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 250–251, Feb 2017.

[27] Mohit Shah, Sairam Arunachalam, Jingcheng Wang, David Blaauw, Dennis Sylvester, Hun-Seok Kim, Jae-Sun Seo, and Chaitali Chakrabarti. A fixed-point neural network architecture for speech applications on resource constrained hardware. *J. Signal Process. Syst.*, 90(5):727–741, May 2018.

[28] K. D. Choo, L. Xu, Y. Kim, J. Seol, X. Wu, D. Sylvester, and D. Blaauw. 5.2 energy-efficient low-noise cmos image sensor with capacitor array-assisted charge-injection sar adc for motion-triggered low-power iot applications. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 96–98, Feb 2019.

[29] 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.

[30] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, A. Kumar, S. K. Gb, R. Ramanarayanan, V. Erraguntla, J. Howard, S. Vangal, S. Dighe, G. Ruhl, P. Aseron, H. Wilson, N. Borkar, V. De, and S. Borkar. A 280mv-to-1.2v wide-operating-range ia-32 processor in 32nm cmos. In *2012 IEEE International Solid-State Circuits Conference*, pages 66–68, Feb 2012.

[31] S. K. Hsu, A. Agarwal, M. A. Anders, S. K. Mathew, H. Kaul, F. Sheikh, and R. K. Krishnamurthy. A 280 mv-to-1.1 v 256b reconfigurable simd vector permutation engine with 2-dimensional shuffle in 22 nm tri-gate cmos. *IEEE Journal of Solid-State Circuits*, 48(1):118–127, Jan 2013.

[32] G. Chen, M. A. Anders, H. Kaul, S. K. Satpathy, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, S. Borkar, and V. De. 16.1 a 340mv-to-0.9v

20.2tb/s source-synchronous hybrid packet/circuit-switched 16×16 network-on-chip in 22nm tri-gate cmos. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 276–277, Feb 2014.

[33] M. Chen, L. Chen, M. Chang, S. Yang, Y. Kuo, J. Wu, M. Ho, H. Su, Y. Chu, W. Wu, T. Yang, and H. Yamauchi. A 260mv l-shaped 7t sram with bit-line (bl) swing expansion schemes based on boosted bl, asymmetric-vthread-port, and offset cell vdd biasing techniques. In *2012 Symposium on VLSI Circuits (VLSIC)*, pages 112–113, June 2012.

[34] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley Publishing Company, USA, 4th edition, 2010.

[35] P. Singh, J. Seo, D. Blaauw, and D. Sylvester. Self-timed regenerators for high-speed and low-power on-chip global interconnect. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(6):673–677, June 2008.

[36] A. Nalamalpu, S. Srinivasan, and W. P. Burleson. Boosters for driving long onchip interconnects - design issues, interconnect synthesis, and comparison with repeaters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(1):50–62, Jan 2002.

[37] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: Implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, March 1995.

[38] M. Huang, M. Mehalel, R. Arvapalli, and S. He. An energy efficient 32-nm 20-mb shared on-die l3 cache for intel® xeon® processor e5 family. *IEEE Journal of Solid-State Circuits*, 48(8):1954–1962, Aug 2013.

[39] D. G. Elliott, W. M. Snelgrove, and M. Stumm. Computational ram: A memory-simd hybrid and its application to dsp. In *1992 Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 30.6.1–30.6.4, May 1992.

[40] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, June 2017.

[41] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[43] Z. Li, J. Wang, D. Sylvester, D. Blaauw, and H. Kim. A1920 × 1080 25fps, 2.4tops/w unified optical flow and depth 6d vision processor for energy-efficient, low power autonomous navigation. In *2018 IEEE Symposium on VLSI Circuits*, pages 135–136, June 2018.

[44] S. Smets, T. Goedemé, A. Mittal, and M. Verhelst. 2.2 a 978gops/w flexible streaming processor for real-time image processing applications in 22nm fdsoi. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 44–46, Feb 2019.

[45] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 288–301, Oct 2017.

[46] M. N. Bojnordi and E. Ipek. Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–13, March 2016.

[47] W. Chen, K. Li, W. Lin, K. Hsu, P. Li, C. Yang, C. Xue, E. Yang, Y. Chen, Y. Chang, T. Hsu, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang. A 65nm 1mb nonvolatile computing-in-memory reram macro with sub-16ns multiply-and-accumulate for binary dnn ai edge processors. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 494–496, Feb 2018.

[48] L. Fick, D. Blaauw, D. Sylvester, S. Skrzyniarz, M. Parikh, and D. Fick. Analog in-memory subthreshold deep neural network accelerator. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, April 2017.

[49] J. Zhang, Z. Wang, and N. Verma. In-memory computation of a machine-learning classifier in a standard 6t sram array. *IEEE Journal of Solid-State Circuits*, 52(4):915–924, April 2017.

[50] A. Biswas and A. P. Chandrakasan. Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 488–490, Feb 2018.

[51] S. K. Gonugondla, M. Kang, and N. Shanbhag. A 42pj/decision 3.12tops/w robust in-memory machine learning classifier with on-chip training. In *2018*

*IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 490–492, Feb 2018.

[52] W. Khwa, J. Chen, J. Li, X. Si, E. Yang, X. Sun, R. Liu, P. Chen, Q. Li, S. Yu, and M. Chang. A 65nm 4kb algorithm-dependent computing-in-memory sram unit-macro with 2.3ns and 55.8tops/w fully parallel product-sum operation for binary dnn edge processors. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 496–498, Feb 2018.

[53] Y. Zhang, L. Xu, Q. Dong, J. Wang, D. Blaauw, and D. Sylvester. Recryptor: A reconfigurable cryptographic cortex-m0 processor with in-memory and near-memory computing for iot security. *IEEE Journal of Solid-State Circuits*, 53(4):995–1005, April 2018.

[54] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester. A 0.3v vddmin 4+2t sram for searching and in-memory computing using 55nm ddc technology. In *2017 Symposium on VLSI Circuits*, pages C160–C161, June 2017.

[55] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw. A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory. *IEEE Journal of Solid-State Circuits*, 51(4):1009–1021, April 2016.

[56] J. Wang, X. Wang, C. Eckert, A. Subramaniyan, R. Das, D. Blaauw, and D. Sylvester. 14.2 a compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 224–226, Feb 2019.

[57] K. E. Batcher. Bit-serial parallel processing systems. *IEEE Transactions on Computers*, C-31(5):377–384, May 1982.

[58] P. B. Denyer and David Renshaw. *VLSI Signal Processing; A Bit-Serial Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1985.

[59] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo. Unpu: A 50.6tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 218–220, Feb 2018.

[60] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura. Quest: A 7.49tops multi-purpose log-quantized dnn inference engine stacked on 96mb 3d sram using inductive-coupling technology in 40nm cmos. In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pages 216–218, Feb 2018.

[61] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das. Neural cache: Bit-serial in-cache acceleration of deep neural networks. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 383–396, June 2018.

[62] J. Seo, B. Brezzo, Y. Liu, B. D. Parker, S. K. Esser, R. K. Montoye, B. Rajendran, J. A. Tierno, L. Chang, D. S. Modha, and D. J. Friedman. A 45nm cmos neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *2011 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–4, Sep. 2011.

[63] ARM Inc. Arm cortex-m series. `http://www.arm.com/products/processors/cortex-m`. Accessed: Oct. 2017.

[64] Google Inc. Arm cortex-m series. `https://code.google.com/archive/p/cuda-convnet/`. Accessed: Sept. 2018.

[65] Nicholas D. Lane and Petko Georgiev. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 117–122, New York, NY, USA, 2015. ACM.

[66] G. Kim, Y. Lee, Zhiyoong Foo, P. Pannuto, Ye-Sheng Kuo, B. Kempke, M. H. Ghaed, Suyoung Bang, Inhee Lee, Yejoong Kim, Seokhyeon Jeong, P. Dutta, D. Sylvester, and D. Blaauw. A millimeter-scale wireless imaging system with continuous motion detection and energy harvesting. In *2014 Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2, June 2014.

[67] Y. Lee, G. Kim, S. Bang, Y. Kim, I. Lee, P. Dutta, D. Sylvester, and D. Blaauw. A modular 1mm3die-stacked sensing platform with optical communication and multi-modal energy harvesting. In *2012 IEEE International Solid-State Circuits Conference*, pages 402–404, Feb 2012.

[68] Y. Chen, T. Krishna, J. Emer, and V. Sze. 14.5 eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 262–263, Jan 2016.

[69] B. Moons and M. Verhelst. A 0.3–2.6 tops/w precision-scalable processor for real-time large-scale convnets. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*, pages 1–2, June 2016.

[70] Patrick Judd, Jorge Albericio, Tayler H. Hetherington, Tor M. Aamodt, Natalie D. Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-precision strategies for bounded memory in deep neural nets. *CoRR*, abs/1511.05236, 2015.

[71] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, Oct 2016.

[72] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan. Axnn: Energy-efficient neuromorphic systems using approximate computing. In *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 27–32, Aug 2014.

[73] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 246–247, Feb 2017.

[74] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2008.

[75] L. Chang, D. M. Fried, J. Hergenrother, J. W. Sleight, R. H. Dennard, R. K. Montoye, L. Sekaric, S. J. McNab, A. W. Topol, C. D. Adams, K. W. Guarini, and W. Haensch. Stable sram cell design for the 32 nm node and beyond. In *Digest of Technical Papers. 2005 Symposium on VLSI Technology, 2005.*, pages 128–129, June 2005.

[76] L. Chang, Y. Nakamura, R. K. Montoye, J. Sawada, A. K. Martin, K. Kinoshita, F. H. Gebara, K. B. Agarwal, D. J. Acharyya, W. Haensch, K. Hosokawa, and D. Jamsek. A 5.3ghz 8t-sram with operation down to 0.41v in 65nm cmos. In *2007 IEEE Symposium on VLSI Circuits*, pages 252–253, June 2007.

[77] J. Wu, Y. Chen, M. Chang, P. Chou, C. Chen, H. Liao, M. Chen, Y. Chu, W. Wu, and H. Yamauchi. A large$\sigma v_{th}$/vdd tolerant zigzag 8t sram with area-efficient decoupled differential sensing and fast write-back scheme. *IEEE Journal of Solid-State Circuits*, 46(4):815–827, April 2011.

[78] G. Chen, M. Fojtik, D. Kim, D. Fick, J. Park, M. Seok, M. Chen, Z. Foo, D. Sylvester, and D. Blaauw. Millimeter-scale nearly perpetual sensor system with stacked battery and solar cells. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 288–289, Feb 2010.

[79] T. Fukuda, K. Kohara, T. Dozaka, Y. Takeyama, T. Midorikawa, K. Hashimoto, I. Wakiyama, S. Miyano, and T. Hojo. 13.4 a 7ns-access-time 25µw/mhz 128kb sram for low-power fast wake-up mcu in 65nm cmos with 27fa/b retention current. In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 236–237, Feb 2014.

[80] R. Ranica, N. Planes, O. Weber, O. Thomas, S. Haendler, D. Noblet, D. Croain, C. Gardin, and F. Arnaud. Fdsoi process/design full solutions for ultra low leakage, high speed and low voltage srams. In *2013 Symposium on VLSI Technology*, pages T210–T211, June 2013.

[81] M. Yabuuchi, K. Nii, S. Tanaka, Y. Shinozaki, Y. Yamamoto, T. Hasegawa, H. Shinkawata, and S. Kamohara. A 65 nm 1.0 v 1.84 ns silicon-on-thin-box (sotb) embedded sram with 13.72 nw/mbit standby power for smart iot. In *2017 Symposium on VLSI Technology*, pages C220–C221, June 2017.

[82] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang, and M. Bohr. A 4.0 ghz 291mb voltage-scalable sram design in 32nm

high-$\varkappa$ metal-gate cmos with integrated power management. In *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 456–457,457a, Feb 2009.

[83] J. Chang, M. Huang, J. Shoemaker, J. Benoit, S. Chen, W. Chen, S. Chiu, R. Ganesan, G. Leong, V. Lukka, S. Rusu, and D. Srivastava. The 65-nm 16-mb shared on-die l3 cache for the dual-core intel xeon processor 7100 series. *IEEE Journal of Solid-State Circuits*, 42(4):846–852, April 2007.

[84] T. Kim, J. Liu, and C. H. Kim. A voltage scalable 0.26v, 64kb 8t sram with vmin lowering techniques and deep sleep mode. In *2008 IEEE Custom Integrated Circuits Conference*, pages 407–410, Sep. 2008.

[85] F. Hamzaoglu, K. Zhang, Y. Wang, H. J. Ahn, U. Bhattacharya, Z. Chen, Y. Ng, A. Pavlov, K. Smits, and M. Bohr. A 153mb-sram design with dynamic stability enhancement and leakage reduction in 45nm high-K metal-gate cmos technology. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 376–621, Feb 2008.

[86] N. Verma and A. P. Chandrakasan. A 256 kb 65 nm 8t subthreshold sram employing sense-amplifier redundancy. *IEEE Journal of Solid-State Circuits*, 43(1):141–149, Jan 2008.

[87] K. Blutman, A. Kapoor, A. Majumdar, J. G. Martinez, J. Echeverri, L. Sevat, A. P. van der Wel, H. Fatemi, K. A. A. Makinwa, and J. P. de Gyvez. A low-power microcontroller in a 40-nm cmos using charge recycling. *IEEE Journal of Solid-State Circuits*, 52(4):950–960, April 2017.

[88] J. M. Wilson, M. R. Fojtik, J. W. Poulton, X. Chen, S. G. Tell, T. H. Greer, C. T. Gray, and W. J. Dally. 8.6 a 6.5-to-23.3fj/b/mm balanced charge-recycling bus in 16nm finfet cmos at 1.7-to-2.6gb/s/wire with clock forwarding and low-crosstalk contraflow wiring. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 156–157, Jan 2016.