# LOW-POWER EMBEDDED DESIGN SOLUTIONS AND LOW-LATENCY ON-CHIP INTERCONNECT ARCHITECTURE FOR SYSTEM-ON-CHIP DESIGN

A Dissertation

by

YOON SEOK YANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2012

Major Subject: Computer Engineering

# LOW-POWER EMBEDDED DESIGN SOLUTIONS AND LOW-LATENCY ON-CHIP INTERCONNECT ARCHITECTURE FOR SYSTEM-ON-CHIP DESIGN

A Dissertation

by

YOON SEOK YANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

| | |
|---|---|
| Chair of Committee, | Gwan S. Choi |
| Committee Members, | Paul V. Gratz |
| | Laszlo B. Kish |
| | Vivek Sarin |
| Head of Department, | Costas N. Georphiades |

August 2012

Major Subject: Computer Engineering

## ABSTRACT

Low-Power Embedded Design Solutions and Low-Latency On-Chip Interconnect Architecture for System-On-Chip Design. (August 2012)

Yoon Seok Yang, B.S., Hanyang University;

M.S., University of California, Irvine

Chair of Advisory Committee: Dr. Gwan S. Choi

This dissertation presents three design solutions to support several key system-on-chip (SoC) issues to achieve low-power and high performance. These are: 1) joint source and channel decoding (JSCD) schemes for low-power SoCs used in portable multimedia systems, 2) efficient on-chip interconnect architecture for massive multimedia data streaming on multiprocessor SoCs (MPSoCs), and 3) data processing architecture for low-power SoCs in distributed sensor network (DSS) systems and its implementation.

The first part includes a low-power embedded low density parity check code (LDPC)-H.264 joint decoding architecture to lower the baseband energy consumption of a channel decoder using joint source decoding and dynamic voltage and frequency scaling (DVFS). A low-power multiple-input multiple-output (MIMO) and H.264 video joint detector/decoder design that minimizes energy for portable, wireless embedded systems is also designed.

In the second part, a link-level quality of service (QoS) scheme using unequal error protection (UEP) for low-power network-on-chip (NoC) and low latency on-chip network designs for MPSoCs is proposed. This part contains WaveSync, a low-latency focused network-on-chip architecture for globally-asynchronous locally-synchronous (GALS) designs and a simultaneous dual-path routing (SDPR) scheme utilizing path diversity present in typical mesh topology network-on-chips. SDPR is akin to having a higher link width but without the significant hardware overhead associated with simple bus width scaling.

The last part shows data processing unit designs for embedded SoCs. We propose a data processing and control logic design for a new radiation detection sensor system generating data at or above Peta-bits-per-second level. Implementation results show that the intended clock rate is achieved within the power target of less than 200mW. We also present a digital signal processing (DSP) accelerator supporting configurable MAC, FFT, FIR, and 3-D cross product operations for embedded SoCs. It consumes 12.35mW along with 0.167mm$^2$ area at 333MHz.

To my family

## ACKNOWLEDGMENTS

I would like to first and foremost thank my advisor Dr. Gwan S. Choi for his advice and direction in my PhD research. I sincerely appreciate his consistent encourage and support during the period of my studies. I also thank my committee members including Dr. Paul V. Gratz, Dr. Laszlo B. Kish and Dr. Vivek Sarin for their valuable suggestions, comments, and advices on all aspects of my research. My thanks also go to other students in Dr. Choi's group including Pankaj Bhagawat, Reeshav Kumar, Hrishikesh Deshpande, Ehsan Rohani, Jingwei Xu, and Wil Bassett for their valuable discussion on research with me and help during my Ph.D. study.

I thank my friends in College Station: Kyu-Nam Shim, Yun-Bum Jung, Yong-Ho Lee, Yong-Tae Kim, and Jung-Kyu Lee and their family for giving me countless assistant during my Ph.D. study and making my life happy and joyful. I also thank MaGee's family (Russell, Karmen, Molly, Besty, Hayley and Matthew) for supporting my life in College Station. They have supported me a lot to make my stay here worthwhile. I thank my Bible study members, Thomas, Rachel, Derick, and Callie, too.

I would like to give heartfelt thanks to my parents, sisters and brothers in low for their selfless love, trust, and support. I thank my wife, Won-Hee Kim, with all my heart for her endless love, support, patience and encouragement.

# TABLE OF CONTENTS

# LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

# CHAPTER I

# INTRODUCTION

Today's high-performance and multiprocessor SoCs (MPSoCs) incorporate hundreds of IP blocks. With this trend, current SoC design and its applications are migrating from single processor-based computation model to communication intensive multiprocessing. Future multimedia and other data intensive applications require massive computing power possibly tractable through low latency on-chip network and numerous application specific IPs or processing units. We envision that future SoCs will be composed of a mixture of heterogeneous IPs and communication-centric architecture. Energy and performance issues will be the focal points for such SoC designs. This thesis presents three solutions to address emerging SoC design with specific emphasis on low-power and high performance. These are: 1) joint source and channel decoding (JSCD) schemes for portable multimedia applications, 2) efficient on-chip interconnect architecture for massive multimedia data streaming on MPSoCs, and 3) data processing architecture for low-power SoCs in distributed sensor network (DSS) systems and its implementation.

The first contribution of this dissertation includes a low-power embedded low density parity check code (LDPC)-H.264 joint source-channel decoding (JSCD) architecture to lower the baseband energy consumption of a channel decoder using joint source decoding and dynamic voltage and frequency scaling (DVFS). With the continuous increase in the capabilities of portable multimedia devices and services, the demand to improve the energy efficiency and error robustness motivates the interest in joint source-channel decoding with unequal error protection (UEP). We propose a configuration search scheme based on UEP to trade-off power and performance on power sensitive mobile devices. We also presents a

This dissertation follows the style of *IEEE Transactions on Computers.*

low-power multiple-input multiple-output (MIMO) and H.264 video joint detector/decoder design that minimizes energy for portable, wireless embedded systems.

The second contribution contains a link-level quality of service (QoS) scheme using UEP for low-power network-on-chip (NoC) and low latency on-chip network designs for MPSoCs. VLSI process technology scaling has provided ever more transistors with both higher performance and lower power consumption. As VLSI technology moves forward, however, many positive VLSI scaling trends are being replaced with negative trends [1]. Transistor leakage, leading to greater power consumption, has forced a practical plateau in VLSI clock frequency based performance gains. Instead, increasing transistor density is yielding performance gains solely through growing the number of cores integrated on a single chip, placing a great burden on the communication between those cores. Therefore, on-chip networks have become more widely accepted for the communication of many cores on a chip such as RAW, TRIPS, Teraflop, and Tilera [2–5]. Dally et al. advocated routing packets not wires since packet switching can surmount many of difficulties in on-chip communications [6]. Our contribution in this work also contains two novel NoC designs, WaveSync, an low-latency focused network-on-chip architecture for globally-asynchronous locally-synchronous (GALS) designs and a simultaneous dual-path routing (SDPR) scheme utilizing path diversity present in typical mesh topology network-on-chips. SDPR is akin to having a higher link width but without the significant hardware overhead associated with simple bus width scaling.

The last contribution of this thesis is data processing accelerator designs for embedded SoCs. Next generation SoCs will include sensors for detection, acceleration, momentum, location, heading, temperature, pressure, sound and light. These sensing and data processing components can be designed by micro-controller, dedicated hardware, or digital signal processor. However, digital signal processing (DSP) or micro-controller units have limits on performance and power necessary for processing input data captured by multiple sensors on a sensor network. We need a dedicated hardware or accelerator to overcome this limits. This thesis aims to provide a solution for a data processing and control logic design that initiates a new sensor SoC system for radiation detection generating data at or above Peta-bits-per-second level. We also present a DSP accelerator supporting multiple multiply and accumulates (MACs), FFT, FIR, and 3-D cross product operations used in fundamental signal processing for embedded systems.

## A. Joint Source Channel Decoding Method for Low-Power Portable and Wireless SoC Systems

This section presents three JSCD-based low-power decoding solutions for low-power portable and wireless SoC systems using a novel UEP scheme. A UEP scheme utilizes prioritized data information; the aim is to protect more important data from errors. This is in contrast to an equal error protection (EEP) scheme where all received frames are treated equally. Xiao, Stoufs, Parrein, and Yip et. al. present the efficiency of UEP schemes over traditional EEP schemes on multimedia video transmission [7–10], showing that UEP can significantly outperform EEP on average.

The first and second designs are developed for portable applications over AWGN channels, configured by exploiting importance and error severity in each data frame. These proposed JSCD schemes are based on LDPC and H.264 video decoding schemes. The first JSCD is devised to operate at a fixed frame-decode-time loop regardless of the quality of data received. Within each loop, optimal sub frequencies and voltage levels are dynamically configured to minimize the energy spent for each frame. This design meets the real-time requirements of motion picture reproduction and minimizes overall power consumption. The design is synthesized using TSMC 0.13 micron technology and is capable of jointly decoding QCIF (176x144) video stream at 30 frame per second (FPS) over wireless channel with 80% code rate. As a result, up to 39% power reduction can be achieved in Foreman, Akiyo, and Mobile, when compared to a fixed-iteration-based joint source channel decoder.

The second design presents a low-power design scheme to lower baseband energy consumption using JSCD and DVFS. The aim of this work is to find a near-optimal configuration search algorithm to maximize energy utilization while receiving and reproducing a video stream through LDPC-H.264 joint decoding without significant loss in video quality. UEP is exploited to obtain the optimal number of iterations for each priority type. The decreased number of iterations reduces power consumption with DVFS that scales voltage and frequency [11]. Using the proposed method, we determine LDPC decoding configurations that achieve minimum energy consumption while satisfying pre-specified image quality at the receiver. The implementation results yield 17%, 37%, 52% power reductions with 0, 0.3, 1.1 dB peak signal to noise ratio (PSNR) degradations at 3.6dB SNR in Foreman test stream respectively.

In the third solution, we propose a low-power MIMO and H.264 video joint detector/decoder design that minimizes energy for portable, wireless embedded systems. The design combines the UEP scheme with a variable fidelity of MIMO detection, making performance and energy consumption tradeoffs as required by the importance of each coded video frame. Using search space reduction/truncation strategies in MIMO detection along with H.264 data partitioning (DP) method, we determine the decoding configurations that yield minimum energy consumption for pre-specified image qualities at the receiver. The synthesis result of this scalable MIMO detector using a 45nm technology library yields 52%, 57%, and 58% energy reductions at 24dB SNR in Foreman, Akiyo, and Mobile test streams respectively. The tradeoff is negligible 0.3dB degradation in PSNR.

## B. Low-Latency On-Chip Interconnect Architecture for System-On-Chip Design

Multimedia components on an NoC/SoC often need to communicate with modules that may be placed far away from them on the chip. For instance, channel decoding part may be located on one part of the chip while the DSP processor may be located several hops away from it since design constraints necessitate it be placed close to other modules with which communication takes places more frequently. Signal reliability is a major issue for the transfer of multimedia streams between modules placed several hops apart in NoC settings because transmission of data over several hops is especially vulnerable to single integrity loss and delay uncertainty. For on-chip interconnection, it takes long time to charge/discharge large capacitances and the propagation delay is further deteriorated by the coupling capacitances. While there are several factors contributing to reliability degradation in on-chip interconnection, crosstalk induced errors still constitute a majority. Existing information theoretic models for soft errors grossly underestimate the impact of crosstalk errors in interconnections in advanced technologies. Individual schemes for crosstalk prevention and error correction on bus may not be sufficient to guarantee acceptable error rates for video applications.

In this dissertation, we propose link-level quality-of-service (QoS) using UEP for a low-power on-chip network. We explore the possibility of providing different levels of protection against crosstalk induced errors for different priority video data on NoC links by combining several crosstalk avoidance and error correction schemes to find the combination which provides acceptable performance at the least power expense. In the results, UEP on links using TransSync, TransSync 2 lines and RecSync schemes have been demonstrated for a video decoder on an NoC with H.264 video test streams. For Akiyo test stream transmitted over 3mm long link wires, UEP can lead to as much as 20% of power savings with 3dB of degradation in average PSNR.

We also propose two on-chip interconnect designs for minimum latency NoCs. Although on-chip interconnects or network-on-chips (NoCs) are trivially scalable and provide very high bandwidth, the worst-case, no-load latency to traverse network can be high as well; approaching the access latency of off-chip DRAM for a 64-node, 2D mesh network. The primary cause of this high latency is the traversal of multiple router pipeline stages at each node in the network. Inter- and intra-processor latency has been shown to place direct constraints on system performance, and hence there is a critical need to address interconnect latency for future many-core CMP architectures to be viable [12–14].

The first low-latency on-chip network design is WaveSync, an low-latency focused network-on-chip architecture for globally-asynchronous locally-synchronous (GALS) designs. WaveSync facilitates low-latency communication leveraging the source-synchronous clock sent with the data, to time components in the downstream routers, reducing the number of synchronizations needed. WaveSync accomplishes this by partitioning the router components at each node into different clock-domains, each synchronized with one of the the orthogonal incoming source synchronous clocks in a GALS 2D mesh network. The data and clock subsequently propagate through each node/router, synchronously, until destination is reached regardless of the number of hops it may take. As long as the data travel in the path of clock propagation, and no congestion is encountered, it will be propagated without latching, as if in a long-combinatorial path, with both the clock and the data accruing delay at the same rate. Result is that the need for synchronization between the mesochronous nodes and/or the asynchronous control associated with a typical GALS network is completely eliminated. We also evaluate the performance of a near-half-cycle synchronizer architecture to reduce synchronization latency when synchronization is unavoidable, further reducing per-hop latency. The proposed WaveSync design results in an improvement in average latency of 68% over the baseline GALS and 55% over ABC router across SPLASH-2 benchmark traffic.

In the second on-chip interconnect design, we propose a simultaneous dual-path routing (SDPR) scheme. NoCs have been adopted by emerging multi-core designs as a flexible, scalable, and high power efficient solution. Deterministic routing algorithm such as DOR is widely used in a 2D mesh NoC because it provides simple algorithm and low-cost implementation. However, its performance in latency can be insufficient due to no path diversity. We observe that a significant component of latency in NoCs is due to the serialization of long packets. Increasing the data/link widths across the network may considerably alleviate this problem but is a costly proposition both in terms of device area and of power. Alternatively, we propose a dual-path router architecture that efficiently exploits path diversity to attain low latency without significant hardware overhead. By 1) doubling the number of injection and ejection ports, 2) splitting packets into two halves, 3) recomposing routing policy to support path diversity, and 4) provisioning the network hardware design, we can considerably enhance network resource utilization to achieve much higher performance in latency. The SDPR architecture statically exploits the path diversity in the network to improve link utilization. In particular, the proposed SDPR technique mitigates the lack of path diversity and utilization of DOR by splitting a packet to two halves that involve the same source-destination address and by injecting them simultaneously in parallel via separate and independent orthogonal two paths (i.e. XY and YX). Our experiment results show that the proposed simultaneous dual-path routing (SDPR) scheme outperforms the conventional dimension order routing (DOR) technique across all workloads with 31-40% average latency reduction on long packets running on a 49-core CMP, when compared to the baseline XY DOR router. The fully synthesizable SDPR router occupies 30.89mW power and 0.091mm$^2$ area with 3.7% and 4.7% power and area overheads over the baseline router respectively.

## C. Data Processing Accelerator Architecture for Low-Power SoCs in Distributed Sensor Network Systems

We presents two hardware accelerators for low-power SoCs on a sensor network. The first accelerator shows a data processing and control logic design for a new radiation detection sensor system that can generate data at or above Peta-bits-per-second level. The logic consists of novel data processing components and operation strategies including low-power and network-on-wafer solutions. The aim of this design is to achieve subtle data reduction before the information is ferried to the network, and redundant processing and channels to minimize the loss of information. The result is a radiation detection system that can operate at scan-rate of billion frames per second. Simulation results show that the intended clock rate is achieved within the power target of less than 200mW.

In the second design, we propose a low-power digital signal processing (DSP) accelerator supporting multiple multiply and accumulates (MACs), FFT, FIR, and 3-D cross product operations used in fundamental signal processing for embedded SoCs. Power consumption is a major concern as demands on the application processor to keep up with a constant stream of sensor data diminish opportunities for power conserving sleep. A low-power sensor hub SoC capable of managing the various sensors, aggregating and filtering sensor signals, and notifying the application processor of significant events is needed. Digital signal processors are widely used to support this data processing for sensor hubs due to the flexible programmable ability and powerful data processing, but power consumption of DSPs are relatively higher than dedicated hardware accelerators. This paper presents a low-power DSP accelerator design satisfying both requirements of data processing ability and low-power consumption for sensor hub SoCs. In the evaluation of the proposed design, the DSP accelerator synthesized on TSMC 65nm worst case library takes 2080 cycles for 256-point complex FFT, consuming 12.35mW power and 0.167mm$^2$ area at 333MHz.

The rest of this paper is organized as follows: Chapter II discusses the proposed joint source and channel decoding work. Chapter III discusses our proposed NoC designs aiming to enhance performance in latency. Chapter IV presents data processing accelerator designs and implementations for low-power embedded SoCs. Finally, we conclude and present our future work in Chapter V.

# CHAPTER II

# JOINT SOURCE CHANNEL DECODING METHOD FOR LOW-POWER PORTABLE AND WIRELESS SOC SYSTEMS

H.264/AVC standard provides some error resilience features for unequal error protection such as flexible macro-block ordering (FMO), redundant slice, and data partitioning (DP). Thomos et al. present that the use of FMO associated with a UEP scheme outperforms classical H.264/AVC transmission schemes in terms of decoded video quality [15]. The utilization of DP in H.264/AVC can yield a lower percentage of entirely lost frames [16]. An extensive study of prioritization and layering techniques for H.264/AVC shows that the combination of DP, turbo codes (TC), and flexible modulation techniques outperforms the combination of DP and TC only [17].

JSCD architectures which combine LDPC and H.264 video coding for UEP are presented by Guo, Wang, Qi, Kumar, and Yang et. al. [18–22]. Guo and Wang et. al. propose LDPC-based unequal error protection algorithms using data partitioning [18, 19]. The idea of the LDPC-based unequal error protection is the high priority data are allocated to low code rate, and low priority data are allocated to high code rate to protect more important partitions from channel errors. Qi et. al propose a dynamic rate selection forward error correction (FEC) scheme utilizing LDPC codes and Reed-Solomon (RS) code for robust video communication [20].

The studies above focus on improving received data quality or robustness of transmission using UEP. In contrast, Wang, Lu, and Zhang et al. consider minimizing both processing power for JSCD and transmitting energy for constrained video quality with RS channel coding [23–25]. Eisenberg et al. propose an unequal iterative decoding approach minimizing the power consumption of a channel decoder with data partitioning and turbo decoding [26]. Higher number of iterations for turbo decoding is used for high priority data to minimize the receiver power while meeting distortion constraints specified by the video decoder at a given channel rate.

Another method to reduce the power consumption of LDPC decoding is presented by Dielissen et. al. [27]. That method exploits scalable sub-block parallelism to achieve efficient LDPC decoding implementations for DVB-S2, enabling lower operating frequency by reducing the parallelism of the LDPC decoder instead of using UEP. However, the scalable parallelism cannot be varied according to the demand of tradeoffs between decoded data quality and low-power requirement.

Wang et. al. present LDPC decoder architecture improving power efficiency through adaptively adjusting the number of iterations of LDPC decoding to meet a required quality for each incoming frame [28–30]. The advantage of this approach is that the LDPC encoder/decoder does not require rate adaptation, thereby simplifying encoder/decoder hardware solutions. The early termination of the iterative process is determined by the constraints of check errors during the decoding of each individual frame. This scheme leads to energy reduction, compared to a fixed iteration technique.

To mitigate the bandwidth limit, MIMO wireless systems that offer higher throughput when compared to single input and single output (SISO) wireless systems have been developed [31]. Thus, to accommodate the increasing capabilities of mobile multimedia devices and services, a video over MIMO joint decoding design using UEP can improve energy efficiency and error robustness in these devices. MIMO-based UEP schemes are presented in [32–35]. These studies demonstrate that MIMO with UEP improves not only the capacity of the system but also error resilience compared to EEP and overcomes frequency selective effects of broadband wireless channels. Yang et. al. propose a hybrid MIMO system, which consists of spatial multiplexing (SM) for low priority data and spatial diversity (SD) for high priority data to achieve better performance in terms of BER and PSNR [32]. Liu et. al. similarly divide H.264 information into two parts according to priority as well [33]. Li et. al utilize two modes, transmission diversity (TD) mode for high error protection and spatial multiplexing (SM) mode for high data rate [34]. These researches use the unequal number of information bits in their channel error correction codes.

The following sections discuss the proposed JSCD schemes using a low-power LDPC decoding architecture, UEP-based configuration set search algorithm, H.264 data partitioning, and DVFS. We also propose a novel MIMO-H.264 JSCD scheme using the UEP-based configuration set search algorithm to reduce power on MIMO detection.

## A. Joint Source Channel Decoding Method Using Unequal Error Protection and LDPC Check Error Levels

In this section, we present the proposed UEP scheme using a low-power LDPC decoding architecture, H.264 data partitioning, and DVFS. The approach includes a partitioning scheme that labels each frame into high, medium, and low priority data. The proposed UEP scheme assigns the unequal number of LDPC decoding iterations to each frame according to the partitioning information on the fly. The aim is to treat each frame with only appropriate degree of error protection, thereby minimizing energy spent for each decoding cycle. The level of decoding effort is then further scaled by the amount of additive noise received from a gaussian channel. Our overall aim is to provide maximum battery life while receiving and reproducing video stream without performance loss in video quality. Therefore, we have explored the tradeoffs between the degradation of reconstructed video quality and the amount of energy reduced by lowering decoding efforts. To further lower power dissipation, our proposed scheme dynamically switches decoder frequency and voltage levels. DVFS is used to guarantee data rates and constant processing time of each decoding frame as well as to achieve desired energy reduction [36, 37].

We present a UEP-LDPC decoding architecture using a DVFS technique for the en-

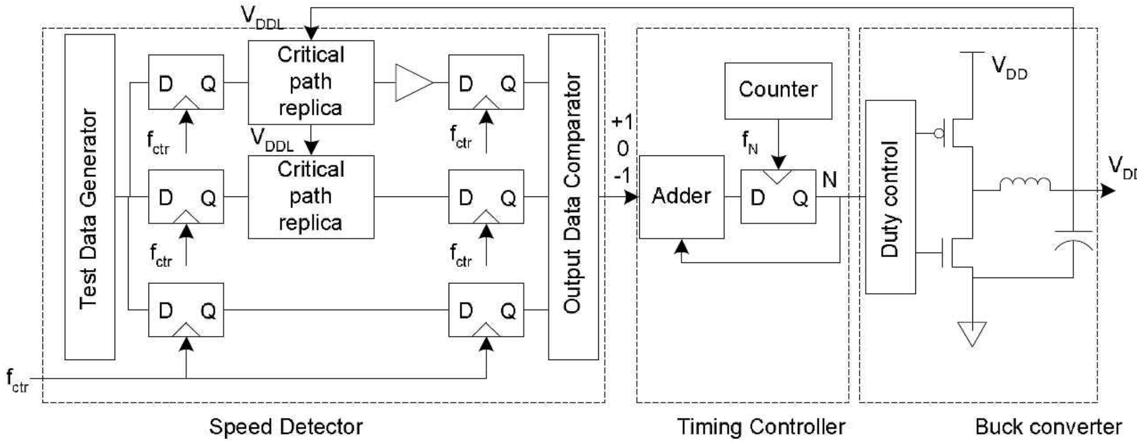


Fig. 1.: Variable supply voltage logic

ergy reduction. This technique is employed for adjusting frequency ($Freq_{LDPC}$) and voltage ($V_{DDL}$) to supply optimal power to the LDPC decoder. Power consumption of a digital CMOS circuit is

$$P = \alpha \cdot C_{eff} \cdot V^2 \cdot f \tag{2.1}$$

where $\alpha$ is switching factor, $C_{eff}$ is effective capacitance, $V$ is operating voltage, and $f$ is operating frequency. Energy required to run a task during time T is

$$E = P \cdot T \propto V^2. \tag{2.2}$$

From (2.1) and (2.2), lowering $V$ yields a quadratic reduction in energy consumption. A variable voltage scheme [11] is presented as shown in Fig. 1. In this figure, $f_{ctr}$ is a main clock, and a supply voltage is generated such that a replicated critical timing path is stably clocked at $f_{ctr}$.

Fig. 2 illustrates the proposed JSCD architecture based on LDPC and H.264 video coding for UEP with a power-aware scheme that configures adaptively the frequency and voltage levels. Firstly, a video stream is encoded into packets with varying priority levels at

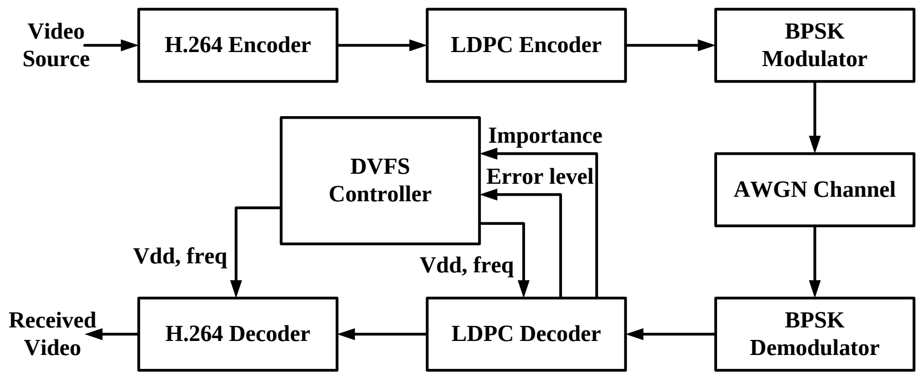


Fig. 2.: Proposed joint source-channel decoding scheme

fixed length by H.264 data partitioning. Secondly, the stream is coded by a LDPC encoder and transmitted over AWGN channels through binary phase shift keying (BPSK) modulation. Next, an unequal error protection technique provides a different level of protection for each frame in a LDPC decoding process. Lastly, a received video stream is reconstructed by a H.264 decoder with error concealment, and then the quality degradation of the received video is measured by peak signal to noise ratio (PSNR), typically used in an image quality estimation.

Dynamic voltage and frequency scaling is used to guarantee data rates and constant processing time of each decoding frame as well as to achieve desired energy efficiency [36, 37]. An illustration of this is shown in Fig. 3. In Fig. 3(a), typically a joint source channel decoder operates at full voltage level for a time duration necessary to successfully decode a frame. In an early terminated iteration approach with clock/power gating techniques [38], wider idle intervals than the intervals of Fig. 3(a) occur between LDPC decoding tasks as shown in Fig. 3(b). In each idle period, the fixed-voltage decoder goes to clock gating mode to reduce power dissipation until new frame data arrives. This process is repeated for each subsequent frame. Alternatively, our DVFS approach estimates the number of LDPC decoding iterations necessary to accommodate the importance of the frame. Then it schedules

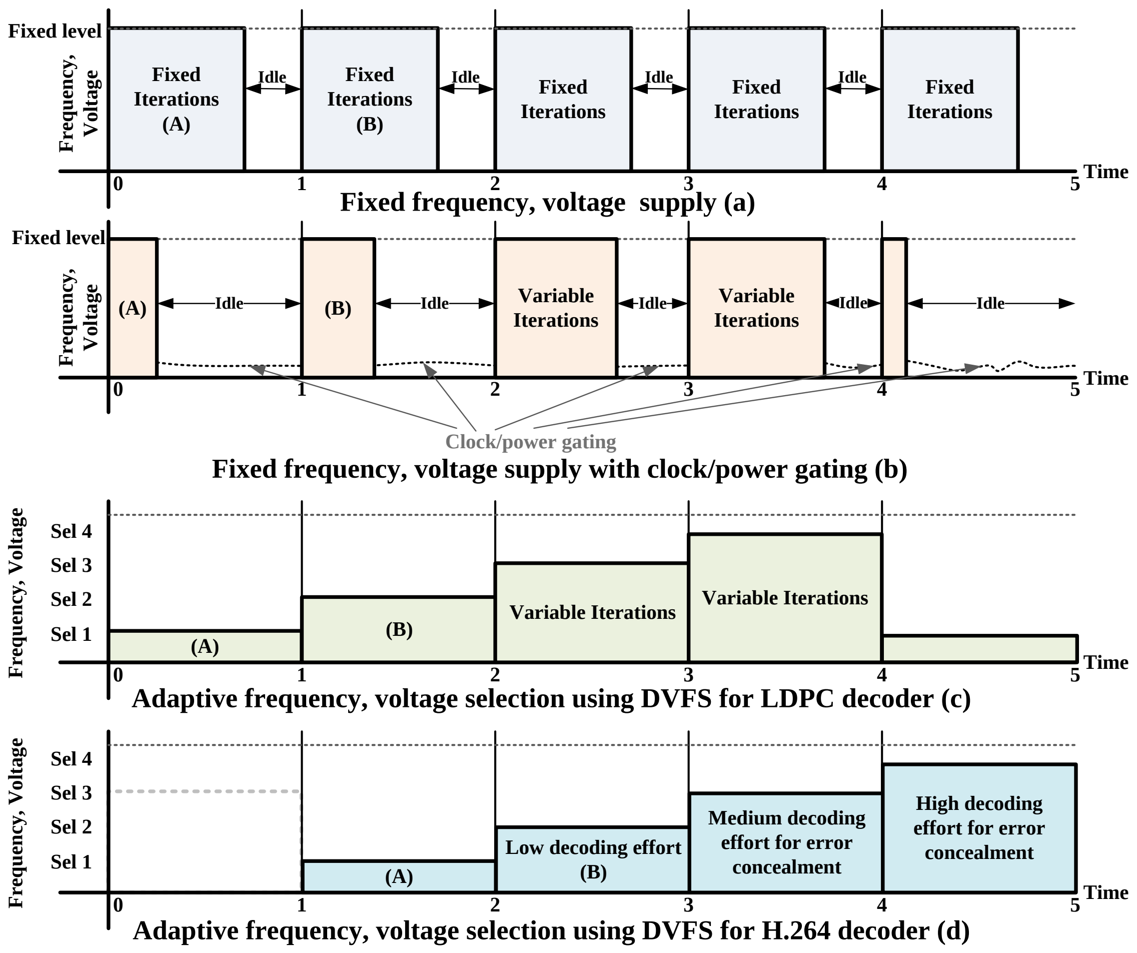

Fixed level
Frequency, Voltage
Fixed Iterations (A)
Idle
Fixed Iterations (B)
Fixed Iterations
Time
0
1
2
3
4
5
Fixed frequency, voltage supply (a)
(A)
(B)
Variable Iterations
Clock/power gating
Fixed frequency, voltage supply with clock/power gating (b)
Sel 4
Sel 3
Sel 2
Sel 1
Adaptive frequency, voltage selection using DVFS for LDPC decoder (c)
Low decoding effort (B)
Medium decoding effort for error concealment
High decoding effort for error concealment
Adaptive frequency, voltage selection using DVFS for H.264 decoder (d)

Fig. 3.: Performance scheduling for energy reduction

exactly that number of iterations until the time point at which next frame is due, reducing the voltage level appropriately for the reduced frequency. This is shown in Fig. 3(c). To support this predictive scheduling of decoding iterations, a combined method is used that 1) estimates the severity of error in each frame by sampling the number of failed parity check equations from the LDPC decoder and 2) assesses the ”importance” information associated with each frame during H.264 coding. Fig. 3(d) illustrates the dynamic voltage-frequency adjustment for the H.264 decoder.

The video decoder employs concealment techniques to mitigate the effects of packet loss [16, 39, 40]. In the video decoder, high error data needs more decoding efforts to conceal and recover the errors while providing acceptable video quality. High priority and check error frames, similarly, require high protection of data over errors. Therefore, it is necessary to exploit more decoding efforts either. In Fig. 3, (A) uses less number of iterations than (B) for LDPC decoding since it is involved in less error environment (less check errors) and less importance; it also consumes less video decoding power. An important real-time constraint for this process is that each frame should complete decoding within a fixed frame-cycle to support constant frame rate in video reproduction. Simulation results of the joint source-channel decoder design with DVFS based on priority and check errors point out clearly that the proposed UEP scheme reduces power dissipation over AWGN channels without performance loss in video quality.

We also propose a near-optimal configuration search method to maximize energy utilization while receiving and reproducing a video stream through LDPC-H.264 joint decoding without significant loss in video quality. An empirical analysis for the configurations of LDPC decoding is studied to quantify the trade-off between the number of decoding iterations and peak signal to noise ratio (PSNR) associated with the reconstructed image quality. In this analysis, UEP is employed to determine the optimal number of iterations for each priority type. The determined iterations are used during runtime to schedule the LDPC decoding. Low priority data allows less iterations in the LDPC decoding process; however, the lowered number of iterations leads to degradation in PSNR performance, but saves energy when used with DVFS that scales voltage and frequency.

## 1. Background: H.264 Unequal Error Protection and LDPC Channel Decoding

### a. H.264 Video Coding

**H.264 overview:** H.264 is a video compression standard known as MPEG-4 Part 10, or MPEG-4 advanced video coding (AVC). It is developed as a joint standard by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). H.264 video coding standard includes the same basic functional elements as previous standards (MPEG-1, MPEG-2, MPEG-4 part 2, H.261, and H.263) such as quantization for bit-rate control, transform for reduction of spatial correlation, motion compensated prediction for reduction of temporal correlation, and entropy encoding for reduction of statistical correlation. The significant improvements in H.264 involve intra-picture prediction, a new 4x4 integer transform, multiple reference pictures, variable block sizes and a quarter-pel precision for motion compensation, a de-blocking filter, and improved entropy coding.

**H.264 unequal error protection:** Multimedia data is especially vulnerable to channel errors due to the predictive coding techniques used in compression schemes such as H.264. Moreover, different portions of video bitstream have different importance to the reconstructed video quality, thereby giving rise to different quality-of-service (QoS) requirements. In order to prevent the degradation caused by errors, one of error resilience techniques is layered video transmission with unequal error protection. UEP provides different levels of protection to the different parts of video data that have unequal degrees of importance. Basically, UEP changes the distribution of errors without incurring extra resource consumption. The aim is to reduce bit errors in more important data. To achieve UEP, a layered video coding scheme needs to be employed to encode the video source into two or more layers with different priorities. Currently, layered video coding is supported by major video compression standards, such as MPEG-2, MPEG-4 and H.264. Data partitioning is

Table I.: H.264 data partition and prioritization

| Priority | NAL Type | NAL Payload |
|---|---|---|
| Priority A | 2 | MB headers, MVs, etc |
| | 5 | IDR picture |
| | 6 | SEI |
| | 7 | SPS |
| | 8 | PPS |
| Priority B | 3 | Intra residual |
| Priority C | 4 | Inter residual |

the simplest form of layered coding. It provides the ability to separate more important and less important syntax elements into different packets of data, and enables the application of unequal error protection (UEP) and other techniques for improving error-loss robustness.

In this dissertation, we used the H.264/MPEG-4 AVC (Advanced Video Coding) standard developed by the joint video team (JVT) of ISO/IEC and ITU-T (Telecommunication Standardization sector) and the reference software JM14.2 of H.264 as a source coder for the video decoding process [41, 42]. At the source side, The data partitioning is used to obtain layered video compression data. At the receiver, because of error bits, the H.264 decoder reconstructs video with error concealment. This coding standard covers two layers, namely, video coding layer (VCL) and network abstraction layer (NAL). Table. I shows NAL units containing the coded video data partitioned into $priority_A$, $priority_B$, and $priority_C$ for the UEP scheme. We exploited the data partitioning technique to prioritize the NAL units into three priority groups where the units in each priority group contain certain coded video elements of same importance to the reconstructed video quality. The first part (priority A) contains the most important data, namely, macro blocks headers, motion vectors (MVs), quantization parameters, IDR picture, and parameter sets. Intra residual data are in the second partition (priority B), and inter residual data are in the last partition (priority C). Since high important and check error frames demand high protection and error concealment for data reconstruction over errors, it is necessary to exploit more decoding efforts. The actions at the video decoder when losing some parts of DPs are described as follows [39]:

- The loss of C (available A and B): conceal using MVs from partition A and texture from Partition B; intra concealment is optional.
- The loss of B (available A and C): conceal using MVs from partition A and inter info from Partition C; inter texture concealment is optional.
- The loss of A (available B and/or C): drop partitions B and C, and use MVs of the spatially above macro block (MB) row for each lost MB.

For instance, if DP B is lost, the motion vectors of DP A and the inter information of DP C can be used for the concealment. Inter texture concealment is optional. We quantified the impact of error in each of these groups empirically.

The quality of reconstructed video frames can be measured by PSNR. PSNR was originally designed to measure distortion in still images due to effects such as lossy compression, commonly used for motion pictures. PSNR is can be calculated by Eqn. (2.3) and (2.4).

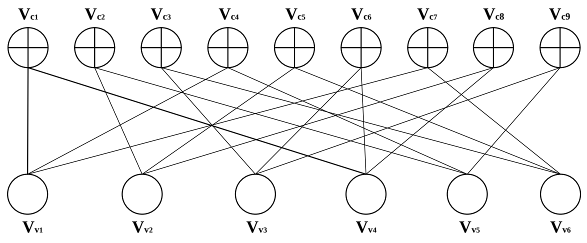
$$PSNR = 10\log_{10}\frac{2^n-1}{MSE} \tag{2.3}$$

$$MSE = \frac{1}{M\times N}\sum_{x,y}[p(x,y)-p'(x,y)]^2 \tag{2.4}$$

where MSE is mean square error computed based on the pixel values of original image ($p(x,y)$) and reconstructed image ($p'(x,y)$). $n$ is the number of bits describing the color of each pixel, and $M$ and $N$ represent the width and height of the image respectively.

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(a) $H$ matrix



(b) Tanner graph

Fig. 4.: An example of $(d_c, d_v) = (3, 2)$ regular LDPC

b. LDPC Coding

LDPC code is an error correcting block code originally proposed by Gallager in 1960's and rediscovered in late 1990's [43, 44]. They describe an iterative two-phase message passing algorithm (TPMP) which involves check-node update and variable-node update as two phase schedule. This code is defined by a sparse parity check matrix $H$ that consists mostly of 0's and described frequently by Tanner graph [45].

In the $H$ matrix, each column contains a small fixed number $d_v$ of 1's, and each row contains a small fixed number $d_c > d_v$ of 1's. Fig. 4 illustrates an example H matrix of $(d_c, d_v) = (3, 2)$ regular LDPC code and corresponding Tanner graph. Tanner graph is a bipartite graph such that its vertex set $V$ can be partitioned into two disjoint subsets, the set of check nodes ($V_c$) and the set of variable nodes ($V_v$). The element of H is 1 if and only if there is an edge in the edge set connecting the check node $V_c$ and the variable node $V_v$. The block length of this code is $n$ which is equal to the number of columns in the $H$ matrix. Suppose that the number of data bits before the channel encoding is $l$, then the number of rows of this $H$ matrix is $m = n - l$. Rate of this code is defined as $l/n = 1 - d_v/d_c$. The code words consist of all one-dimensional row vectors that span the null space of the parity check $H$ matrix. The number for $d_v$ and $d_c$ should be no less than 3 and 6, respectively, for good coding performance. Another type of LDPC code is irregular codes, in which the number of 1's in each row and column is not constant.

The LDPC decoder is typically set to run for data convergence until a prescribed maximum number of iterations depending on the code rate. However, the actual number of decoding iterations varies from frame to frame. As a result, the decoder often remains idle since for most frames, the decoding process ends far earlier than the maximum number of iterations. Thus it is not power efficient. We based our design on recently developed dynamic control to improve the system efficiency by adjusting maximum number of decoding iterations. This design is built on the early termination technique satisfying the required energy constraint for each incoming frame. There is research on the early termination of frame that can not be decoded even if the maximum iterations are applied [46,47]. In both papers, the early termination of the iterative process is determined by checking the messages during the decoding. Their attempts are to dynamically switch off the hardware when no additional iterations will amount to improvement in decoding performance. Wang et al. present an early termination scheme for layered LDPC decoders devised by evaluating the number of checks in error for each frame [28–30].

To update check nodes, various algorithms are used such as sum of products (SP), min-sum (MS) and Jacobian based BCJR (named after its discoverers Bahl, Cocke, Jelinik and Raviv). MS is an approximation of the SP belief propagation algorithm. A quantitative performance comparison for different check updates is given by [48]. Their research shows that the performance degradation of offset based min-sum with 5-bit quantization is less than 0.1dB in SNR as compared with that of floating point SP and BCJR. Mansour et al. introduce the concept of turbo decoding message passing (TDMP), called as layered decoding, using BCJR for their architecture-aware LDPC (AA-LDPC) codes [49,50]. TDMP yields twice throughput and significant memory benefits, reducing the number of iterations required by up to 50% without performance loss when compared to TPMP. This scheme is later studied and applied for different LDPC codes using the sum of products algorithm and its variations in [51]. Our paper presents a JSCD scheme using the layered LDPC decoding based on the offset-min-sum algorithm, early termination strategy, H.264 data partitioning technique for UEP, and DVFS over AWGN channels.

**Two phase message passing LDPC decoding:** The iterative two phase message passing algorithm is computed in two phases [48,52]. One is check node processing and the other is variable node processing. In the check node step, each row of the parity matrix is checked to verify that parity check constraints are satisfied.

$$R_{mn}^{(i)} = \delta_{mn}^{(i)} \max\left(\kappa_{mn}^{(i)} - \beta, 0\right) \tag{2.5}$$

$$\kappa_{mn}^{(i)} = |R_{mn}^{(i)}| = \min_{n' \in N(m)\backslash n} |Q_{n'm}^{(i-1)}| \tag{2.6}$$

For the $i^{th}$ iteration, $Q_{nm}^{(i)}$ is the message from variable node $n$ to check node $m$, $R_{mn}^{(i)}$ is the message from check node $m$ to variable node $n$. A positive constant $\beta$ depends on the code parameters [48]. The sign of check-node message $R_{mn}^{(i)}$ is defined as

$$\delta_{mn}^{(i)} = \Big(\prod_{n' \in N(m)\backslash n} \mathsf{sgn}(Q_{n'm}^{(i-1)})\Big) \tag{2.7}$$

In the variable node step, processing the probability will be updated by summing up the other probabilities from the rest of the rows and the a priori probabilities from the channel output.

$$Q_n = L_n^{(0)} + \sum_{m \in M(n) \backslash m} R_{mn}^{(i)} \tag{2.8}$$

where the log-likelihood ratio of bit $n$ is $L_n^{(0)} = y_n$. In the last step, a hard decision is taken by setting $\hat{x}_n = 0$ if $P_n(x_n) \geq 0$, and $\hat{x}_n = 1$ if $P_n(x_n) < 0$. If $\hat{x}_n H^T = 0$, the decoding process is finished with $\hat{x}_n$ as the decoder output; otherwise, go to the first step.

$$P_n = L_n^{(0)} + \sum_{m \in M(n)} R_{mn}^{(i)} \tag{2.9}$$

If the decoding process does not end within predefined maximum number of iterations, $it_{max}$, stop and output an error message flag and proceed to the decoding of the next data frame.

**Turbo decoding message passing LDPC decoding:** In contrast with two phase message passing algorithm, where all check-nodes are updated simultaneously in each iteration, layered decoding (or called as TDMP) views the $H$ matrix as a concatenation of $j = d_v$ sub-codes. One of advantages of layered decoding is that the LDPC decoding can be performed at each layered level. This increase the efficiency of the iterative decoding process. Mathematically, the layered decoding algorithm can be described as shown in equation (2.10)-(2.13)

$\forall i = 1, 2, ..., it_{max}$,[Iteration loop]

$\forall l = 1, 2, ..., d_v$,[Sub-iteration loop]

$\forall n = 1, 2, ..., d_c$,[Block column loop]

$$\overrightarrow{R}_{l,n}^{(0)} = 0, \overrightarrow{P}_n = \overrightarrow{L}_n^{(0)} \tag{2.10}$$

$$[\overrightarrow{Q}_{l,n}^{(i)}]^{S(l,n)} = [\overrightarrow{P}_n]^{S(l,n)} - \overrightarrow{R}_{l,n}^{(i-1)} \tag{2.11}$$

$$\overrightarrow{R}_{l,n}^{(i)} = f([\overrightarrow{Q}_{l,n'}^{(i)}]^{S(l,n')}) \tag{2.12}$$

$$[\overrightarrow{P}_n]^{S(l,n)} = [\overrightarrow{Q}_{l,n}^{(i)}]^{S(l,n)} + \overrightarrow{R}_{l,n}^{(i)} \tag{2.13}$$

The vectors $\overrightarrow{R}_{l,n}^{(i)}$ and $\overrightarrow{Q}_{l,n}^{(i)}$ represent all the $R$ and $Q$ messages in each block of the $H$ matrix. $s(l,n)$ denotes the shift coefficient for the block in $l^{th}$ block row. $f()$ denotes the check-node processing, which can be done using min-sum algorithm. Due to the structure of layered LDPC $H$ matrix, the updated sum $[\overrightarrow{P}_n]^{S(l,n)}$ in each block column n needs to go through either a) a cyclic down shift of $n-1$ or b)cyclic up shift of $(j-1)n$ if we do layered decoding.

## 2. Proposed Low-Power JSCD Scheme Using DVFS

The proposed layered LDPC code is developed on the adaptive decoding scheme described in [29, 30, 38]. The number of iterations for LDPC decoding is assigned by the threshold of check errors and importance (priority). There are two iteration parameters: inner loop iterations (*iter*=outer loop iterations $\times$ the number of layers($d_v$)) and outer loop iterations ($i = \lceil iter/d_v \rceil$) known as ordinary LDPC iterations. For example, 12 iterations in the regular LDPC decoder are completed at the same time as $12 \times 5 = 60$ sub-iterations when $d_v$ equals 5 in the layered LDPC decoder. The number of sub-iterations depends on the degree of check errors since more check errors (severity of noise impairment) require more decoding efforts. The priority information of the incoming frame thus determines the number of iterations.

Next, the adaptive LDPC decoding routine is combined with UEP. The iterative decoding routine runs until *iter* reaches the sub-iterations. In the decoding process, we used three threshold parameters ($cErrThre_1$, $cErrThre_2$, $cErrThre_3$), and these produce four

Table II.: Check error level for each error scope

| Check error level | Check error scope |
|---|---|
| 1 | $\infty \sim cErrThre_1$ |
| 2 | $cErrThre_1 \sim cErrThre_2$ |
| 3 | $cErrThre_2 \sim cErrThre_3$ |
| 4 | $cErrThre_3 \sim 0$ |

Table III.: Parameters of sub-iterations ($Iter_{Error\ level,Priority}$)

| Check error level | $Priority_A$ | $Priority_B$ | $Priority_C$ |
|---|---|---|---|
| 1 | $Iter_{1,A}$ | $Iter_{1,B}$ | $Iter_{1,C}$ |
| 2 | $Iter_{2,A}$ | $Iter_{2,B}$ | $Iter_{2,C}$ |
| 3 | $Iter_{3,A}$ | $Iter_{3,B}$ | $Iter_{3,C}$ |
| 4 | $Iter_{4,A}$ | $Iter_{4,B}$ | $Iter_{4,C}$ |

check error levels as shown in Table II. Combining the three priority (Table I) and the four check error levels, there are twelve sub-iteration parameters (4 error levels $\times$ 3 priorities = 12), where each parameter requires its specific error coverage (i.e. decoding iterations), as summarized in Table III. The three threshold values ($cErrThre_1$, $cErrThre_2$, $cErrThre_3$) and necessary iterations (Table III) are determined by the empirical analysis of the check error distributions, and stored in a look-up table in pre-processing. Furthermore, the analysis of performance improvement and power consumption was studied to determine optimal parameter sets. The details of this analysis are described in the results and discussion section.

### a. Runtime Process

For runtime decoding process, the joint source-channel encoder inserts NAL unit type bits of the next frame into the current frame data. When the very first frame comes into the LDPC decoder, it decodes the frame using the maximum number of sub-iterations, and

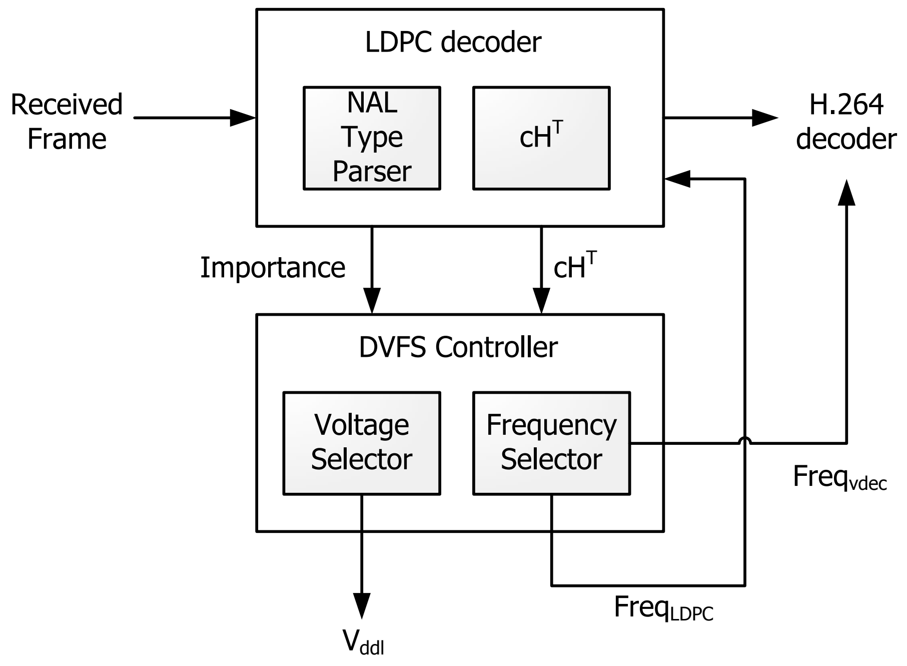


Fig. 5.: DVFS controller

NAL type parser extracts the NAL unit type which denotes the priority of the next frame. The next frame incoming, the LDPC decoder estimates check errors ($cH^T$) and reads the check error thresholds from the look-up table to decide a corresponding check error level according to the check errors. By the priority information acquired from the previous frame decoding and the check error level, the decoder can select the number of sub-iterations from the memory and decode the frame at runtime. However, three check error threshold values and twelve sub-iterations determined by the probability density function of number of check errors in the pre-processing simulation are not updated at runtime.

b. Low-Power JSCD Implementation

We present a low-power joint source-channel decoder architecture and its implementation based on the UEP scheme and the adaptive decoding architecture. Fig. 5 illustrates the block diagram of DVFS controller composed of frequency selector and voltage selector.

In this implementation, we use clock division to select frequency on the fly. This requires no additional delay. For voltage, we have multiple power rails that we switch from. This requires more than one clock cycle to stabilize. However, as long as minimum voltage level to transition up to (or down to) a next selected frequency is met, voltage level can trail that of enveloping minimum voltage level. The frequency selector selects the frequency values ($Freq_{LDPC}$, $Freq_{VDEC}$) based on importance information extracted by NAL type parser and check errors ($cH^T$). The corresponding voltage levels ($V_{ddl}$) for the LDPC decoder and for the video decoder are then chosen by the voltage selector. The power consumption of both decoders can be minimized by the DVFS logic while meeting the real-time constraints required to decode a frame. The DVFS logic operates in $V_{ddl}$ domain, and it is characterized by following Eqn. (2.14) and (2.15) [28].

$$\begin{aligned} P_{total} &= P_{switching} + P_{sc} + P_{leakage} \\ &= \alpha C_L \Delta V V_{dd} f_{clk} + I_{sc} V_{dd} + I_{leakage} V_{dd} \end{aligned} \tag{2.14}$$

$$\tau = \frac{1}{f_{clk}} = \frac{C_L V_{dd}}{I_{dsat}} \propto \frac{V_{dd}}{(V_{dd} - V_{th})^{1.3}} \tag{2.15}$$

where $P_{switching}$ shows the switching power, and $C_L$ is the loading capacitance, $f_{clk}$ is the clock frequency, and $\alpha$ is the node transition factor defined as the probability that a power consuming transition occurs. Mostly, the voltage changing $\Delta V$ is the same as the supply voltage $V_{dd}$. The short circuit power $P_{SC}$ is caused by direct-path short circuit current $I_{SC}$ which arises when both NMOS and PMOS are turned on. $P_{leakage}$ is the leakage component of power, and $I_{leakage}$ is the total leakage current in CMOS circuit. Furthermore, Eqn. (2.15) represents the increment of circuit delay that brings the decrement of voltage supply. The power consumption is estimated assuming 80% dynamic power and 20% static power consumption.

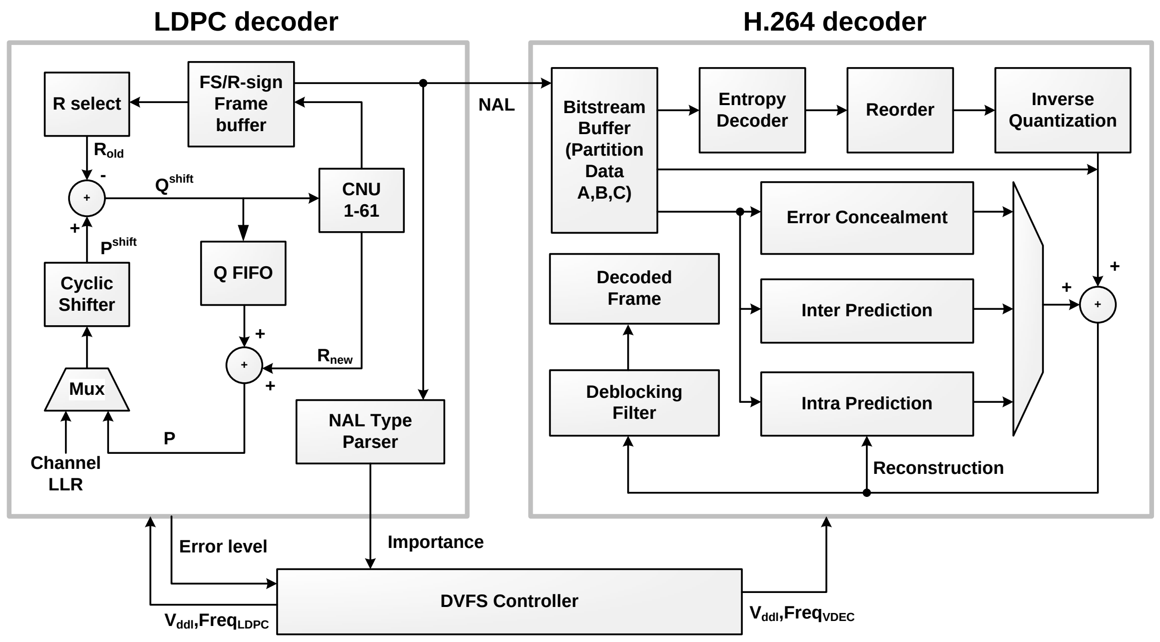


Fig. 6.: The proposed joint source-channel decoder Architecture

The proposed joint source-channel decoder architecture is shown in Fig. 6, including layered decoding architecture and the real-time H.264/AVC baseline decoder. The left part illustrates the architecture of LDPC decoder, and the H.264 decoder is in the right with NAL stream interface to the LDPC decoder. The LDPC decoder includes two particular modules in order to generate the importance information and the number of check errors for UEP and DVFS. The first is NAL type parser, which analyzes NAL header bits to extract NAL unit type. The parsed importance information (i.e. NAL type) is sent to the DVFS controller for the selection of optimal frequency and voltage. The second is the check node message block to provide the check error level to the DVFS logic.

Since the DVFS controller issues variable frequency and voltage to keep constant decoding time, the input NAL stream is transmitted to the video decoder at constant rate. The LDPC decoder in Fig. 6 is derived from the layered LDPC decoder implementation proposed by [52]. The check node units (CNUs) in the figure take variable-node messages associated with each check-node serially and compute the compressed check-node message and index, in which this compressed check-node is the least magnitude of all variable-node messages. These compressed check-node messages are called final states (FS), stored in FS buffers. Check-node messages to each associated variable-node are sent out serially again, selected by index and sign comparison, where the signs of check-node messages ($R$) are stored in sign FIFO. As soon as the check-node message to the first connected variable-node is ready, the corresponding sum message (P) can be computed, and the check error value is sent to the DVFS unit for selecting adjustable voltage and frequency. Then the variable-node message (Q) is ready for the check-node message processing of the next sub-iteration. As such, each CNU operates on two layers of the H matrix simultaneously: selecting check-node message for one layer and computing FS for the next layer.

The proposed H.264 video decoder is built on the open-source low-power H.264 baseline decoder supporting QCIF resolution [53, 54]. We developed several minor modifications to incorporate the UEP scheme. These include additional data partitioning functions and a fixed-length packetization unit. The video decoder is a fully hardwired design without utilizing any general process cores, attributed with the following features:

- Utilization of pipelining and parallelism
- Hybrid and self-adaptive pipeline architecture
- Low cost intra/inter prediction unit
- Error concealment unit
- 5-stage pipelined de-blocking filter
- Clock gating to reduce power

Table IV.: H.264 encoder parameters and configuration

| Source video stream | Foreman, Akiyo, Mobile |
|---|---|
| The number of frames in each source | 150 |
| Frame rate | $30 frame/sec$ |
| Source resolution | $176 \times 144$ (QCIF) |
| Quant. parameter | 28 |

Table V.: Distributions of priority partitions in test video streams, Foreman, Akiyo, and Mobile

| Importance | Distributions(%) | | |
|---|---|---|---|
| | Foreman | Akiyo | Mobile |
| $Priority_A$ | 44.8% | 44.6% | 47.1% |
| $Priority_B$ | 29.6% | 40.4% | 19.8% |
| $Priority_C$ | 25.4% | 14.6% | 32.9% |

### 3. Results and Discussion

In this subsection, we evaluate the proposed, low-power joint source-channel decoding scheme in terms of BER, PSNR, and energy reduction, discussing about the configuration of heuristic parameters and evaluation results.

#### a. Simulation Environment

The environment of simulation is described with specific case studies in this section. Table IV presents parameters and configuration for H.264 encoding. Three video streams, Foreman, Akiyo, and Mobile were simulated to evaluate the performance and energy reduction of the proposed UEP approach. The distributions of the priority A, B, and C partitions in the test video streams are shown in Table V. The distributions significantly

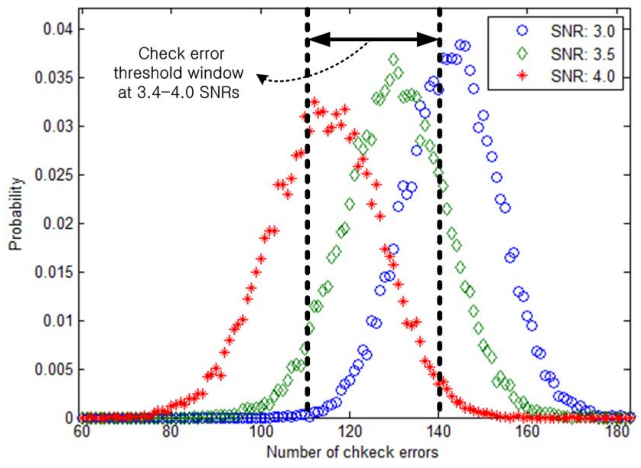


Fig. 7.: Probability density function of number of check errors

Table VI.: Check error threshold values ($cH^T$) used in the simulation

| #Case | $cErrThre_1$ | $cErrThre_2$ | $cErrThre_3$ |
|---|---|---|---|
| Case1 | 140 | 115 | 110 |
| Case2 | 140 | 120 | 110 |
| Case3 | 140 | 125 | 110 |
| Case4 | 140 | 130 | 110 |
| Case5 | 140 | 135 | 110 |

influence the decoding efforts because higher priority groups not only cause more iterative computations but also spend more power. Each priority partition with partition sizes $b(q) = \{b_A(q), b_B(q), b_C(q)\}$, where the size of the partitions $b(q)$, is directly controlled by the applied quantization parameter (QP) $q$ [16]. The parameters, $q$ and channel coding rates $r = r_A, r_B, r_C$, can be constrained by a total bit-rate as

$$N_c(q,r) = \frac{b_A(q)}{r_A} + \frac{b_B(q)}{r_B} + \frac{b_C(q)}{r_C} \leq N_t \tag{2.16}$$

where $N_t$ denotes the total number of bits available for a certain video frame.

Fig. 7 illustrates the probability density function of the number of check errors to find

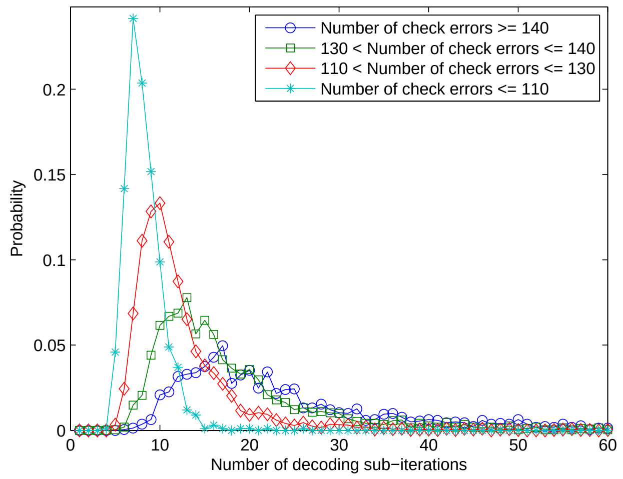


Fig. 8.: Distributions of decoding sub-iterations for different number of check errors

the bounds of the check error thresholds in the layered LDPC code. In the LDPC code, the $H$ matrix was created by $d_c = 25$, $d_v = 5$, and $p = 67$, simulated on AWGN channels. At a given SNR, the number of check errors for received data is consistent with Gaussian distribution. We evaluated the probability density function of check errors at 3.0, 3.5, and 4.0dB SNRs and determined the range of check error thresholds at 3.4-4.0dB SNRs since the LDPC decoder yielded acceptable BER performance at the given SNRs (We will show this in Fig. 9). Hence, $cErrThre_1$ and $cErrThre_3$, upper and lower bounds for the thresholds, were set to be 140 and 110 where the probabilities of check errors appertained to the section of high probability at 3.4-4.0dB SNRs (threshold window in Fig. 7). These heuristic parameter sets are shown in Table VI. In this table, five sets of threshold triplets ($cErrThre_1$, $cErrThre_2$,$cErrThre_3$) were tested for identifying an optimal threshold value set in the UEP simulation.

Fig. 8 shows the distributions of decoding sub-iterations on various numbers of check errors in the LDPC decoder. The average decoding sub-iterations ($E$) at the upper and lower bounds (i.e. the numbers of check errors are 140 and 110) are 19.16 and 8.19, and

Table VII.: The number of sub-iterations used in the simulation

| Check error level | $Priority_A$ | $Priority_B$ | $Priority_C$ |
|---|---|---|---|
| 1 | 60 | 50 | 40 |
| 2 | 45 | 40 | 30 |
| 3 | 30 | 30 | 15 |
| 4 | 15 | 15 | 15 |

the corresponding standard deviations ($\sigma$) are 17.56 and 2.12 respectively. The $E+2\sigma$ points are then 55 and 15. Based on the evaluation, we used 60 and 15 for the maximum and minimum number of sub-iterations in the UEP-LDPC decoder. The twelve parameters of sub-iterations, shown in Table VII, were assessed within the min-max iteration bounds to search minimum number of iterations at the given SNRs in the UEP-LDPC simulation. For evaluation of the UEP-LDPC decoder, we compared the results of UEP decoding with the result of fixed-iteration LDPC decoding. The fixed number of iterations was selected as a half value (i.e. 30 sub-iterations) of the UEP-LDPC maximum number of iterations for a proper evaluation because if we chose a large number of iterations for the fixed-iteration decoding like 40$\sim$60 sub-iterations in the comparison, the energy reduction in the UEP over the fixed-iteration method would be naturally achieved, thus this is not fair.

For the hardware implementation, we synthesized the joint source-channel decoder and the DVFS controller using TSMC 0.13$\mu m$ technology. Critical path of the decoder was extracted from Synopsys design compiler [55]. Extra timing margin (5% on assuming $2\times\sigma <5\%$) was added to the critical path to accommodate variations; it can be increased more when the variations are large [30]. We targeted at a low bit-rate and low throughput portable system such that video bit rate was under 300kbps (H.264 baseline decoder), and LDPC decoder had 1Mbps throughput, which was very low in comparison with the throughput of [30]'s LDPC decoder, 200Mbps. In other words, our LDPC decoder

Table VIII.: Sub-iterations vs. voltage/frequence for DVFS

| #Sub-iterations | Voltage($V_{ddl}$) | LDPC Frequency(MHz) | H.264 Frequency(MHz) |
|---|---|---|---|
| 60 | 1.45 | 16.2 | 3.7 |
| 50 | 1.4 | 15.5 | 3.5 |
| 45 | 1.25 | 12.5 | 2.8 |
| 40 | 1.2 | 11.7 | 2.6 |
| 30 | 1.1 | 9.2 | 2.0 |
| 15 | 1.0 | 7.0 | 1.5 |

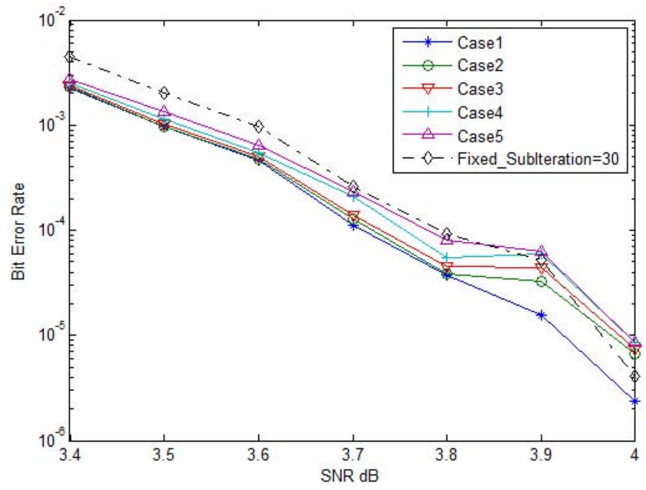


Fig. 9.: BER performance comparison in between the five cases of check error threshold configurations and the fixed-iteration scheme on Foreman

and H.264 decoder, both were designed on low frequency, low performance and high energy saving constraints. In Table VIII, with 1.45∼1.0V voltage supply constrained by the technology, when the number of decoding sub-iterations was chosen among 15∼60, the corresponding frequency was able to be selected from 7.0MHz∼16.2MHz for the LDPC decoder and from 1.5MHz∼3.7MHz for the video decoder.

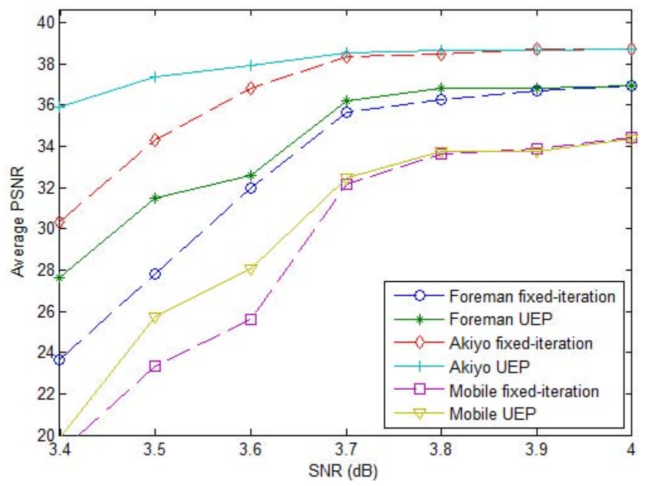


Fig. 10.: PSNR performance comparison in between UEP Case3 and the fixed-iteration scheme on Foreman, Akiyo, and Mobile

b. Simulation Results

The analysis of the amount of power reduction without performance loss is presented here. In order to evaluate the proposed UEP design, the implemented hardware was operated on five parameter cases (Table VI). There are two evaluations for the proposed joint source channel decoder. The first is the comparison of decoding performance between the fixed-iteration simulation and the UEP simulation in terms of UEP-LDPC performance (BER) and reconstructed image quality (PSNR). The other is the measurement of energy reduction while driving the DVFS scheme in the UEP simulation.

Fig. 9 shows the BER performance of the UEP-LDPC decoder, compared with the BER performance of the fixed-iteration decoder on Foreman. In this figure, 3.4∼4.0dB SNRs are used on AWGN channels, this is because we wanted to keep the BER better than $10^{-2}$. (At SNRs of less than 3.4dB, the LDPC decoder's BERs do not get better than $10^{-2}$. At high SNRs, there are few errors or no errors; therefore, comparing the UEP scheme with the fixed iteration scheme is meaningless). Among the five test cases, Case1 is the most

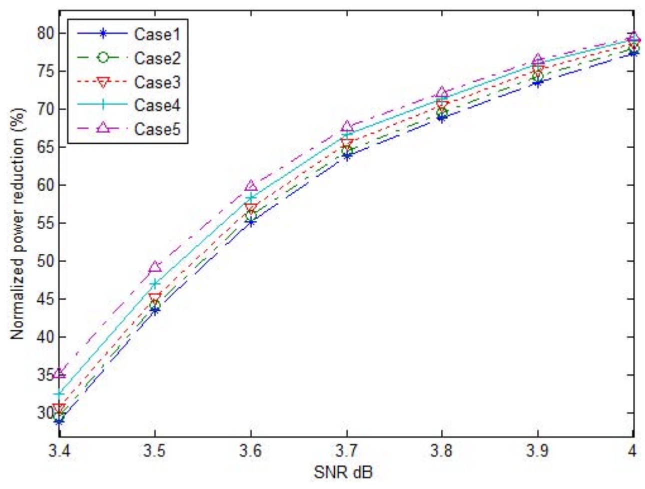


Fig. 11.: Energy reduction (%) of the UEP-LDPC decoder on Foreman

protected in high channel errors so that it requires more iterations (more decoding effort but less power reduction) in the LDPC decoding process. Therefore, it shows higher BER performance than the other cases in Fig. 9.

Fig. 10 shows the PSNR performance of the proposed UEP based decoder and the fixed-iteration decoder on Foreman, Akiyo, and Mobile. Case3 is selected for the performance comparison with the fixed-iteration decoder because the check error thresholds of Case3 are evenly balanced. The PSNR performance of the UEP Case3 is superior than the fixed-iteration performance in 3.4∼3.8dB since the UEP can efficiently protect high priority data that influence the reconstructed video quality over noisy channel. In high SNRs (3.8∼4.0dB), the PSNRs of the UEP and the fixed-iteration are saturated to the maximum PSNR values since there are few errors.

The power analysis (Fig. 11, 12, and 13) presents the power reduction of the proposed joint decoder. In Fig. 11 and 12, the power reduction results of the five simulation cases in the UEP-LDPC decoder and in the UEP-H.264 decoder are shown respectively. The

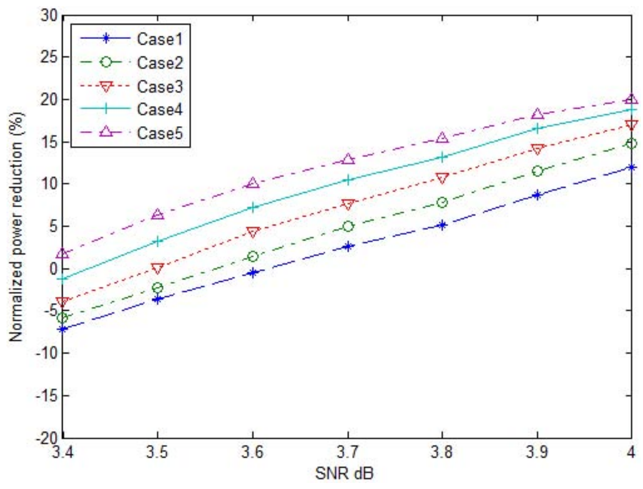


Fig. 12.: Energy reduction (%) of the UEP-H.264 decoder on Foreman

normalized power reduction (%) is computed by Eqn. (2.17).

$$Power_{reduction} = (1 - \frac{Power_{UEP}}{Power_{Fixed}}) \times 100 \tag{2.17}$$

where $Power_{UEP}$ and $Power_{Fixed}$ denote the power dissipation of the UEP decoder and the fixed-iteration decoder respectively. In Case3, the UEP-LDPC scheme reduces 30% power more than the fixed-iteration scheme at 3.4dB. As SNR increasing, more energy reduction can be achieved not only because the average of UEP decoding iterations decreases but because the operating frequency and voltage decrease by the DVFS logic, compared to the fixed-iteration decoding scheme. In Fig. 12, the UEP-video decoder consumes more power than the non-UEP video decoder at low SNRs (3.4 and 3.5dB) since the UEP-video decoder consumes more processing power for intensive error concealment than the non-UEP video decoder in high error environment. However, at high SNRs, the UEP-video decoder consumes less power than the non-UEP video decoder since the UEP-video decoder utilizes less error concealment relatively; this decreases the operating frequency and voltage driven by the DVFS logic.

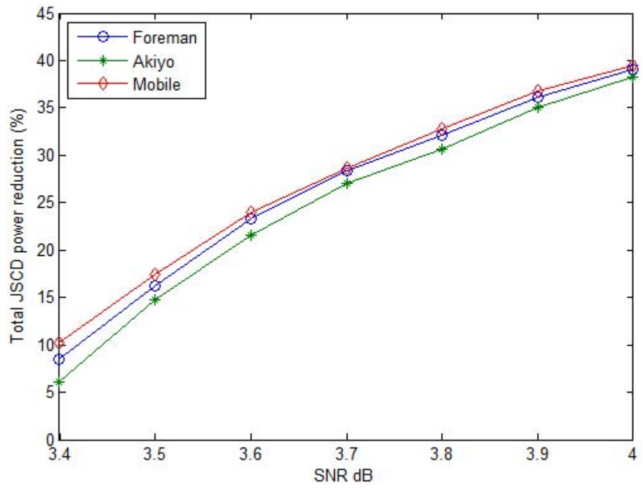


Fig. 13.: Energy reduction (%) of the UEP joint source channel decoder on Foreman, Akiyo, and Mobile

Fig. 13 illustrates the overall power reduction in the joint decoder on Foreman, Akiyo, and Mobile. In the power reduction results of Akiyo and Mobile, we observed that the results were similar to the result of Foreman even though the distributions of the priority group B and C of Foreman, Akiyo, and Mobile were different. The reason is that these video streams have similar distributions in the priority group A; the group A is dominant in performance and energy consumption.

In summary, Table IX presents the overall simulation results on Foreman, Akiyo and Mobile, where F, U, and R denote fixed-iteration, UEP, and power reduction respectively. The power results of the synthesized LDPC and video decoder are presented in Fig. 14. In the results, the LDPC decoder and H.264 decoder consume 256.3$\mu$W and 459.9$\mu$W at 3.4 dB ,respectively, on Foreman; this consumes overall 716.2$\mu$W. According to the evaluation results, it can be concluded that the proposed UEP-joint source channel decoder design provides efficient energy scheduling without performance loss.

Table IX.: Simulation results of the UEP Case3 on Foreman, Akiyo, and Mobile

Foreman

| SNR | $BER_F$ | $BER_U$ | $PSNR_F$ | $PSNR_U$ | LDPC $P_R$ | VDEC $P_R$ | JSCD $P_R$ |
|---|---|---|---|---|---|---|---|
| 3.4 | $4.41\times10^{-3}$ | $2.36\times10^{-3}$ | 26.38 | 30.35 | 30.72 | -3.90 | 8.48 |
| 3.5 | $2.03\times10^{-3}$ | $1.02\times10^{-3}$ | 28.62 | 33.27 | 45.25 | 0.03 | 16.21 |
| 3.6 | $9.78\times10^{-4}$ | $4.99\times10^{-4}$ | 33.26 | 34.60 | 57.07 | 4.38 | 23.23 |
| 3.7 | $2.63\times10^{-4}$ | $1.38\times10^{-4}$ | 35.28 | 36.21 | 65.45 | 7.62 | 28.31 |
| 3.8 | $9.14\times10^{-5}$ | $4.54\times10^{-5}$ | 36.69 | 36.23 | 70.44 | 10.76 | 32.12 |
| 3.9 | $5.24\times10^{-5}$ | $4.42\times10^{-5}$ | 36.82 | 36.41 | 75.16 | 14.26 | 36.05 |
| 4.0 | $4.06\times10^{-6}$ | $7.41\times10^{-6}$ | 36.95 | 36.10 | 78.53 | 16.96 | 38.99 |

Akiyo

| SNR | $BER_F$ | $BER_U$ | $PSNR_F$ | $PSNR_U$ | LDPC $P_R$ | VDEC $P_R$ | JSCD $P_R$ |
|---|---|---|---|---|---|---|---|
| 3.4 | $7.94\times10^{-3}$ | $2.42\times10^{-3}$ | 30.33 | 35.87 | 27.64 | -6.03 | 6.01 |
| 3.5 | $3.65\times10^{-3}$ | $9.65\times10^{-4}$ | 34.30 | 37.36 | 44.30 | -1.61 | 14.81 |
| 3.6 | $1.75\times10^{-3}$ | $5.41\times10^{-4}$ | 36.84 | 37.90 | 55.36 | 2.68 | 21.53 |
| 3.7 | $4.74\times10^{-4}$ | $1.29\times10^{-4}$ | 38.34 | 38.56 | 64.48 | 6.09 | 26.98 |
| 3.8 | $1.64\times10^{-4}$ | $5.24\times10^{-5}$ | 38.47 | 38.65 | 69.37 | 8.99 | 30.59 |
| 3.9 | $9.41\times10^{-5}$ | $6.40\times10^{-5}$ | 38.70 | 38.65 | 74.52 | 13.07 | 35.06 |
| 4.0 | $7.31\times10^{-6}$ | $4.30\times10^{-6}$ | 38.71 | 38.71 | 78.21 | 15.93 | 38.21 |

Mobile

| SNR | $BER_F$ | $BER_U$ | $PSNR_F$ | $PSNR_U$ | LDPC $P_R$ | VDEC $P_R$ | JSCD $P_R$ |
|---|---|---|---|---|---|---|---|
| 3.4 | $2.85\times10^{-3}$ | $2.29\times10^{-3}$ | 19.20 | 19.92 | 32.70 | -2.40 | 10.15 |
| 3.5 | $1.31\times10^{-3}$ | $1.03\times10^{-3}$ | 23.32 | 25.70 | 46.59 | 1.16 | 17.41 |
| 3.6 | $6.32\times10^{-4}$ | $5.49\times10^{-4}$ | 25.60 | 28.06 | 57.60 | 5.20 | 23.95 |
| 3.7 | $1.70\times10^{-4}$ | $1.48\times10^{-4}$ | 32.14 | 32.45 | 65.40 | 8.14 | 28.63 |
| 3.8 | $5.90\times10^{-5}$ | $5.78\times10^{-5}$ | 33.60 | 33.75 | 70.78 | 11.68 | 32.82 |
| 3.9 | $3.38\times10^{-5}$ | $5.11\times10^{-5}$ | 33.90 | 33.78 | 75.57 | 15.19 | 36.79 |
| 4.0 | $2.62\times10^{-6}$ | $9.11\times10^{-6}$ | 34.40 | 34.36 | 78.68 | 17.50 | 39.39 |

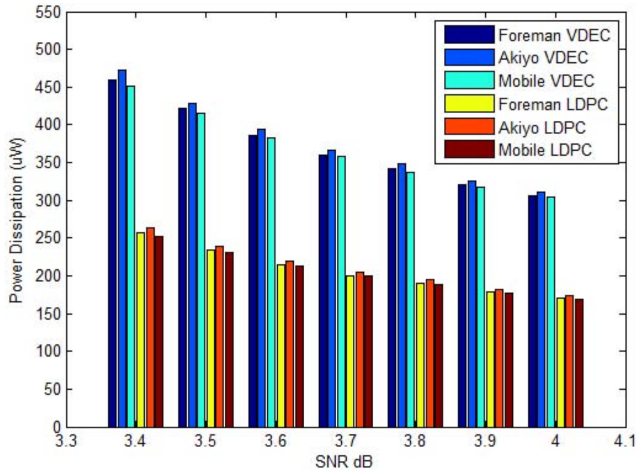


Fig. 14.: Power consumption ($\mu$W) of the UEP decoder on Foreman, Akiyo, and Mobile

## B. Optimal Configuration Search Method for Low-Power Channel Decoder in Embedded LDPC-H.264 Joint Decoding Architecture

In this section, we also propose a near-optimal configuration search method to maximize energy utilization while receiving and reproducing a video stream through LDPC-H.264 joint decoding without significant loss in video quality. An empirical analysis for the configurations of LDPC decoding is studied to quantify the trade-off between the number of decoding iterations and peak signal to noise ratio (PSNR) associated with the reconstructed image quality. In this analysis, UEP is employed to determine the optimal number of iterations for each priority type. The determined iterations are used during runtime to schedule the LDPC decoding. Low priority data allows less iterations in the LDPC decoding process; however, the lowered number of iterations leads to degradation in PSNR performance, but saves energy when used with DVFS that scales voltage and frequency [11].

### 1. Proposed Optimal Configuration Search Method

The presented joint decoding process mainly consists of two parts: offline pre-process and online execution as shown in Fig. 15. In the offline pre-process, a suite of test video streams are encoded through an H.264 video encoding software, and the encoded video frames are delivered to the search process to find optimal iteration sets using UEP. The optimal sets are then determined by the priorities of input frames, and average PSNR values are measured. Using this process, we quantify the relationship between the prioritized video frames and the number of LDPC decoding iterations on targeted PSNRs and find the minimum iteration sets. The searched iteration sets are stored in a lookup table for the online execution process.

During the online process, encoded video streams containing partition information are transferred through the LDPC encoder. When receiving the frames, the LDPC decoder

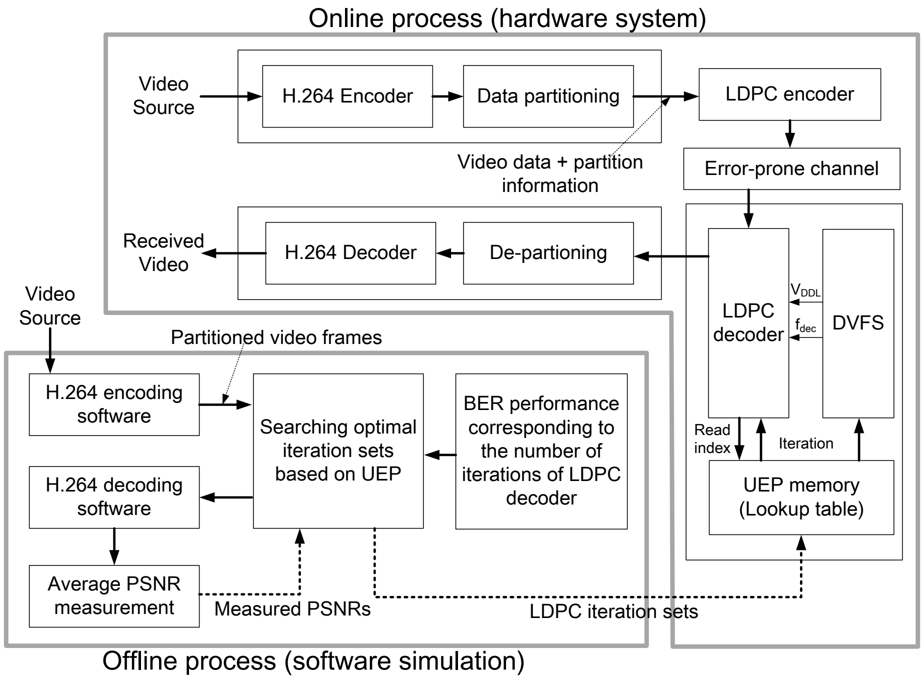


Fig. 15.: The proposed UEP-based joint decoder design

reads the iteration values from the UEP memory using read indices to decode. The read indices are generated by the pre-specified target PSNRs, channel condition (SNR), and priority information on the fly. DVFS controller then selects an operation voltage corresponding to each input frame according to the iteration information loaded from the lookup table.

a. Overview of the proposed search scheme

The proposed method searches the decoding iteration set $(iter_{min,A}, iter_{min,B}, iter_{min,C})$ that yields minimum energy consumption at the given PSNR level, where $iter_{min,A}$, $iter_{min,B}$, $iter_{min,C}$ are the minimum number of iterations for decoding frames annotated with priority type A, B, and C respectively. The search process is composed of two search steps: coarse search (algorithm 1, 2) and fine search (algorithm 3).

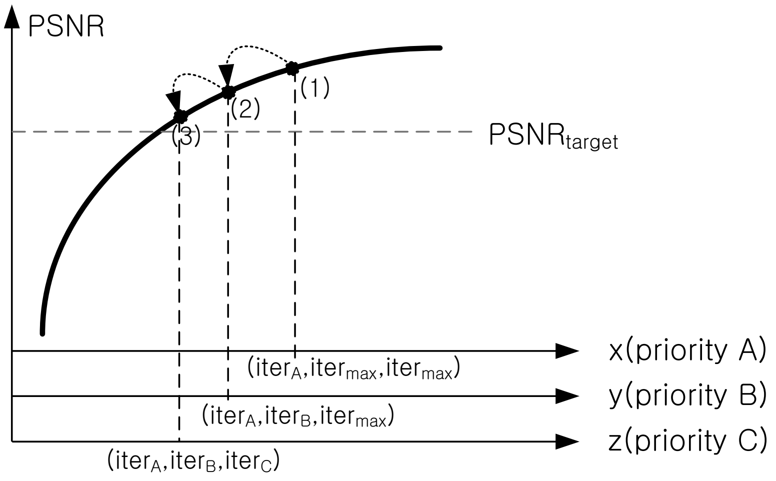


Fig. 16.: Coarse search method

b. Coarse binary search process

In algorithm 1, binary search [56] is used to find a coarse estimate of an iteration set $(iter_A, iter_B, iter_C)$, where $iter_A$ indicates the number of iterations in LDPC decoding for frames of priority A. The iteration set is then obtained when the average PSNR of H.264 reconstructed images marginally satisfies the target PSNR (desired quality of reconstructed images), $PSNR_{target}$. This routine can be graphically represented in Fig. 16:

1. Find minimum $iter_A = i$ satisfying $PSNR_{target} < PSNR(iter_A, iter_{max}, iter_{max})$
2. Find minimum $iter_B = i$ satisfying $PSNR_{target} < PSNR(iter_A, iter_B, iter_{max})$
3. Find minimum $iter_C = i$ satisfying $PSNR_{target} < PSNR(iter_A, iter_B, iter_C)$

In this figure (each axis, x, y, z, shows the search space of iterations), $iter_A$ is first searched since the average PSNR is more sensitive to the impairment of high priority frames. Therefore, the solution of the search routine would quickly converge on the coarse estimate of the iteration set $(iter_A, iter_B, iter_C)$ when the coarse search is conducted in the order of importance $A$, $B$, and $C$. In addition, the initial values of $iter_B$ and $iter_C$ are set to $iter_{max}$ while searching $iter_A$ because we search the set in the direction of reducing iterations (from the large number of iterations to the small number of iterations : $iter_{max} \rightarrow iter_{min}$). To find the coarse iteration set $(iter_A, iter_B, iter_C)$, the binary search algorithm can be performed as algorithm 2.

**Algorithm 1** Coarse Search

```
1:  Coarse_Search($PSNR_i, PSNR_{target}$)
2:  $min = 1, max = iter_{max}$
3:  while $min <= max$ do
4:    $mid = (max + min)/2$
5:    if $PSNR_{mid} < PSNR_{target}$ then
6:      $min = mid + 1$
7:    else if $PSNR_{mid} > PSNR_{target}$ then
8:      $max = mid - 1$
9:      $pos = mid$
10:   else
11:     $pos = mid$
12:     break
13:   end if
14: end while
15: return $pos$
```

**Algorithm 2** Coarse Iteration Set $(iter_A, iter_B, iter_C)$

```
1: $PSNR_i = PSNR(i, iter_{max}, iter_{max})$
2: $iter_A$ = Coarse_Search($PSNR_i, PSNR_{target}$)
3: $PSNR_i = PSNR(iter_A, i, iter_{max})$
4: $iter_B$ = Coarse_Search($PSNR_i, PSNR_{target}$)
5: $PSNR_i = PSNR(iter_A, iter_B, i)$
6: $iter_C$ = Coarse_Search($PSNR_i, PSNR_{target}$)
```

$PSNR(a,b,c)$ shows an average PSNR at $iter_A = a$, $iter_B = b$, and $iter_C = c$. $PSNR$ $(i, iter_{max}, iter_{max})$ means an average PSNR when $iter_A$ is variable i, and $iter_B$, $iter_C$ are fixed to $iter_{max}$.

c. Fine search process using the UEP scheme

The fine search shown in algorithm 3 uses the fact that the increasing of the number of iterations for high importance data leads to enhancement in the average PSNR. Hence, a small increment of $iter_A$ leads to a large decrement of $iter_B$ and $iter_C$ while satisfying the target PSNR. After obtaining a coarse iteration set $(iter_A, iter_B, iter_C)$ from the coarse search, we gradually increase $iter_A$ and decrease $iter_B$ and $iter_C$ until finding a minimum iteration set $(iter_{min,A}, iter_{min,B}, iter_{min,C})$, which maximizes the total reduced iterations $(\Delta iter_B + \Delta iter_C - \Delta iter_A)$. The iteration set can be found as follows:

1. Increase $iter_A$ and decrease $iter_B$ and $iter_C$
2. Find $(iter_A, iter_B, iter_C)$ maximizing $[(\Delta iter_B + \Delta iter_C) - \Delta iter_A]$ (the total number of reduced iterations) and satisfying $PSNR_{target} < PSNR(iter_A, iter_B, iter_C)$

The reduced iterations resulted from the proposed algorithms lead to energy savings along with DVFS.

## 2. Energy Minimization Using DVFS

Fig. 17 represents the hardware implementation of the LDPC decoder and DVFS controller. The offset-min-sum LDPC decoder that we use In this dissertation is based on [30]. The check node units (CNU1-61) take the variable-node message associated with each check-node serially and compute the least magnitude of all variable-node messages. They are stored in Final States (FS) buffers. Check-node messages to each associated variable-node are sent out serially again, and they are selected by index and sign comparison, where the signs of $R$ messages are stored in a sign FIFO. As soon as check-node messages to the first connected variable-node are ready, the corresponding P sum message can be computed. The Q message is then ready for check-node message processing of the next iteration.

**Algorithm 3** Fine search based on unequal error protection

```
1: Fine_Search()
2: $x = iter_A$, $y = iter_B$, $z = iter_C$
3: $iter_{min,A} = iter_A$, $iter_{min,B} = iter_B$, $iter_{min,C} = iter_C$
4: $diff_{max} = 1$
5: while $x < iter_{max}$ do
6:   $x = x + 1$
7:   $y = iter_B$
8:   while $y > iter_{min}$ do
9:     $y = y - 1$
10:     $z = iter_C$
11:     while $z > iter_{min}$ do
12:       $z = z - 1$
13:       $diff = (iter_B - y) + (iter_C - z) - (x - iter_A)$
14:       if $PSNR_{target} < PSNR(x,y,z)$ && $diff_{max} < diff$ then
15:         $iter_{min,A} = x$, $iter_{min,B} = y$, $iter_{min,C} = z$
16:         $diff_{max} = diff$
17:       end if
18:     end while
19:   end while
20: end while
```

The searched sets of minimum iterations for priority frame A, B, and C are loaded to the DVFS controller from the UEP memory for the energy minimization scheme in a real-time system. Frequency information is generated from the iteration values and stored into a frequency selector register in the DVFS controller. The frequency selector generates a Freq_sel signal using the priority information. When the selection is done, the clock divider, such as a phase-loop locker (PLL) [57], divides the fast system clocks ($fs_{ctr}$,$fs_{dec}$) into slower clock signals ($f_{dec}$,$f_{ctr}$) according to the output of the frequency selection register. $f_{dec}$ clocks the decoder for the current frame, and $f_{ctr}$ is sent to the variable supply voltage logic for the voltage control.

## 3. Results and Discussion

We used four different QCIF (176x144) video sources (Foreman, Akiyo, Mobile and News: each 300 frames) at 30fps for the JM14.2 decoder [42] for simulation. The proposed

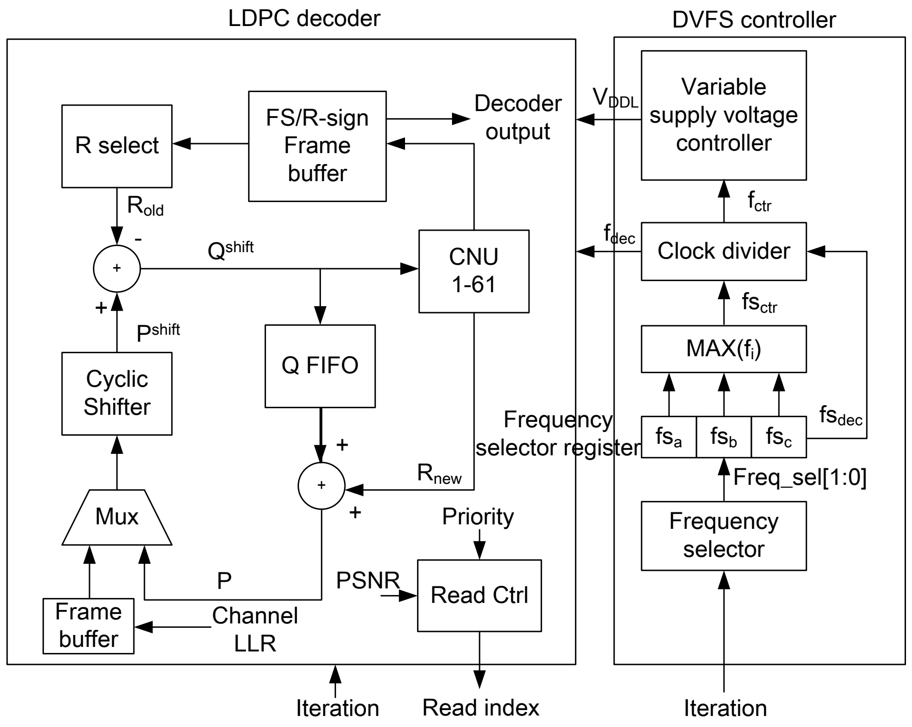


Fig. 17.: DVFS controller for LDPC decoder

method was simulated at 3.5, 3.6 and 3.7dB SNR levels. This was because we simulated three cases keeping the BER performance of the LDPC decoder better than $10^{-3}$ as shown in Fig. 18. In this simulation, LDPC decoder had $4/5$ code rate and 1675 bits code length over an adaptive white gaussian noise channel (AWGN), and our searching scheme can be extended to other LDPC decoder designs. For ASIC design, we synthesized the LDPC decoder and DVFS controller with TSMC $0.13\mu m$ technology (Table X). The critical path of the decoder was extracted from Synopsys Design Compiler.

The DVFS controller demonstrated that the LDPC decoder could be clocked as fast as 175MHz with a 1.5V supply voltage. Fig. 19 shows the relationship between decoding iterations and required supply voltage for the decoding. For instance, when the numbers of iterations were 3, 5, 12, and 18, then the supply voltages could be chosen to 1.02V, 1.05V,

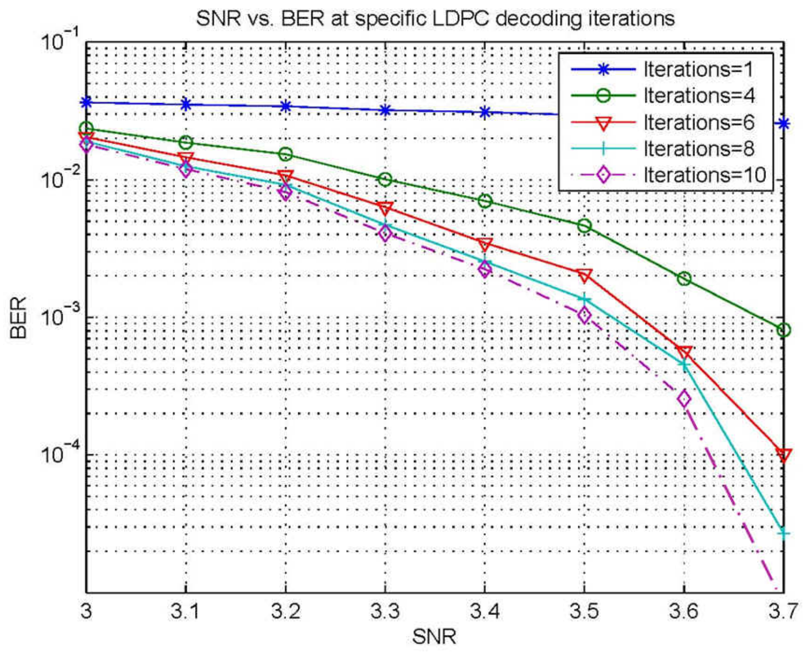


Fig. 18.: BER performance of the LDPC decoder

1.19V, and 1.45V, respectively. The corresponding decoder frequencies then were 66MHz, 82.5MHz, 110MHz, and 165MHz, which could be derived from a 330MHz system clock.

Table XI shows the simulation results in Foreman, Akiyo, Mobile, and News video streams. The number of iterations for the search scheme varied from 1 (minimum iterations) to 20 (maximum iterations), and each result was compared to EEP using 10, 15, and 20 iterations in terms of power consumption. $PSNR_{target}$ values were chosen from PSNRs which mostly covered the PSNR ranges of reconstructed images. To measure the energy reduction in the simulation, we computed energy consumption ratio of the energy consumption of UEP over the energy consumption of EEP. $E_{10}$, $E_{15}$, $E_{20}$ denote the energy consumption ratios, where the numbers of iterations are set to 10, 15, 20, respectively, in the EEP simulation. The results of the coarse and fine search show that a slight increase of iterations in high priority data causes a large amount of decrease of iterations in low priority data with the same PSNR level. The energy consumption ratios can be greater than

Table X.: Synthesis results of LDPC decoder and DVFS controller

| | DVFS logic | LDPC decoder |
|---|---|---|
| Frequency | 175MHz | 175MHz |
| Area | $0.011mm^2$ | $1.2mm^2$ |
| Power | $\sim 10mW$ | $\sim 200mW$ |

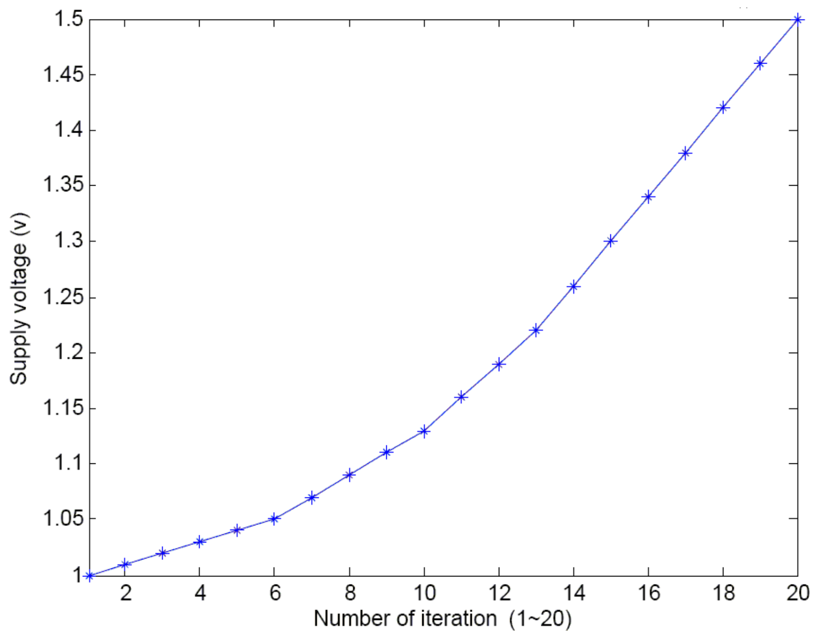


Fig. 19.: Iterations vs. supply voltage

1 in the cases of $E_{10}$ and $E_{15}$ because, in high target PSNRs, the energy consumption of UEP (i.e. the average iterations) can be greater than that of EEP. For example, $E_{10}$ is 1.23 in Foreman when SNR is 3.5dB and target PSNR is 35.2dB.

Fig. 20 graphically shows the results of the proposed joint decoding method in Foreman video stream at 3.5, 3.6, and 3.7dB SNRs. In Fig. 20(a), the results of the coarse search involve high numbers of iterations for priority B and C data. To further minimize the iteration values which were found in the coarse search results, the iterations for the priority B and C data were gradually decreased with a slight increasing of the iterations for the priority A data in the fine search as shown in Fig. 20(b). In this figure, the considerable decrease of the iteration values for the priority B and C data achieves noteworthy power reduction. Fig. 20(c) illustrates the energy consumption ratios of UEP over EEP at given SNRs. When a target PSNR is low, the proposed UEP scheme can reduce power consumption, but cause video quality degradation; therefore, the trade-off between power and quality happens here. As presented in Table XI, EEP PSNRs with 10, 15, 20 iterations are 31.5, 32.7, 33.9 at 3.5dB in Foreman. Once 31.5 is selected as target PSNR, then the proposed decoding scheme can reduce 15%, 35%, 48% energy consumption, but lower 0, 1.2, 2.4dB in PSNR compared to EEP using 10, 15, 20 iterations respectively. In Fig. 20(c), the UEP scheme reduced 17%, 37%, 52% power consumption with 0, 0.3, 1.1dB PSNR losses at 3.6dB, and 18%, 38%, 52% with 0.4, 0.4, 0.4dB PSNR losses at 3.7dB when compared to the results of EEP using 10, 15, 20 iterations.

Table XI.: Simulation results at 3.5, 3.6 and 3.7dB. It shows LDPC configuration sets achieved from the coarse search and the fine search. $E_n$ represents the energy consumption ratio of UEP over EEP$_n$, where n is the number of iterations.

SNR=3.5dB

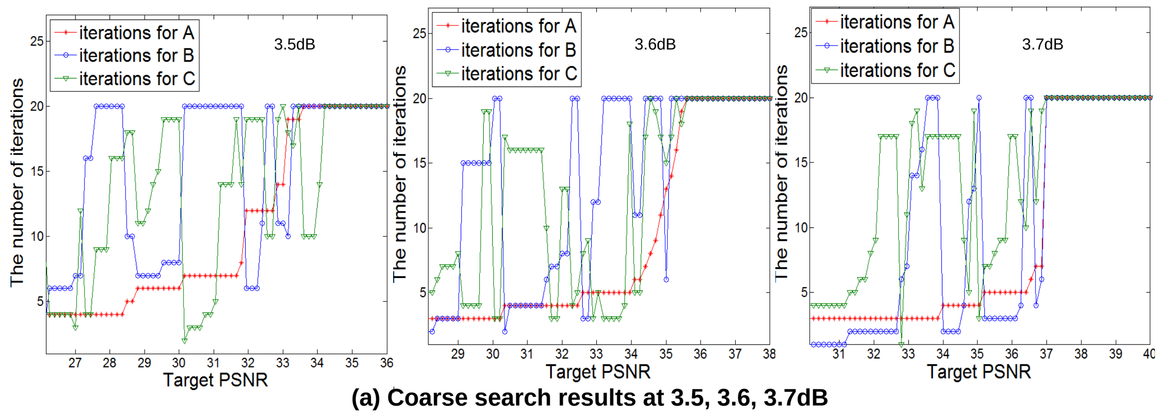
| Foreman (EEP PSNRs with 10/15/20 iterations = 31.5/32.7/33.9) | | | | | | Akiyo (EEP PSNRs with 10/15/20 iterations = 37.7/38.1/38.07) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 30.0 | (6,8,19) | (8,4,1) | 0.86 | 0.65 | 0.49 | 35.5 | (4,4,9) | (6,3,1) | 0.83 | 0.63 | 0.47 |
| 31.0 | (7,20,5) | (10,3,1) | 0.89 | 0.67 | 0.51 | 36.0 | (4,6,13) | (6,4,1) | 0.83 | 0.63 | 0.47 |
| 32.0 | (12,6,19) | (15,3,1) | 1.04 | 0.78 | 0.59 | 37.0 | (6,8,12) | (7,4,1) | 0.85 | 0.64 | 0.48 |
| 33.0 | (14,11,20) | (18,4,3) | 1.16 | 0.88 | 0.66 | 37.5 | (10,6,19) | (11,5,1) | 0.92 | 0.70 | 0.52 |
| 33.5 | (19,20,20) | (20,12,2) | 1.33 | 1.00 | 0.75 | 38.0 | (19,20,19) | (20,6,1) | 1.25 | 0.94 | 0.71 |
| Mobile (EEP PSNRs with 10/15/20 iterations = 26.9/27.1/27.8) | | | | | | News (EEP PSNRs with 10/15/20 iterations = 30.7/32.5/33.2) | | | | | |
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 25.0 | (13,20,5) | (12,5,1) | 0.95 | 0.72 | 0.54 | 29.5 | (7,8,14) | (8,6,1) | 0.87 | 0.66 | 0.50 |
| 26.0 | (13,20,8) | (13,3,1) | 0.96 | 0.73 | 0.55 | 30.5 | (12,8,19) | (13,7,1) | 0.99 | 0.75 | 0.56 |
| 27.0 | (13,20,19) | (12,9,1) | 0.98 | 0.74 | 0.56 | 31.5 | (14,18,11) | (17,8,1) | 1.15 | 0.87 | 0.65 |
| 27.5 | (14,20,10) | (14,8,1) | 1.03 | 0.78 | 0.59 | 32.5 | (15,20,12) | (16,11,2) | 1.15 | 0.87 | 0.65 |
| 28.2 | (15,20,16) | (17,7,2) | 1.14 | 0.86 | 0.65 | 33.4 | (16,20,19) | (19,9,1) | 1.24 | 0.93 | 0.70 |

SNR=3.6dB

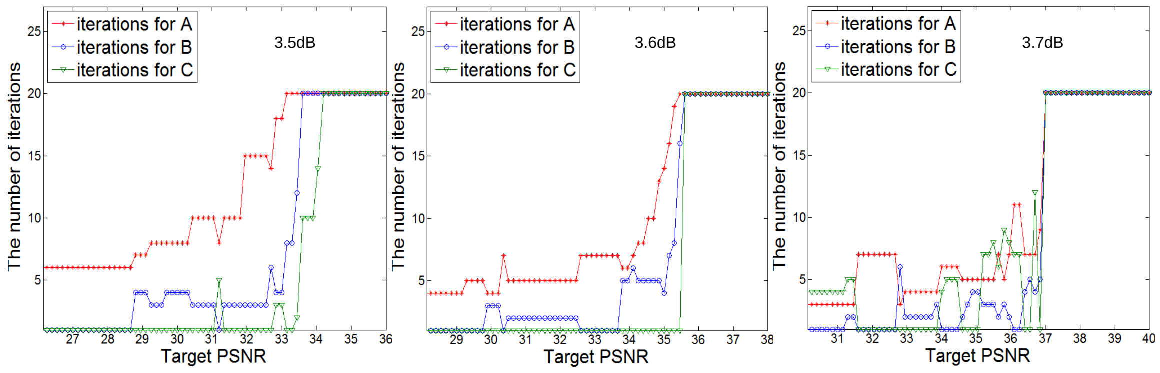
| Foreman (EEP PSNRs with 10/15/20 iterations = 34.3/34.6/35.4) | | | | | | Akiyo (EEP PSNRs with 10/15/20 iterations = 38.2/38.3/38.6) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 32.0 | (4,8,13) | (5,2,1) | 0.82 | 0.62 | 0.46 | 36.5 | (2,20,15) | (3,2,1) | 0.80 | 0.61 | 0.46 |
| 33.0 | (5,12,5) | (7,1,1) | 0.83 | 0.63 | 0.47 | 37.0 | (3,13,19) | (4,3,1) | 0.81 | 0.62 | 0.46 |
| 34.0 | (6,11,5) | (7,6,1) | 0.86 | 0.65 | 0.49 | 37.5 | (4,3,19) | (6,2,6) | 0.84 | 0.64 | 0.48 |
| 34.5 | (8,20,20) | (10,5,1) | 0.90 | 0.68 | 0.51 | 38.0 | (12,7,11) | (13,4,1) | 0.97 | 0.73 | 0.55 |
| 35.2 | (16,20,20) | (19,8,1) | 1.23 | 0.93 | 0.70 | 38.5 | (14,20,15) | (15,7,1) | 1.06 | 0.80 | 0.60 |
| Mobile (EEP PSNRs with 10/15/20 iterations = 29.2/30.8/31.2) | | | | | | News (EEP PSNRs with 10/15/20 iterations = 34.2/34.9/36.0) | | | | | |
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 28.0 | (5,5,11) | (5,5,1) | 0.83 | 0.63 | 0.47 | 32 | (5,3,20) | (6,4,1) | 0.83 | 0.63 | 0.47 |
| 29.0 | (5,6,20) | (6,5,1) | 0.84 | 0.63 | 0.48 | 33 | (5,9,15) | (6,7,1) | 0.85 | 0.64 | 0.48 |
| 30.0 | (5,20,15) | (8,5,1) | 0.87 | 0.66 | 0.49 | 34 | (5,20,10) | (7,8,4) | 0.89 | 0.67 | 0.51 |
| 30.5 | (8,20,15) | (11,7,3) | 0.95 | 0.72 | 0.54 | 35 | (17,20,15) | (20,3,2) | 1.24 | 0.93 | 0.70 |
| 31.1 | (9,20,18) | (17,3,1) | 1.11 | 0.84 | 0.63 | 35.6 | (18,20,19) | (19,8,8) | 1.26 | 0.95 | 0.72 |

SNR=3.7dB

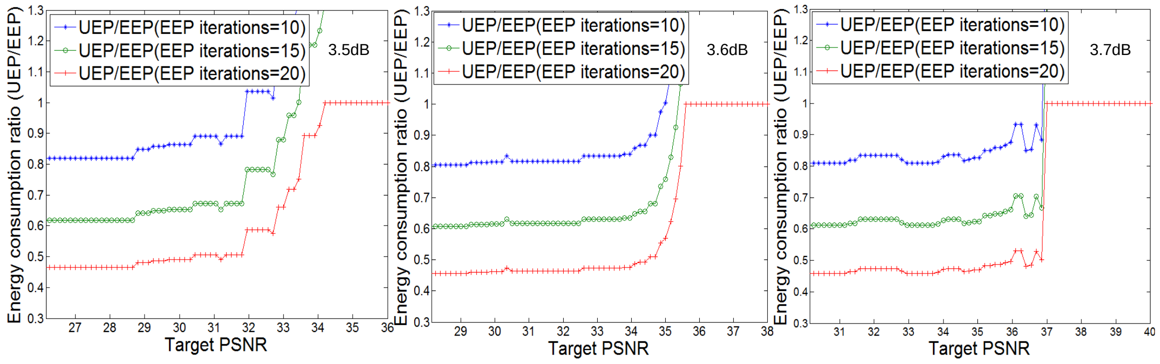
| Foreman (EEP PSNRs with 10/15/20 iterations = 36.9/36.9/36.9) | | | | | | Akiyo (EEP PSNRs with 10/15/20 iterations = 38.7/38.7/38.7) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 32.0 | (3,2,17) | (4,1,1) | 0.80 | 0.61 | 0.46 | 37.0 | (2,1,20) | (4,1,3) | 0.81 | 0.61 | 0.46 |
| 33.0 | (3,7,11) | (5,2,1) | 0.82 | 0.62 | 0.46 | 37.5 | (3,2,14) | (5,1,4) | 0.82 | 0.62 | 0.47 |
| 34.0 | (4,2,17) | (6,1,5) | 0.84 | 0.63 | 0.47 | 38.0 | (3,5,10) | (4,3,1) | 0.81 | 0.62 | 0.46 |
| 35.5 | (4,20,3) | (6,3,7) | 0.86 | 0.65 | 0.49 | 38.4 | (4,12,3) | (5,4,1) | 0.83 | 0.62 | 0.47 |
| 36.5 | (6,20,19) | (9,5,1) | 0.88 | 0.67 | 0.50 | 38.7 | (6,20,9) | (8,6,1) | 0.87 | 0.66 | 0.50 |
| Mobile (EEP PSNRs with 10/15/20 iterations = 34.4/34.4/34.4) | | | | | | News (EEP PSNRs with 10/15/20 iterations = 37.4/37.4/37.4) | | | | | |
| $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ | $PSNR_{target}$ | Coarse | Fine | $E_{10}$ | $E_{15}$ | $E_{20}$ |
| 32.0 | (4,5,13) | (5,4,1) | 0.83 | 0.62 | 0.47 | 33.5 | (4,2,13) | (6,2,2) | 0.83 | 0.63 | 0.47 |
| 33.0 | (5,4,10) | (8,2,1) | 0.85 | 0.65 | 0.48 | 34.0 | (4,3,12) | (7,1,1) | 0.83 | 0.63 | 0.47 |
| 33.5 | (5,4,16) | (11,2,1) | 0.91 | 0.69 | 0.52 | 34.5 | (4,5,10) | (7,2,1) | 0.84 | 0.63 | 0.48 |
| 34.0 | (5,20,16) | (10,6,1) | 0.90 | 0.68 | 0.51 | 36.5 | (4,20,20) | (5,5,1) | 0.83 | 0.63 | 0.47 |
| 34.4 | (11,20,11) | (15,8,5) | 0.91 | 0.82 | 0.62 | 37.3 | (7,6,13) | (8,5,2) | 0.87 | 0.66 | 0.50 |



(a) Coarse search results at 3.5, 3.6, 3.7dB



(b) Coarse + Fine search results at 3.5, 3.6, 3.7dB



(c) Energy consumption results at 3.5, 3.6, 3.7dB

Fig. 20.: (a) Coarse search results, (b) coarse and fine search results, and (c) comparisons of energy consumption ratio in foreman at 3.5, 3.6, and 3.7dB SNRs

## C. Optimal Configuration Search Method for Low-Power MIMO Detector in Embedded MIMO-H.264 Joint Decoding Architecture

The UEP-based MIMO detector design is presented to minimize energy consumption without significant loss in video quality in this section. This approach carries out an empirical analysis, searching the minimum configuration of MIMO detection in terms of power savings. Energy reduction is fulfilled by trading off the search space of MIMO symbols (MIMO candidate vectors, or search paths) and BER associated with the MIMO detection. Reduced (or truncated) search paths lead to degradation in BER performance but reduce energy consumption. To compensate for the BER degradation, we adopt UEP using the H.264 DP method, which partitions image streams into three priority groups, A, B, and C: A) high priority frames, B) medium priority frames, and C) low priority frames [41]. Here, MIMO configuration (or processed paths) denotes the number of MIMO candidate vectors to be searched, and a set of MIMO configurations is defined by [$C_A$,$C_B$,$C_C$], where $C_A$, $C_B$, and $C_C$ are MIMO configurations to detect priority A, B, and C data respectively.

The aim of this scheme is to 1) quantify the relationship between the priorities of video data and MIMO configurations empirically, 2) design a search method such that we used in LDPC-H.264 JSCD system to search minimum energy consuming sets (i.e. the sets of MIMO configurations) to decode prioritized video data under the consideration of reconstructed video quality, and 3) catalogue the searched configurations into a UEP lookup table and use them for the joint MIMO-H.264 decoding in runtime.

### 1. Background: MIMO Detection

Eqn. (2.18) shows a MIMO system composed of $M_T$ transmit and $M_R$ receive antennas [58].

$$y = Hs + n \tag{2.18}$$

where $y$=$[y_1, y_2, ..., y_{M_R}]^T$ is a $M_R \times 1$ received vector, $s$=$[s_1, s_2, ..., s_{M_T}]^T$ is a $M_T \times 1$ transmitted vector, $n$ is a $M_R \times 1$ zero mean complex Gaussian noise vector, and $H$ is a $M_R \times M_T$-dimensional complex matrix. The complex channel gain from the $j^{th}$ transmit antenna to the $i^{th}$ receive antenna can be denoted by $h_{ij}$ in the matrix $H$. MIMO symbol $s_i$ ($i = 1, 2, ..M_T$) is derived from a set $\Omega$ of cardinality $\eta$, complex numbers with their real and imaginary parts of the form $\{-\sqrt{\eta} + (2k-1)\}$ where k=1,2,...$\sqrt{\eta}$. This method is called as a $\eta$-ary quadrature amplitude modulation (QAM). A group of binary bits (size is $\log_2 \eta$) is mapped onto a symbol from $\Omega$, creating the QAM symbols ($s_i$). In Eqn. (2.19), the optimal or the ML estimate $\hat{s}_{ml}$ of $s$ can be achieved by $y$ and $H$ for the MIMO detection [59]:

$$\hat{s}_{ml} = arg \min_{s \in \Omega^{M_T}} ||y - Hs||^2 \tag{2.19}$$

To reduce the cost of searching for $\hat{s}_{ml}$, only a small subset of all the possible vectors can be evaluated, using QR decomposition: $H = QR$, where, $R$ is an upper triangular matrix, and $Q^H$ is the Hermitian of a unitary matrix $Q$. The estimate symbols, $\hat{s}$ is derived as (2.20) [60],

$$\hat{s} = ||y - Hs||^2 = ||\hat{y} - Rs||^2, and\ \hat{y} = Q^H y \tag{2.20}$$

It is well known that this optimization can be considered as a tree search problem with each node of the tree having $\eta$ children [61]. The complexity can be reduced by employing pruning large parts of the tree (sphere decoding algorithm) as shown in [62]. However, we chose to implement a hardware friendly detection algorithm called conditionally ordered successive interference cancelation (COSIC) [63]. In this algorithm the complicated data flow is avoided as opposed to sphere decoding. The resulting ”tree” structure is shown in Fig. 21 (for 64-QAM), where only the root node has 64 children, and the rest of the nodes in the tree have only one child. The key fact we utilized is that the more processed paths (larger search space) in Fig. 21, better BER will be but at higher energy consumption levels.

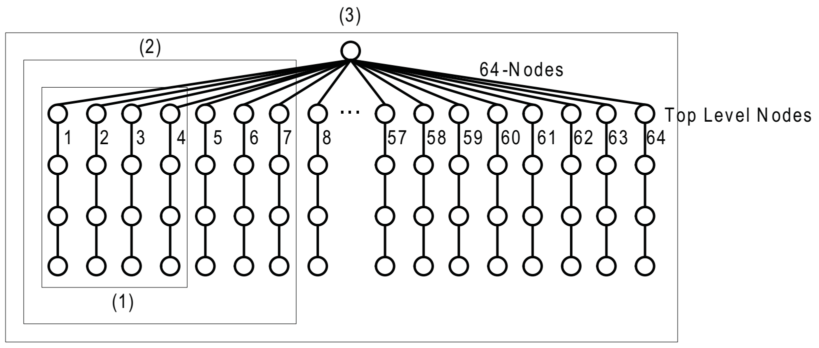


Fig. 21.: Energy scalable MIMO with variable detection effort

We use these two divergent behaviors along with the fact that H.264 inherently has unequal error protection requirements. In view of that, we can scale the energy consumption by varying the number of paths used in the MIMO detection since more/less paths lead to more/less energy needed.

For example, in Fig. 21, (1),(2),(3) indicate three differently sized search spaces that computes 4, 7, and 64. In particular, we have used a variant of the staggered architecture presented in [60]. The only difference is that [60] employs pruning during runtime, but in this work we do not use pruning to reduce the search space, instead, the energy savings is attained by dynamically changing the size of the search space to keep the analysis simple. This analysis can be applied when pruning is present. Fig. 22 shows the MIMO hardware architecture, where the metric computation unit (MCU) is used as a computational node to detect 64-QAM symbols. This detection flow is implemented as a systolic architecture carefully configured to minimize power-costly register/memory accesses. The MCU for each level is composed of shift/add, adder, slicer, and norm computational logics to compute the search function implicated in Eqn. (2.20).

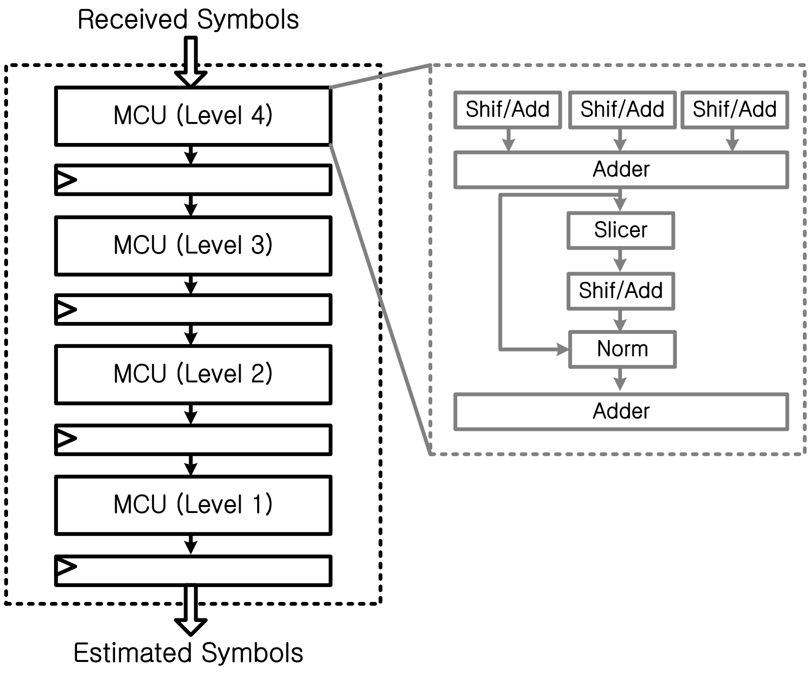


Fig. 22.: MIMO architecture

### 2. The Proposed Low-Power MIMO-H.264 Joint Decoder Design

Our MIMO-H.264 joint decoder design using UEP is shown in Fig. 23. This design mainly consists of two parts: initialization (i.e. pre-process) and runtime execution. During initialization, *1)* a suite of test video streams are encoded through an H.264 video encoding software [42], and *2)* encoded video frames are sent to the search process with the priority information of the frames and MIMO information (BERs and the number of processed paths corresponding to each BER). *3)* The encoded frames are corrupted by injecting errors using BER information and sent to an H.264 software decoder [42] to measure the average PSNRs of decoded pictures. *4)* The measured PSNRs are sent back to the search process, and the sets of MIMO configurations are determined by the presented search scheme. Using this scheme, we quantify the relationship between prioritized video data and MIMO configurations on various PSNRs and find minimum sets for a low-power MIMO detec-

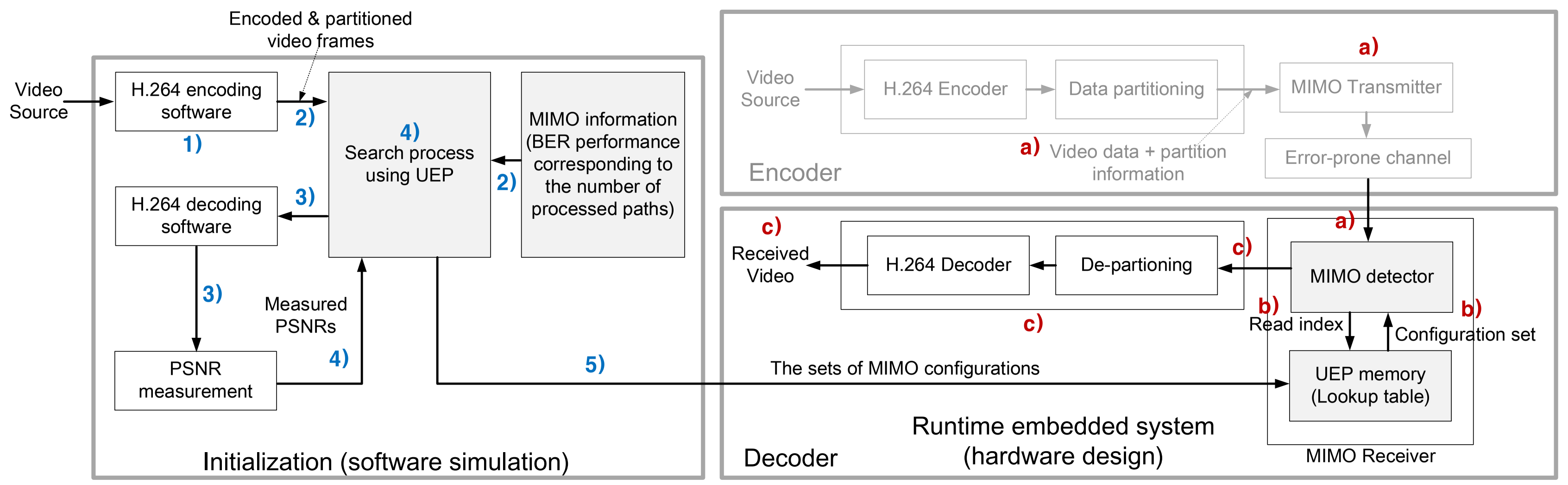

Encoded & partitioned video frames
Video Source
H.264 encoding software
1)
2)
4) Search process using UEP
MIMO information (BER performance corresponding to the number of processed paths)
2)
H.264 decoding software
3)
3)
Measured PSNRs
PSNR measurement
4)
5)
Initialization (software simulation)
Video Source
H.264 Encoder
Data partitioning
a)
MIMO Transmitter
a) Video data + partition information
Error-prone channel
Encoder
c)
Received Video
H.264 Decoder
De-partioning
c)
a)
MIMO detector
c)
b)
Read index
b)
Configuration set
The sets of MIMO configurations
UEP memory (Lookup table)
Runtime embedded system (hardware design)
Decoder
MIMO Receiver

Fig. 23.: The proposed joint decoder system

tor. *5)* Last, the searched sets are saved into UEP memory (a lookup table) for runtime execution.

During runtime, *a)* encoded video streams are sent to the MIMO detector with partition information over error-prone channel. *b)* The MIMO detector generates read indices using a given target PSNR, channel condition (SNR), and the priority information of input frames, reading a set of MIMO configurations from the UEP memory. This is to be done on-the-fly. *c)* The output data of the MIMO detector are sent to an embedded video decoder to reconstruct output images. For this evaluation, the encoded video streams are generated by a software video encoder [42] and Matlab MIMO transmitter (dim blocks in Fig. 23).

The current implementation is based on a static scheme that uses pre-searched configurations for the MIMO detection in runtime. However, these configurations can be re-evaluated if application or channel conditions calls for a change. The optimization problem for searching the minimum energy paths (i.e. the sets of configurations) of the MIMO detector can be now stated by prioritized video sequences, utilization of paths, and PSNRs of reconstructed video frames. The main purpose of our search scheme is to determine a minimum MIMO configurations set [$C_{min,A}$, $C_{min,B}$, $C_{min,C}$] while minimizing PSNR degradation. The search scheme is composed of two search steps: *1)* coarse search and *2)* fine search.

The evaluation of quality degradation in video transmission over wireless channel is necessary for this work since the variable effort of the MIMO detection is also affected by the quality of reconstructed video.

### a. Coarse Search Process

We exploit binary search (BS) to find a coarse estimate of the set of configurations $[C_{co,A}, C_{co,B}, C_{co,C}]$ in the MIMO detection. The coarse set is searched when $PSNR(a,b,c)$ (an average PSNR of decoded video frames at $C_A = a$, $C_B = b$, and $C_C = c$) marginally satisfies target PSNR (the desired quality of reconstructed images), $\tau$. The search is detailed as follows.

1. Find $C_{co,A}$ satisfying $PSNR(C_{co,A}, C_{max}, C_{max}) > \tau$
2. Find $C_{co,B}$ satisfying $PSNR(C_{co,A}, C_{co,B}, C_{max}) > \tau$
3. Find $C_{co,C}$ satisfying $PSNR(C_{co,A}, C_{co,B}, C_{co,C}) > \tau$

where $C_{max}$ is the maximum MIMO configuration, 64. In the coarse search, $C_{co,A}$ is firstly searched since the average PSNR is more sensitive to the impairment of high priority frames. Therefore, when the coarse search is performed in order of importance (or priority), $A$, $B$, and $C$, the search result would be quickly converged. Plus, the initial values of $C_B$ and $C_C$ are $C_{max}$ while searching $C_{co,A}$ because we search the set in the direction of reducing energy consumption (from the large number of paths to the small number of paths : 64$\rightarrow$1).

b. Refining the Coarse Search Using UEP

The fine search refining the coarse search (binary search + fine search: BFS) result shown in Algorithm 4 uses the fact that the increasing of the processed paths of high importance frames leads to enhancement in the average PSNR due to UEP. Hence, a small increment of $C_A$ (x) leads to a large decrement of $C_B$ (y) and $C_C$ (z) while meeting the target PSNR. By this fact, we gradually increase x and decrease y and z until finding a minimum processed paths set $[C_{min,A}, C_{min,B}, C_{min,C}]$. A processed paths set which produces the optimal performance in terms of energy reduction on a given target PSNR is searched as follows (Algorithm 4).

1. Increase $C_A$ and decrease $C_B$ and $C_C$
2. Find a minimum set, $[C_{min,A}, C_{min,B}, C_{min,C}]$ maximizing the number of reduced paths, $[(\Delta C_B + \Delta C_C) - \Delta C_A]$ and satisfying $PSNR(C_{min,A}, C_{min,B}, C_{min,C}) > \tau$

**Algorithm 4** Fine search algorithm refining coarse search result using UEP

```
1: Fine_Search(PSNR,Coarse Configurations Set)
2: $C_{max} = 64, C_{min} = 1$
3: $x = C_{co,A}, y = C_{co,B}, z = C_{co,C}$
4: $C_{min,A} = C_{co,A}, C_{min,B} = C_{co,B}, C_{min,C} = C_{co,C}$
5: $diff_{max} = 1$
6: while $x < C_{max}$ do
7:   $x = x+1$
8:   $y = C_{co,B}$
9:   while $y > C_{min}$ do
10:     $y = y-1$
11:     $z = C_{co,C}$
12:     while $z > C_{min}$ do
13:       $z = z-1$
14:       $diff = (C_{co,B} - y) + (C_{co,C} - z) - (x - C_{co,A})$
15:       if $PSNR(x,y,z) > \tau$ && $diff_{max} < diff$ then
16:         $C_{min,A} = x, C_{min,B} = y, C_{min,C} = z$
17:         $diff_{max} = diff$
18:       end if
19:     end while
20:   end while
21: end while
```

where $C_{min}$ is 1 (the minimum number of paths), and $diff_{max}$ denotes the maximum reduction of processed paths, which implies an amount of energy reduction $((\Delta C_B + \Delta C_C) - \Delta C_A)$. As a result, the minimum sets of MIMO configurations can be searched using the presented UEP scheme.

## 3. Results and Discussion

As aforementioned, the more paths considered for the MIMO detection, the better BER gets. However, energy consumption increases as well. This relationship is shown in Fig. 24. In this experiment, we wanted to keep the un-coded BER better than $10^{-3}$ since the H.264 decoder could not work properly for high impaired data. The MIMO detection's BER at 20dB did not get better than $10^{-3}$, thus we simulated only for the range of 22 and 24dB. The energy consumption of the MIMO detector in Fig. 24 was computed by the synthesis

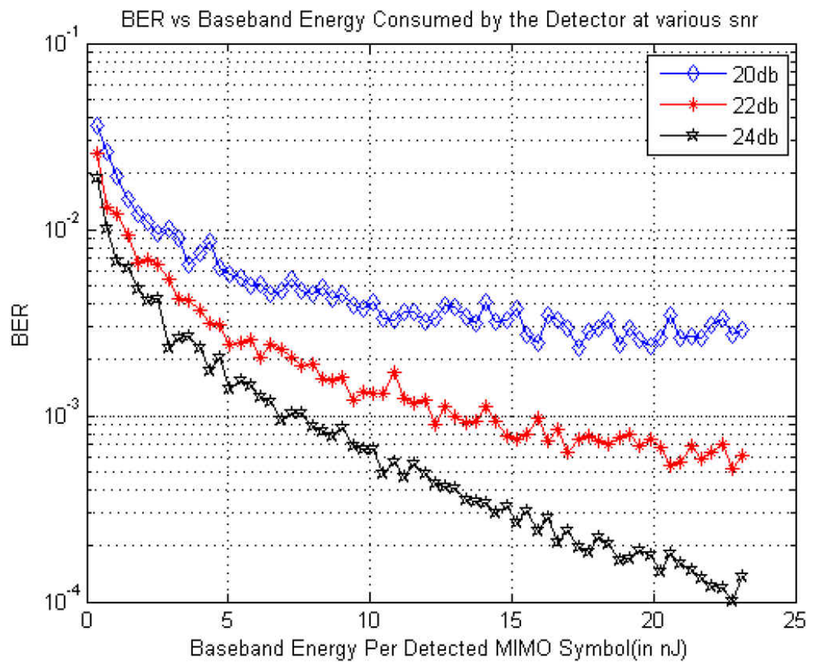


Fig. 24.: MIMO energy vs. BER performance: The more energy is needed for the lower BER.

result shown in Table XII. These design attribute estimates were achieved using Synopsys design compiler [55] and OSU PDK 45nm CMOS predictive standard cell library [64]. The area of MCUs increased because more computations were needed as we went down the tree (i.e. level 4 $\rightarrow$ level 1).

For the evaluation, we simulated three different QCIF (176x144) video sources (Foreman, Akiyo, and Mobile: each 300 frames) generally used for video test at 30fps. The amount of energy reduction varied according to the distribution of priority in a video sequence since a high priority part utilized more resources (processed paths). Table XIII shows the simulation results of the presented UEP-based MIMO design in Foreman, Akiyo, and Mobile video streams. The normalized energy reduction (ER) can be computed by Eqn. (2.21).

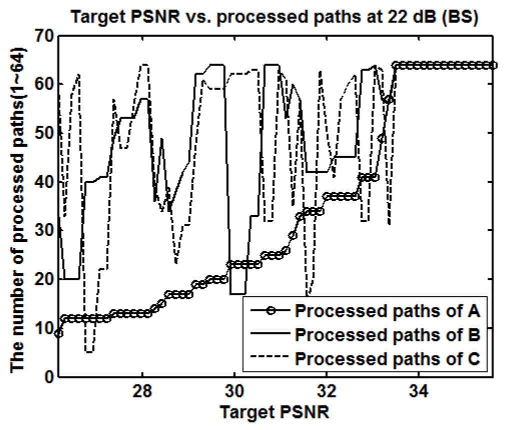
$$ER_{normalized} = 1 - E_{x,y,z}/E_{N=64} \tag{2.21}$$

Table XII.: Synthesis results of MIMO detector using 45nm CMOS predictive standard cell library

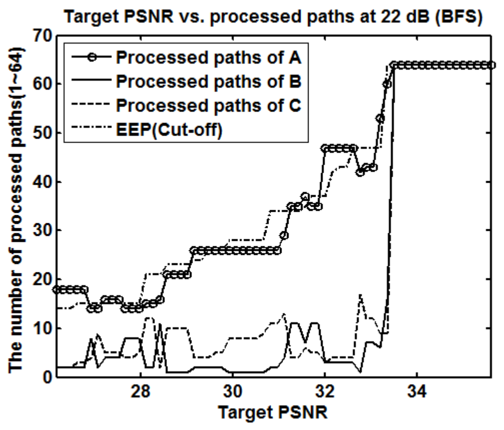
| | Area | Energy (nJ) | Delay (ns) |
|---|---|---|---|
| Level 1 | 7325 | 0.148 | 15.68 |
| Level 2 | 5997 | 0.114 | 14.65 |
| Level 3 | 4158 | 0.076 | 12.84 |
| Level 4 | 2982 | 0.023 | 6.9 |

where $E_{x,y,z}$ is energy consumption at a processed paths set (x,y,z) , and $E_{N=64}$ is energy consumption when using *N=64* paths (maximum EEP energy consumption). In this table, the ER results were attained from UEP-BFS and EEP (N=64). We observed that the results of Akiyo and Mobile were similar to that of Foreman. Although the distributions of the priority group B and C of Foreman, Akiyo, and Mobile were different, the results were analogous one another in terms of energy consumption. The reason was that these video streams had similar distributions in the priority group A. (Group A was dominant in performance and energy consumption because it exploited more paths than B and C.) It leaded to the comparable results in terms of energy reduction.

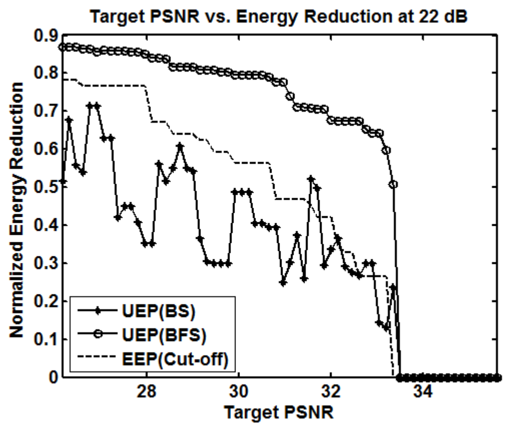
Fig. 25(a) and (d) show the UEP-BS results in the simulation of Foreman at 22dB and 24dB respectively. In these figures, the number of paths for B and C are considerably higher than the number of paths for A. For energy reduction, the fine search algorithm was combined. The UEP-BFS results in Foreman are shown in Fig. 25(b) and (e). In these results, the numbers of utilized paths for B and C decreased under 10$\sim$20 with increasing the number paths for A. It leaded to energy reduction in the MIMO detector. We thus demonstrated the number of processed paths for EEP that equally cut off the MIMO search paths (called ”EEP-cutoff”) regardless of the priority of data for comparison with the UEP results. For instance, when target PSNR is 33.6dB at 22dB SNR, the processed paths set is (57,56,31)



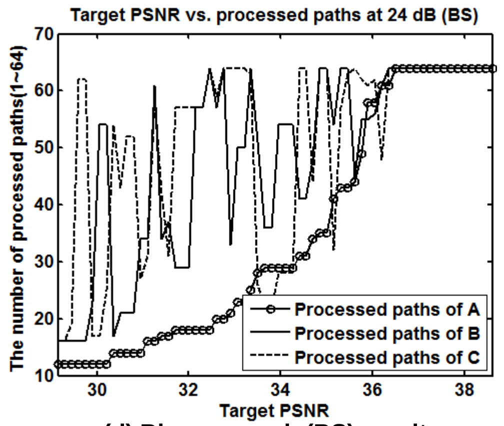
(a) Binary search (BS) result



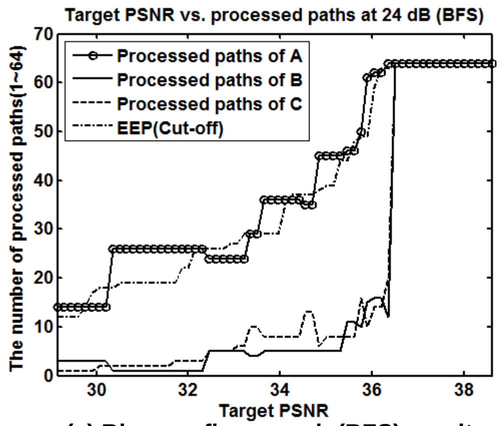
(b) Binary + fine search (BFS) result



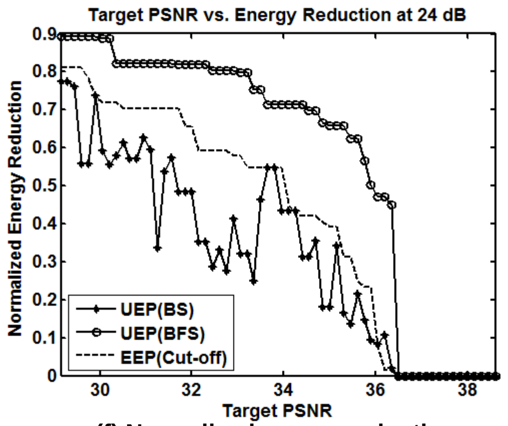
(c) Normalized energy reduction



(d) Binary search (BS) result



(e) Binary + fine search (BFS) result



(f) Normalized energy reduction

Fig. 25.: (a-c) show simulation results at 22dB in Foreman: (a) coarse binary search (BS), (b) fine search (BFS), and (c) the energy reduction of our UEP-base energy efficient MIMO detector, (d-f) show simulation results at 24dB in Foreman: (d) BS, (e) BFS, and (f) energy reduction

Table XIII.: Simulation results at 22dB and 24dB SNRs. It shows the sets of MIMO configurations resulted in the coarse binary search (*BS*) and the fine search refining the binary search result (*BFS*). The percentage value of normalized energy reduction (*ER*) was calculated by Eqn. (2.21).

SNR=22dB

| $\tau$ | Foreman | | | Akiyo | | | Mobile | | |
|---|---|---|---|---|---|---|---|---|---|
| | BS | BFS | ER(%) | BS | BFS | ER(%) | BS | BFS | ER(%) |
| 30.0$dB$ | (23,17,62) | (26,1,8) | 82 | (25,40,52) | (28,10,3) | 79 | (25,20,59) | (27,5,4) | 81 |
| 31.0$dB$ | (25,64,63) | (26,2,11) | 80 | (28,60,59) | (30,7,13) | 74 | (27,60,61) | (29,4,10) | 78 |
| 32.0$dB$ | (37,42,50) | (47,3,3) | 73 | (38,56,60) | (45,6,4) | 71 | (35,63,60) | (39,7,3) | 74 |
| 33.0$dB$ | (41,64,64) | (43,7,12) | 68 | (43,63,61) | (50,9,2) | 68 | (42,64,63) | (46,6,9) | 68 |
| 33.6$dB$ | (57,56,31) | (60,16,9) | 56 | (60,64,55) | (62,12,3) | 60 | (56,64,58) | (60,15,7) | 57 |

SNR=24dB

| $\tau$ | Foreman | | | Akiyo | | | Mobile | | |
|---|---|---|---|---|---|---|---|---|---|
| | BS | BFS | ER(%) | BS | BFS | ER(%) | BS | BFS | ER(%) |
| 32.0$dB$ | (18,29,57) | (26,1,3) | 84 | (19,38,60) | (25,5,7) | 81 | (23,37,49) | (25,3,3) | 84 |
| 33.0$dB$ | (23,50,64) | (24,5,6) | 82 | (27,62,62) | (30,6,10) | 76 | (27,54,58) | (30,4,7) | 79 |
| 34.0$dB$ | (29,54,28) | (36,5,8) | 74 | (34,63,42) | (37,8,11) | 71 | (32,56,51) | (36,3,10) | 74 |
| 35.0$dB$ | (35,64,64) | (45,5,8) | 70 | (39,60,64) | (44,7,12) | 67 | (40,53,60) | (44,7,9) | 69 |
| 36.2$dB$ | (61,61,48) | (62,16,14) | 52 | (60,63,58) | (62,10,11) | 57 | (60,64,63) | (63,10,8) | 58 |

in UEP-BS, (60,16,9) in UEP-BFS, and (56,56,56) in EEP-cutoff. The maximum energy reduction can be achieved by UEP-BFS that decreases considerably the number of paths for B and C through increasing slightly the number of paths for A. Fig. 25(c) and (f) show UEP and EEP-cutoff energy reduction vs. target PSNR in Foreman. In the simulation of Foreman, our MIMO-H.264 joint decoder with 33.6 and 36.2dB target PSNRs yielded 56% and 52% energy reductions respectively while trading off 0.29dB and 0.3dB acceptable degradation, when compared to EEP (N=64) at 22dB and 24dB SNRs.

Although our joint decoder design significantly reduced energy consumption in run-time, it requires additional works such as: 1) a dynamic scheme for updating new configurations in real-time, 2) evaluation and analysis using some problematic test streams, and 3) a baseband logic implementation with a video processor. These issues are currently addressed along with algorithm improvement for low-power video decoding.

## D. Conclusions

This chapter presents a JSCD-based low-power decoder design that uses a novel UEP scheme and DVFS. In Section A, we propose an LDPC-H.264 JSCD scheme for portable applications over AWGN channels, configured by exploiting importance and error severity in each data frame. The proposed JSCD scheme is devised to operate at a fixed frame-decode-time loop regardless of the quality of data received. Within each loop, optimal sub frequencies and voltage levels are dynamically configured to minimize the energy spent for each frame. This design meets the real-time requirements of motion picture reproduction and minimizes overall power consumption. The design is synthesized using TSMC 0.13 micron technology and is capable of jointly decoding QCIF (176x144) video stream at 30 fps over wireless channel with 80% code rate. As a result, up to 39% power reduction can be achieved in Foreman, Akiyo, and Mobile, compared to a fixed-iteration-based joint source channel decoder.

Section B shows the proposed optimal configuration search scheme for LDPC-H.264 JSCD to reduce power consumption of LDPC channel decoder. As a result, the low-power decoding framework not only provides the trade-off between power reduction and video quality, but proposes a method for efficient resource utilization to save power.

We also present a low-power MIMO-H.264 joint decoder design using the optimal configuration searching algorithm based on unequal error protection in Section C. The design is developed for video mobile applications over MIMO and configured to make minimum tradeoffs between energy consumption and performance. The results show that our design significantly reduces overall energy consumption and compromises picture quality negligibly.

# CHAPTER III

# LOW-LATENCY ON-CHIP INTERCONNECT ARCHITECTURE FOR SYSTEM-ON-CHIP DESIGN

We introduce link-level QoS usng UEP for low-power on-chip interconnect and two on-chip interconnect architectures to address this low-latency demand for CMPs.

- Low-power interconnect design for NoC using UEP: Unequal protection against crosstalk induced errors on link wires can result in considerable power savings with acceptable degradation in performance

- WaveSync: Low-latency source synchronous bypass network-on-chip architecture for globally-asynchronous locally-synchronous (GALS) designs

- SDPR: Dual-path router architecture that efficiently exploits path diversity to attain low latency without significant hardware overhead

In the following sections, we discuss the proposed QoS scheme that reduces power consumption on links and WaveSync that facilitates low-latency communication leveraging the source-synchronous clock sent with the data. We also discuss SDPR doubling the number of injection and ejection ports, splitting packets into two halves, and traversing multiple-paths simultaneously to achieve much higher performance in latency.

## A. Link-Level QoS for Low-Power On-Chip Network

In the previous chapter, we exploited UEP to reduce power consumption in JSCD systems. In this section, we present a low-power on-chip interconnect design using UEP against crosstalk induced errors on links. Video packets transmitted on links between nodes can be classified into different priority groups. High priority parts require more efforts in protecting than low priority parts against crosstalk induced errors on link wires. Therefore, we can leverage this to employ a different level of protection to each priority category, exploring tradeoff between power consumption and quality requirement in video decoding.

Capacitive coupling between wires in NoC/SoC paradigm has been widely studied to minimize the impact of crosstalk on signal transition [65, 66]. Increased wire spacing (DBS), shielding of the wires are explored as options for reducing the impact of crosstalk by Arunachalam et al. in [67]. The theory behind self-shielding or crosstalk-prevention codes (CPC) and the methods for generating these code-words is presented by Victor et al. [68]. Pande et al. propose the use of crosstalk avoidance codes (CAC) and modification in the structure of the data packets to incorporate CAC schemes in the NoC data stream to address both crosstalk and energy dissipation [69] .

Error correction codes (ECC) are able to detect and correct the error bits based on an information theoretic model [70]. Unified framework of coding schemes for system on-chip with CAC and ECC to solve delay, power, and reliability problems jointly are presented in [71–74]. A joint error correction coding scheme using duplication with parity (DAP) and triplication error correction (TEC) with Green bus coding for crosstalk avoidances to guard against crosstalk induced errors is presented in [75, 76] respectively.

We adopt TransSync and RecSync schemes [77] to provide different levels of protection on different priority groups. TransSync and RecSync allow them to be switched on and off dynamically as and when required. In this approach, high priority data can be protected by TransSync and RecSync against crosstalk induced errors, but the protection schemes for the low priority parts can be switched-off due to a relatively small impact on the reconstructed video quality.

### 1. TransSync-RecSync Technique

TransSync-RecSync technique mitigates crosstalk induced transition skew [77]. TransSync preemptively eliminates transition skew amongst the bit-lines by calculating the expected link traversal delay for each transition on the fly. RecSync tries to eliminate the accrued intra-flit on link wires at the receiving node by forcing all the transitions to become aligned before they are relayed to the receiving buffers. Using TranSync and RecSync along with UEP can substantially reduce energy on links at the cost of relatively small loss of system performance (i.e. slight degradation in reconstructed video quality). In the result and discussion, we will evaluate the merits of UEP exploiting TranSync and RecSync on NoC with H.264/VC test data streams.

### 2. UEP with TransSync-RecSync on NoC

Table XIV shows the different configurations of protection schemes studied to evaluate the benefits of UEP on links against crosstalk induced errors. We choose to implement UEP with only TransSync 2 lines, TransSync and RecSync schemes since these schemes have the highest performance in terms of energy savings vs. quality loss. We will discuss this performance comparison in the results and discussion subsection.

TransSync and RecSync techniques are also dynamically switched on/off to provide different levels of protection for different priority data on the same link. Since the BER performance of scheme employed for protecting higher priority parts is better than those of the schemes used with lower priority parts, more energy is typically spent on securing higher priority parts of the data. Table XIV shows all possible UEP configurations obtained by combining RecSync, TransSync and TransSync 2 lines. In these combinations, protection schemes for lower priority data have the same or worse BER performance when compared to those for the higher priority parts. Case 4, Case 7 and Case 10 are EEP schemes employing the same level of protection for all priority data parts. Case 15 is an example of UEP which uses RecSync module for eliminating intra-flit skew for only $Priority_A$ data parts. During the transmission of $Priority_B$ and $Priority_C$ data parts in Case 15, the RecSync module is switched off at the receiving node and TransSync 2 lines circuit is switched on

Table XIV.: Protection schemes employed on links with data partitions for different UEP configurations studied

| Configuration | *Priority* $_A$ | *Priority* $_B$ | *Priority* $_C$ |
|---|---|---|---|
| Case 1 | Baseline | Baseline | Baseline |
| Case 2 | TransSync | Baseline | Baseline |
| Case 3 | TransSync | TransSync | Baseline |
| Case 4 | TransSync | TransSync | TransSync |
| Case 5 | TransSync 2L | Baseline | Baseline |
| Case 6 | TransSync 2L | TransSync 2L | Baseline |
| Case 7 | TransSync 2L | TransSync 2L | TransSync 2L |
| Case 8 | RecSync | Baseline | Baseline |
| Case 9 | RecSync | RecSync | Baseline |
| Case 10 | RecSync | RecSync | RecSync |
| Case 11 | RecSync | TransSync | Baseline |
| Case 12 | RecSync | TransSync | TransSync |
| Case 13 | RecSync | RecSync | TransSync |
| Case 14 | RecSync | TransSync 2L | Baseline |
| Case 15 | RecSync | TransSync 2L | TransSync 2L |
| Case 16 | RecSync | RecSync | TransSync 2L |

at the transmitting node.

## 3. Results and Discussion

### a. Simulation Environment

For the comparison of crosstalk avoidance and error correction performance between protection schemes, we evaluated the PRNR of the reconstructed video streams with EEP. We measured PSNRs when the different crosstalk avoidance and error correction schemes were employed to protect against crosstalk induced errors on links for all data partitions. Fig. 26 shows the simulation setup used for evaluating the quality of reconstructed video streams with different protections schemes on links under EEP. The encoded video frames were impaired by randomly flipping bits in accordance with the BER of protection scheme em-

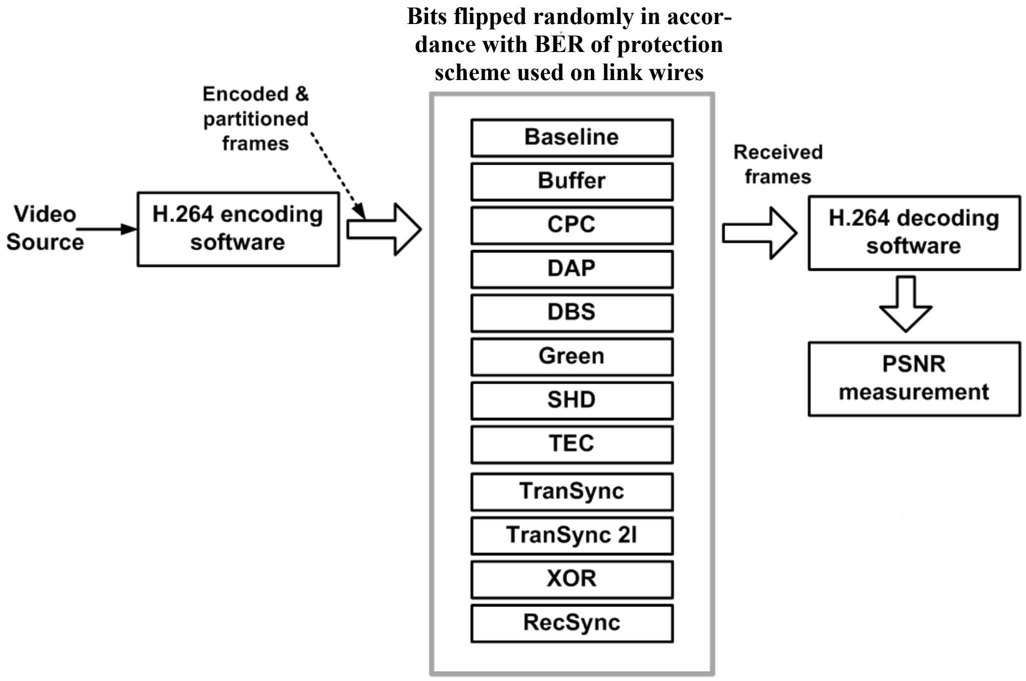


Fig. 26.: Simulation setup for evaluating the quality of reconstructed frames with different protection schemes on links

ployed on the link wires before they were sent to the sender/receiver circuit. The received frames were sent to an H.264 software decoder [42] and the decoder reconstructed video frames to measure the average PSNRs. Damaged macro-blocks were concealed using motion copy [39] during the frame reconstruction process.

We computed a design metric called *Merit* (Eqn. 3.1) to quantify performance and overhead. *Merit* is the ratio of the average PSNR of the reconstructed video stream with the protection scheme employed on links and the product of the normalized area and normalized energy consumption in the scheme. The area and energy consumption of all schemes are normalized to that of the baseline design to obtain the Merit figures.

$$Merit = \frac{PSNR}{NormalizedArea \times NormalizedEnergy} \tag{3.1}$$

Fig. 27 and Fig. 28 show the merit of different protection schemes for 2*mm* and 3*mm* long link wires. In these figures, TransSync 2 lines, TransSync, and RecSync schemes present the highest *Merit* amongst all the schemes studied. That is the reason why we exploited TransSync and RecSync for UEP on on-chip links.

### b. UEP Results

Fig. 29 and Fig. 30 present the BER performance of the different UEP configurations analyzed, arranged in the increasing order of their average power consumption from left to right for 2mm and 3mm long links respectively. The lowest tolerable average BER levels vary between different applications. Fig. 29 and Fig. 30 enable the designer to select the UEP configuration with the least energy consumption to meet a specified BER performance. For example, if the PEs are placed 3mm apart on the die and the application requires a minimum average PSNR of 35dB (dotted line in Fig. 30) for Akiyo streams, then the leftmost UEP configuration in Fig. 30 for which the average PSNR is greater than 35dB should be chosen as the solutions. For the given example, Case 15 offers an average PSNR performance of 35.04dB at approximately 20% lesser overall energy consumption when compared to Case 15 which offers 38dB of average PSNR. We have therefore presented a design methodology which allows the designers to achieve the required performance levels with the least energy consumption using unequal error protection on link wires against crosstalk induced errors.

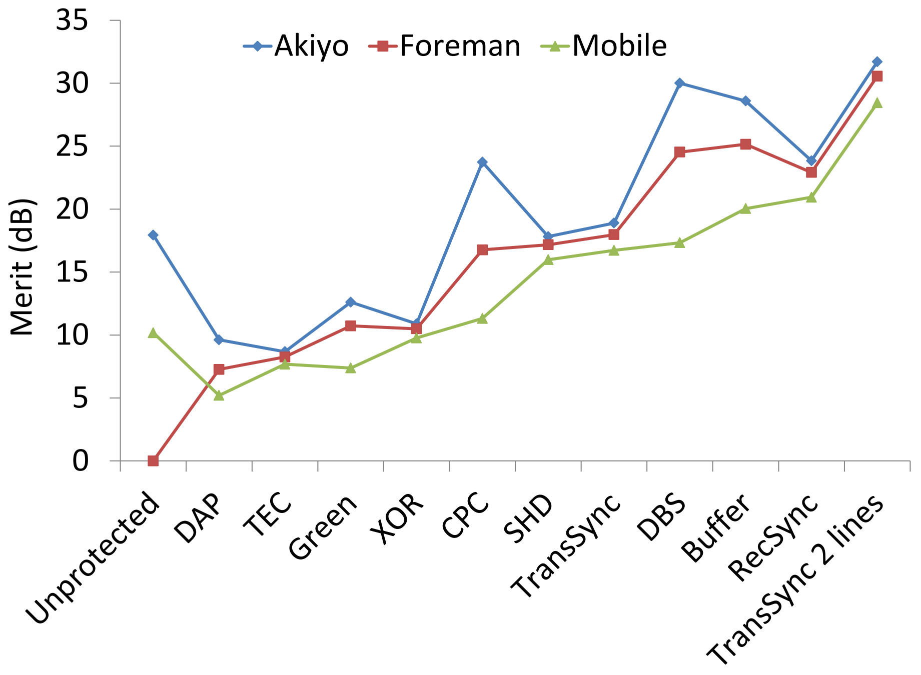


Fig. 27.: Merit of different protection schemes on 2mm long link wires

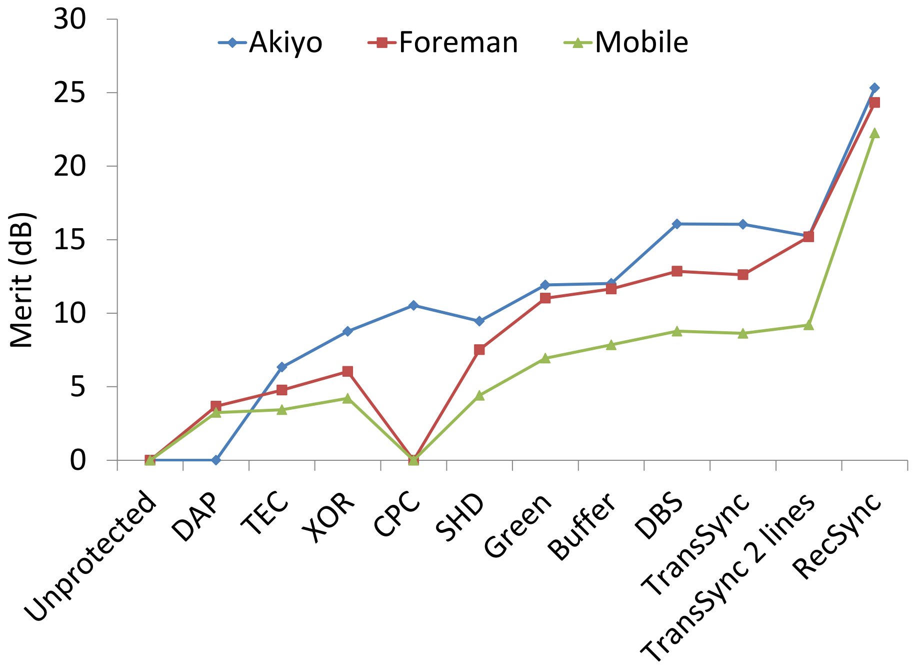


Fig. 28.: Merit of different protection schemes on 3mm long link wires

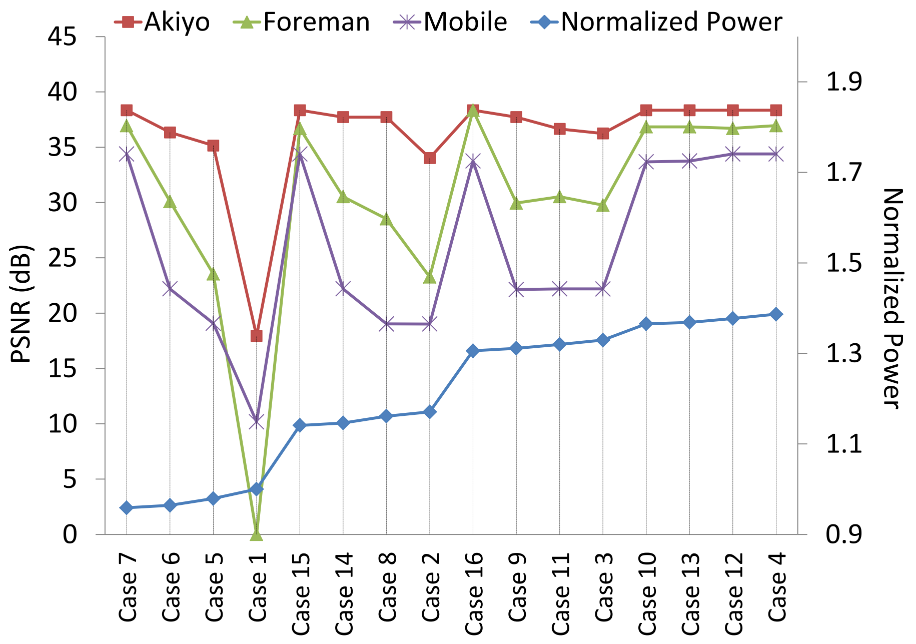


Fig. 29.: Results for 2mm long link wires for the UEP schemes analyzed

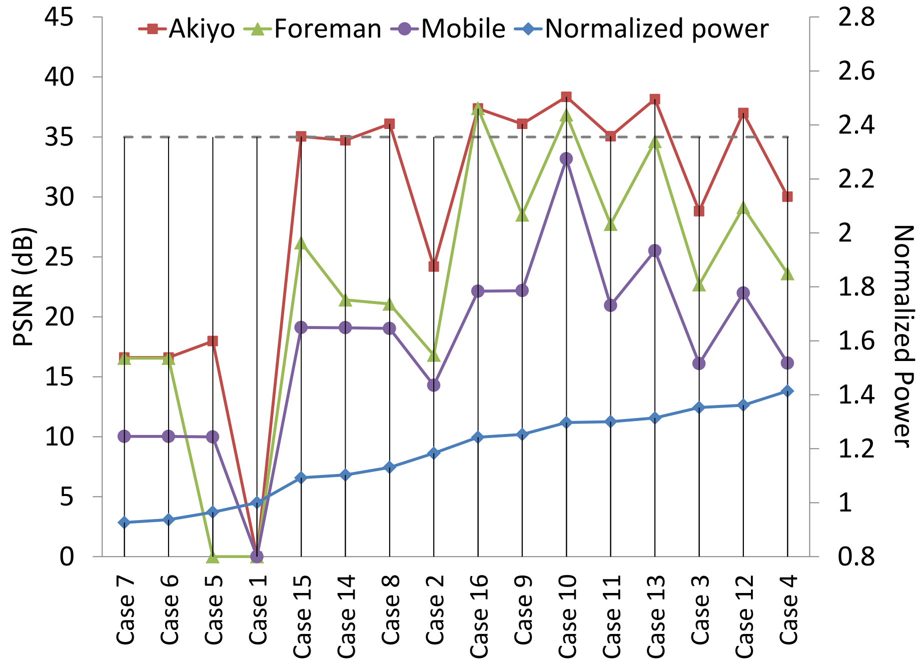


Fig. 30.: Results for 3mm long link wires for the UEP schemes analyzed

## B. WaveSync: Low-Latency Source Synchronous Bypass Network-On-Chip Architecture

As greater numbers of devices are connected to the same clock tree, the power consumption necessary to ensure all nodes are fully synchronous is becoming prohibitive [78]. Globally-asynchronous, locally-synchronous (GALS) clocking has been proposed to reduce the power burden of globally synchronous clocking, at the cost of the synchronization required for cross-clock domain communication [79]. Achieving performance in current multi- and future many-core architectures requires solutions which address their cross-clock-domain communication needs [80, 81]. They also require redundant process to reconfigure asynchronous channel or asynchronous components to support a simple arbitration.

A significant issue in a fully synchronous design is clock tree power consumption. Studies have shown that synchronous clock trees can consume as much as 30$\sim$40% of the total chip power in real designs [82–84]. These power costs are expected to increase in future process technologies as device count increases. The GALS design style is a well known technique to reduce clock tree power consumption [79]. GALS designs are composed of large synchronous blocks which communicate with each other asynchronously. By eliminating the global clock, a major source of power consumption is eliminated. Asynchronous inter-core communication in these designs is typically achieved by through source synchronous data transmission (i.e. the transmit clock is sent with the data), followed by receiver synchronization. Synchronization is required to avoid metastability as packets cross the mesochronous clock domains in GALS NoCs. This synchronization, however, accounts for a major portion of the communication latency, impacting system performance [12]. Synchronization can often double or triple the latency overheads of inter-tile communication, exacerbating larger NoC designs' already substantial packet latency issues.

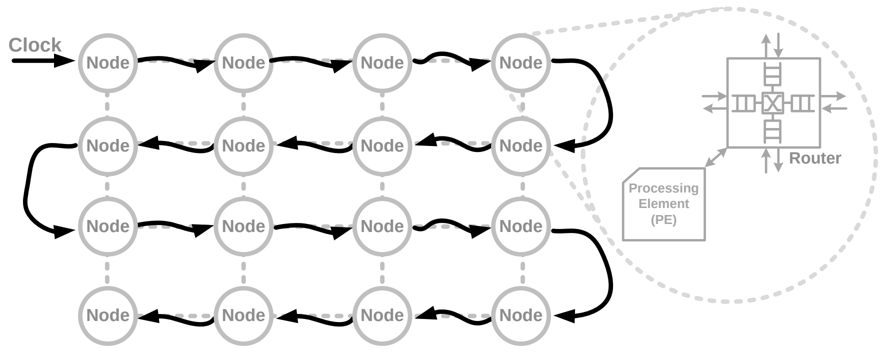


Fig. 31.: Typical GALS clocking scheme on 4x4 mesh NoC

Fig. 31 shows a diagram of a typical GALS CMP, composed of processing elements (PEs) and network adapters interconnected via a mesh NoC and clocked with a GALS clocking scheme. A GALS clock is ”lazily” propagated from node to node in a daisy-chain-like arrangement. We observe that, in this typical GALS scheme, not only is the clock propagated to each node via the clock net; the NoC must also propagate a clock along with the NoC link to provide for synchronization of data transfer between clock domains. This clock redundancy represents both waste and an opportunity which we will leverage the WaveSync design. Since the link already contains a clock signal for data synchronization, we propose using the datas clock as the clock for the entire processing node connected to that the router. Maintaining synchronization between the link and processing element eliminates the need for send and receive synchronization on to the network. Further, since clock is already propagated with data in any source synchronous GALS design, we allow the packet to propagate without latching at each intervening hop in our design. We also exploit a flow control that bypasses buffering/pipelining when there is no congestion, similar to a technique proposed by Jain et al. [85]. In this scheme, flits which traverse a given router on their way to more distant nodes may accumulate only wire and crossbar logic delays each hop, without incurring additional latency due to needless latches.

Therefore, WaveSync can attain low latency via avoiding synchronization and bypassing input buffers at the intermediate nodes on a multi-synchronous network in low congestion levels. This approach outperforms even fully synchronous designs, turning the liability of mesochronous communicating clock domains into a feature.

The following subsections discuss the proposed WaveSync clock distribution and network, router, and synchronization architecture. We also propose We also evaluate the presented design and provide simulation results and implementation.

## 1. WaveSync Design

In a typical GALS NoC, each incoming link contains a source synchronous clock signal, which is used to synchronize the incoming data into the local clock domain/node. By contrast, WaveSync uses incoming clocks to time the routing components (and the PE itself if that link is the designated clock source for it), eliminating the need for synchronization between the incoming data and the local node assuming no turns are needed. In either case, a dedicated clock distribution network is unnecessary and thus, the difficult problem of building a fully synchronous clock network is avoided.

In WaveSync, straight path packets are stored in FIFOs only in the event that its header arrives during transmission of another source's packet or if no available downstream credit is available. Our implementation assumes a 2-D mesh topology, credit based, and wormhole flow control where each packet is divided to header, body, and tail flits [86]. A dimension order routed (DOR) 2-D mesh wormhole router typically allows turns from the first ordered dimension to the second. These turns, however, imply complexity in terms of crossbar and VC allocation and crossbar ports. To reduce router complexity, the WaveSync router disallows all left turns and removes most VCs; any arriving packet may only go straight, turn right, or enter the local PE. This ”Right-turn only” routing is a new, deterministic, minimal routing algorithm we propose, co-designed explicitly to reduce the need for synchronization given our network and clock architecture. This routing algorithm favors paths that are less likely to require synchronization. This algorithm requires one extra VC in one ordinal direction to ensure deadlock freedom. The primary purpose of our right-turn only routing policy, a key contribution of the work proposed here, is to reduce latency, first by minimize the number of synchronizations in a network, and second by reducing router complexity, hence reducing router pipeline stages, through minimizing the number of legal paths through the crossbar. While we expect that reducing routing policy choices, as the right-turn only algorithm does, will decrease load balance in the network somewhat; this comes at the benefit of lower low-load latency. Taken as a whole, these router implementation choices have the effect of trading some bandwidth for lower latency; however, we note that bandwidth is easily attained through extra link width and/or extra network layers, while lower latency is much more difficult to attain.

#### a. Clock Distribution

We evaluated a number of clock assignment patterns possible with our scheme. Among the many possible patterns, three patterns, A, B, and C, are chosen for evaluation and shown in Fig. 32. The arrows in PE blocks denote the direction of clocks that individual PEs select for clocking internal logic. Each pattern was examined to determine the PE clock assignment pattern that minimized overall latency in the network. To evaluate the relative merits of each pattern, we computed the average number of synchronizations required under no-load, uniform random traffic. In Fig. 32(a), pattern A uses northbound clock (south clock) for all PEs. In Fig. 32(b), pattern B, two clock sources, east and west clocks, are used for each half PEs located in east half and west half respectively. Pattern C exploits east, west, north, and south incoming clocks for the nodes of each quadrant as shown in Fig. 32(c). Fig. 32(d,e,f) shows the number of synchronizations necessary to arrive at any given node for data originating from the **S** node on the three patterns shown in

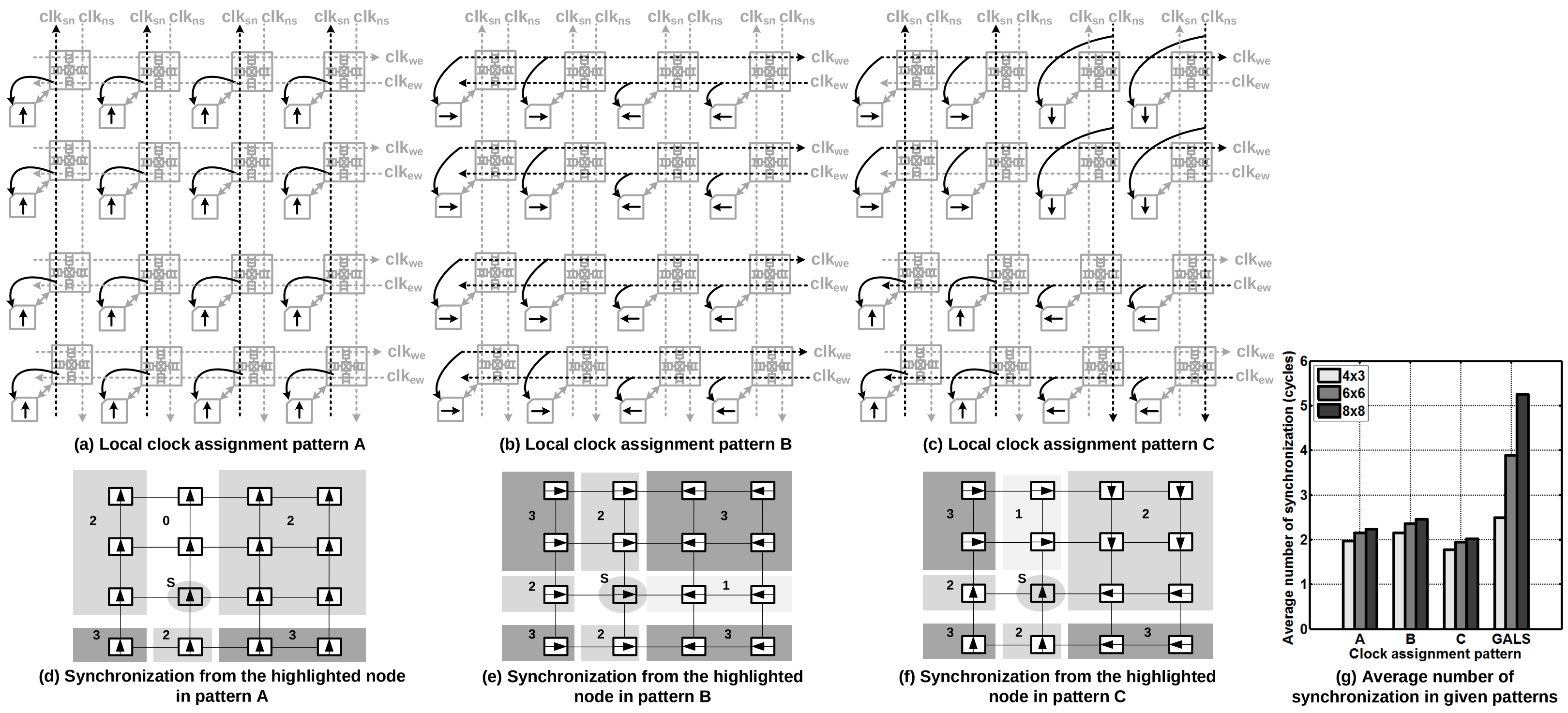

clk$_{sn}$ clk$_{ns}$
clk$_{we}$
clk$_{ew}$
(a) Local clock assignment pattern A
(b) Local clock assignment pattern B
(c) Local clock assignment pattern C
(d) Synchronization from the highlighted node in pattern A
(e) Synchronization from the highlighted node in pattern B
(f) Synchronization from the highlighted node in pattern C
Average number of synchronization (cycles)
4x3
6x6
8x8
A
B
C
GALS
Clock assignment pattern
(g) Average number of synchronization in given patterns

Fig. 32.: WaveSync clock distribution and network : pattern A is clocked by single local clock (south clock), pattern B uses two local clocks (west and east clocks), and pattern C employs a different local clock (east, west, south, or north) in each quadrant. The arrow denotes the direction of clocks in a node, and each number in a node group in (d),(e),(f) denotes the number of synchronizations needed when sending packets from source (**S**) to each node

Fig. 32(a,b,c). Packet propagation with zero synchronization is possible when incoming, outgoing, and local paths are in the same clock domain. For instance, the synchronization count from **S** to upper two nodes is zero in Fig. 32(d) because source clock (outgoing clock), intermediate node traversing clock (incoming clock), and destination clock (local clock) are in the same clock domain (i.e. same direction).

Fig. 32(g) shows average number of synchronizations in three patterns and baseline GALS, based upon the assumption that all sources produce packets for all destinations with equal probability (results analytically determined). In baseline, every packet is latched at each node, requiring synchronization; hence, the average number of synchronizations for GALS largely increases with network size. Clock pattern C, a symmetric and right turn bounded pattern, yields the minimum average number of synchronizations when used in conjunction with the right-turn only routing. In this pattern, data and clock are most likely to be in the same direction (i.e. not in need of synchronization). Thus, we will use this clock distribution pattern (pattern C) for the remainder of this work.

### b. Router Microarchitecture

Fig. 33(a) shows a block diagram of the WaveSync router, with output submodules for east, west, north, and south ($R_e$, $R_w$, $R_n$, $R_s$), associated with output ports for each of the orthogonal directions. On the figure we see the four data inputs (Nin, Sin, Win, Ein), four data outputs (Nout, Sout, Wout, Eout), a local data input (Lin), four local data outputs (Louts), and clocks for the incoming data (clk$_{ns}$, clk$_{sn}$, clk$_{we}$, clk$_{ew}$). Each submodule is clocked with its designated clock source, and is of its own clock domain. In Fig. 33(b-c), the clock and data routes for each output submodule are denoted in black (e.g. in Fig. 33(b), the output submodule uses the clock received from the south link, clk$_{sn}$).

Each output submodule and port may select among three incoming paths (bypass, right-turn, and local in). The bypass path is clocked by the upstream clock and does not

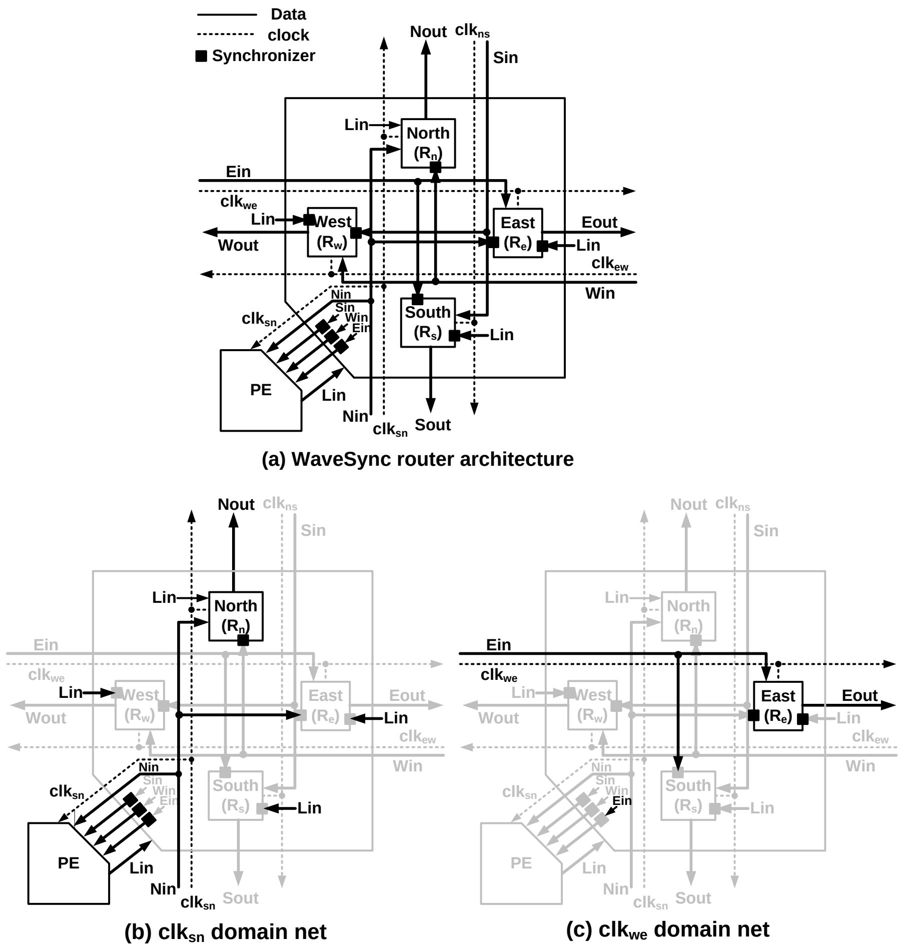


Fig. 33.: (a) WaveSync router top block and (b-c) north and east clock domain nets in WaveSync

require synchronization; however, packets coming from a different input port than the one clocking the PE should be synchronized before they are consumed. In this example, we assume the PE is clocked by clk$_{sn}$ (i.e. it is a router from the southwest quadrant), hence the north output submodule ($R_n$) does not require synchronization for the traffic injected from the PE, nor the traffic received from the south link addressed to propagate northward. Further, the incoming traffic from the south can enter the PE port without synchronization since the PE clock is originated from the source-synchronous clock coming in from south.

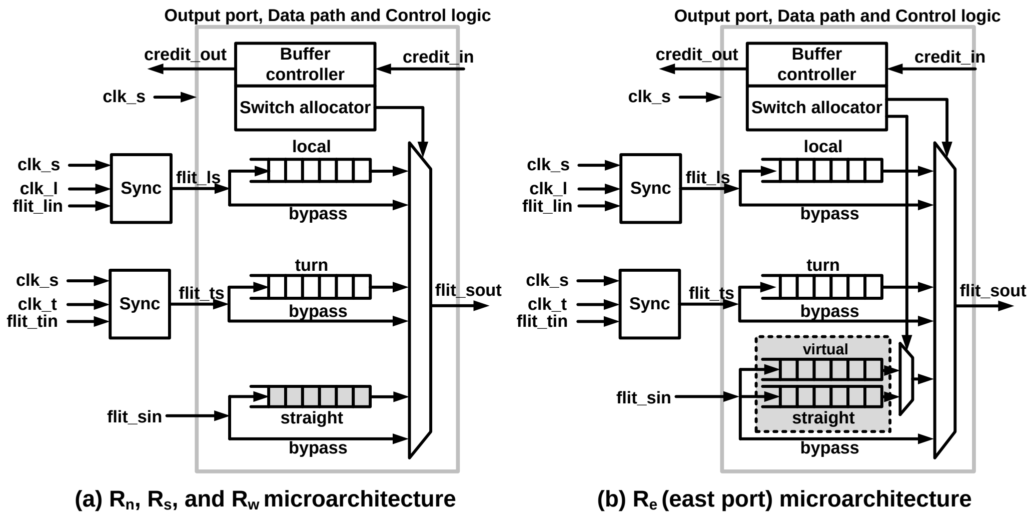


Fig. 34.: (a) Microarchitecture of north, south, and west output submodules and (b) microarchitecture of east output submodule including a virtual channel for deadlock avoidance

The data incoming from right turn path (Win) must be synchronized because the incoming data is clocked by a different clock domain (clk$_{ew}$). Since the local clock (i.e. processing element clock) is sourced from the south clock that is different from the propagation clocks of south, east, and west output submodules ($R_e$, $R_w$, $R_s$), local input/output and turn paths require synchronization. As a result, total number of synchronizers required for the WaveSync router top block is ten because $R_e$, $R_w$, and $R_s$ each need two synchronizers, $R_n$ requires one (for right turn path since the turn path is in different clock domain), and three for south, west, and east local output data.

Fig. 34 shows the microarchitecture for each of the output submodules. Each output submodule is connected to three input ports, straight (flit_sin), turn (flit_tin), and local (flit_lin) and incoming clocks (clock_s, clock_t, clock_l). Three FIFO paths (east logic contains four FIFO paths due to an extra virtual channel) and three bypass paths are connected to the output multiplexer of the router. Flits which traverse the router along the straight

Table XV.: Flit structure

| Field | Field description |
|---|---|
| Packet type | 00 : none, 01 : head flit, 10 : body flit, 11 : tail flit |
| D0: initial direction | 00 : north, 01 : south, 10 : east, 11 : west |
| D1-12: 2-bit routing address for next hops | 00 : straight, 01: turn, 10:stop |

path (flit_sin), are not synchronized since the flits are clocked by straight (i.e. bypass) clock (clock_s) which is the same as their output clock. Flits incoming from the turn input must be synchronized to the straight clock because they are in different clock domain (clock_t), and multiplexed output data are transferred to the next node along with the straight path clock (clock_s). If clock_s is the same as the local clock (clock_l) timing the PE, then no synchronization of local output flits for the PE is required, yielding a reduction of latency. Internal output submodule logic, such as FIFOs, buffer controller, and switch allocator operate on the straight clock. Packet routing information, denoting straight, turn right, or enter the PE, is decoded and used on the fly as header flits of the packets are propagated through the router.

For buffer management, we use a credit based flow control. If lack of upstream credit (credit_in) indicates there is congestion, then all incoming packets will be buffered into local FIFOs. The switch allocator controls the output multiplexer switching bypass mode or buffer mode based on the traversal paths of incoming packets, the status of buffers owned by the node, and credit_in signal. A two input mutiplexer is exploited to select a virtual channel or a straight channel in the east port (Fig. 34(b)).

Fig. 35 and Table XV show a flit format in WaveSync. A header flit consists of 2 bits of packet type for header, body, and tail information and up to $(2N-1)\times 2$ bits of source encoded route, where $N$ is the network diameter (e.g. a maximum of 26-bits of source information can be encoded in the header for a 7x7 NoC). The first 2 bits of the source

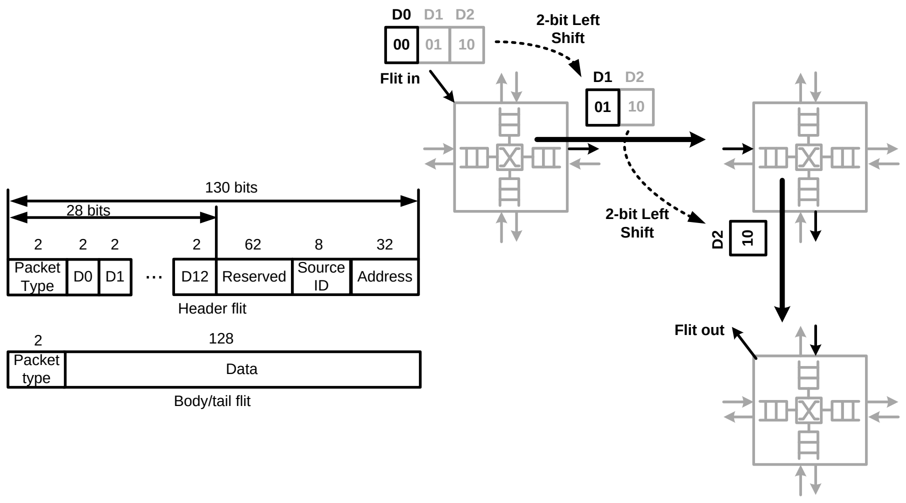


Fig. 35.: Flit format in WaveSync and source route decoding using a shifter logic

encoded route indicate a routing direction (north, south, east and west) at the initial node. After each hop, the current routing bits (2 bits) are shifted away and new routing bits used at the next node. This simple source routing scheme is used to reduce the logical complexity of routing and the potential for skew between the routing bits and the remainder of the header as it propagates along the bypass path. This scheme requires no actual shifting logic as it is a static wire rename operation essentially; the two bits used in a given router are not propagated and the remaining wires are renamed, 0s are inserted into the missing bits. Using this logic, the next hop information can be retrieved at the node without impacting the skew between the flits and its clock signal. The remainder of the header flit can be used for address and source ID, up to the bus bit-width of 130 bits. Body and tail flits are composed of 2 bits of packet type and 128 bits data. Given the wide bit-widths available in on-chip interconnection networks, header flits are not typically fully utilized. Therefore, we argue that source encoded routing should not require any extra packet flits for moderately large networks. If header bits become a constraint we could shift to a denser form of source route encoding, at the cost of slightly more frequent de-skew operations.

### c. De-skewer for suppressing intra-flit skew on links

The movement of flits on bypass paths in low latency NoC designs [87–89] like WaveSync is very similar to the flow of data in wave pipelines [90]. Like wave pipelining, different bits of a given flit which bypasses through several hops without synchronization or latching at the intervening hops, experience different wire delays. Factors which contribute to temporal skewing between bits belonging to the same flit as the flit makes its way on asynchronous bypass paths in low latency designs include: crosstalk coupling between link wires, design irregularities, timing variations at switching and multiplexing logic on the bypass paths, process and temperature variations and physical changes like electro-migration. Since the source synchronous clock is also carried on the links, the use of this incoming clock signal on the link to facilitate bypass and also to synchronize incoming data bits to the local clock at the receiving node can lead to packet errors from violation of setup and hold constraints. It is therefore necessary to periodically eliminate the accrued skew between bits of flits as they are bypassed.

At the 45nm technology node, given a positive edge triggered system with clock period of $1ns$, we determined via spice simulation that skewing between bits of a given flit would be less than half the clock period, when asynchronously bypassed over three hops [77]. De-skewing can therefore be easily achieved by latching the bus data to the negative edge of source synchronous clock every three hops. The inverted source synchronous clock used for latching now becomes the new source clock on links upon de-skewing. To guarantee error free operation, we only need to ensure that the de-skewing is performed every time a flit travels three hops on the bypass paths. The placement of de-skewing blocks for static de-skewing on a 7X7 mesh is done by simply turning off the bypass mode and latching all the traffic at the 3rd column and 3rd row nodes.

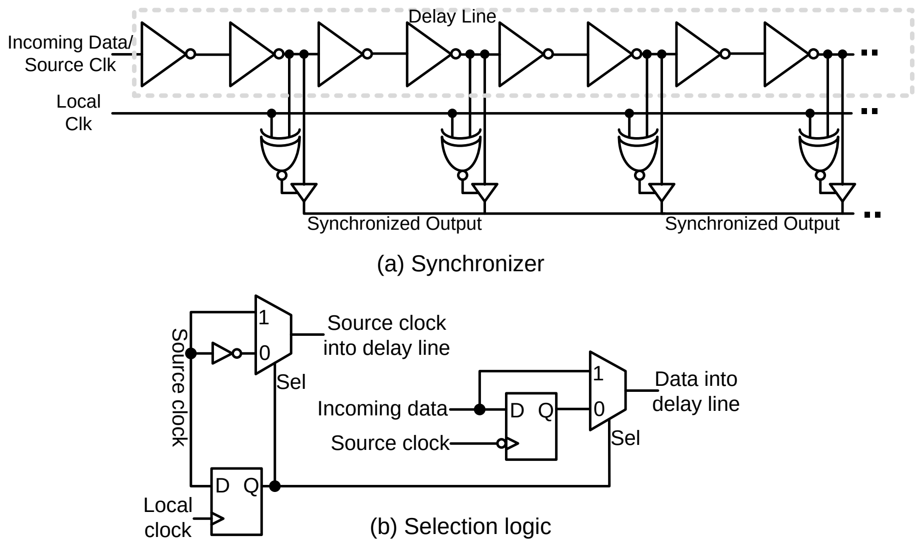


Fig. 36.: Proposed synchronizer: (a) schematic of the proposed synchronizer, (b) selection logic for the proposed synchronizer

d. Synchronizer architecture for half cycle synchronization latency

The proposed synchronizer, shown in Fig. 36(a), calculates the skew between the source clock arriving on the link and the receiving node's local clock, and applies appropriate compensation for the calculated skew to the incoming data to synchronize them to the local clock with an average synchronization latency of only half a clock cycle. The incoming source clock along with each incoming data bit is successively delayed by the delay cells comprised of inverters on similar delay lines. At each delay stage of the delay line, the delayed incoming clock signal is compared to the local clock by taking the exclusive NOR of the two signals. The output of the exclusive NOR gate drives the enable signal of a tri-state buffer and outputs of all the delay stages are hard-wire ORed. When the incoming clock has been delayed appropriately on the delay line such that it is synchronized to the local clock, the output of the exclusive NOR gate is logic one for the entire period of the local clock and the data from this delay stage is passed to the output. The delay is capable of providing a phase skew of $\pi$ radians. If the incoming clock signal lags the local clock by a phase less than $\pi$ radian, the incoming data bits need to be delayed appropriately. On the other hand, for 50% duty cycle clock if the incoming clock lags the local clock edge by a phase greater than $\pi$ radian, then the incoming data bits need to be delayed by a phase equivalent to the skew between the rising edge of the local clock and the falling edge of the incoming clock and another $\pi$ radians. Fig. 36(b) shows the circuit that performs this selection.

## 2. Experiments and Evaluation

In this evaluation, we first discuss our simulation methodology. This is followed by our evaluation of WaveSync's performance versus competing designs under synthetic and realistic workloads.

### a. Simulation methodology

We evaluated two versions of our proposed WaveSync NoC: 1) WaveSync using a typical BIFIFO; and 2) using our low delay synchronizer. These designs are compared against a baseline GALS NoC, an asynchronous bypass channel (ABC) NoC [85] and a fully synchronous NoC design. A fully synthesizable Verilog implementation of a 7x7, 2-D mesh WaveSync NoC is simulated. A single-stage, non-pipelined baseline GALS router design is also evaluated on 7x7 2-D mesh network with XY DOR routing. In the baseline GALS, packets need to be synchronized at every hop incurring 2.5 cycles of synchronization delay and one cycle of pipeline delay per hop. The baseline router has two, eight-flit deep virtual channels (VCs) at every port. The fully synchronous NoC design presented by Kim et al. [91] is also implemented. Performance was measured under three types of synthetic workloads (uniform, transpose and bit-complement) and realistic traces taken from SPLASH-2 benchmarks [86, 92].

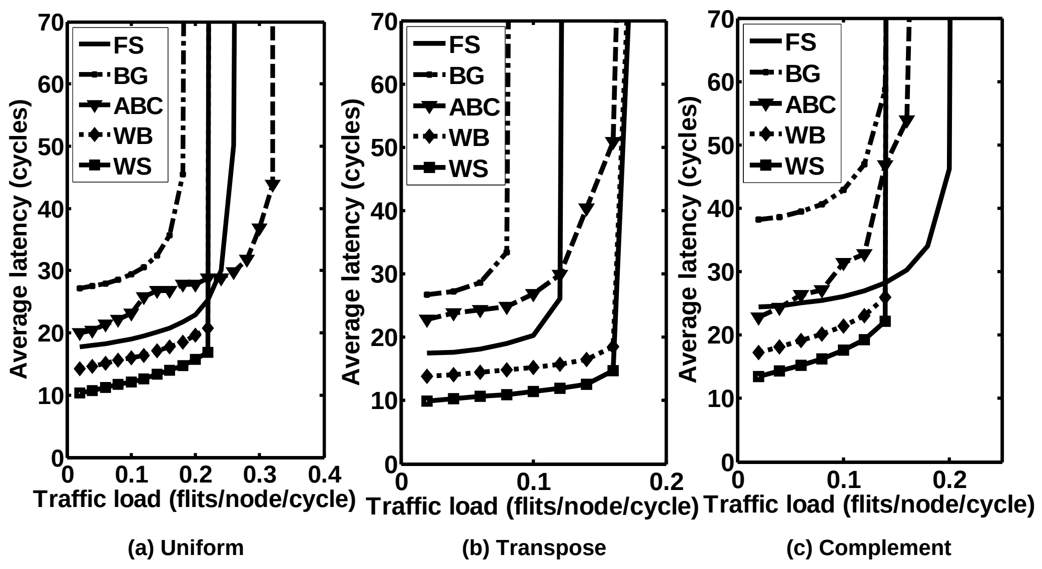


Fig. 37.: Simulation results of synthetic traffic patterns (a) uniform, (b) transpose, and (c) complement on fully synchronous router (FS), baseline GALS (BG), ABC, WaveSync with BIFIFO (WB), and WaveSync with our synchronizer (WS)

In the synthetic workloads, packet length was varied randomly between two to five flits, and the simulation was run for 100,000 cycles including 1000 warm-up cycles. Synthetic workloads of a given uniform random injection process are known to converge fairly rapidly to a given average latency. Beyond 100000 clock cycles the average packet latencies do not change significantly. The SPLASH-2 workloads are composed of last-level cache spills and fills as well as coherence traffic packet traces, taken from a 49-core CMP. For an unbiased evaluation of SPLASH-2, we use 500000 cycles from the middle of traces with 1000 warm-up cycles.

### b. Synthetic workloads

The three synthetic workloads, uniform random, transpose and bit-complement, were chosen because they represent well balanced (uniform), and skewed corner case (transpose and bit-complement) traffic patterns that may be seen in realistic workloads. These results are shown in Fig. 37. In all three cases, the WaveSync design has significantly lower, low-load latency than all competing designs, achieving the goal of lower latency at the typical loads seen in NoCs.

For uniform traffic, the latency performance of WaveSync is better at low injection rates than those of baseline and ABC as shown in Fig. 37(a); although, ABC outperforms WaveSync at injection rates over 22%. This is because ABC router provides higher bandwidth than the WaveSync router due to the extra "wrap-around" links in its serpentine topology. Under transpose traffic (Fig. 37(b)), the WaveSync router yields the lowest latency at all injection rates since the wrap-around paths of ABC in the serpentine topology are not available. The traffic path range of bit-complement is relatively narrower than other traffic patterns; and therefore, the link loading is higher than in other patterns. In the bit-complement, ABC outperforms WaveSync in higher traffic loads (Fig. 37(c)), however, WaveSync with our low latency synchronizer outperforms others regardless of the variability of traffic patterns in low network load because in that low load cases packets can bypass more intermediate nodes. We note that the de-skewing logic adds an extra delay of 1$\sim$3 cycles. We also note that generally, use of the proposed synchronizer reduces the synchronization latency by almost 2$\sim$3 cycles when compared to BIFIFO.

### c. Realistic workloads

Previous analysis has shown that realistic workloads typically have relatively low average injection rates and many applications are highly sensitive to latency [12, 13, 93]. Fig. 38 shows the performance of the different router and network designs under traces taken from benchmarks in the SPLASH-2 shared-memory, multi-threaded benchmark suite.

Generally the pattern exhibited in Fig. 38 mirrors that seen for low loads with the synthetic workloads. The average latency for WaveSync is smaller than that of ABC and

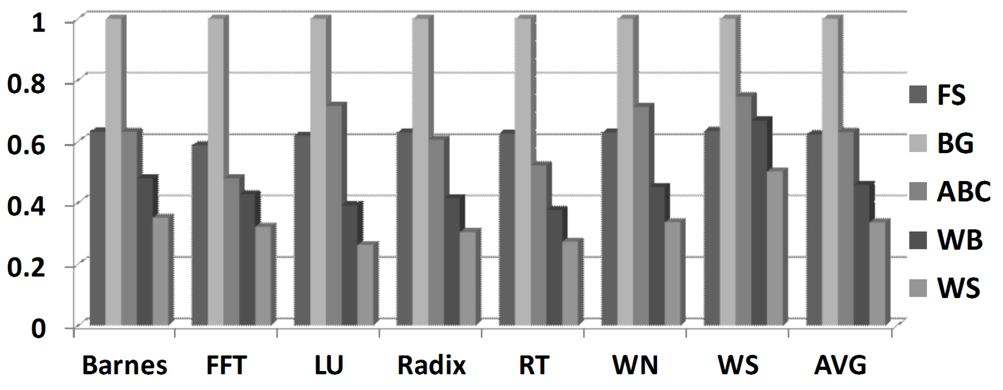


Fig. 38.: Normalized latency results of SPLASH-2 realistic traffic patterns (RT: Raytrace, WN:Water-nsquared, WS:Water-spatial, AVG:total average) on FS, BG, ABC, WB, and WS

baseline GALS routers for all benchmarks. On an average, WaveSync with the proposed novel synchronizer results in an improvement in latency of 68% over the baseline and 55% over ABC router. Furthermore, we see that WaveSync yields an improvement of 54% over even a fully synchronous NoC design. These results reflect WaveSync's ability to support extremely low latency communication at low loads and the fact that generally the SPLASH-2 benchmarks have low injection rates.

### 3. Design Implementation

The WaveSync router is implemented in Verilog and synthesized using TSMC 45nm library at a operating frequency of 1GHz. We used an eight-deep buffer for each FIFO in the router design. For comparison, a fully synchronous router configured by two, eight-flit entry, virtual channels a port and XY DOR routing is synthesized for the same technology, consuming approximately 15% greater area than the WaveSync router. A baseline GALS router has been built on the synchronous router design using the BIFIFO for GALS. Ta-

Table XVI.: Synthesis results of the WaveSync router @1GHz. Clock power denotes clock tree power per node (mW). CDP stands for clock distribution power.

| Network router | # of FIFOs | Router power | Synchronizer power | Total power per node | Area ($mm^2$) |
|---|---|---|---|---|---|
| Synchronous router | 10 | 29.9 | - | 29.9 + CDP | 0.133 |
| Baseline GALS | 10 | 29.9 | 0.471 | 30.371 | 0.145 |
| WaveSync + BIFIFO | 13 | 34.7 | 0.422 | 35.122 | 0.116 |
| WaveSync + Synchronizer | 13 | 34.7 | 0.518 | 35.218 | 0.149 |

ble XVI shows the synthesis results of the WaveSync router, the fully synchronous router, and the baseline GALS router. For clock tree power itself, we expect the synchronous 7x7 NoC to have a high clock tree power, we found one such example in the Xilinx Virtex6 chip, which has a clock tree power of 10.5W and is expect to be approximately the same area [94]. For the Baseline GALS and WaveSync NoCs, the four directional clocks are transmitted over the regular link wires connecting nodes and therefore do not require any special distribution scheme. The power associated with this clock distribution scheme is therefore simply the power associated with transmitting data on four regular link wires interconnecting nodes.

Although the router power for WaveSync is $\sim$15% higher than that of baseline, it is considerably less than the power required by a completely synchronous NoC+clock tree. The increased power consumption in WaveSync over traditional GALS is a small price to pay for improved network performance. Also, the power consumption in WaveSync is expected to scale with frequency in similar fashion as in baseline but with much better performance.

## C. SDPR: Exploiting Path Diversity for Low-Latency through Simultaneous Dual Path Routing

One of emerging issues in chip-multiprocessor (CMP) and multiprocessor systems-on-chip (MPSoC) designs for mobile terminals is a massive data communication such as multimedia streaming between heterogeneous cores and components [95,96]. The massive internal data handling for portable multimedia devices such as smartphones over on-chip networks requires low-power and minimum latency requirements. An increasingly large number of integrated components (processors, memory arrays, application specific IPs such as baseband processors and video processing units) oblige the use of NoCs [97,98] to permit high system-level throughput. NoC design efforts to-date have largely been aimed at reducing latency to relieve congestion [99–101]. For instance, Peh and Dally introduced router delay models for the pipelined routers and proposed a microarchitecture for a speculative virtual-channel router to reduce latency [102]. However, a significant portion of network traffic in MPSoCs for multimedia devices is lengthy multimedia streaming data. Lee et al. have explored multimedia applications containing video block packets [97]. In the work, one block data in a frame was ($8\times8\times16$ bits) divided into 64-flit length packets with 16-bit flit size. Another example is the study of NoC designs for MPEG-4/H.264 parallel coding [98]. Multiple video streams are coded simultaneously in parallel while video stream data are subsequently distributed to processing elements (PEs).

These studies point the need for an efficient on-chip interconnect architecture for massive data streaming. One solution for such applications is to increase the link widths. However, while this may reduce congestion and serialization latency, it comes at a high cost of increased power consumption [103]. We propose the simultaneous dual-path routing (SDPR) scheme that utilizes the path diversity present in typical mesh topology NoCs. This approach is akin to having a higher link width but without the significant hardware overhead associated with simple bus width scaling.

Multipath routing has widely been explored in the networking community. It has been recognized to yield reduced network congestion and traffic hotspots. This paper focuses on the same objectives but with observation that we can prescribe specific multiple paths and inject packets *simultaneously* through multiple I/O ports (and simultaneously eject at the receiving node) to leverage the network's path diversity on 2-D meshes. This is in contrast with the multi-dimension routing works or O1TURN routing selecting one of alternative paths among available multi-paths and sending flits through the path sequentially [104]. C. Izu et al. have studied effects of multiple injection ports on highly congested network and concluded that injecting multiple complete packets into network might not help in performance [105]. However it is assumed that injection rate will increase at injection ports, and this makes the network resource too scarce to accept all the packets, thus causing early saturation. We design a network adapter which takes a packet, splits it into two halves and send them uniformly to the injection port queues. This makes the injection port twice wider than a single port injection router. Hence, SDPR does not exert unnecessary injection pressure on network throughput. To the best of our knowledge, this is the first work to explore leveraging path diversity via packet splitting and injecting simultaneously through multi-ports.

For most source-destination pairs in a mesh network there are two statically determined non-intersecting output links from the source node in the direction of the destination, XY and YX dimension order routing (DOR) paths, shown in Fig. 39. But only one can be used when a packet is injected because of a single injection port. Even in the adaptive routing networks, though a packet can travel in multiple routes based on the congestion, it still uses the network bandwidth equal to its flit width. By doubling the number of the injection port and ejection port and splitting packets into two halves, we can leverage the available path diversity and cheaply mimic a higher bandwidth network. The motivation

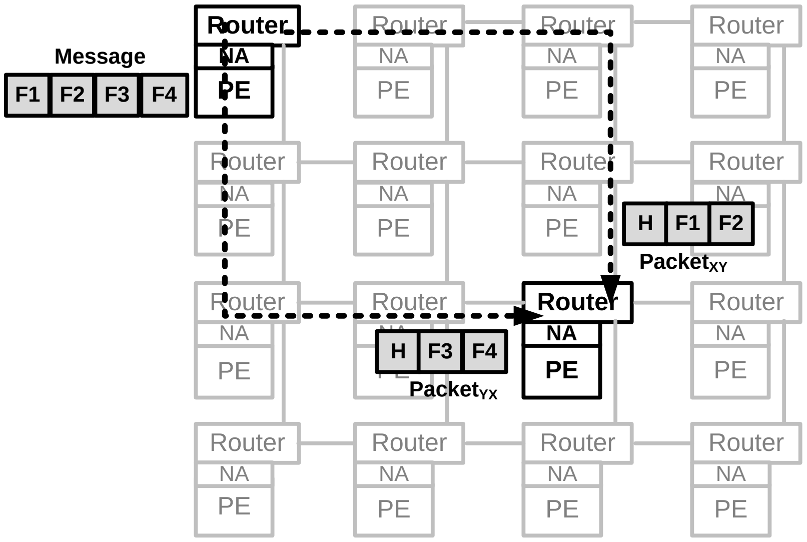


Fig. 39.: Dual-path routing on a 4x4 NoC system

of packet splitting is that injecting two packets via the XY and YX DOR paths simultaneously can improve performance in latency and increase link utilization. But the packets injected through the XY and YX minimal paths have to keep the same destination address to traverse to the destination.

The reconstruction of packets can be done using the order information stored in the head flit. We will focus on direct memory access (DMA) data transfers for this work, and therefore providing a reconstruction order number in the header is sufficient to allow for reconstruction within the DMA buffer at the destination, eliding the need for a dedicated packet reconstruction buffer. Given this design, the SDPR router can ideally reduce serialization latency by 50%. We note that the SDPR router provides greater benefit for large or medium size packets such as video streams. Segmenting a short packet into two halves can incur significant overhead in header flit generation. This can negate potential latency improvement realized in serialization.

The remainder of this section summarizes relevant work on path diversity, outlines the proposed SDPR scheme, and describes the microarchitecture of the SDPR router. We also evaluate experiment results and examine hardware implementation overhead shown in the synthesis result.

## 1. Related Work

This subsection summarizes the prior work on on-chip interconnect architecture for path diversity. Heuristic approaches have been extensively explored [106, 107]. Banner et al. extends the discussion to feasibility of non-minimal paths as well [108]. Implementing these complex algorithms in NoCs constrained by power, latency and hardware complexity overheads, however, is not readily feasible.

There are oblivious, minimal, path diverse routing schemes such as Valiant [109], ROMM [110], O1TURN [104], and PROM [111]. Their path diversity leads to improving throughput because of increasing routing flexibility. O1TURN routing randomly routes packets in one of orthogonal paths (XY and YX) with equal probability (i.e. 50%). Valiant routes each packet through a random intermediate node. As Valiant, ROMM is also one of probabilistic routing algorithms, but it restricts the intermediate nodes to the minimal routing area. PROM performs local randomized decisions at each hop based on probabilistic oblivious routing policy. ROMM, Valiant, O1TURN, and PROM are able to encounter out-of-order packet arrivals at destination dissimilar to DOR, so the destination requires enough buffer to reorder the packets. Even though they all achieve better path diversity than DOR [111] with minimal hardware overhead, they do not overcome the serialization latency of a long packet incurred due to single injection port. As we showed in the SDPR experiments, this problem can be mitigated by two injection ports injecting split halves of a packet via separate and independent orthogonal two paths (i.e. XY and YX) at the same time.

Murali et al. describes a multipath routing strategy for in-order packet delivery in

NoCs [112]. The packets are sent through non-intersecting paths and the lookup table is employed at the switch of re-convergent node to support in-ordering. Michelogiannakis et al. introduced multi-dimension routing concept for bufferless flow control [113] in which flits can travel in any productive directions. Common to these approaches is the aim to better utilize the link diversity attainable at a given node or a link to reduce latency and hotspots.

## 2. Dual-Path Network Architecture

Daeho Seo et al. reviewed the path diversity in 2-D mesh networks in O1TURN routing [104] by injecting the packets in XY DOR and YX DOR paths with equal probability. However it is done with a single injection port which does nothing to reduce the serialization latency of packet injection. The motivation behind the SDPR router is to employ complete parallelism in packet traversal to reduce the serialization latency. The number of minimal non-intersecting paths where we can send the parts of the packet simultaneously can at most be two. The best possible split size of the packet is half which traverse through two independent paths simultaneously in parallel, leading to a 50% reduction in serialization latency ideally under no loads. Under the synthetic and realistic loads, we still get reduction in serialization latency, but the benefit is reduced by difference in arrival time of these packet pieces at the destination. Thus, the network adapter splits a message into two halves and these packets are injected to travel XY DOR and YX DOR respectively, leveraging the two available and productive output ports at the destination to increase the effective network bandwidth. Each SDPR router consists of two injection ports and two ejection ports along with minimum overhead. Injected packets approach the destination from different directions and get absorbed by different ejectors in parallel. The baseline and SDPR router details are described in this section.

a. Dual-Path Routing Scheme

SDPR can reduce the latency of serialization by accommodating additional injection and ejection ports and exploiting two inherent minimal XY and YX DOR paths on a mesh network, compared with baseline using XY DOR only. Fig. 40 presents the proposed SDPR scheme. When source and destination nodes are not in a line (i.e. two minimal paths are available) as shown in Fig. 40(a), a packet is split to two packets, and they are simultaneously sent along the XY DOR path and the YX DOR path in parallel. In this approach, the split two packets should retain the same source and destination addresses in their head flits since they composed a packet and were supposed to be sent to the same destination before being split. Therefore, SDPR requires preprocessing to divide a packet to two halves and re-packetize them into two split packets. This task can be performed in the packet generation process handled by a network adapter module as shown in Fig. 41. In particular, this makes our method significantly different and independent from the previous multi-path studies that send packets in serial via a path selected from multiple paths (e.g. O1TURN [104], ROMM [110], Valiant [109], PROM [111], or odd-even routing [114]). SDPR utilizes path diversity as the previous multi-path studies but contrastively injects the divided portions of a packet via multiple paths at the same time. This increases parallelism and path utilization as well as decreases serialization latency.

If there are minimal or non-minimal multiple paths where packets can traverse, SDPR can be extended to accommodate the multiple splitting and simultaneous sending scheme. In Fig. 40, payload data are divided into two payload groups for split packet 1 and split packet 2, which are packetized into two split packets with header (H) including flit type, DOR path and VC information, source address (S0), destination address (D14), and payload data respectively. The DOR path information denotes which path is allotted for each split packet. In the figure, packet A, one of two halves, is injected into inject port 1 (Inj1)

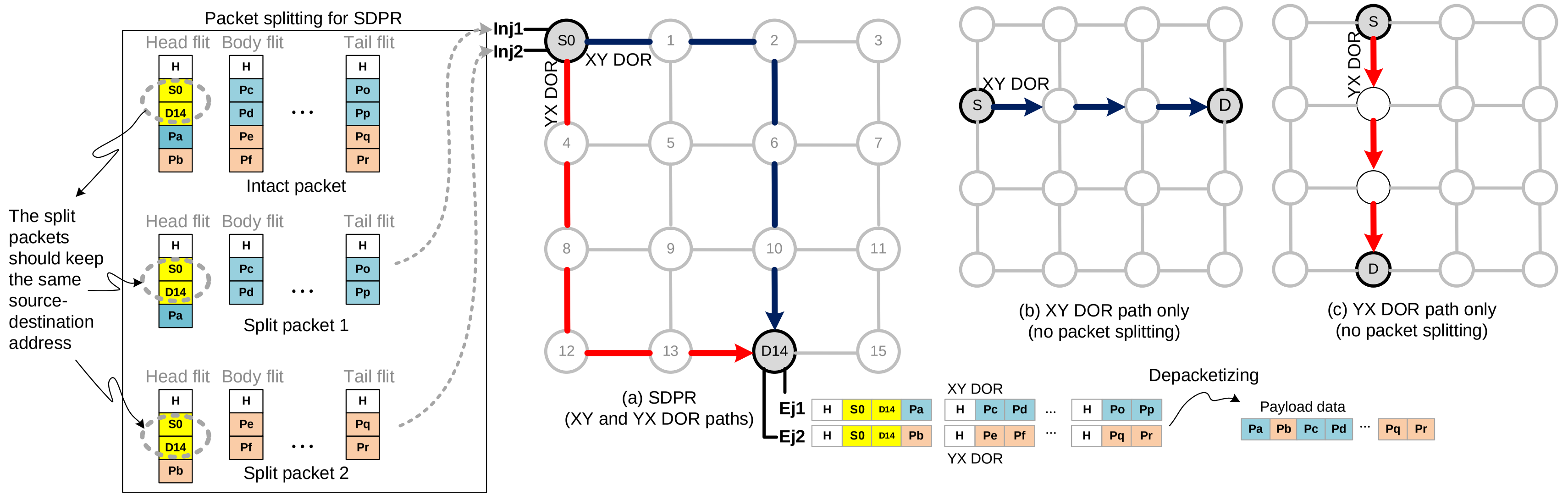

Packet splitting for SDPR
Head flit
Body flit
Tail flit
H
S0
D14
Pa
Pb
Pc
Pd
Pe
Pf
Po
Pp
Pq
Pr
Intact packet
Split packet 1
Split packet 2
The split packets should keep the same source-destination address
Inj1
Inj2
XY DOR
YX DOR
Ej1
Ej2
(a) SDPR (XY and YX DOR paths)
(b) XY DOR path only (no packet splitting)
(c) YX DOR path only (no packet splitting)
Depacketizing
Payload data

Fig. 40.: The proposed SDPR scheme (a) There are two minimum DOR paths from the source node (S0) to the destination node (D14). Two split packets are simultaneously sent via the XY and YX DOR paths respectively. (b and c) the destination and source nodes are in the same line (X axis or Y axis). Therefore, packets are not split, sent via XY or YX path.

at the source S0, traverses through the XY DOR path (i.e. S0→1→2→6→10→D14) and ejects out of ejection port 1 (Ej1) at the destination D14. Packet B, the other half, simultaneously traverses from Inj2 to Ej2 along the YX DOR path (i.e. S0→4→8→12→13→D14). Once both A and B packets arrive at the destination node, they are retrieved and de-packetized by the network adapter. The latency of a intact packet transmission is measured from the injection time of the first injected packet in two halves at the source to the ejection time of the last arrived packet at the destination. If the time difference of arrival between two split packets at the destination increases due to congestion in the network, the packet latency will also increase.

If there exists only one minimal path between source and destination like traversing only on X or Y axis as shown in Fig. 40(b and c), the packet is sent intact without packet splitting and dual-path routing. In this case, the performance of the SDPR router relatively degrades, compared with that of the baseline router using the same number of virtual channels (VCs) of the SDPR router. This is because VCs for XY DOR are separate from VCs for YX DOR in SDPR to avoid deadlock. When there is only one minimal path (XY or YX) traversed, a set of VCs for the minimal path will be used. It means that VCs for the other minimal path are not utilized. This causes inefficiency in terms of resource utilization and performance degradation in SDPR.

Table XVII shows the distribution equations of dual path node pairs over whole node pairs under given traffic patterns, where *N* denotes network dimension. The motivation of this examination is that serialization latency reduction in SDPR is highly dependent on the distribution of source and destination pairs including both minimal DOR paths on a traffic pattern. The reduction in serialization latency will be high at high levels of dual-path node pairs in distribution. The distributions of dual path node pairs are 73%, 86%, 73% at uniform random, transpose, and bit-complement on a 7x7 mesh network respectively. It implies that the serialization latencies of SDPR across the three traffic patterns can be

Table XVII.: Distribution of dual path node pairs and serialization latency reduction in SDPR where N=7

| Traffic pattern | Distribution (%) | Ideal reduction in serialization latency (%) |
|---|---|---|
| Uniform random | $[N^2 \cdot (N-1)^2]/(N^2 \cdot N^2)$ $= (N-1)^2/N^2 = 73$ | 37 |
| Transpose | $(N^2 - N)/N^2$ $= (N-1)/N = 86$ | 43 |
| Bit-complement | 100 (N:even) $(N-1)^2/N^2 = 73$ (N:odd) | 50 (N:even) 37 (N:odd) |

ideally reduced by 37%, 43%, 37%, respectively, (i.e. half of the level of distribution because of packet splitting and injecting in parallel) without accounting for the overhead increase.

DOR is inherently deadlock free [115]. In SDPR, separate channels are allocated for XY DOR and YX DOR respectively to remove the cyclic dependencies in the resources sharing a physical channel; hence, SDPR is also deadlock free. Virtual channels are also provided in these separate channels to avoid the head-of-line blocking in XY and YX DOR paths.

b. Network Adapter

The network adapter (NA) splits messages and injects them through separate injection ports into the separate paths as shown in Fig. 41. When a DMA block has a message (i.e. packet) to send, it is forwarded from memory to the packet splitting block, and then split into two packets of half the size, and re-packetized with modified header and tail flits. An initial message with *n* flits is divided into two packets of *n/2 + 1(header)* flits each. New head flits for the half packets ($P_{xy}$ and $P_{yx}$) are built and injected to the SDPR router through

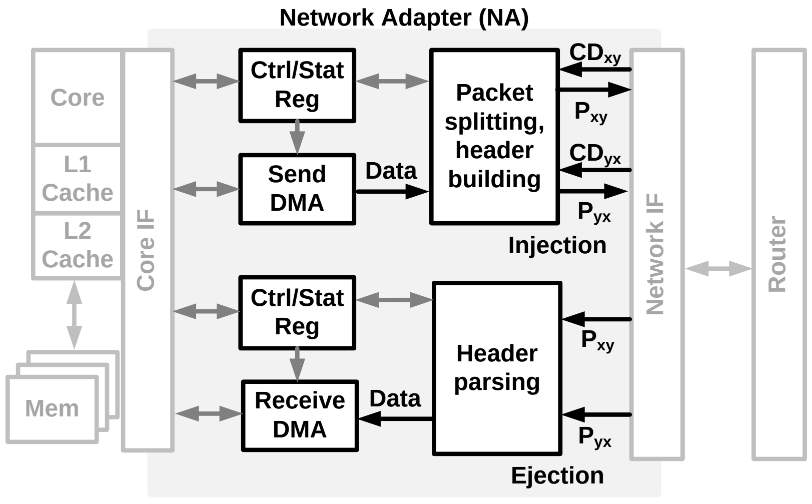


Fig. 41.: Network adapter architecture for packet splitting

network interface. At the destination, two ejected packets (P$_{xy}$ and P$_{yx}$) are delivered to NA for header parsing and DMA transfers the parsed packets to memory. When DMA finishes sending two ejected packets to the memory, a processing element (PE) reconstructs the split packets.

We have observed that the reconstruction of the separated halves could be done using small amount of cache memory in the experiment for realistic video benchmarks. This is because the network load of realistic video traffic was low, leading to 27-34 cycles difference of ejection time of two split long packets on average at the destination. We will demonstrate this result more in the experiments section. Furthermore, we note that in many applications such as cache line or DMA data transfers, memory has already been set aside for the reception of the data, thus even for the worst case ejection time difference, a reconstruction buffer is unnecessary. Therefore, in the cache line transfers or DMA data transfers, the reconstruction order number (i.e. ADDR field in the header) can sufficiently support to reconstruct the separate packets within the cache or DMA buffer at the destination.

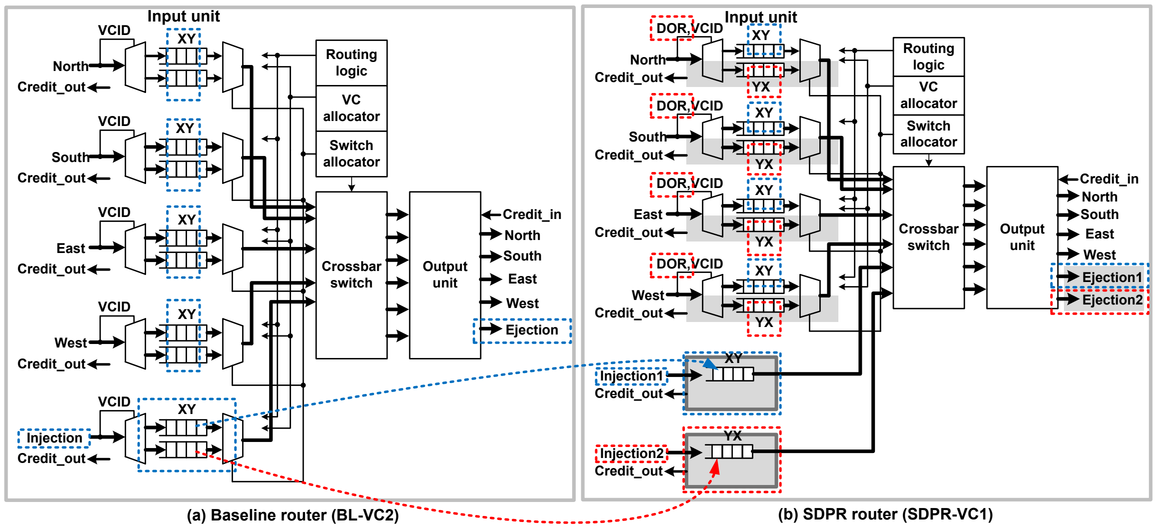


Fig. 42.: Microarchitectures of (a) baseline (i.e. single-path) router with two virtual channels and (b) SDPR router with one virtual channel for each XY or YX DOR. They exploit equivalent resources in terms of total number of buffers used.

### c. Baseline Router

The baseline is a standard 2D mesh, pipelined router with virtual channels [102]. The pipeline consists of 2 stages: route computation and arbitration at output port. When a flit enters a router, it is sent to the particular VC depending on the VC identification (VCID) carried by the flit. If the flit is a header carrying the current node output port (CNOP) information, arbitration is requested for acquiring the VC at next node as conventional wormhole routers.

### d. SDPR Router

**Microarchitecture:** Unlike the baseline router as shown in Fig. 42(a), the SDPR router has an extra local injection and ejection port to make SDPR beneficial. In Fig. 42(b), the SDPR router consists of four directional input/output ports, two local injection ports, credit

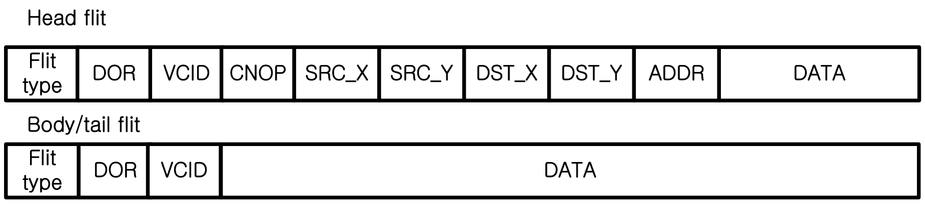


Fig. 43.: Packet structure for the SDPR router

signals, and two ejection ports. Each directional input port contains two channels, one for XY DOR and the other for YX DOR, supporting two completely disjoint paths from source to destination. The XY or YX path in a packet is determined by 1-bit DOR information encoded in the packet. If there are multiple VCs for each path, VCID indicates which virtual channel is occupied by this packet. In Fig. 42(b), the SDPR router exploits only one virtual channel, hence VCID should be 0. In case of using two VCs for each path, VCID can be 0 or 1 to denote which one is occupied. The credit based flow-control system is used for buffer management. As highlighted in grey on the figure, the hardware overhead of the SDPR router is minimal as the only additions are DOR information in a flit, extra ejection port, and hence wider crossbar with slightly modified control logic. To keep the same utilization of VC buffers, the two injection VCs are separated to two injection ports. Therefore, no additional buffers are required because total VCs used in injection ports as well as total VCs used in direction ports are equal in SDPR and baseline. SDPR thus does not require extra channel ports, just static assignment of each injector to two of the output directions.

**Packet structure:** Fig. 43 and Table XVIII shows the packet format. The size of a flit is 64 bits and 2 MSBs indicate the flit type. The DOR bit determines XY DOR or YX DOR, and the VCID bit is the same as that in baseline. We note that 1-bit VCID was used to provision maximum 2 VCs per port in our experiment, but the number of VCs can be extensible. CNOP (3 bits) denotes the output port for the packet at the current node and is required to

Table XVIII.: Packet structure for the SDPR router

| Flit type | DOR | VCID | CNOP | SRC | DST |
|---|---|---|---|---|---|
| | | | 000: inj1 | | |
| 00 : head | 0: | 0: | 001: north | | |
| 01 : body | XY | 1 VC | 010: south | Coord. | Coord. |
| 10 : tail | 1: | 0,1: | 011: east | for sou | for des |
| 11 : single | YX | 2 VCs | 100: west | rce | tination |
| | | | 101: inj2 | | |

request the output port. Once the output port is acquired, the CNOP is updated for the next node. The source and destination addresses are used to update the CNOP for the next node (i.e. lookahead routing). 10-bit ADDR followed by payload denotes the order information of split packets for reconstruction. For the body and tail flits, it only has the flit type, DOR, and VCID fields, and the rest is payload. Table XVIII explains the information stored in the flit and its meaning.

**Packet processing at source and destination:** Each source node switches dual-path routing or single-path routing based on the destination address of packets. If the destination is ”in-line”, there are no two minimal paths to the destination and hence packet is sent intact without splitting via XY DOR path (i.e. DOR=0). When there are two minimal paths to the destination, the network adapter splits the packet into two halves and assigns the header information (i.e. flit type, DOR=0/1, VCID, CNOP=000/101) to both packets at the source. The re-packetized packets are injected simultaneously through dual injection ports. When the split packets reach the destination, they exit through the ejection ports. Injection1 and Ejection1 ports are dedicated for XY DOR path, and Injection2 and Ejection2 ports are dedicated for YX DOR path. There are no mixed ejections between port1 and port2 since these ports are separate in different channels for deadlock prevention. Packet splitting incurs overhead of an extra head flit. In particular, for short packets (2-5 flits), this overhead results in reducing the performance of SDPR. In the experiments section, we will demonstrate the influence of overhead on short packets. Therefore, we target medium-long length traffic such as video streams where the packet lengths are longer and serialization latency is dominant. Splitting the long packets and thus injecting them simultaneously provide double network bandwidth and cause at most 50% reduction in serialization latency with a relatively minor overhead. While the proposed SDPR approach statically splits packets, in the future we plan to explore the dynamic splitting of packets and balancing of lengths between paths dependent on network load metrics.

## 3. Experiments and Evaluation

In this subsection, we evaluate the proposed SDPR scheme and the SDPR router experimentally to analyze performance under different types of synthetic and realistic workloads. We compare the performance of the SDPR router against that of the baseline router.

### a. Methodology

We developed a fully synthesizable network consisting of the SDPR routers connected in the 7x7 2-D mesh topology to obtain the performance numbers. All simulation models for SDPR and baseline routers were coded in Verilog and synthesized using Synopsys Design Compiler [55]. In the baseline router, we used 2 VCs and 4 VCs (BL-VC2 and BL-VC4, BL stands for baseline) each with buffer depths of 5, 8, and 12. The buffer depth of 5 is the minimum to ensure no pipeline bubbles due to credit return time in the credit-based flow control [86]. For a fair comparison, we allocated the same number of buffers for the baseline and SDPR routers. We compared the baseline routers with the SDPR routers using 1 VC and 2 VCs per DOR path (SDPR-VC1 and BL-VC2) since BL-VC2 and BL-VC4 consist of the same FIFO buffering and resources as SDPR-VC1 and SDPR-VC2 respectively. Hence, total buffers per port are two for BL-VC2 and SDPR-VC1 and four

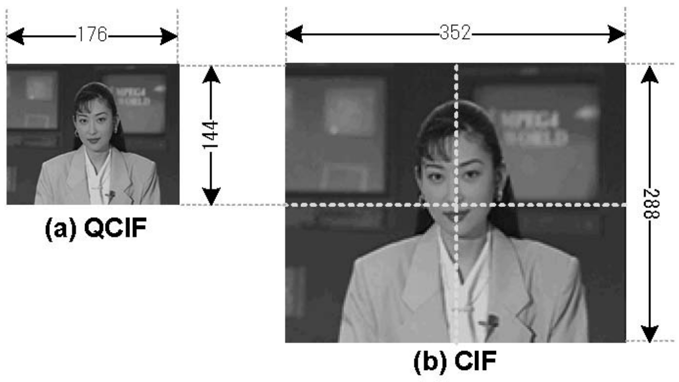


Fig. 44.: QCIF and CIF frame resolutions on Akiyo

for BL-VC4 and SDPR-VC2.

The performance of the proposed router was evaluated across uniform random, transpose, and bit-complement synthetic workloads [86] and H.264 video test streams [116] and SPLASH-2 benchmarks [86,92] for realistic workloads with variations. Each of the routers was experimented with a buffer of depth five. Table XIX details the network configuration and the variations used in the experiments. In the synthetic workload simulation, we used long, medium, and short packets in which the average packet lengths (APLs) were 100, 25, 3.5 flits respectively.

We also evaluated SDPR and baseline across four different QCIF (176x144) 10 frames and CIF (352x288) 10 frames video sources (Foreman, Akiyo, Mother, and Mobile streams as configured in Table XX) generally used as video benchmarks for the evaluation on video applications [117]. Fig. 44 shows CIF and QCIF frame resolutions on the Akiyo stream. We used the H.264/MPEG-4 AVC (advanced video coding) standard developed by the joint video team (JVT) of ISO/IEC and ITU-T (Telecommunication Standardization sector) and the reference software of H.264 as a source coder for the video encoding [41, 42]. The five sources (i.e. four video streams and all mixed stream) were encoded by the software H.264 encoder and uniformly distributed through source-destination pairs. Therefore, the

Table XIX.: Network configuration and variations for baseline and SDPR routers

| Characteristic | Network configuration | Variations |
|---|---|---|
| Topology | 7x7 2D Mesh | - |
| Routing | XY DOR for baseline, XY and YX DORs for SDPR | - |
| Router architecture | Two-stage pipelined architecture | - |
| Per-hop latency | 3 cycles: 2 in router, 1 to cross channel | - |
| Virtual Channels/Port | 2(baseline), 1(SDPR) | 4(baseline), 2(SDPR) |
| Flit buffers/VC | 5 | 8, 12 |
| Flit size in bits | 64 | - |
| Traffic workload | Transpose, Bit-complement, Uniform random | H.264 video traces<br>SPLASH-2 traces |
| Average packet length (flits per packet) | Synthetic: 100(long), 25(medium), 3.5(short) | H.264 video: 37(medium-long), 8.35(short)<br>SPLASH-2: 3.5(short) |
| Simulation warmup | 10,000 (cycles) | - |
| Analyzed packets | 100,000 (cycles) | H.264 video: CIF 10 (frames), QCIF 10 (frames)<br>SPLASH-2: 500,000 (cycles) |

Table XX.: H.264 video traces

| Source video stream | Foreman, Akiyo, Mobile, Mother |
|---|---|
| The number of frames | $10(QCIF), 10(CIF)$ |
| Frame rate | $30 frame/sec$ |
| Source resolution | $176 \times 144$ (QCIF)<br>$352 \times 288$ (CIF) |
| Max. Bitrate | $192(QCIF), 768(CIF) kbit/s$ |

video packets included H.264 encoded video bitstreams as payloads. The packet length of the video streams were 33.14 (CIF) and 8.35 (QCIF) flits on average for medium and short respectively, where the video streams suitably provided a realistic benchmark for evaluating our design on short to long packet sizes.

SPLASH-2 [86,92] is a shared-memory, multi-threaded benchmark suite. For an unbiased evaluation of the SPLASH-2 benchmarks (Barnes, FFT, LU, Radix, Raytrace, Water-nsqeuared, and Water-spatial), we used 500,000 cycles from the medium of traces with 10000 warm-up cycles. The packets consisted of 2-5 flits per packet (i.e. short packets) and the average packet length of the SPLASH-2 traces was 3.5 flits. As we mentioned before, the latency of the ejected packet was measured as time difference between the injection of the first half packet at source and the ejection of the last half packet at destination.

b. Results

**Standard synthetic loads**: Fig. 45, 46, 47 show the packet latency averaged across uniform random, transpose, and bit-complement synthetic loads using long, medium, and short length packets at 2-50% injection bandwidths on BL-VC2, BL-VC4, SDPR-VC1, and SDPR-VC2. As expected, SDPR with long packet significantly outperformed baseline in latency across all traffic patterns. The latency results, 32%, 40%, 31% approached to

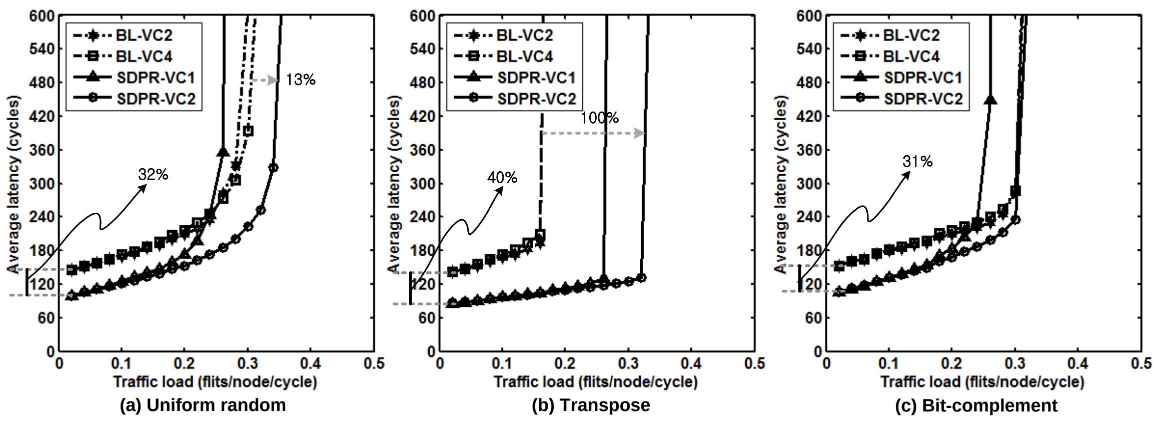


Fig. 45.: Results of synthetic long length packets (average packet length=100)

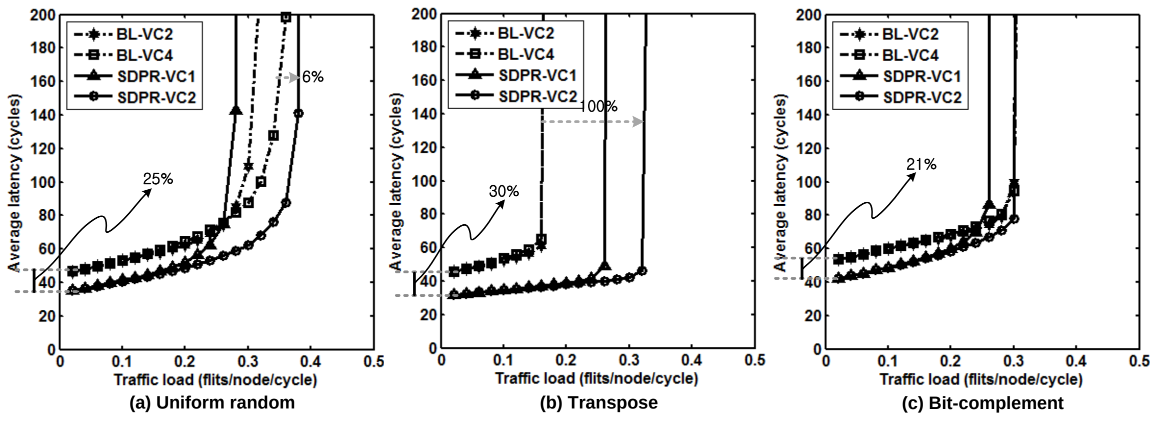


Fig. 46.: Results of synthetic medium length packets (average packet length=25)

the ideal reduction gains in serialization latency (37, 43, 37% as discussed in Table XVII) under three synthetic traffic patterns because SDPR provided better utilization of path diversity than baseline under long packet. In particular, the latency and saturation throughput of SDPR were much better than them of the baseline router under transpose traffic in Fig. 45(b), 46(b), 47(b) because the transpose pattern is ideally balanced under SDPR and increases the probability of packet splitting and parallel traversal through the dual-path routes. The results show the improvement of saturation bandwidth by 100%, 100%, 75%

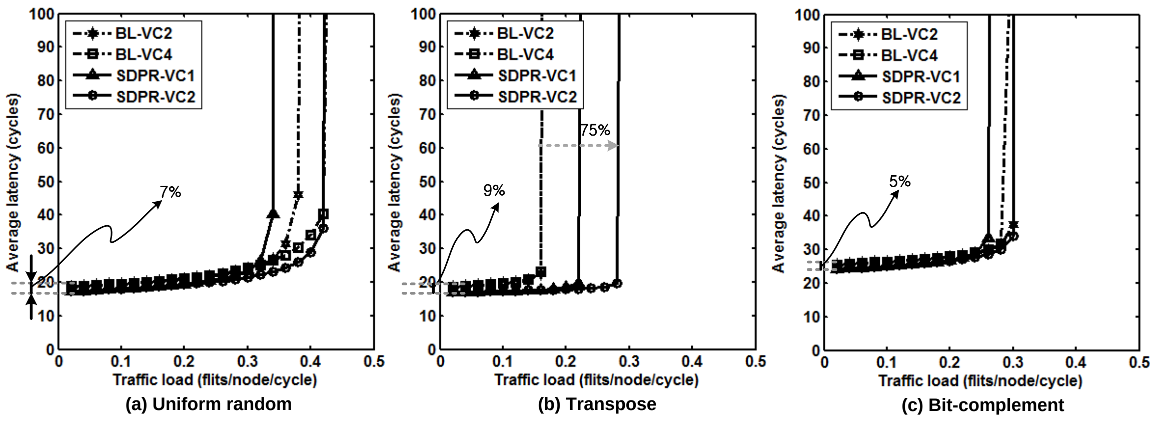


Fig. 47.: Results of synthetic short length packets (average packet length=3.5)

on long, medium, short packets respectively. On the other hand, bit-complement traffic mainly uses horizontal and vertical network bisections and provides less chance of parallel traversal than the transpose traffic, leading to the slightly less performance improvement in latency and saturation bandwidth than the transpose and random uniform loads as shown in Fig. 45(c), 46(c), 47(c). However, it still achieved up to 32% latency reduction in the long packet traffic pattern. Fig. 45, 46, 47 also show that the saturation throughput of SDPR-VC1 is less than BL-VC2 under uniform random and bit-complement. This was because SDPR-VC1 used only one VC for each DOR path, leading to head-of-line blocking which the BL-VC2 design did not experience. SDPR-VC1 nevertheless achieved significantly lower no-load latencies than BL-VC2.

Fig. 46 shows medium packet results. The APL of the medium packets was 25, greater then the packet length boundary (20). Accordingly, latency reductions were improved to 25, 30, 21% on random uniform, transpose, and bit-complement which were 3-4 times the short packet latency results across the synthetic traffic patterns. Long packet results are shown in Fig. 45, where APL is 100.

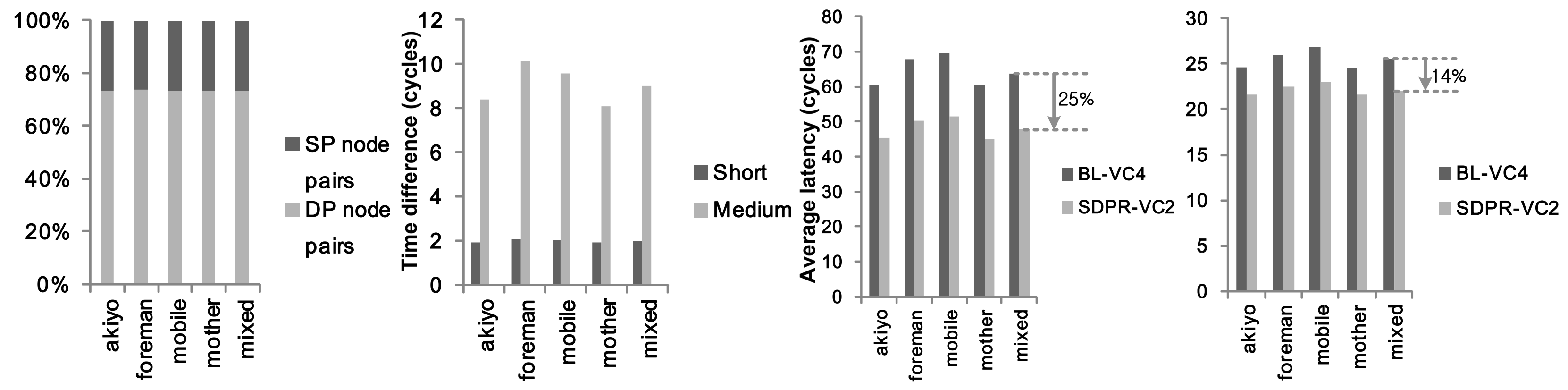
Fig. 47 shows the results of short packet simulation (APL=3.5) across synthetic traffic

Table XXI.: A summary of average latency reductions on long packet traffic and closeness to ideal latency reductions under SDPR on a 7x7 mesh NoC

| Traffic pattern | Ideal latency reduction (%) | Average latency reduction (%) | Closeness to ideal latency reduction (%) |
|---|---|---|---|
| Uniform Random | 37 | 32 | 86 |
| Transpose | 43 | 40 | 93 |
| Bit-complement | 37 | 31 | 84 |

patterns. Short length packets degraded the latency performance in SDPR due to the overhead of header in split packets. In the figure, the average latencies gained by SDPR over baseline were 7, 9, 5% on uniform random, transpose, and bit-complement respectively. Also as expected, these improvements were much less than the ideal maximum gains under the synthetic traffic patterns, 37, 43, 37% (we mentioned in Table XVII) in SDPR. Table XXI summarizes the average serialization latency reductions across synthetic workloads on long packet traffic and compares them with ideal serialization latency reductions under SDPR.

**Realistic loads-H.264 video streams**: As we discussed, realistic H.264 video streams were used to evaluate our SDPR scheme because lengthy video streaming data held significant portion of network traffic, causing the system performance bottlenecks in the network. Fig. 48 shows the video packet latency averaged across five test streams under medium and short packet traffic patterns generated by H.264 CIF and QCIF encoded video workloads. When we compared SDPR to baseline on medium packets (APL=33.14), the average latencies of SDPR-VC2 were outperformed by 25%. The latency reduction of SDPR-VC2 with short packets (APL=8.35) under the mixed video stream was 14% on average, compared with BL-VC4. We were able to estimate the maximum reduction in serialization latency from the distribution of dual-path node pairs across the five video traffic patterns as shown



(a) Distribution (b) Time difference of arrival on SDPR-VC2 (c) Avg. latency (medium: APL=33.14) (d) Avg. latency (short: APL=8.35)

Fig. 48.: Results of realistic video workloads: akiyo, foreman, mobile, mother, and mixed CIF (medium packet, APL=33.14) and QCIF (short packet, APL= 8.35)

in Fig. 48(a). The distribution of dual-path node pairs was 75% on average at the patterns; thereby the maximum reduction in serialization latency on SDPR was expected as 38%.

Fig. 48(b) shows the distance of arrival time between two split packets at a destination node across video patterns. When the time difference of arrival increases, processing elements require more memory space to hold the payloads of the split packets for reconstruction. As shown in the figure, average time differences were 29.8, 5.3, 1.9 on the mixed video patterns with medium and short packet length respectively. The reconstruction can be performed using small amount of cache memory. Even for the worst case, cache memory is enough to handle this reconstruction without any additional buffers in the network adapter.

We concluded from this analysis, SDPR attained a significant improvement over conventional XY DOR for all traffic loads due to its better utilization of path diversity. Especially, the SDPR scheme outperformed XY DOR under lengthy streaming applications such as video streaming since they mostly require long packet communication for high resolution between nodes in NoCs. SDPR also provided better performance in latency than baseline across all buffer depths and injection rates we experimented. The reconstruction of split packets could be done in memory space belonging to main processor units.

**Realistic loads-SPLASH-2**: Realistic traces taken from SPLASH-2 benchmarks were exploited for evaluation. The SPLASH-2 workloads are composed of static traces of short packets (APL=3.5) from the portion of the memory hierarchy which directly communicates via the NoC. We performed SDPR simulations across the SPLASH-2 traces and obtained the experimental result as shown in Fig. 49. The average packet length of SPLASH-2 was short (i.e. 3.5) as that of synthetic short packet traffic, therefore, as expected, the latency reduction (5-6%) attained from the experiments was low due to the short packet length like the results of synthetic short packet traces. Fig. 49(a) shows the distribution of single-path packets and dual-path packets implying reduction degree. Raytrace consists mainly

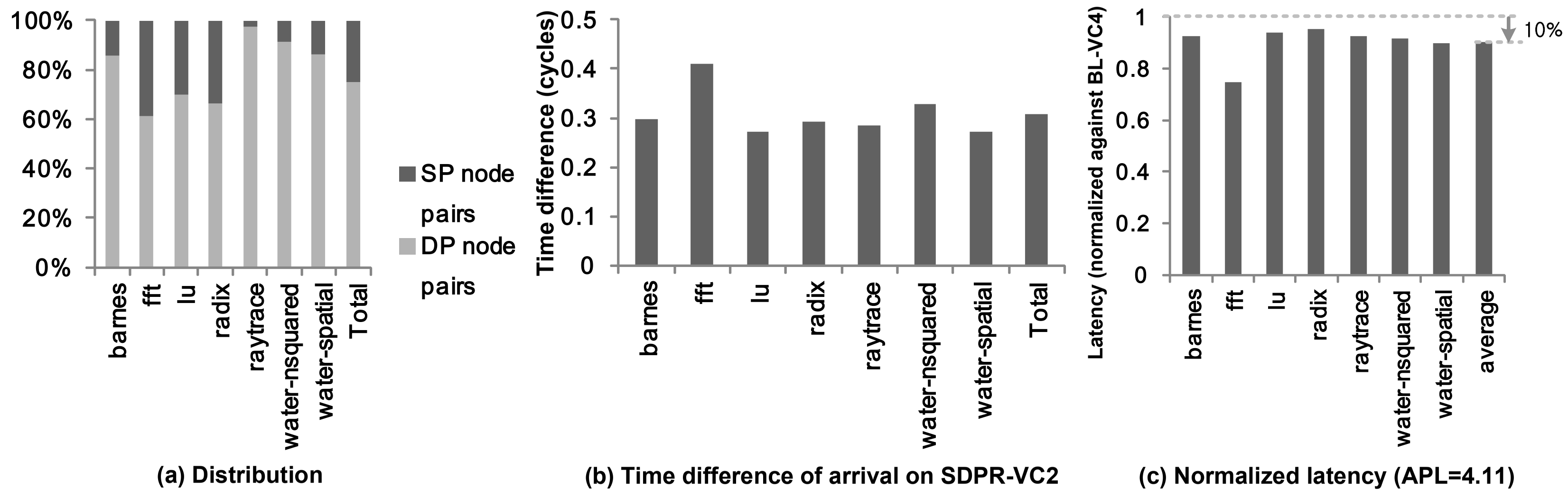


Fig. 49.: Experiment results across SPLASH-2 benchmarks (APL=3.5): Barnes, FFT, LU, Radix, Raytrace, Water-nsqeuared, and Water-spatial

of dual-path packets, so SDPR can improve the efficiency of routing on this traffic pattern. But it mostly included short packets (i.e. 2-5 filts per packet), thus the benefic of SDPR was attenuated due to head flit overhead. The time arrival difference between two split and ejected packets on SDPR-VC2 across the SPLASH-2 benchmarks is shown in Fig. 49(b). The time difference was less than one cycle, which meant the split packets were ejected at almost the same time.

### c. Discussion

We examined the effect of packet length on our SDPR scheme, compared with the baseline router simulation. A source and destination pair was uniformly and randomly selected, and 10,000 payload flits were packetized and traversed along with single-path or dual-path from the source to the destination on the 7x7 network. This experiment was repeated with eight different pairs of nodes. The results of transfer time in cycle were measured and averaged with 1,000 warmup cycles as shown in Fig. 50. In Fig. 50(a), transfer time results on BL-VC4 were measured while packet length per injection port was varied from 2 to 100 to examine the optimal length of packets in terms of transfer time. We used short packets for background nodes excluding the selected source-destination pair to minimize the side effects of packet length variation in the background nodes. To investigate the influence of buffer depth over the transmission, we used three buffer depth variations, 5, 8 and 12. Fig. 50(b) shows the results of SDPR-VC2 simulation under 5, 8, and 12 buffer depths. Injection rate thus varied from 3 to 30% to demonstrate the effect of congestion over BL-VC4 and SDPR-VC2. Packet length boundary to achieve subminimal transfer time was 20 flit per injection port in both BL-VC4 and SDPR-VC2. If the packet length is less than the boundary, transfer time will gradually increase. The performance gain of SDPR-VC2 over BL-VC4 was about 36% across all injection rates and buffer depths. As a result, SDPR with packet length boundary attained a significant improvement in latency performance

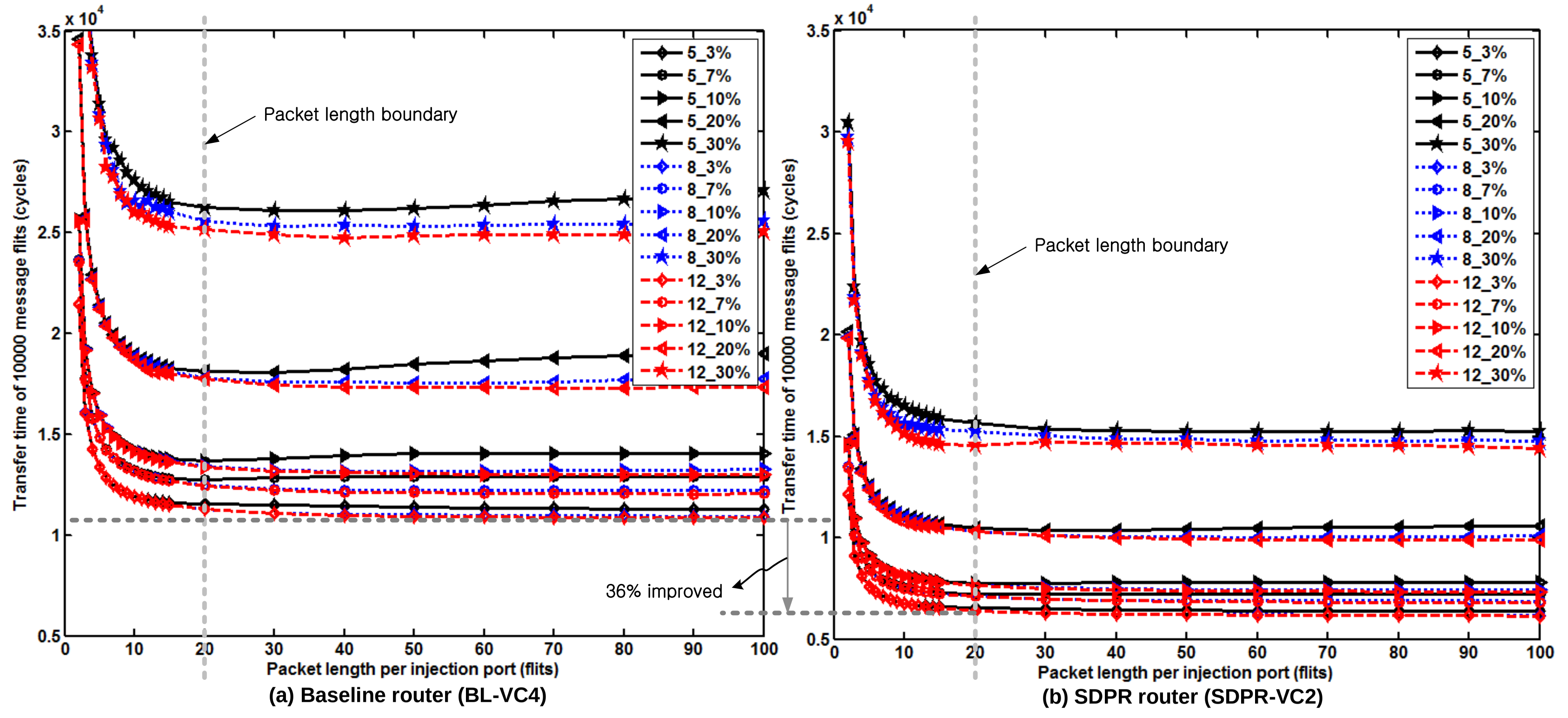


Fig. 50.: Transfer time of 10,000 message flits under various packet lengths, buffer depths, and injection rates in baseline and SDPR NoCs. Solid, dotted, and dashed lines show the simulation results using 5, 8, 12 buffer depths respectively.

regardless of buffer depth and injection rate.

### 4. Synthesis Results

We also examined the hardware overhead of the SDPR router to tradeoff strengths achieved from the improved link utilization and weaknesses caused by the hardware overhead for external injection and ejection ports and additional logic. Fig. 51 profiles and compares the synthesis results of the fully synchronous baseline and SDPR routers. We synthesized these two routers using Synopsys Design Compiler on TSMC 45nm technology with 20% default switching activities. Internal blocks of each router such as input unit, VC allocator, switch allocator, crossbar switch, output unit, and router top were synthesized to demonstrate power and area overheads. The power and area results of input unit were relatively greater than other blocks because it included twenty 4-depth, 64-bit width FIFOs (i.e. four FIFOs times five ports). The results of SDPR-VC2 denote slightly higher power/area overheads than BL-VC4 due to the supplementary allocation and switch logic. However, these overheads involved are inexpensive changes relative to the benefit of SDPR in terms of improvement.

## D. Conclusions

We present the WaveSync NoC design, which enables very low-latency communication in GALS NoC designs under low injection rates. By clocking portions the downstream node and processing element with the incoming source synchronous clocks on links, the WaveSync architecture allows packets propagating along the same path as the clock to skip synchronization entirely thereby allowing data to move as fast as it would in a long-combinatorial path. We also evaluate the performance of a near-half-cycle synchronizer architecture to reduce synchronization latency when synchronization is unavoidable, fur-

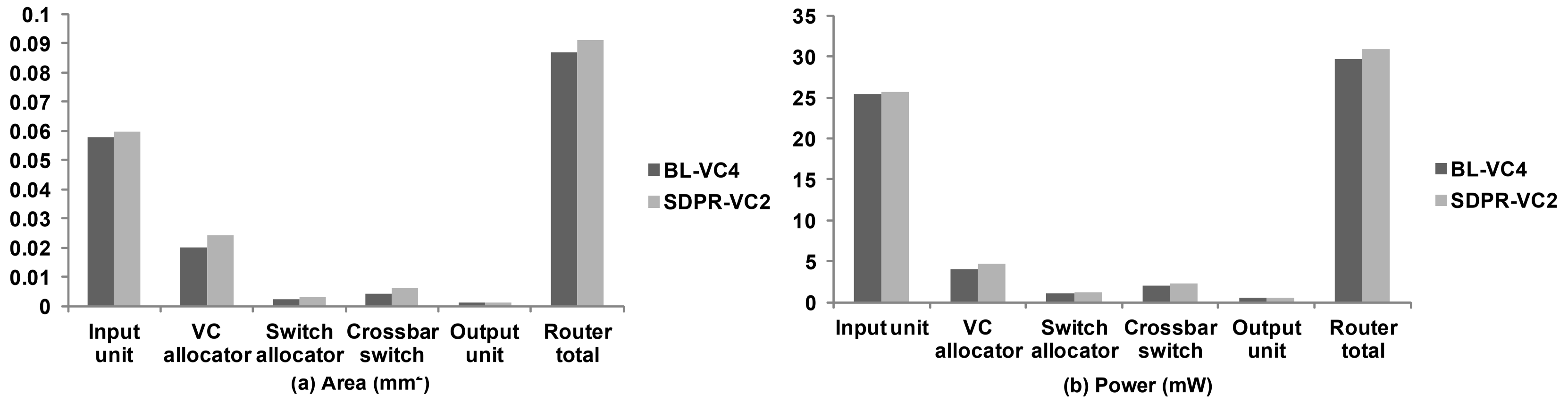


Fig. 51.: Synthesis results of BL-VC4 and SDPR-VC2 with 4-depth FIFO@1GHz

ther reducing per-hop latency. The proposed WaveSync design results in an improvement in average latency of 68% over the baseline GALS and 55% over ABC across the SPLASH-2 benchmarks.

Deterministic routing algorithm such as DOR is widely used in 2D mesh NoC because it provides simple algorithm and low-cost implementation. However, its performance in latency can be insufficient due to no path diversity. We observe that there are two inherent minimal paths (XY and YX DOR paths) on a 2-D mesh network. Simultaneous packet injecting via the two paths can enhance performance and gain better link utilization, but the two packets should have the same destination address to be injected at the same time.

We also propose a solution for this problem using the SDPR scheme. The SDPR architecture statically exploits the path diversity in the network to improve link utilization. In particular, the proposed SDPR technique mitigates the lack of path diversity and utilization of DOR by splitting a packet to two halves that involve the same source-destination address and injecting them simultaneously in parallel via separate and independent orthogonal two paths (i.e. XY and YX). By using dual injection and ejection ports dedicated at the router, parallel traversal along with XY DOR and YX DOR channels incurs a marginal logic and power overhead.

The experiment results demonstrate that SDPR can outperform the traditional DOR-based single injection scheme under long packets across all traffic patterns. We performed the SDPR evaluation using different synthetic workloads and realistic H.264 video traces and SPLASH-2 benchmarks to show significant reduction in the average packet latency. In the results, the SDPR router achieves 31-40% average reduction in latency across all synthetic under long packet simulations and 10% across SPLASH-2 realistic workloads, compared to the baseline router. The fully synthesizable SDPR router occupies 30.89mW power and 0.091mm$^2$ area with 3.7% and 4.7% power and area overheads over the baseline router respectively.

# CHAPTER IV

# DATA PROCESSING ACCELERATOR ARCHITECTURE FOR LOW-POWER SOCS IN DISTRIBUTED SENSOR NETWORK SYSTEMS

We present a data processing and control logic design for a new radiation detection sensor system that can generate data at or above Peta-bits-per-second level. The logic consists of novel data lossy compression components and operation strategies including low-power and network-on-wafer solutions. The design goal is to achieve subtle data compression before the information is ferried to the network, and redundant processing and channels to minimize the loss of information. The result is a radiation detection system that can operate at scan-rate of billion frames per second.

## A. Data Processing Logic for Stacked Wafer-Scale CMOS Radiation Sensor Network

To further ensure safety from unauthorized transport and distributed radioactive material, a detection system is needed to perform monitoring and serve as an early warning system. This has led to the development of various detection systems with wireless sensor networks [118, 119]. In [118], distributed sensor network (DSS) systems composed of commercial hardware for radiation detection are studied and simulated. A radiation sensor network for emergency prototype was presented in [119] to handle radiation information through a sensor network.

This section presents a data processing architecture for a low-power radiation sensor system and its implementation. This system is developed to detect radiological sources such as nuclear weapons, improvised nuclear devices, radionuclide materials, and space radiations. The design provides a data processing logic and implementation with a router design for an on-chip network. The novelty of our design includes signal processing logic for data compression and management, on-chip routers for a low-power multi-layer wafer-scale radiation sensor network. Fig. 52 shows the whole mechanism of the sensor network. In the figure, when radiation strikes a wafer, a photodiode emits a current pulse through one row and one column of the sensor array [120]. The position and severity of detection events can be recorded by sensor arrays and compressed into packets by data processing logic. With the information, we can construct three dimensional images of multiple particle interactions. To connect the sensor arrays and to transmit the compressed packets to main controllers (redundant) in each wafer, we employ network-on-wafer (NoW) architecture.

### 1. Data Processing Logic

One of key challenges of our detector design is minimizing the amount of data recorded and transmitted from a detection event without losing critical information. Lossless compression, often used in medical imaging applications, guarantees the integrity of the data without distortion. In contrast, lossy compression reduces data with reasonable distortions but can achieve higher compression rates [121]. In our design, a novel lossy data compression scheme for sensor arrays is implemented. Fig. 52(e) shows a sensor unit used as a basic detection unit on a detection wafer. The row and column axes of the sensor unit have 1024 lines respectively, and each line is enabled when a radiation strikes one of photodiodes in the line. As a result, we need to process up to 1024x1024 bits per array where each bit indicates a detection or lack thereof. For instance, when we use 10 multi-layer sensor wafers and each with 100 sensor units at 1 GHz operating clock, the amount of data needed to be processed reach to $10^{18}$ (*quintillion*) bits per second in the worst case unlikely scenario. To reduce the volume of processing data, our compression scheme extracts only necessary information from the detection data of the sensor arrays.

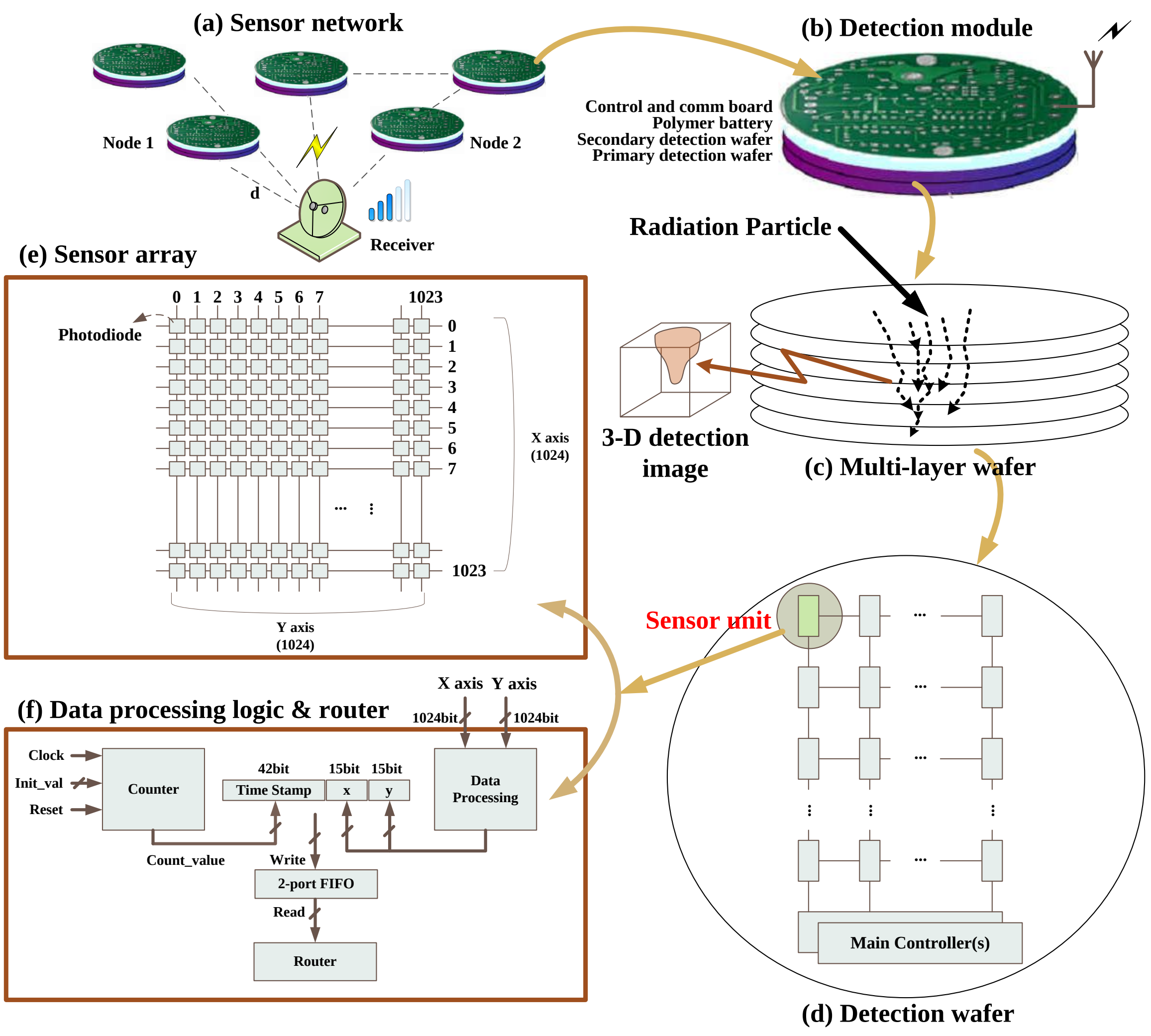


Fig. 52.: Overview of the proposed radiation detection sensor system

### a. Overall Architecture of Data Processing Unit

Fig. 52(f) illustrates the architecture of the proposed data processing unit, consisting of counter, data compression unit, 2-port FIFO, and an on-chip router. The counter unit generates time stamp information and inserts it into every packet with detection information created by the data compression unit every event if any particles are detected. In particular, the data processing logic is capable of compressing 1024x1024 bit/array input to 72-bit/array output packet per clock cycle. The output packet is composed of 42-bit time stamp recoding the detection event time (a clock counter value) and two compressed results (x and y) filled with 1-bit detection flag, 4-bit resolution, and 10-bit address. The detection output data, x and y, show the range (distance or resolution) and the representative position (address) of detected pixels at every event. Using the resolution and address information, we can construct time-varying 3-D images and estimate the angle of incidence, intensity, and the type of radiation source.

### b. Proposed Data Compression Algorithm

The main idea of the detection algorithm is that when the sensor unit detects multiple particles, the compression unit does not need to send all information of detected positions but sends minimal information necessary to reconstruct estimated detection information representing the position and severity of detection events since the detected particles are more likely to be gathered in a specific area. Therefore, we can compress 1024 bits indicating the detection positions of hit particles in an axis to 1-bit detection flag, 4-bit resolution data, and 10-bit address designating the smallest index in the 1024 detection positions. Table XXII shows the resolution of detected bits in a row or column axis. If the 4-bit resolution is 5, it means that the maximum and minimum number of detected particles are 32 ($2^{resolution} = 2^5$) and 17 ($2^{resolution-1} + 1 = 2^4 + 1$) respectively.

Table XXII.: Resolution and detection bits

| Resolution | $n$ ($0 \sim 10$) |
|---|---|
| Detection bits | $2^n$ ($1 \sim 1024$) |

Fig. 53 illustrates this compression scheme. In EX1, the detection positions of input data are widely distributed from 0 to 1023 in an axis. In this case, 4-bit resolution representing detection range is 10, and 10-bit address, the smallest index of a detected particle, is 0. As a result, 15-bit detection data is '1_1010_0000000000' in binary. In EX4, the resolution is 8 because the range of detection is from in[356] to in[530] (range=$530-356+1=175<2^8$), and the address is 356.

## c. Data Compression Unit

The data processing unit for compression is mainly composed of address generation and resolution generation. Fig. 54 presents the architecture of data processing logic. There are 10 pipeline stages in the block, and we exploit divide and conquer strategy to build address and resolution outputs at each pipeline stage. In the first stage, Init1$\sim$512 blocks that receive 2-bit detection data through input ports generate next 3-bit output data that contain 1-bit detection flag, 1-bit resolution output, and 1-bit address output. In the next stage, Main1$\sim$256 blocks generate new 2-bit resolution/address (i.e. 1-bit flag, 2-bit resolution, and 2-bit address) using two 1-bit resolution/address inputs. In the last stage, Main9 block creates a 15-bit detection packet composed of 1-bit flag, 4-bit resolution, and 10-bit address.

The address and resolution generators creating a next address and resolution are shown in the left of Fig. 54. The address generator selects the smallest index among detected inputs, therefore if there are two detected bits (latched_in[1:0]), a smaller index will be chosen. For instance, when in[0]=1(detected) and in[1]=1(detected), the result of address

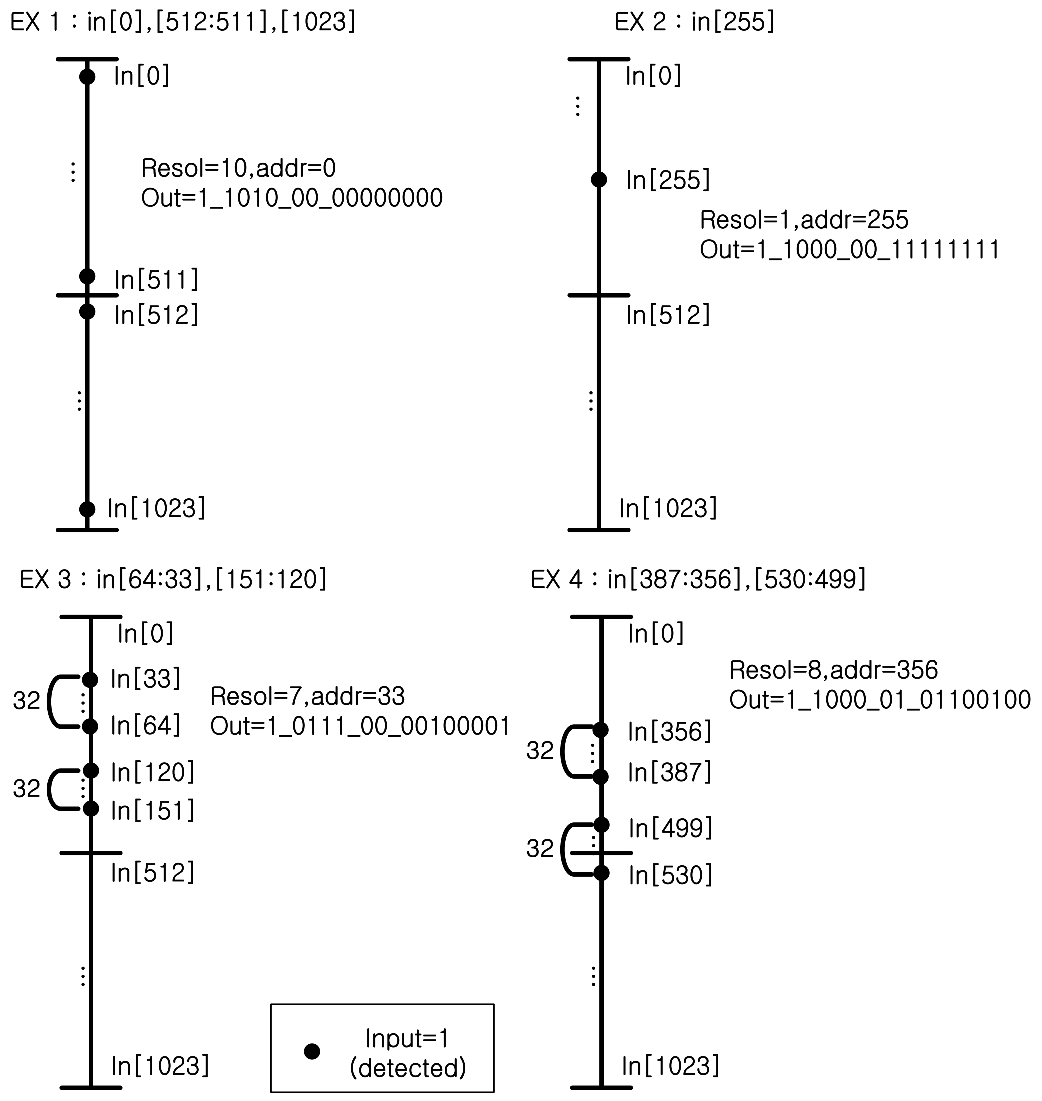


Fig. 53.: Data compression scheme

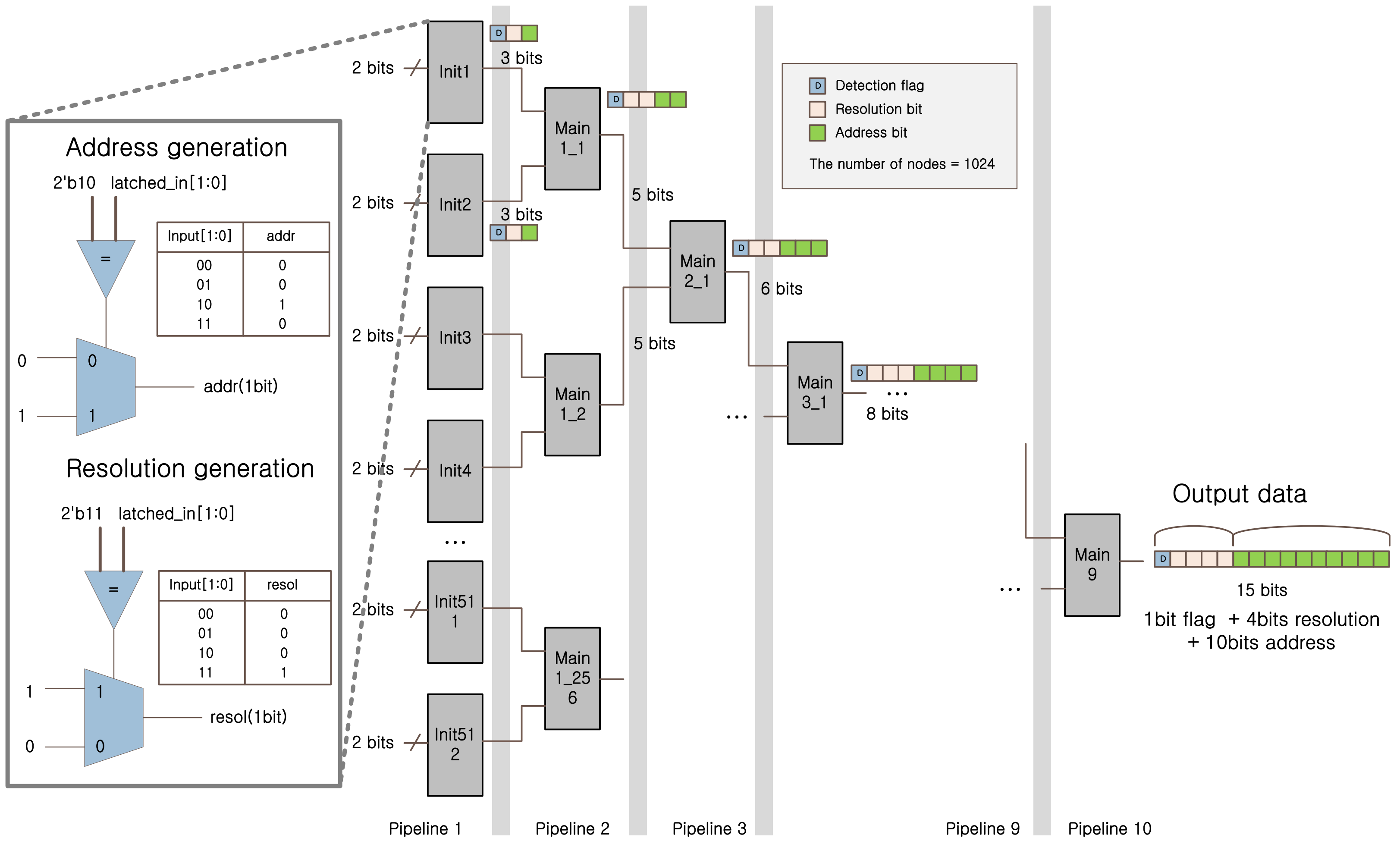


Fig. 54.: Data processing unit for compression

generator is 0 which is the smallest index. The resolution generator creates new resolutions by using two resolution inputs generated in the previous pipeline stage. In Fig. 54, the resolution generator receives two 1-bit detection inputs and computes a 2-bit resolution output. For example, when in[0]=1(detected) and in[1]=1(detected), the updated resolution will be 1 since there are two detected particles (i.e. $2^1 = 2$).

Fig. 55 shows the method generating a new address and resolution output in Module C using address and resolution inputs delivered from the previous Module A and B. First, a new address can be determined by the smallest index of A and B. Accordingly, the address of C will be the address of A, 15. A new resolution can be computed by the addresses of A and B and the resolution of B. As the resolution of B (b) implies the maximum distance from the address of B in the Module B, total distance from the address of A to the end of detection area of B in the Module C will be (a)+(b), which equals (c). In this case, the resolution of the Module C will be 10 since the address range (c) is 661.

### 2. On-chip Router Design

Network-on-Chips (NoC) has emerged as an alternative of the traditional bus-based interconnecting between sensing and processing array elements (SPE), to increase bandwidth and reduce interconnection complexity on a chip. As shown in Fig. 56, each SPE in an NoC architecture is linked through a router, and data are transferred in the form of packets which are subdivided by flits (Header, Body, and Tail). In our system, the idea of NoC is simply expanded to the wafer-scale while achieving power efficient data communication. The NoC paradigm fails when reliability of all components (SPEs and interconnections) on the wafer cannot be guaranteed. To tolerate component failure on a wafer, protocols are placed to ensure the system as a whole can continue to function. In our system, the redundant main controllers on each wafer are responsible for determining failures of individual links or nodes. This built-in self-test (BIST) will dynamically modify the routing

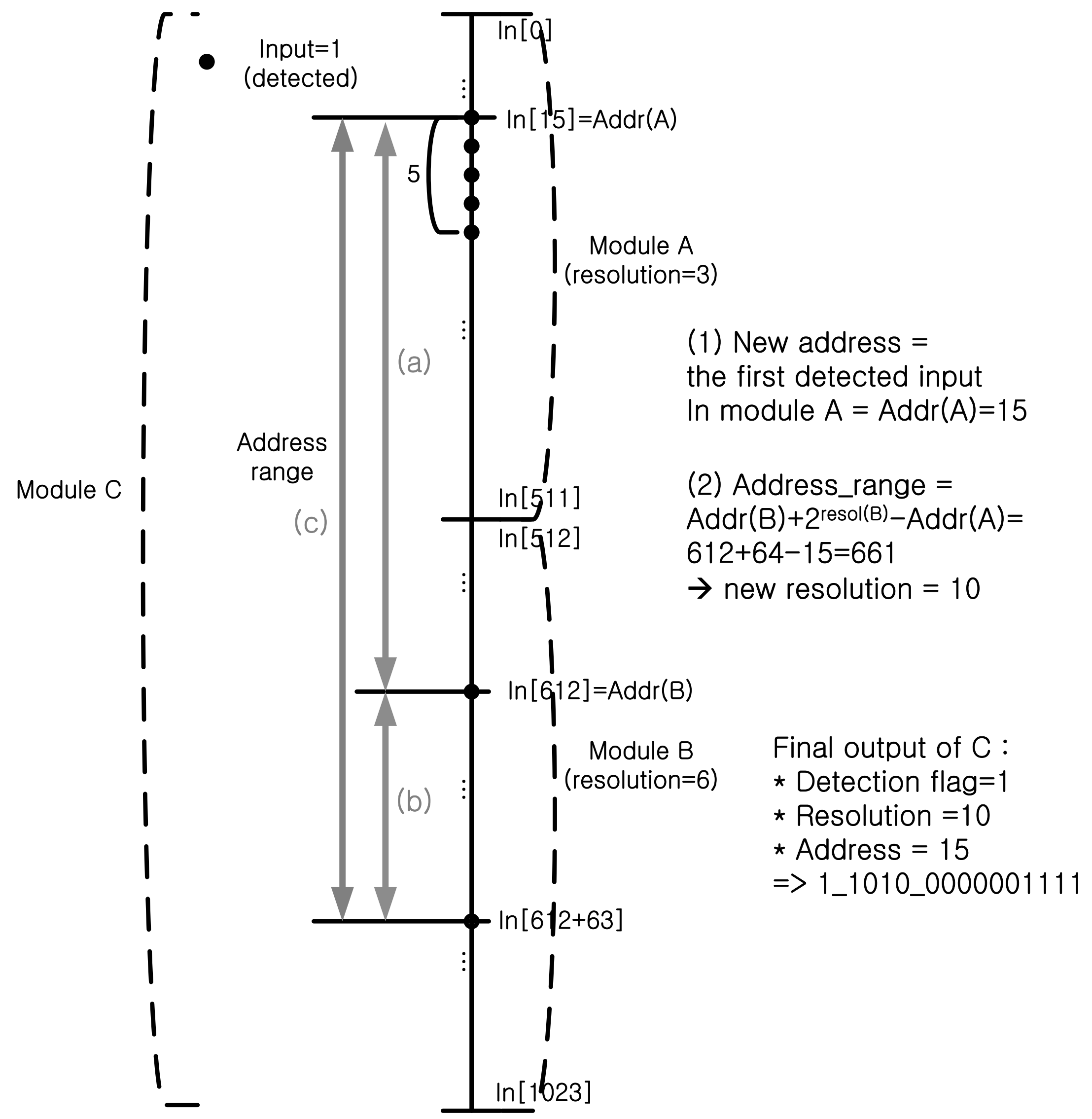


Fig. 55.: Generating new address and resolution outputs using address and resolution inputs created in the previous pipeline stage

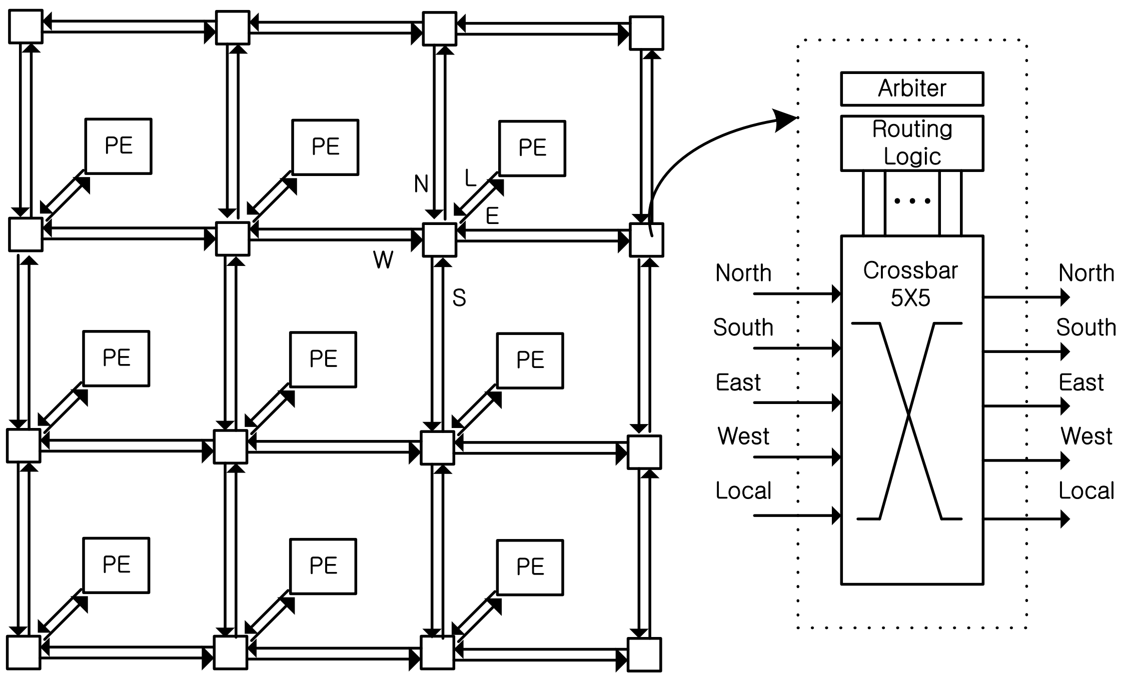


Fig. 56.: Baseline router

table to navigate data transmission around these failures and defects. Equally important as reliability is the goal of reducing power consumption. A globally asynchronous locally synchronous (GALS) architecture is also proposed for our NoW design, eliminating the need for complex clock trees, which are expensive in both cost and power. This architecture reduces power consumption to near zero during standby/detection mode [122]. Another substantial challenge facing designers of NoWs is the need for resilience. A unified framework of coding for SoCs with crosstalk avoidance codes and error control codes was proposed [123]. The paper investigates combining existing coding schemes and provides practical coding schemes to reduce delay and energy and increases reliability. The baseline router we used in the proposed design is a standard 2D mesh, pipelined router with virtual channels (VC) [86, 102]. The pipeline is composed of 2 stages (i.e. route computation and arbitration) at output port.

Table XXIII.: Synthesis results of the data processing logic and baseline router at 1GHz

| | Data processing logic | Router |
|---|---|---|
| Target Library | Input : 1024x1024 bit/cycle<br>Output : 72 bit/cycle | 128 bit/cycle |
| Compression Rate | 99.99% | - |
| Maximum Frequency | 1GHz | 1GHz |
| Area | 60358 | 32060 |
| Power Consumption | 112.1718mW | 79mW |

## 3. Experiments

The data processing logic and baseline router was implemented in Verilog. We synthesized them using Nangate 45nm open library at 1GHz (Table XXIII). In the design, the reduction rate of data was 99.99% because input (1024x1024 bit/cycle) was compressed to 72 bit/cycle. In the synthesis result, the data processing logic consumed 112.17mW @ 1GHz, and router consumed 79mW @ 1GHz with 128-bit data width. The data compression block consists of a pipelined architecture (10 pipelines), therefore, it could be synthesized at high frequency, 1GHz.

## B. DSP Accelerator for Low-Power Sensor Hub SoCs

A sensor hub SoC typically contains a heterogeneous mix of hardware blocks such as a small embedded microprocessor managing sensors, DSPs or dedicated hardware accelerators to perform complex DSP algorithms for massive data. The data can be sensed by angular momentum, GPS location, magnetic compass heading, temperature, pressure, sound and light. In this sensor hub SoC design, the main components dominantly consuming power and area are signal processing cores. General purpose DSPs can provide developers flexibility and quick development of DSP-oriented algorithms. However, the chip area and energy consumption per operation of DSP is relatively higher than dedicated hardware logics performing the same operations due to overheads to support wide functionality. In particular, power consumption is a major concern as demands on the sensor hub SoCs since they are typically used in portable and battery restricted sensor systems. All these requirements for low-power data processing for the sensor hub SoC make extensive use of a DSP accelerator. The DSP accelerator provides a compromise between the performance gains of fixed-functionality hardware and the flexibility of software-programmable tasks, enabling energy-efficient digital signal processing.

In this paper we make the following contributions to the DSP accelerator design:

- We propose a DSP accelerator design for low-power and low-cost SoCs for sensor network systems. The low-power DSP accelerator performs DSP operations such as single instruction multiple data (SIMD) style multiply and accumulate (MAC), FFT/IFFT, FIR, and 3-D cross product (CP). This accelerator is developed as a dedicated hardware to obtain small area and low-power performance but provides programmable functionality to run fundamental DSP algorithms when compared traditional hardwired accelerators.

- The DSP accelerator does not contain large internal registers such as general purpose registers (GPRs). The GPRs are typically used in DSPs to load data from a larger memory and to hold temporarily data and intermediately result, but demanding additional area and power. The DSP accelerator loads and stores data from and to external SRAM through AHB bus; it leads to low area and power consumption like dedicated hardware accelerators.

- We propose fast FFT, CP, FIR executions using a pipeline architecture that supports large loop operations effectively. The result shows the proposed DSP accelerator outperforms general purpose DSPs in throughput performance, close to that of a hardwired accelerator with programmable control as well as with low-power requirement.

The detail features of the proposed DSP accelerator will be shown in the next subsection.

### 1. Background: DSP Algorithms

#### a. Vector Dot Product

Vector dot product (or sum of products) is the most fundamental operation in DSP. It is widely used for convolutional algorithms such as finite impulse response (FIR) filters. FIR filters compute the convolution between the filter coefficients and the delay line values. This is described in equation (4.1).

$$Y_n = \sum_{i=0}^{N-1} C_i X_{n-i} \tag{4.1}$$

where N is the number of coefficients, $C_i$ are the coefficients of the filter, and $X_n$, $Y_n$ are the n$th$ terms of the input and output sequences respectively. This vector dot product algorithm can be implemented by multiply and accumulate (MAC) operations with zero-overhead looping technique. MAC is a very common low-level operation used in many DSP algorithms. This operation multiplies two numbers and adds into an accumulator register. The DSP accelerator provides 8-,16-,32-bit SIMD MAC operations for programmable FIR filters.

b. 3-Dimensional Cross Product

The 3-D CP calculates a new vector from two vectors in 3-D space. The resulting vector is perpendicular to the two original vectors. This operation is used to compute the normal for a triangle or polygon. It is also used for computational geometry applications in computer graphics. We design a fast 8-,16-,32-bit CP operation for these computationally intensive applications. For vectors A=($a_1$, $a_2$, $a_3$), B=($b_1$, $b_2$, $b_3$), C=($c_1$, $c_2$, $c_3$) in $R^3$, the CP is defined by equation (4.2).

$$C = A \times B = (a_2b_3 - a_3b_2)i + (a_3b_1 - a_1b_3)j + (a_1b_2 - a_2b_1)k = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \tag{4.2}$$

where i,j,k are unit vectors.

c. Fast Fourier Transform (FFT)

The fast Fourier transform (FFT) is an efficient algorithm to calculate the discrete Fourier transform (DFT). It is also one of the most primary used operations in digital signal processing. The DFT transforms N discrete-time samples (n=0 to N-1) to the same number of discrete frequency samples, and is defined as

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)W_N^{nk} \tag{4.3}$$

where $k = 0$ to $N-1$, and $W_N$, the twiddle factor, is defined as

$$W_N = e^{-j2\pi/N} \tag{4.4}$$

In practice for large series, DFT takes significant time proportional to the square of the number on points. A much faster algorithm has been developed by Cooley and Tukey in 1965 called the FFT. Radix-2 Cooley-Tukey decimation-in-time (DIT) FFT breaks each DFT computation into the combination of two DFTs, one for even-indexed inputs and another for odd-indexed inputs [124]. The decomposition continues until a DFT of just two inputs remains. The 2-point DFT is called a butterfly, and it is the simplest computational kernel of radix-2 FFT algorithms.

## 2. DSP Accelerator Features

Table XXIV presents the features of the DSP accelerator. It supports 8-, 16- and 32- fixed-point data and operates in SIMD operations for handling MAC, FFT/IFFT, CP, and FIR filter processing. In the accelerator, the SIMD supports up to 4 operations (4x8=32 bits) at the same time. It can generate up to two read-memory addresses and one write-memory address per cycle using three address generate units (AGUs). A 32-bit multiplier is built on 16x8-bit multipliers to support 8-, 16-bit and 32-bit multiplications. The 8-, 16- and 32-bit multiplied results are accumulated into a 80-bit accumulator with saturation process, and an auxiliary 60-bit accumulator is implemented to accelerate 8-, 16- and 32-bit FIR operations. The accelerator supports programmable shift, guard bits, rounding, and saturation to provide shifted, rounded, saturated and guarded outputs in the accumulator. Special addressing modes such as bit-reverse for decimation-in-time (DIT) radix-2 FFT/IFFT operation, programable increment/decrement by n, and zero overhead looping for optimized control flow and background processing are also supported.

## 3. DSP Accelerator Architecture

SoC top block and the architecture of DSP accelerator are shown in Fig. 57, 58. The accelerator includes separated read and write channels to process pipelined operations for FFT/IFFT, MAC, CP, and FIR. For two read channels and one write channel, three 32-

Table XXIV.: DSP accelerator features

| | |
|---|---|
| Numeric features | 8/16/32-bit fixed point |
| Memory channel | 3 x 32-bit read/write channels |
| Multiplication | 16 x 8-bit multipliers |
| Accumulator | 80-bit (+ 60-bit for FIR) |
| SIMD | 4-way 8-bit / 2-way 16-bit / 1-way 32-bit |
| Numeric fidelity | Programmable shifter, guard bits, rounding, saturation |
| Addressing modes | Bit-reverse, cyclic, inc/dec by n, zero overhead looping |
| Operations | 8/16/32-bit dot product, |
| | 8/16/32-bit cross product, |
| | 32-bit complex (16-bit real and 16-bit imaginary) radix-2 DIT FFT/IFFT |
| | 8/16/32-bit FIR |

bit address and data ports are built in the accelerator. When two operands are loaded by two read channels, a 32-bit output will be written back through the write channel in the next cycle. Thus the operands read, execution, and output write-back can performed in a pipelined fashion to increase operating speed. Typically DSPs exploit general purpose registers to store any transient data required by the program. The DSP accelerator, however, does not accommodate any memory inside the block to store operands, outcomes, and any transient data. Therefore, whenever the accelerator executes a command (or instruction), it is necessary to load two operands or transient data every time from outside SRAM memory through AHB bus. This gives both pros and cons. The accelerator wastes SRAM read/write cycles to load input data and write back output data through AHB bus for every instruction, but it keeps small area and low-power consumption due to not including power dominant data register block. Control unit controls address generation, receives 96-bit commands (i.e. instructions) from a micro-controller through AHB$^{TM}$ [125] slave port, and stores them into internal command FIFOs (i.e. instruction memory).

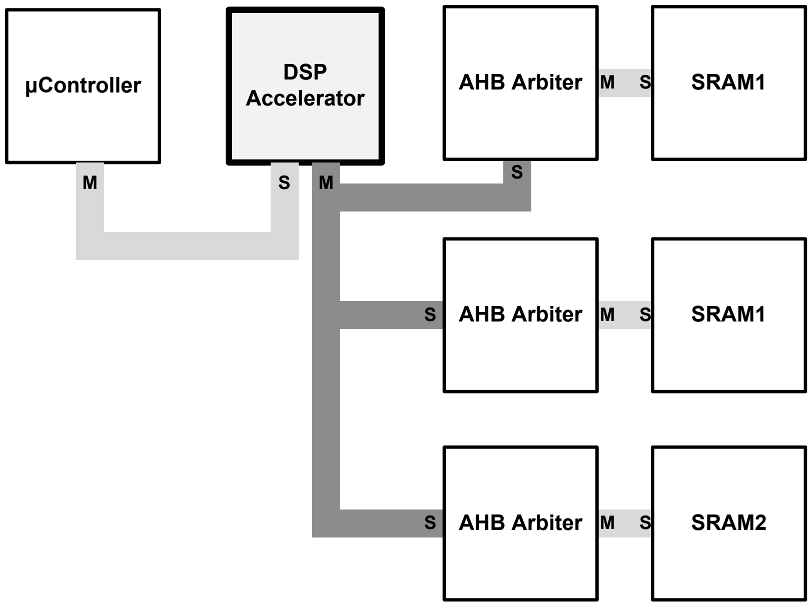


Fig. 57.: Micro-controller, DSP accelerator and data memory interface in the proposed sensor hub SoC

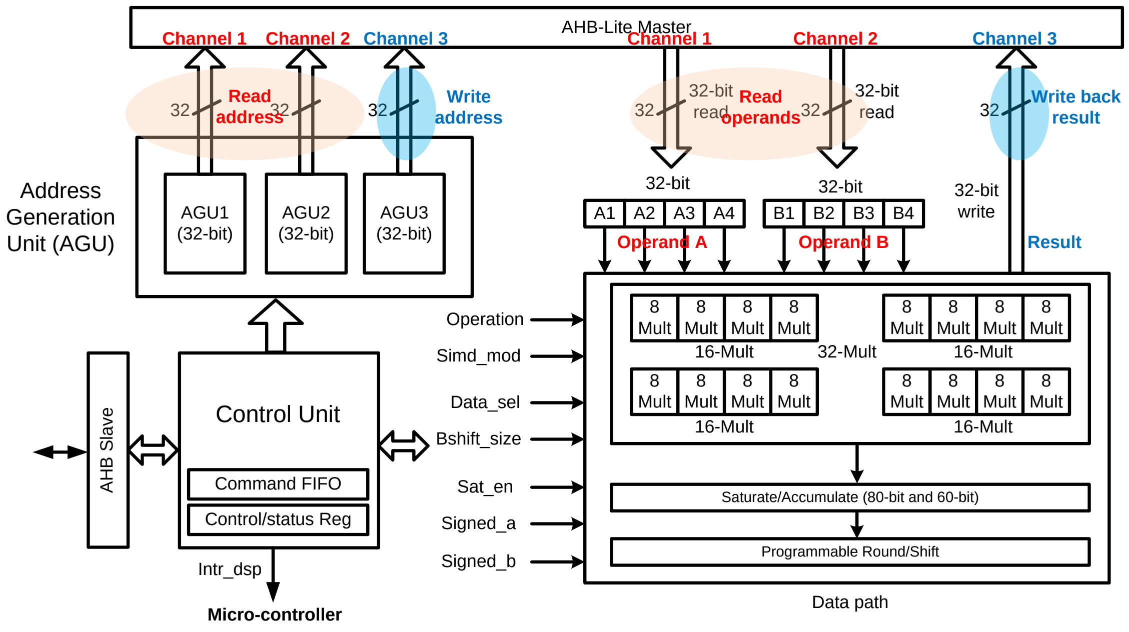


Fig. 58.: DSP accelerator architecture

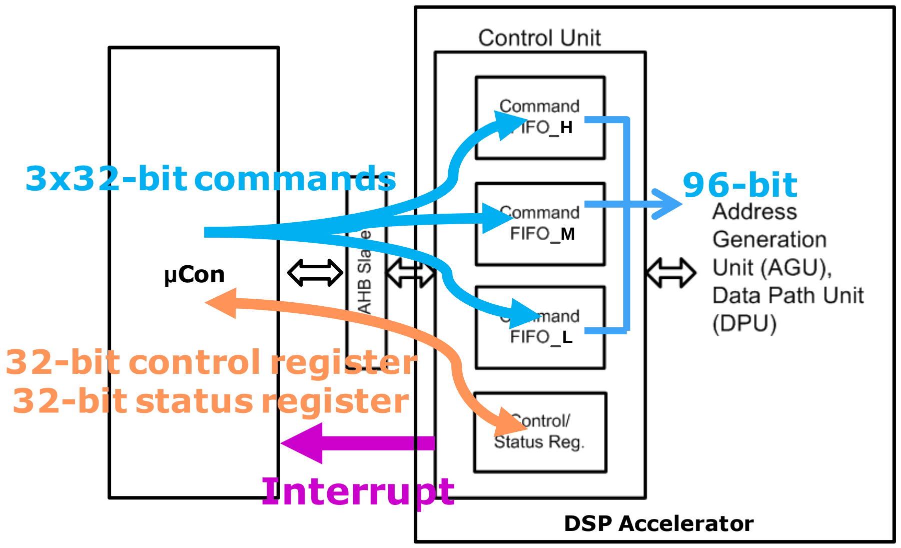


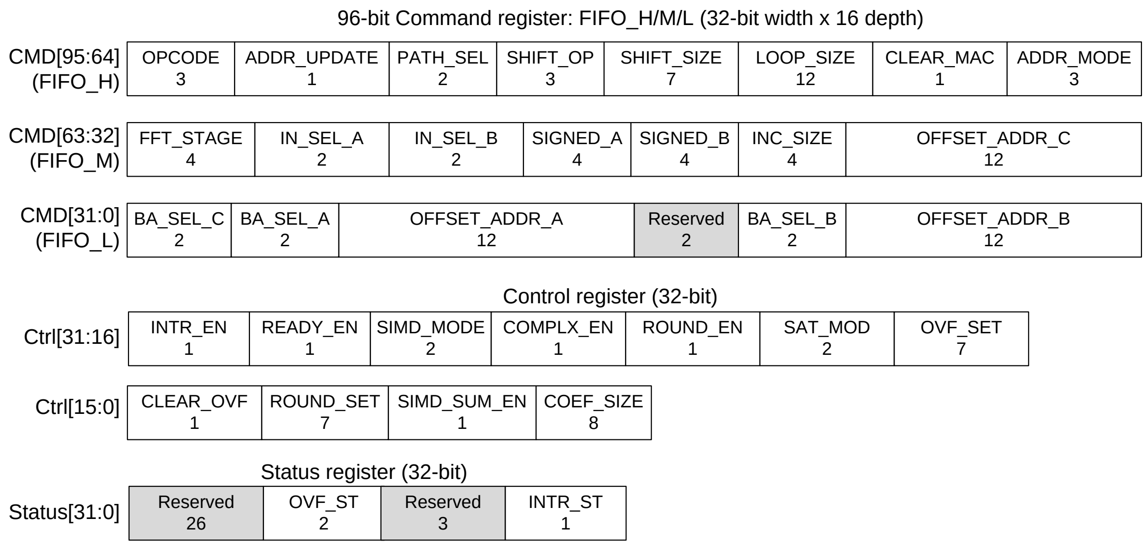
Fig. 59.: Interface with $\mu$Con

a. Address generate unit (AGU)

Three AGUs generate 32-bit addresses for two 32-bit read data and one 32-bit write data per cycle. AGUs support special address modes such as bit-reverse, increment/decrement and cyclic addressing.

b. Data path unit (DPU)

AHB channel 1 and 2 are assigned for two read operands, and AHB channel 3 is used for writing back a 32-bit result. When two operands are loaded for an operation, DPU performs the operation using loop-level pipeline. We will discuss this pipeline design for loops in the pipeline architecture. The DPU consists of 16x8-bit multipliers and a 80-bit accumulator for 8-, 16- and 32-bit MAC operations. FIR requires a 60-bit supplement accumulator to diagonally accelerate 8- and16-bit FIR filtering processes. Rounding, shifting, and saturating are also provided by the data path. The arithmetic logic unit (ALU) of the DPU incorporates the 16x8-bit multipliers and 80-bit and 60-bit accumulators. All operations



96-bit Command register: FIFO_H/M/L (32-bit width x 16 depth)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| CMD[95:64] (FIFO_H) | OPCODE 3 | ADDR_UPDATE 1 | PATH_SEL 2 | SHIFT_OP 3 | SHIFT_SIZE 7 | LOOP_SIZE 12 | CLEAR_MAC 1 | ADDR_MODE 3 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| CMD[63:32] (FIFO_M) | FFT_STAGE 4 | IN_SEL_A 2 | IN_SEL_B 2 | SIGNED_A 4 | SIGNED_B 4 | INC_SIZE 4 | OFFSET_ADDR_C 12 |

| | | | | | | |
|---|---|---|---|---|---|---|
| CMD[31:0] (FIFO_L) | BA_SEL_C 2 | BA_SEL_A 2 | OFFSET_ADDR_A 12 | Reserved 2 | BA_SEL_B 2 | OFFSET_ADDR_B 12 |

Control register (32-bit)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Ctrl[31:16] | INTR_EN 1 | READY_EN 1 | SIMD_MODE 2 | COMPLX_EN 1 | ROUND_EN 1 | SAT_MOD 2 | OVF_SET 7 |

| | | | | |
|---|---|---|---|---|
| Ctrl[15:0] | CLEAR_OVF 1 | ROUND_SET 7 | SIMD_SUM_EN 1 | COEF_SIZE 8 |

Status register (32-bit)

| | | | | |
|---|---|---|---|---|
| Status[31:0] | Reserved 26 | OVF_ST 2 | Reserved 3 | INTR_ST 1 |

Fig. 60.: Configurations of command, control and status registers

performed in the DSP accelerator share the ALU block. This leads to the minimization of hardware overhead and the increase of resource utilization. For instance, the high-order FFT/IFFT and 3-D CP operations can be implemented by sharing the data path with minor control logic overhead and configuration registers for programmable features.

c. Control unit (CU)

CU block controls address generation units and data path based on commands and control information configured by an on-chip micro-controller. Fig. 59 shows interface between the micro-controller (i.e. $\mu$Con) and the DSP accelerator. $\mu$Con sends a 96-bit (3x32bit) command by AHB slave interface, and the 96-bit command is divided to high, middle, low 32-bit commands and stored into command FIFO_H,_M,_L, respectively. Each FIFO has 32-bit data width and 16 depth; therefore, total 16 x 96-bit commands can be stored in the FIFOs. When commands are available, CU reads a 96-bit command from FIFO_H,_M,_L, at a time, generating control signals for AGU and DPU. Control information in a command is only valid for the command. However, 32-bit control information in the control register of CU is effective over the DSP accelerator. So the 32-bit control register stores global control configurations such as interrupt enable, SIMD mode, complex number enable for FFT, and so on. The 32-bit status register reports overflow and interrupt occurrence.

d. Command memory

Fig. 60 depicts these 96-bit command configurations (CMD[95:0]), 32-bit control (Ctrl[31:0) and status (Status[31:0]) registers. The command fields in the figure denote as follows.

- OPCODE (3-bit) signifies operation mode. 0: no operation, 1: multiplication, 2: MAC addition, 3: MAC subtract, 4: CP, 5: FFT, 6: IFFT, 7: FIR
- ADDR_UPDATE (1-bit): When this bit is enabled, three read/write addresses in AGUs are initialized by base addresses and offset addresses.
- PATH_SEL (2-bit) selects a path to apply the shift operation. 0: operand A, 1:operand B, 2: MAC
- SHIFT_OP (3-bit) configures the mode of shift operation. 0: bypass shifter, 1: logical right shift, 2: logical left shift, 3: arithmetic right shift, 4: arithmetic left shift
- SHIFT_SIZE (7-bit) shows the magnitude of shift.
- LOOP_SIZE (12-bit) denotes m when the number of points is $2^m$ in FFT/IFFT. This field also denotes the number of input samples to compute loop size in FIR. Otherwise, it represents the number of loops in an operation.
- CLEAR_MAC (1-bit) initializes accumulators for MAC

- ADDR_MODE (3-bit) selects the address mode. 0: none, 1: increment (+INC_SIZE), 2: decrement (-INC_SIZE), 3: cyclic-increment, 4: cyclic-decrement, 5: bit-reverse, 6: FIR

- FFT_STAGE (4-bit) denotes FFT/IFFT stage number in a command. The FFT/IFFT operation needs m commands to operate m stages in $2^m$ point FFT/IFFT. Therefore, each command contains a corresponding stage number.

- IN_SEL_A (B) (2-bit) selects two operand sources (A and B) from input sources. 0: A, 1: B, 2: accumulator. e.g. IN_SEL_A=0 and IN_SEL_B=2 means operand A and B are connected to input channel A and accumulator output respectively.

- SIGNED_A (B) (4-bit) indicates the sign of each byte in 4x8-bit data (i.e. 32-bit data). e.g. SIGNED_A=1010 then A[31:24]: signed, A[23:16]: unsigned, A[15:8]: signed, A[7:0]: unsigned

- INC_SIZE (4-bit) configures address increment/decrement size.

- BA_SEL_A (B) (C) (2-bit) selects base address (BA) for operands (A and B) or outcome (C). e.g. BA_SEL_A = 0 : select BA register 0 for loading operand A, BA_SEL_A = 1 : select BA register 1 for loading operand A, BA_SEL_A = 2 : select BA register 2 for loading operand A

- OFFSET_ADDR (12-bit) represents 12-bit immediate value for address offset. Effective address (EA) can be computed as EA = BA + OFFSET_ADDR.

e. Control register

This register involves global configurations to control the accelerator. The field description of the 32-bit control register shown as follows.

- INTR_EN (1-bit) enables interrupt generation. When this is enabled, CU creates a cycle valid interrupt signal to the micro-controller for interrupt handling.
- READY_EN (1-bit) supports interconnect with the micro-controller using hready signal. 0: DSP accelerator does not reflect its busy/available status on hready_resp 1: DSP accelerator reflects its busy/available status on hready_resp
- SIMD_MOD (2-bit): 0 : 8-bit 4-way, 1: 16-bit 2-way, 2: 32-bit 1-way
- COMPLX_EN (1-bit) enables 32-bit complex multiplication. SIMD_MOD must be 1 (16-bit) to perform 4x16-bit real and imaginary multiplications.
- ROUND_EN (1-bit) enables round operation
- SAT_MOD (2-bit) controls saturation mode. 0 : no saturation 1 : saturation for normal overflow (set guard bit) 2 : saturation for super-overflow (overflow guard bit)
- OVF_SET (7-bit) sets overflow position. e.g. when OVF_SET=7, data size is equal to 1-bit sign + 7-bit magnitude.
- CLEAR_OVF (1-bit) clears overflow bit in the status register.
- ROUND_SET (7-bit) denotes round position. e.g. when ROUND_SET = 7,
  if round_data[7:0]>"01111111" then round_data[8]=1; else round_data[8]=0.
- SIMD_SUM_EN (1-bit) enables the sum of results in 8/16-bit SIMD mode.
- COEFF_SIZE (8-bit) configures the number of coefficients in FIR filter.

f. Status register

This register informs the status of overflow and interrupt to the micro-controller. The field description of the 32-bit status register is presented as follows.

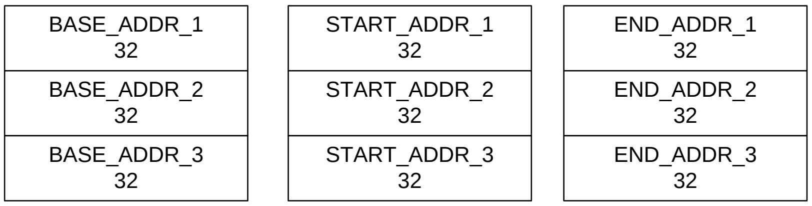


Fig. 61.: 32-bit address registers for the DSP accelerator. ADDR_1,_2,_3 represent addresses for channel 1,2,3 respectively.

- OVF_ST reflects the status of overflow in an operation. This field can be cleared by CLEAR_OVF. OVF_ST[1] denotes super-overflow which denotes the overflow of guard-bits. OVF_ST[0] presents normal overflow in the operation.
- INTR_ST shows interrupt status. When micro-controller reads this status, this field will be reset automatically.

g. Address register and map

Fig. 61 shows address registers for data read and write. There are three address categories, 1,2,3 for memory channel 1,2,3 respectively. Base addresses are used to denote read points for two operands and write point to write back an execution result. Effective address for data access is computed as EA = 32-bit BASE_ADDR + 12-bit OFFSET_ADDR. Start (START_ADDR) and end (END_ADDR) addresses specify begin and end address points for cyclic addressing mode.

Fig. 62 depicts the address map of the accelerator, showing all register addresses and external SRAM addresses. DSP_BaseAddress is the base address to access registers in the DSP accelerator. Sram_1,2,3_BaseAddress stand for the base addresses to access external SRAM1,2,3 by AHB bus respectively.

| Register | Address |
|---|---|
| **FIFO_H** | DSP_BaseAddress + 0x0 |
| **FIFO_M** | DSP_BaseAddress + 0x4 |
| **FIFO_L** | DSP_BaseAddress + 0x8 |
| **Control Register** | DSP_BaseAddress + 0x10 |
| **Status Register** | DSP_BaseAddress + 0x14 |
| **Base Address Register 1** | DSP_BaseAddress + 0x20 |
| **Base Address Register 2** | DSP_BaseAddress + 0x24 |
| **Base Address Register 3** | DSP_BaseAddress + 0x28 |
| **Start Address Register 1** | DSP_BaseAddress + 0x30 |
| **Start Address Register 2** | DSP_BaseAddress + 0x34 |
| **Start Address Register 3** | DSP_BaseAddress + 0x38 |
| **End Address Register 1** | DSP_BaseAddress + 0x40 |
| **End Address Register 2** | DSP_BaseAddress + 0x44 |
| **End Address Register 3** | DSP_BaseAddress + 0x48 |

| Memory | Address |
|---|---|
| **RAM_1** | Sram_1_BaseAddress (0x80000000) |
| | Sram_1_AddressRange (0x10000) |
| **RAM_2** | Sram_2_BaseAddress (0x80010000) |
| | Sram_2_AddressRange (0x10000) |
| **RAM_3** | Sram_3_BaseAddress (0x80060000) |
| | Sram_3_AddressRange (0x10000) |

Fig. 62.: Address map

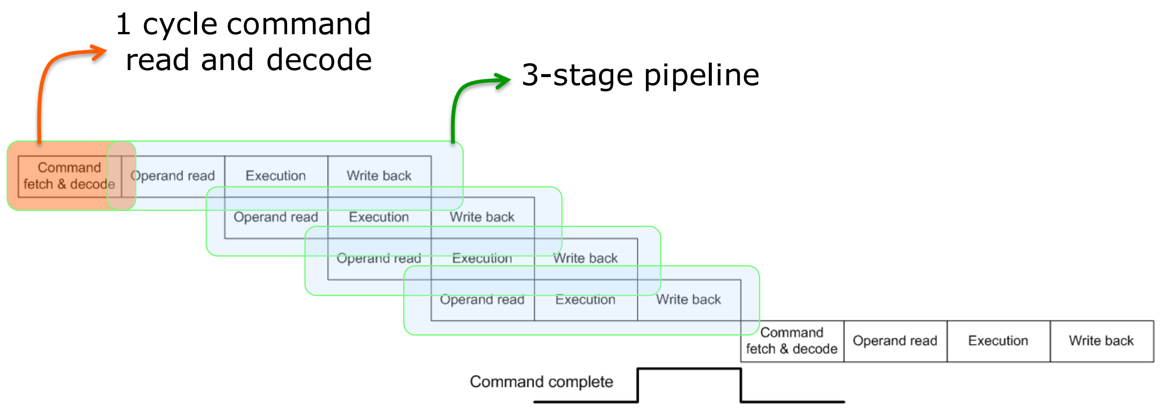
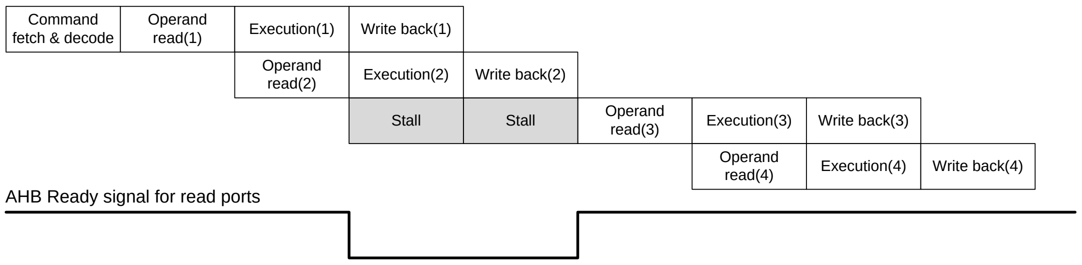


Fig. 63.: Pipeline architecture in the DSP accelerator

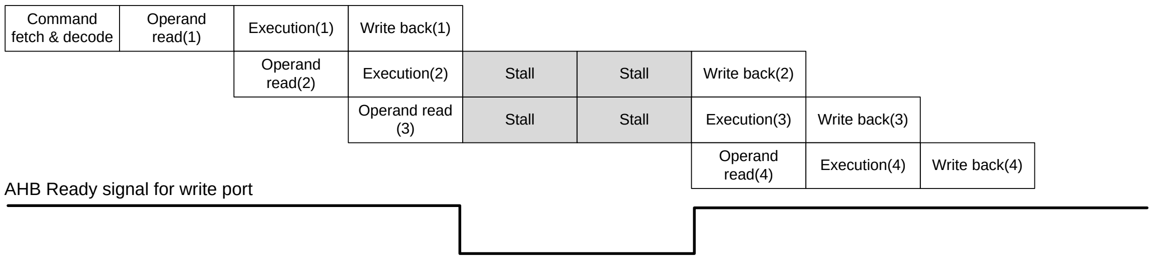
## h. Pipeline architecture

The DSP accelerator supports a three-stage pipeline to increase throughput performance. Fig. 63 shows the pipeline architecture for the proposed DSP accelerator. In the beginning of operating a command, the accelerator fetches the command and decodes it. This accelerator is devised to execute DSP oriented operations such as looping MAC operations or high-point FFT, so a command usually involves large loops. Thus we implement that the pipeline occurs only in multiple loops. We call this *loop-level pipeline*. It also implies that the accelerator does not support instruction (or command) level pipeline. This makes its architecture simple, achieving low area and power consumption. Moreover, this can prevent occurring intricate data/control hazard that is not easy to be solved in the AHB-based read/write architecture. The DSP accelerator does not include any general purpose registers occupying large area and consuming high energy, so every data (i.e. operands) should be directly loaded through AHB interface from an external SRAM or memory. Problem is that when the AHB read or write is not ready, the execution of pipelined loops will be delayed till the bus is available. This is also not predictable.

Once a command is fetched and decoded, the pipelined execution occurs in a loops



(a) AHB read port is not available − pipeline stall



(b) AHB write port is not available − pipeline stall

Fig. 64.: Pipeline stalls when AHB ready signals for read/write channels are not ready

operation. Fig. 63 depicts the pipelined loops operation when LOOP_SIZE=4. The first stage is operand read. Two operands are loaded through two read AHB channels simultaneously per cycle. The execution of an operation is performed in the second stage, and in the last stage the 32-bit execution result is written back to external memory through the write AHB channel.

When AHB channel access is not available, pipeline loops operation should be hold until the bus is ready to access. Fig. 64 shows three cases necessary to stall pipeline. When one of AHB read channels is not available, the AHB read bus asserts this using the AHB ready signal. In Fig. 64(a), the AHB ready signal is not enabled, so the pipeline for the loops operation should be stalled. Once the ready is enabled again, the pipeline stall is released. The pipeline stall for AHB write ready also occurs in the same way as shown in Fig. 64(b).

### 4. DSP Accelerator Operations

#### a. 16-/32-Bit MACs

Fig. 65 illustrates a 16-bit multiplier and accumulator implementation using four 8-bit multipliers. The DSP accelerator contains eight 8-bit multipliers so four 16-bit multiplications can be conducted in a SIMD style. The signed or unsigned multiplication is configured by SIGNED_A (B) field in each command. In the figure SIGNED_A="0010", SIGNED_B="0010", so A[15:8] (a0), B[15:8] (b0) are signed bytes, but A[7:0], B[7:0] are unsigned bytes. Four 8-bit multipliers are used to compute four signed 8-bit multiplications, a1xb1, a0xb1, a1xb0, a0xb0 and sum up all 8-bit multiplication results in a 40-bit result including 8-bit guard bits. If MAC_EN=1 (i.e. MAC is enabled), the 40-bit multiplication result should be accumulated to the 40-bit accumulator register (acc0). When MAC_EN=0, the 40-bit acc0 stores the 40-bit multiplication result without accumulation.

A 32-bit multiplier can be implemented in the same way as shown in Fig. 66. Four 16-bit multipliers can construct a 32-bit multiplier, and the signed or unsigned operation is also controlled by the SIGNED_A (B) field. The result of 80-bit multiplication are accumulated into 80-bit accumulator (acc) with 16-bit guard bits. The accumulator result can be saturated, rounded, or shifted according to the configurations of command and control register fields such as SHIFT_OP, SHIFT_SIZE, ROUND_EN, ROUND_SET, and SAT_MOD.

#### b. 8-/16-/32-Bit Cross Product

The DSP accelerator provides 8-/16-/32-bit 3 dimensional CP operations. These operations basically utilize 8-/16-/32-bit multipliers to compute equation (4.2). In Fig. 67(a) showing 8-bit 3-D CP, 32-bit operand A and B contain 8-bit 3-D vector input A$_n$[31:8]=($a_{3n-2}$, $a_{3n-1}$, $a_{3n}$) and B$_n$[31:8]=($b_{3n-2}$, $b_{3n-1}$, $b_{3n}$), respectively, where n denotes the index of a sample vector. LSB bytes in A and B (i.e. A$_n$[7:0], B$_n$[7:0]) are filled with 0. 8-bit vector

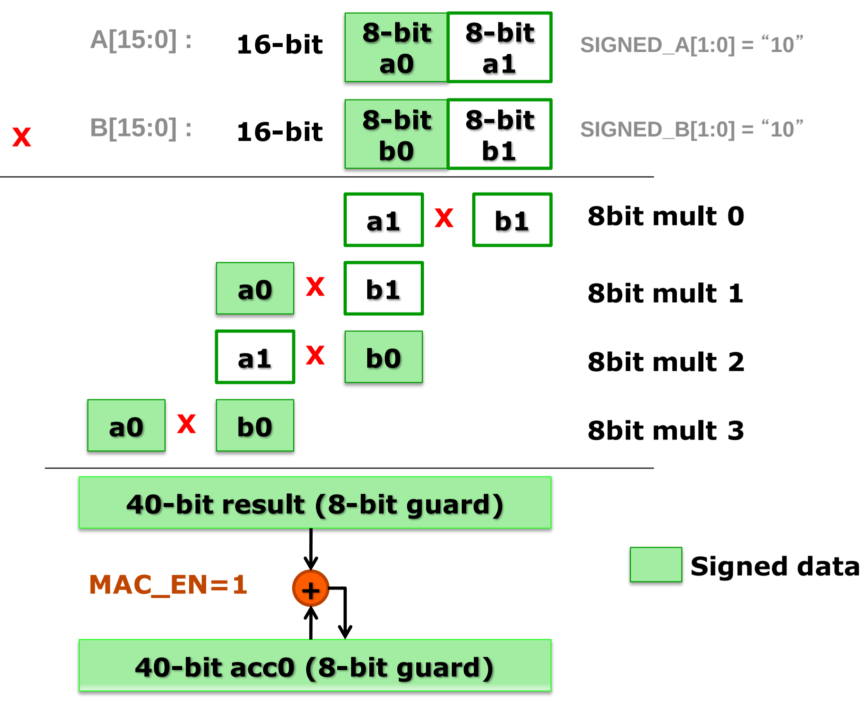


Fig. 65.: 16-bit multiplication and 40-bit accumulator register for 16-bit MAC

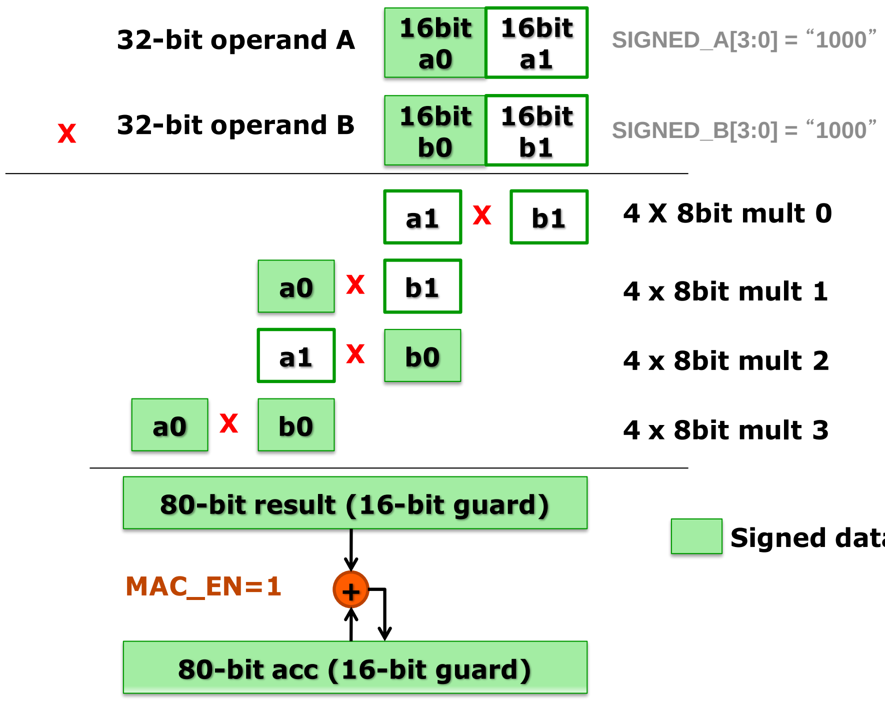
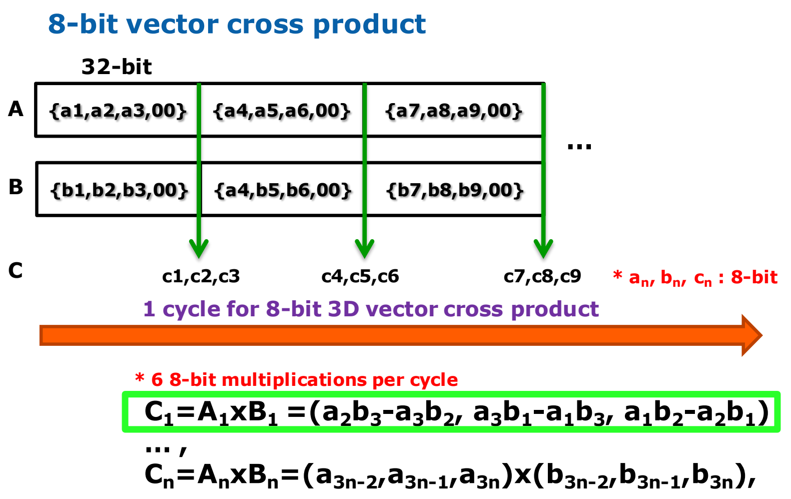
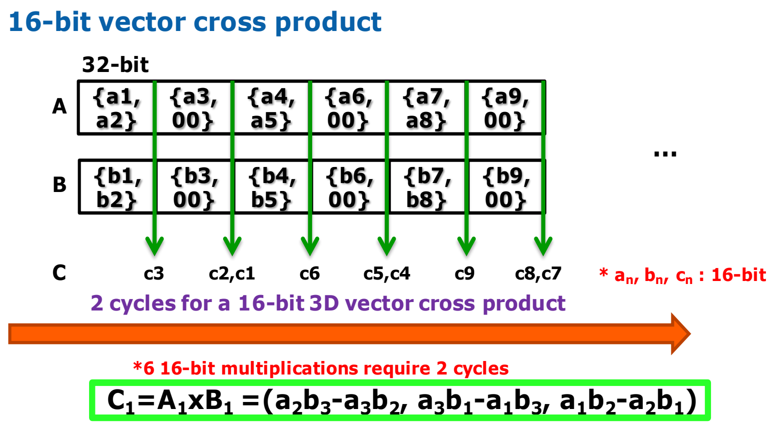


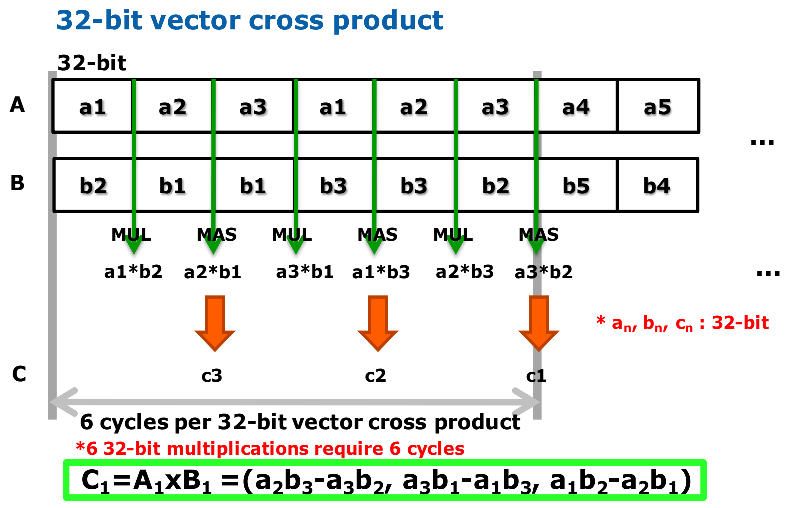
Fig. 66.: 32-bit multiplication and 80-bit accumulator register for 32-bit MAC


8-bit vector cross product
32-bit
A
{a1,a2,a3,00}
{a4,a5,a6,00}
{a7,a8,a9,00}
...
B
{b1,b2,b3,00}
{a4,b5,b6,00}
{b7,b8,b9,00}
C
c1,c2,c3
c4,c5,c6
c7,c8,c9
* $a_n$, $b_n$, $c_n$ : 8-bit
1 cycle for 8-bit 3D vector cross product
* 6 8-bit multiplications per cycle
$C_1=A_1xB_1 =(a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1)$
... ,
$C_n=A_nxB_n=(a_{3n-2},a_{3n-1},a_{3n})x(b_{3n-2},b_{3n-1},b_{3n})$,

(a) 8-bit 3-D cross product


16-bit vector cross product
32-bit
A
{a1, a2}
{a3, 00}
{a4, a5}
{a6, 00}
{a7, a8}
{a9, 00}
...
B
{b1, b2}
{b3, 00}
{b4, b5}
{b6, 00}
{b7, b8}
{b9, 00}
C
c3
c2,c1
c6
c5,c4
c9
c8,c7
* $a_n$, $b_n$, $c_n$ : 16-bit
2 cycles for a 16-bit 3D vector cross product
*6 16-bit multiplications require 2 cycles
$C_1=A_1xB_1 =(a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1)$

(b) 16-bit 3-D cross product


32-bit vector cross product
32-bit
A
a1
a2
a3
a1
a2
a3
a4
a5
...
B
b2
b1
b1
b3
b3
b2
b5
b4
MUL
MAS
MUL
MAS
MUL
MAS
a1*b2
a2*b1
a3*b1
a1*b3
a2*b3
a3*b2
...
* $a_n$, $b_n$, $c_n$ : 32-bit
C
c3
c2
c1
6 cycles per 32-bit vector cross product
*6 32-bit multiplications require 6 cycles
$C_1=A_1xB_1 =(a_2b_3-a_3b_2, a_3b_1-a_1b_3, a_1b_2-a_2b_1)$

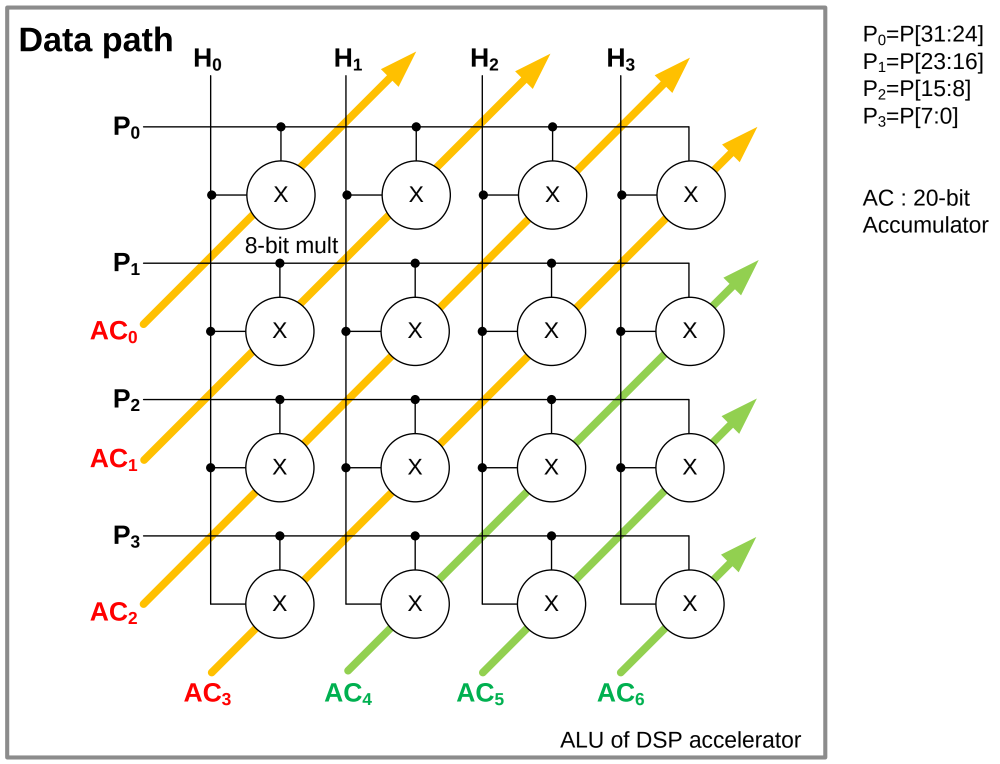
(c) 32-bit 3-D cross product

Fig. 67.: 8-/16-/32-bit cross product operations

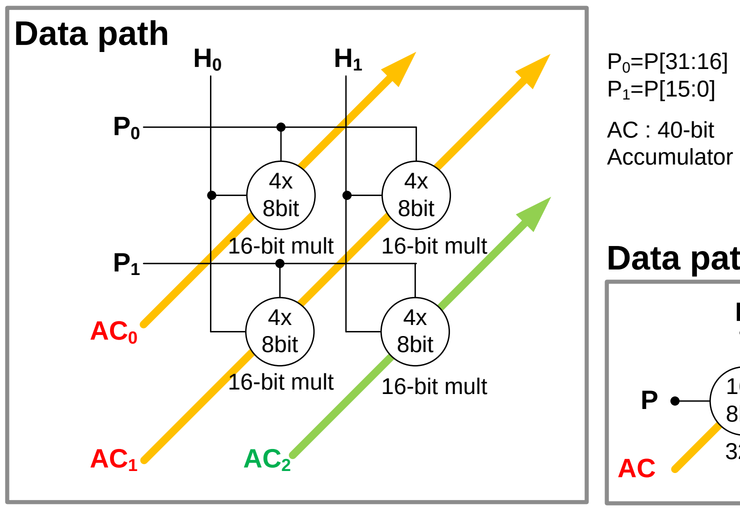
output, $C_n = (c_1, c_2, c_3) = (a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)$ can be computed using 6 8-bit multiplications. The DSP supports max 8 8-bit multiplications per cycle, therefore the execution of 8-bit 3-D CP takes only 1 cycle. The 16-bit 3-D CPs are illustrated in Fig. 67(b). It requires two cycles to calculate 6 16-bit multiplications. In the first cycle, two 16-bit elements in a vector, $a_1$, $a_2$ ($b_1$, $b_2$) are loaded into a 32-bit operand A (B). The loaded elements ($a_1$, $a_2$, $b_1$, $b_2$) are used to compute $c_3 = a_1b_2 - a_2b_1$ and temporarily stored in an internal register to reuse them in the next cycle. The rest elements $a_3$, $b_3$ are loaded in the next cycle and used along with $a_1$, $a_2$, $b_1$, $b_2$ stored in the temporary register to compute $c_1 = a_2b_3 - a_3b_2$, $c_2 = a_3b_1 - a_1b_3$. Fig. 67(c) shows the 32-bit 3-D CP operation. It demands 6 32-bit multiplications, taking 6 cycles since the DSP accelerator can provide a 32-bit multiplication per cycle. In the first cycle, 32-bit $a_1$, $b_2$ are loaded and multiplied. The result of the 32-bit multiplication is stored in the 80-bit accumulator register to reuse it for computing $c_3$. In the next cycle, 32-bit $a_2$, $b_1$ are loaded, multiplied, and accumulated into the accumulator to calculate $c_3 = a_1b_2 - a_2b_1$. In the same way, $c_2$, $c_1$ can be attained.

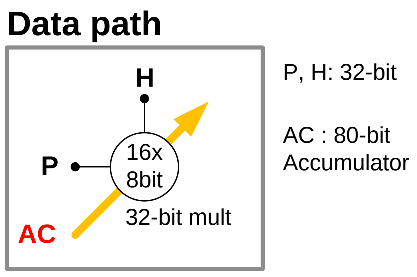
c. FIR Operation

The DSP accelerator supports 8-/16-/32 bit FIR operations. We propose a novel fast FIR scheme using *diagonal accumulation*. Fig. 68(a) shows 8-bit FIR filter operation using 16 8-bit multipliers and 7 20-bit diagonal accumulators ($AC_0$-$AC_7$) built in the data path. The data path consists of 16 8-bit multipliers and 80-bit main and 60-bit auxiliary accumulators. The 16 8-bit multipliers can be reformed as 4 16-bit multipliers or a 32-bit multiplier (see the data path part and 16-/32-bit MACs). The results of 8-bit multiplications ($P_i$ x $H_i$, i=0, .., 3) are accumulated in the diagonal direction in the 8-bit FIR filter operation. $AC_0$-$AC_3$ are implemented by the 80-bit main accumulator. $AC_4$, $AC_5$, $AC_6$ are built on the 60-bit auxiliary accumulator for the 8-bit FIR filter operation. The aim of the diagonal accumulators is to maximize the utilization of multipliers in the data path. When we exploit the



**(a) 8-bit FIR filter using the data path**





**(b) 16-bit FIR filter using the data path**

**(c) 32-bit FIR filter using the data path**

Fig. 68.: 8-/16-/32-bit FIR filter operations implemented by diagonal accumulators and shared multipliers

diagonal accumulators, we can keep using all 16 8-bit multipliers at each cycle. This leads to the maximum utilization of resources in data path, providing significant improvement in throughput performance.

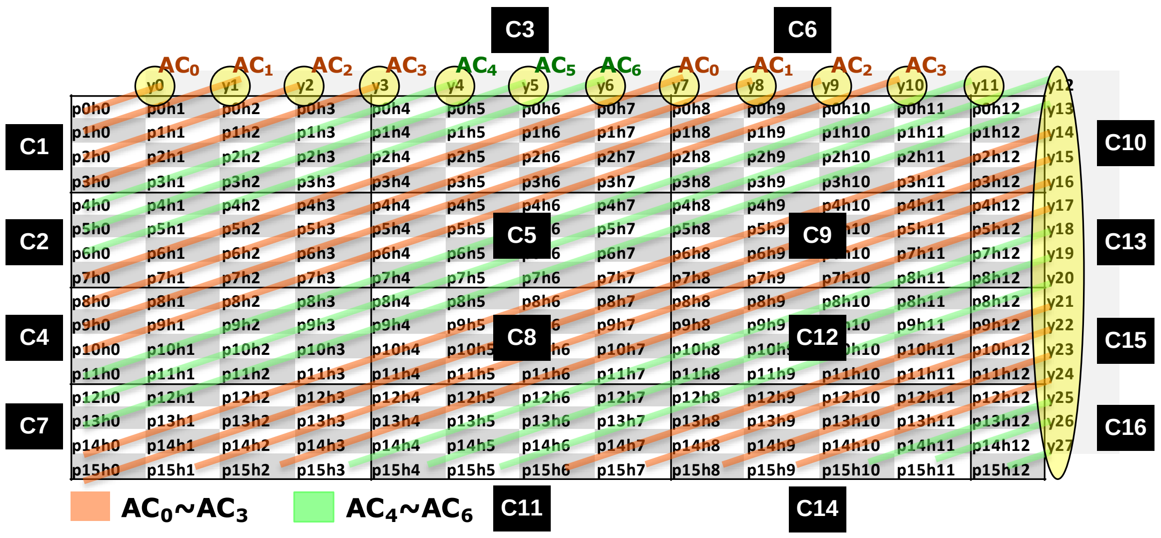
In Fig. 68(b), 4 16-bit multiplications and 3 40-bit diagonal accumulators $AC_0$, $AC_1$, $AC_2$ are used to implement the 16-bit FIR operation. $AC_0$, $AC_1$ are built on the 80-bit main accumulator, and $AC_2$ is stored in the 60-bit auxiliary accumulator. In the same way, 32-bit FIR filter can be implemented using a 32-bit multiplication and a 80-bit diagonal accumulator (AC) built on the 80-bit main accumulator.

Here, we present how the diagonal accumulators accelerate the FIR filter operations. We explain this scheme using a $12th$ order 8-bit FIR filter example as shown in Fig. 69. The filter consists of 13 8-bit coefficients (h0-h12), 16 8-bit inputs (p0-p15), and 16+12 8-bit outputs (y0-y27). Fig. 69(a) shows coefficients, inputs and outputs for a 8-bit FIR filter operation at cycle1-16 for computing 28 8-bit outputs, y0-y27. For instance, when the DSP accelerator loads 4 8-bit coefficients (h0-h3), 4 8-bit inputs (p0-p3) through 32-bit operand channels, the coefficients and inputs are multiplied and accumulated in the diagonal direction, yielding 4 8-bit outputs (y0-y3 in the black box) at cycle1. The next 32-bit (i.e. 4 8-bit) outputs, y4-y7, are obtained at cycle3 from the diagonal accumulators. In this way, the last 32-bit outputs, y24-y27 can be computed at cycle16. In this operation, all 16 8-bit multipliers are used to compute 16 8-bit multiplications at each cycle. This increases performance in throughput. Total execution cycles in the 8-bit FIR operation using the diagonal accumulators can be calculated by the number of sample words (32-bit) x the number of coefficient words (32-bit) = 4x4 = 16 cycles. Throughput is 28 bytes / 16 cycles = 1.75 bytes per cycle. We note that samples and coefficients must be 32-bit aligned data in this design, so if not aligned in a 32-bit word, they must be padded by zeros.

The diagonal accumulation is illustrated in Fig. 69(b). In this figure, a small rectangular box denotes a 8-bit multiplication of input and coefficient (i.e. p*h) and a large box

| | | | |
|---|---|---|---|
| cycle 1<br>h0 h1 h2 h3<br>p0 p1 p2 p3<br>y0 y1 y2 y3 | cycle 3<br>h4 h5 h6 h7<br>p0 p1 p2 p3<br>y4 y5 y6 y7 | cycle 6<br>h8 h9 h10 h11<br>p0 p1 p2 p3<br>y8 y9 y10 y11 | cycle 10<br>h12 0 0 0<br>p0 p1 p2 p3<br>y12 y13 y14 y15 |
| cycle 2<br>h0 h1 h2 h3<br>p4 p5 p6 p7 | cycle 5<br>h4 h5 h6 h7<br>p4 p5 p6 p7 | cycle 9<br>h8 h9 h10 h11<br>p4 p5 p6 p7 | cycle 13<br>h12 0 0 0<br>p4 p5 p6 p7<br>y16~19 |
| cycle 4<br>h0 h1 h2 h3<br>p8 p9 p10 p11 | cycle 8<br>h4 h5 h6 h7<br>p8 p9 p10 p11 | cycle 12<br>h8 h9 h10 h11<br>p8 p9 p10 p11 | cycle 15<br>h12 0 0 0<br>p8 p9 p10 p11<br>y20~23 |
| cycle 7<br>h0 h1 h2 h3<br>p12 p13 p14 p15 | cycle 11<br>h4 h5 h6 h7<br>p12 p13 p14 p15 | cycle 14<br>h8 h9 h10 h11<br>p12 p13 p14 p15 | cycle 16<br>h12 0 0 0<br>p12 p13 p14 p15<br>y24~27 |

(a) Coefficients, inputs and outputs for a 8-bit FIR filter operation at each cycle


C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
C16
$AC_0$ $AC_1$ $AC_2$ $AC_3$ $AC_4$ $AC_5$ $AC_6$ $AC_0$ $AC_1$ $AC_2$ $AC_3$
y0 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11
y12 y13 y14 y15 y16 y17 y18 y19 y20 y21 y22 y23 y24 y25 y26 y27
p0h0 p0h1 p0h2 … p0h12
p15h0 p15h1 p15h2 … p15h12
🟧 $AC_0$~$AC_3$
🟩 $AC_4$~$AC_6$

(b) Diagonal accumulation in the 8-bit FIR filter operation

Fig. 69.: 12th order 8-bit FIR (13 coefficients) with 16 pixels (8-bit data), (b) illustrates the proposed diagonal accumulation scheme accelerating the 8-bit FIR operation.

labeled by C1-C16 (i.e. cycle1-cycle16) including the 16 small boxes presents the execution of 16 8-bit multiplications in the data path at each cycle. $AC_0$-$AC_6$ denotes the 7 20-bit diagonal accumulators. In the first cycle (C1), the outputs can by calculated by

$$y0 = AC_0 = p0 * h0$$

$$y1 = AC_1 = p1 * h0 + p0 * h1$$

$$y2 = AC_2 = p2 * h0 + p1 * h1 + p0 * h2$$

$$y3 = AC_3 = p3 * h0 + p2 * h1 + p1 * h2 + p0 * h3$$

The next outputs, y4-y7, can be obtained by the diagonal accumulators, $AC_4$, $AC_5$, $AC_6$, $AC_0$ at C3 (cycle3), and the accumulations are shown as the diagonal lines in the figure as follows:

$$y4 = AC_4 = p4 * h0 + p3 * h1 + p2 * h2 + p1 * h3 + p0 * h4$$

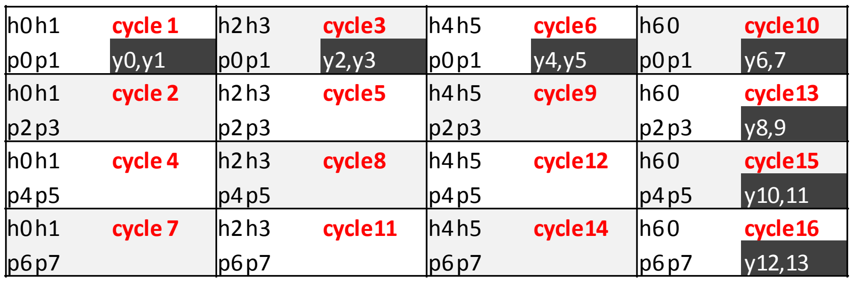
$$y5 = AC_5 = p5 * h0 + p4 * h1 + p3 * h2 + p2 * h3 + p1 * h4 + p0 * h5$$

$$y6 = AC_6 = p6 * h0 + p5 * h1 + p4 * h2 + p3 * h3 + p2 * h4 + p1 * h5 + p0 * h6$$

$$y7 = AC_0 = p7 * h0 + p6 * h1 + p5 * h2 + p4 * h3 + p3 * h4 + p2 * h5 + p1 * h6 + p0 * h7$$

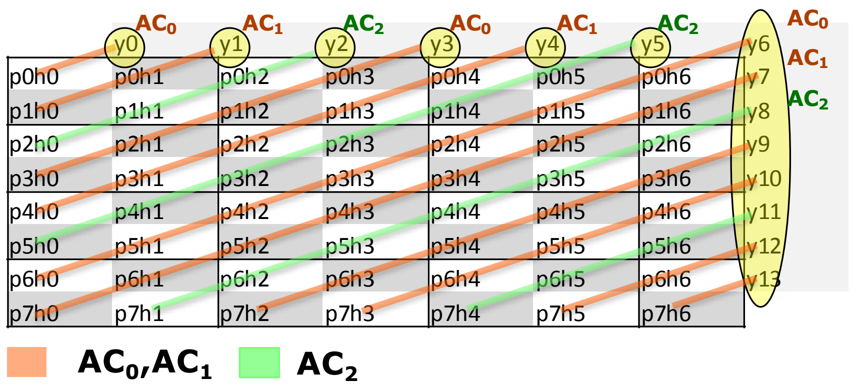
In this scheme, 7 20-bit accumulators ($AC_0$-$AC_6$) are used to prevent overwriting FIR results into the non-available accumulators.

Fig. 70 shows the 16-bit FIR filter operation using 3 40-bit diagonal accumulators. This example shows the 6th order 16-bit FIR filter operation that has 7 16-bit coefficients and 8 half-word inputs (16-bit data), yielding 14 16-bit outputs, y0-y13. The mechanism of diagonal accumulation is the same as that of the 8-bit FIR filtering process. In Fig. 70(b), 40-bit accumulators, $AC_0$, $AC_1$, $AC_2$, accumulate the results of 16-bit multiplication (a 16-bit coefficient x a 16-bit input) in the diagonal directions. Like the 8-bit FIR, all 4 16-bit multipliers are utilized to calculate 4 16-bit multiplications by means of the diagonal accumulators at each cycle. This also maximizes the utilization of multipliers. In this



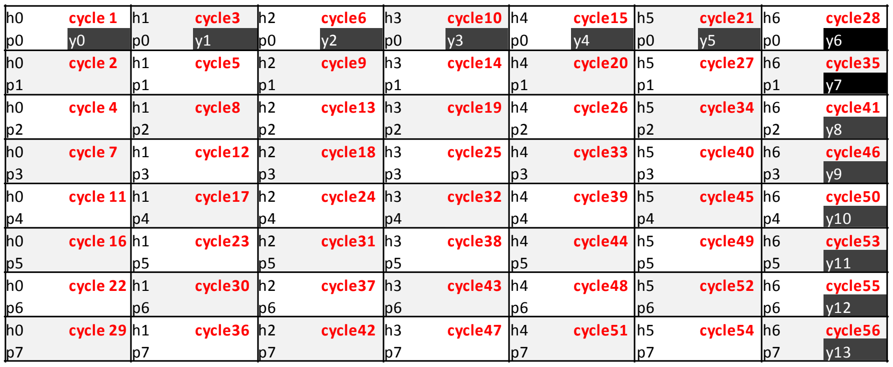
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| h0 h1 | cycle 1 | h2 h3 | cycle3 | h4 h5 | cycle6 | h6 0 | cycle10 |
| p0 p1 | y0,y1 | p0 p1 | y2,y3 | p0 p1 | y4,y5 | p0 p1 | y6,7 |
| h0 h1 | cycle 2 | h2 h3 | cycle5 | h4 h5 | cycle9 | h6 0 | cycle13 |
| p2 p3 | | p2 p3 | | p2 p3 | | p2 p3 | y8,9 |
| h0 h1 | cycle 4 | h2 h3 | cycle8 | h4 h5 | cycle12 | h6 0 | cycle15 |
| p4 p5 | | p4 p5 | | p4 p5 | | p4 p5 | y10,11 |
| h0 h1 | cycle 7 | h2 h3 | cycle11 | h4 h5 | cycle14 | h6 0 | cycle16 |
| p6 p7 | | p6 p7 | | p6 p7 | | p6 p7 | y12,13 |

(a) Coefficients, inputs and outputs for a 16-bit FIR filter operation at each cycle


AC0 AC1 AC2 AC0 AC1 AC2
y0 y1 y2 y3 y4 y5
p0h0 p0h1 p0h2 p0h3 p0h4 p0h5 p0h6
p1h0 p1h1 p1h2 p1h3 p1h4 p1h5 p1h6
p2h0 p2h1 p2h2 p2h3 p2h4 p2h5 p2h6
p3h0 p3h1 p3h2 p3h3 p3h4 p3h5 p3h6
p4h0 p4h1 p4h2 p4h3 p4h4 p4h5 p4h6
p5h0 p5h1 p5h2 p5h3 p5h4 p5h5 p5h6
p6h0 p6h1 p6h2 p6h3 p6h4 p6h5 p6h6
p7h0 p7h1 p7h2 p7h3 p7h4 p7h5 p7h6
y6 y7 y8 y9 y10 y11 y12 y13
AC0 AC1 AC2
AC0,AC1
AC2

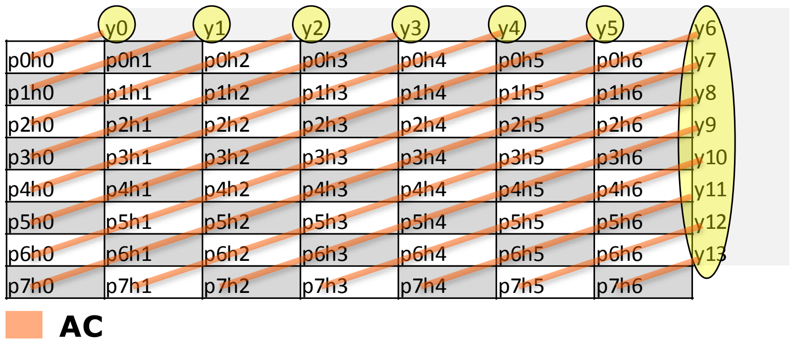
(b) Diagonal accumulation in the 16-bit FIR filter operation

Fig. 70.: 6th order 16-bit FIR (7 coefficients) with 8 half-words (16-bit data), (b) illustrates the proposed diagonal accumulation scheme accelerating the 16-bit FIR operation.



| | | | | | | |
|---|---|---|---|---|---|---|
| h0 p0 cycle 1 y0 | h1 p0 cycle3 y1 | h2 p0 cycle6 y2 | h3 p0 cycle10 y3 | h4 p0 cycle15 y4 | h5 p0 cycle21 y5 | h6 p0 cycle28 y6 |
| h0 p1 cycle 2 | h1 p1 cycle5 | h2 p1 cycle9 | h3 p1 cycle14 | h4 p1 cycle20 | h5 p1 cycle27 | h6 p1 cycle35 y7 |
| h0 p2 cycle 4 | h1 p2 cycle8 | h2 p2 cycle13 | h3 p2 cycle19 | h4 p2 cycle26 | h5 p2 cycle34 | h6 p2 cycle41 y8 |
| h0 p3 cycle 7 | h1 p3 cycle12 | h2 p3 cycle18 | h3 p3 cycle25 | h4 p3 cycle33 | h5 p3 cycle40 | h6 p3 cycle46 y9 |
| h0 p4 cycle 11 | h1 p4 cycle17 | h2 p4 cycle24 | h3 p4 cycle32 | h4 p4 cycle39 | h5 p4 cycle45 | h6 p4 cycle50 y10 |
| h0 p5 cycle 16 | h1 p5 cycle23 | h2 p5 cycle31 | h3 p5 cycle38 | h4 p5 cycle44 | h5 p5 cycle49 | h6 p5 cycle53 y11 |
| h0 p6 cycle 22 | h1 p6 cycle30 | h2 p6 cycle37 | h3 p6 cycle43 | h4 p6 cycle48 | h5 p6 cycle52 | h6 p6 cycle55 y12 |
| h0 p7 cycle 29 | h1 p7 cycle36 | h2 p7 cycle42 | h3 p7 cycle47 | h4 p7 cycle51 | h5 p7 cycle54 | h6 p7 cycle56 y13 |

(a) Coefficients, inputs and outputs for a 16-bit FIR filter operation at each cycle


y0
y1
y2
y3
y4
y5
y6
y7
y8
y9
y10
y11
y12
y13
p0h0 p0h1 p0h2 p0h3 p0h4 p0h5 p0h6
p1h0 p1h1 p1h2 p1h3 p1h4 p1h5 p1h6
p2h0 p2h1 p2h2 p2h3 p2h4 p2h5 p2h6
p3h0 p3h1 p3h2 p3h3 p3h4 p3h5 p3h6
p4h0 p4h1 p4h2 p4h3 p4h4 p4h5 p4h6
p5h0 p5h1 p5h2 p5h3 p5h4 p5h5 p5h6
p6h0 p6h1 p6h2 p6h3 p6h4 p6h5 p6h6
p7h0 p7h1 p7h2 p7h3 p7h4 p7h5 p7h6
AC

(b) Diagonal accumulation in the 16-bit FIR filter operation

Fig. 71.: 6th order 32-bit FIR (13 coefficients) with 8 words (32-bit data), (b) illustrates the proposed diagonal accumulation scheme accelerating the 32-bit FIR operation.

example, total execution cycles can be computed by total cycles = the number of sample words x the number of coefficient words = 4x4 = 16 cycles. Throughput is 28 bytes / 16 cycles = 1.75 bytes per cycle.

Fig. 71 shows the 32-bit FIR filter operation using a 80-bit diagonal accumulator. The 6th order 32-bit FIR filter operation that has 7 32-bit coefficients and 8 word inputs (32-bit data) is illustrated in the figure. In this example, the operation yields 14 word outputs, y0-y13. This operation does not utilize the 60-bit supplement accumulator, accumulating the results of multiplication of two 32-bit operands in the 80-bit main accumulator (AC). The efficiency of the 32-bit FIR filter operation is inferior to that of the 8-/16-bit FIR filters due to less parallel accumulation performed by the auxiliary accumulators in the 8-/16-bit FIR mode. Total execution cycles is calculated in the same way: total cycles = the number of sample words x the number of coefficient words = 8x7 = 56 cycles. Throughput is 56 bytes / 56 cycles = 1 bytes per cycle.

d. FFT Operation

The interconnected butterflies of an 8-point radix-2 DIT FFT is illustrated in Fig. 72. The inputs to the FFT are indexed in bit-reversed order (0, 4, 2, 6, 1, 5, 3, 7) and the outputs are indexed in sequential order (0, 1, 2, 3, 4, 5, 6, 7). Computation of a radix-2 DIT FFT requires the input vector to be in bit-reversed order, and generates an output vector in sequential order. The DSP accelerator supports this bit-reverse addressing mode in the command register (ADDR_MODE=5).

Pipelined execution considerably improves throughput when loop size is large. Fig. 73 shows an example of pipeline loops for 8-point radix-2 DIT FFT. When the number of points is $n = 2^m$ in FFT, the number of stages is $log_2(n) = m$, and $2^m/2$ butterfly operations are required. Since a butterfly operation performs a complex multiplication and two complex addition, the complexity of radix-2 DIT FFT involves $n/2 \times log_2(n) = n/2 \times m$

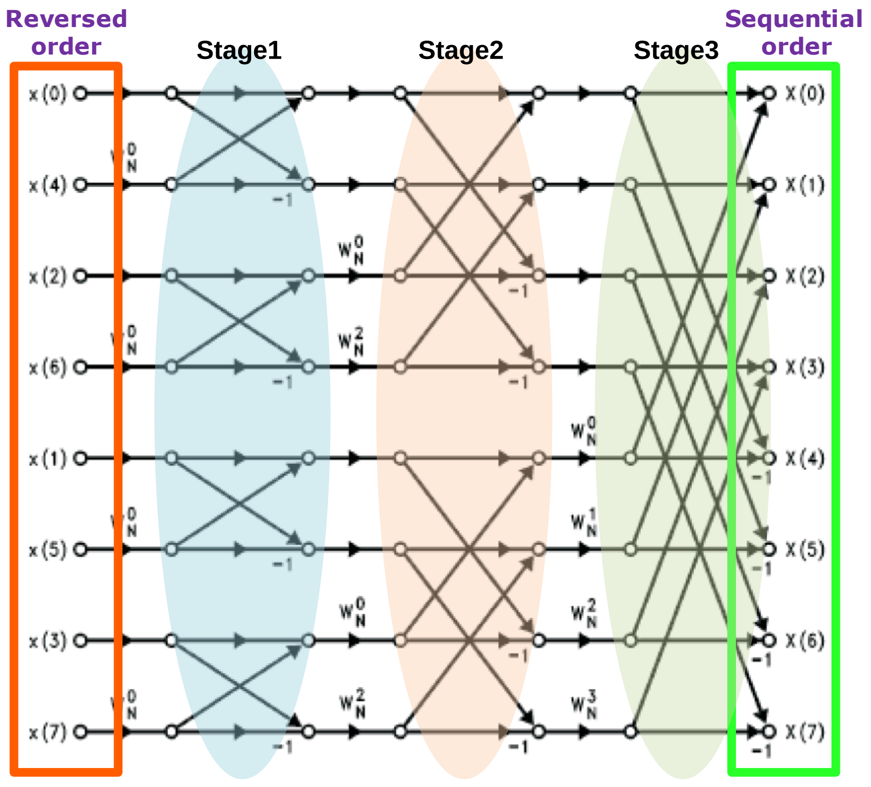


Fig. 72.: The interconnected butterflies of an 8-point radix-2 DIT FFT

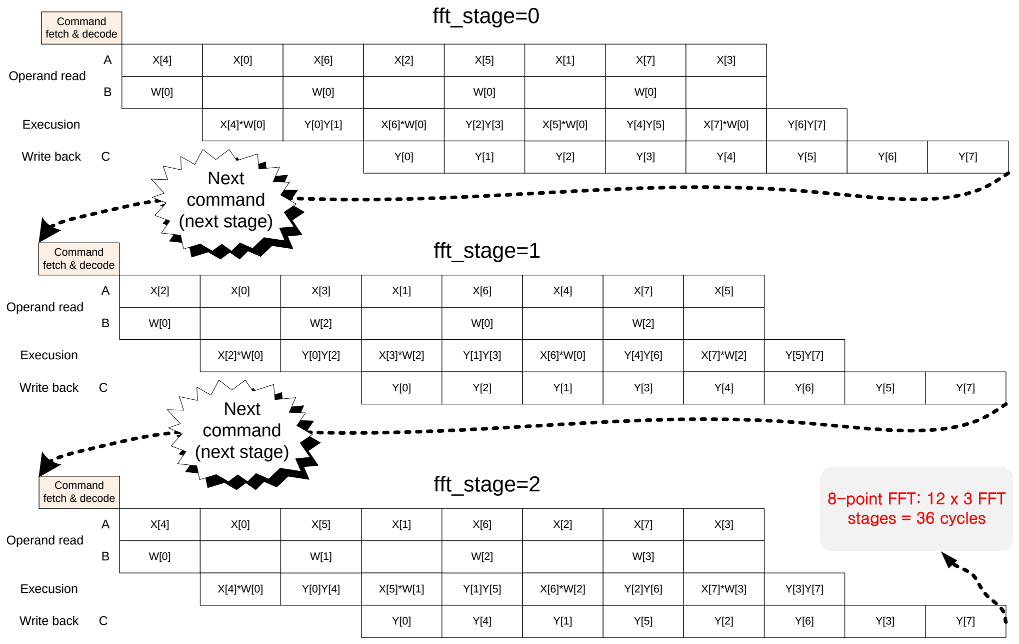

fft_stage=0
Command fetch & decode
Operand read
A
B
Execusion
Write back
C
X[4] X[0] X[6] X[2] X[5] X[1] X[7] X[3]
W[0] W[0] W[0] W[0]
X[4]*W[0] Y[0]Y[1] X[6]*W[0] Y[2]Y[3] X[5]*W[0] Y[4]Y[5] X[7]*W[0] Y[6]Y[7]
Y[0] Y[1] Y[2] Y[3] Y[4] Y[5] Y[6] Y[7]
Next command (next stage)
fft_stage=1
X[2] X[0] X[3] X[1] X[6] X[4] X[7] X[5]
W[0] W[2] W[0] W[2]
X[2]*W[0] Y[0]Y[2] X[3]*W[2] Y[1]Y[3] X[6]*W[0] Y[4]Y[6] X[7]*W[2] Y[5]Y[7]
Y[0] Y[2] Y[1] Y[3] Y[4] Y[6] Y[5] Y[7]
Next command (next stage)
fft_stage=2
X[4] X[0] X[5] X[1] X[6] X[2] X[7] X[3]
W[0] W[1] W[2] W[3]
X[4]*W[0] Y[0]Y[4] X[5]*W[1] Y[1]Y[5] X[6]*W[2] Y[2]Y[6] X[7]*W[3] Y[3]Y[7]
Y[0] Y[4] Y[1] Y[5] Y[2] Y[6] Y[3] Y[7]
8−point FFT: 12 x 3 FFT stages = 36 cycles

Fig. 73.: An example of pipelined execution for 8-point FFT operation. A FFT stage operation can be performed by a command. The 8-point FFT has 3 FFT stages ($2^{stage}$=n points)

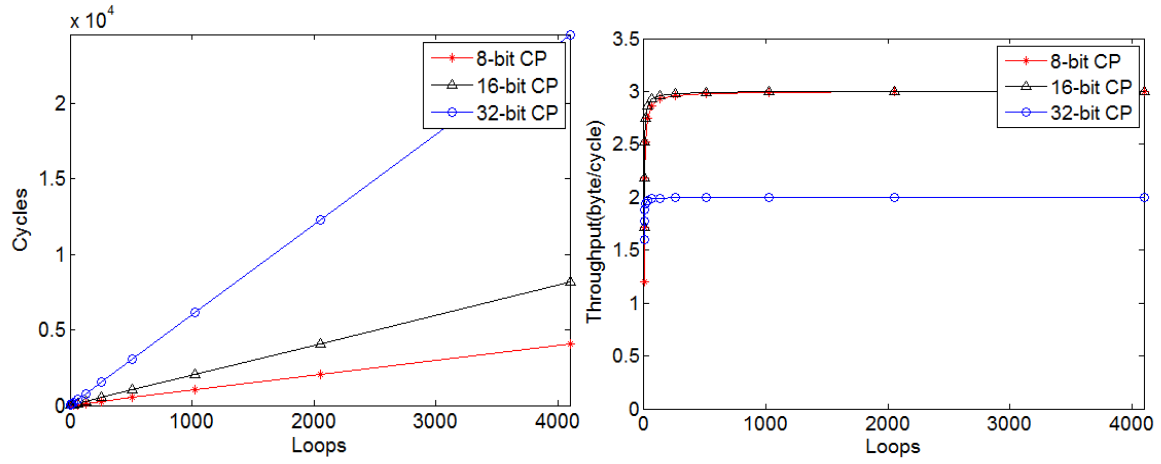
Table XXV.: Execution cycles of the DSP accelerator operations, $C_{FD}$ is command fetch and decode cycles, $P_{normal}$ is pipeline delay cycles in normal mode not including FFT, $P_{FFT}$ is pipeline delay cycles in FFT mode. $C_{FD}$=1 cycle, $P_{normal}$=2 cycles, $P_{FFT}$=3 cycles in the DSP accelerator.

| Operation | Cycles |
|---|---|
| 8-/16-/32-bit MAC | $C_{FD}$ + $P_{normal}$ + #of loops |
| 8-bit CP | $C_{FD}$ + $P_{normal}$ + 1 (cycles per a 8-bit CP) x #of loops |
| 16-bit CP | $C_{FD}$ + $P_{normal}$ + 2 (cycles per a 16-bit CP) x #of loops |
| 32-bit CP | $C_{FD}$ + $P_{normal}$ + 6 (cycles per a 32-bit CP) x #of loops |
| 8-/16-/32-bit FIR | $C_{FD}$ + $P_{normal}$ + #of sample words x # of coefficient words |
| 32-bit complex FFT | ($C_{FD}$ + $P_{FFT}$ + #of points) x #of FFT stages |

complex multiplications, and $n \times log_2(n) = n \times m$ complex additions. In the FFT function, each stage can be conducted by a command. Therefore, $2^m$-point radix-2 DIT FFT requires $m$ commands. The pipeline execution occurs in a stage for running $2^m/2$ butterfly operation loops. In Fig. 73 the 8-point FFT consists of 3 FFT stages where each stage conducts 4 butterfly operations. The execution cycles in each stage is 1 for command fetch and decode + 3 for pipeline latency + 8 for 4 butterfly executions = 12 cycles. Hence, the total execution cycles taken in the 8-point FFT is 12 for cycles in a stage x 3 for the number of FFT stages = 36 cycles. We note that the commands are individually performed, not pipelined, so when a command is completed, the next command can be fetched and executed. In the command structure, the number of FFT stages is configured by the LOOP_SIZE field, and a corresponding stage number in a command is set by the FFT_STAGE field.

## 5. Experiment Results

Table XXV presents the execution time of the DSP accelerator operations. Delay cycles in pipeline took normally 2 cycles except FFT since the FFT operation required one more delay cycle to execute addition/subtraction from a complex input to the complex multipli-



(a) Execution cycles in cross product (CP) (b) Throughput in cross product (CP)

Fig. 74.: Results of execution cycles and throughput in cross product

cation (the other complex input * a twiddle factor).

The execution cycles and throughput performance of 8-/16-/32-bit CPs are shown in Fig. 74. For instance, 4095 loops 3D CP took 4098, 8193, 24573 cycles at 8-/16-/32-bit CP operations respectively. The 4095 is the maximum number of loop size in the proposed design but can be expandable with the negligible overhead of field bit extension (e.g. extend 12-bit LOOP_SIZE to 16-bit LOOP_SIZE). Throughput of 8-/16-bit CPs were 1.5 times greater than that of 32-bit CP in performance. This was because 8-/16-bit CPs read multiple operands (3 operands in 8-bit, 2 operands in 16-bit mode) at once per each operand channel, executing multiple 8-/16-bit multiplications in parallel. This technique was similarly exploited to accelerate the 8-/16-bit FIR operations.

Fig. 75 shows the results of 8-/16-/32-bit FIR filter operations. This experiment was performed on the $12th$ order FIR filter with varying the number of input samples and data width (8-/16-/32-bit) of the samples. In Fig. 75(a), 8-/16-/32-bit FIR operations with 128 samples took 131, 451, 1667 cycles respectively. The figure also shows the comparison of the FIR filter performance with $12th$ order 32-bit FIR filter (general purpose) on TI-67x

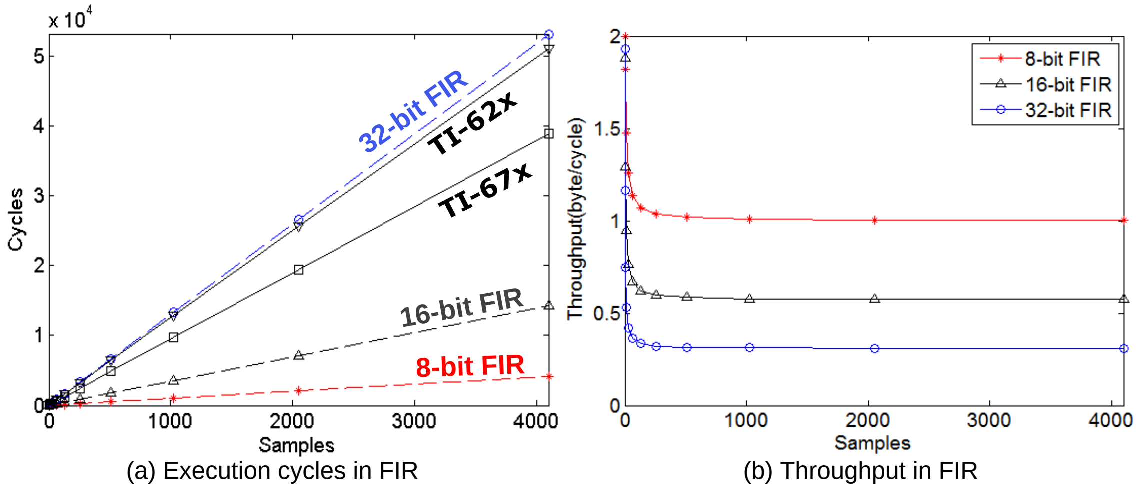


Fig. 75.: Results of execution cycles and throughput in FIR and comparison with 12*th* order 32-bit FIR filter on TI-67x and TI-62x DSPs

and TI-62x DSPs [126, 127]. The 32-bit FIR performance of the DSP accelerator was close to that of TI-62x. The 8-/16-bit FIRs took much less execution cycles than the 32-bit FIR filters (32-bit FIR, TI-62x, TI-67x). We will discuss this reason in the throughput result.

The throughput performance of the FIR operations is show in Fig. 75(b). As we discussed in the 8-/16-bit CPs, the 8-bit FIR operation yielded around three and two times more outputs per cycle than 16-bit and 32-bit FIRs respectively. The 16-bit FIR was also approximately twice better than the 32-bit FIR in throughput. Since multiple operands (4 operands in 8-bit, 2 operands in 16-bit) per each read channel were loaded, multiplied, and diagonally accumulated in parallel, the utilization of multipliers could be maximized as we discussed. Hence, the throughput could be considerably improved.

Fig. 76 shows the comparison of execution cycles between the proposed DSP accelerator and other processing components in radix-2 Complex DIT FFT. The DSP accelerator provided a superior performance in execution cycles than CPU [128] and DSPs (TI-67x and TI-62x) that support more programmable functions. However, it denoted a less per-

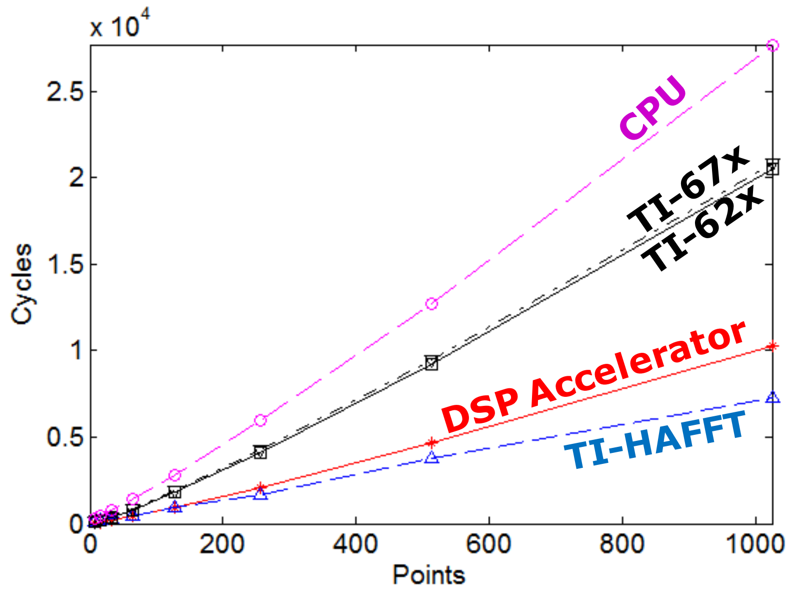


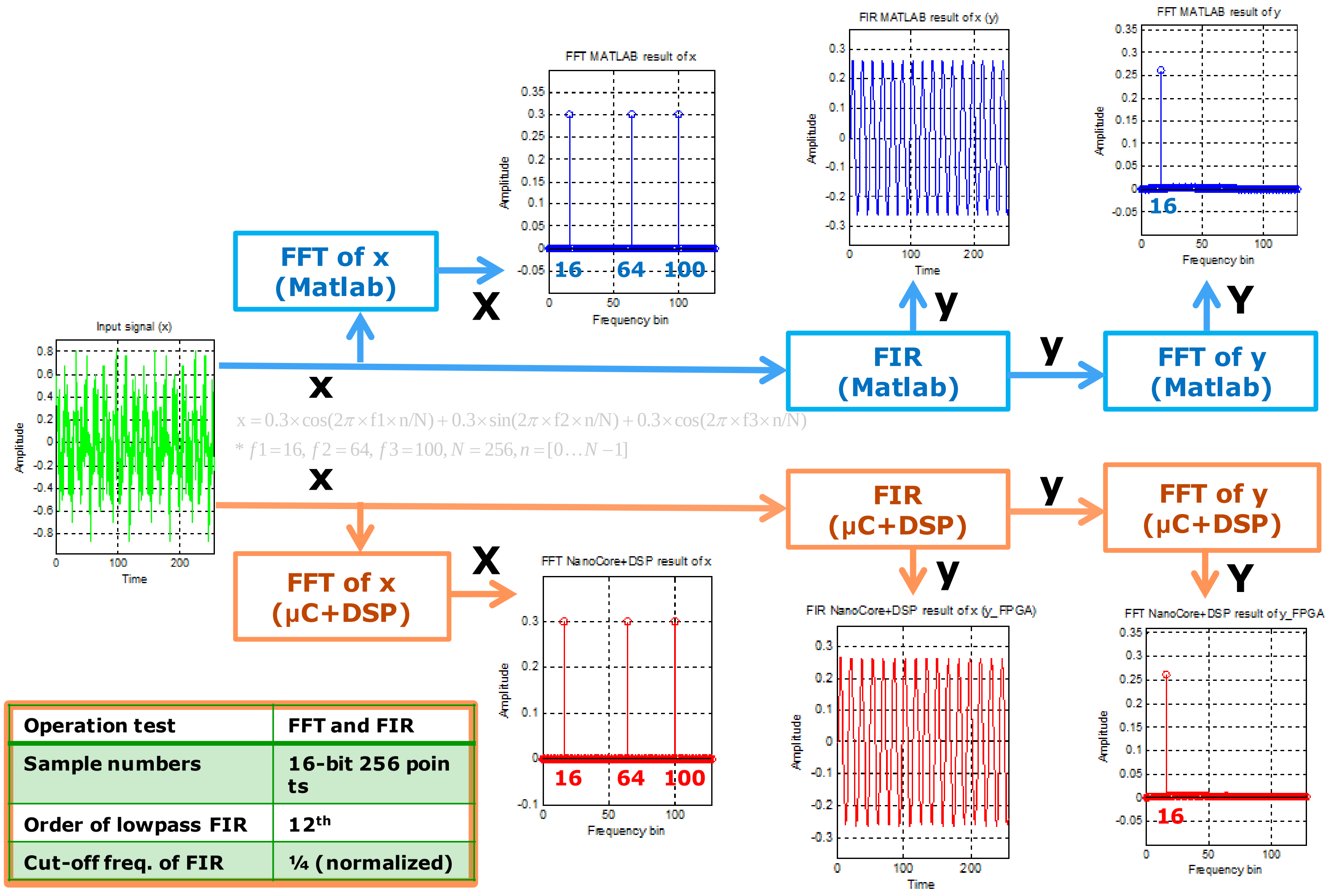
Fig. 76.: Comparison of execution cycles in FFT

Table XXVI.: Comparison of the 256-point complex FFT performance (radix-2) to other DSPs

| | DSP accelerator | ConnX D2 | TI-55x |
|---|---|---|---|
| Cycles | 2080 | 3740 | 4786 |
| Normalized cycles against DSP accelerator | 1 | 1.8 (+44%) | 2.3 (+57%) |

formance than TI FFT hardware accelerator (HWAFFT) [128] because HWAFFT is more optimized to compute FFT in hardware. Table. XXVI compares the 256-point radix-2 FFT performance of the DSP accelerator to ConnX D2 [129] and TI-55x [130]. The proposed accelerator could achieve 44% and 130% reduction in execution cycles, compared to ConnX D2 and TI-55x in the FFT operation respectively.

We also examined the area and power consumption of the accelerator. Table. XXVII compares the synthesis result of the DSP accelerator to the result of ConnX D2. We synthesized the DSP accelerator using Synopsys Design Compiler on TSMC 65nm-GP process


Input signal (x)
Amplitude
Time
x
FFT of x (Matlab)
FFT MATLAB result of x
Frequency bin
16
64
100
X
FIR (Matlab)
y
FIR MATLAB result of x (y)
FFT of y (Matlab)
Y
FFT MATLAB result of y
x = 0.3×cos(2π×f1×n/N) + 0.3×sin(2π×f2×n/N) + 0.3×cos(2π×f3×n/N)
* f1 = 16, f2 = 64, f3 = 100, N = 256, n = [0…N − 1]
FFT of x (µC+DSP)
FFT NanoCore+DSP result of x
FIR (µC+DSP)
FIR NanoCore+DSP result of x (y_FPGA)
FFT of y (µC+DSP)
FFT NanoCore+DSP result of y_FPGA

| Operation test | FFT and FIR |
|---|---|
| Sample numbers | 16-bit 256 poin ts |
| Order of lowpass FIR | 12th |
| Cut-off freq. of FIR | ¼ (normalized) |

Fig. 77.: Demonstration of the proposed DSP accelerator on ML605 FPGA board. The verification of the operated results on the board was performed by comparison with the Matlab results

Table XXVII.: Synthesis results using TSMC 65GS technology

| | DSP accelerator | ConnX D2 |
|---|---|---|
| Max. Frequency (MHz) | 333 | 605 |
| Area (mm$^2$) | 0.167 | 0.18 |
| Power consumption (mW/MHz) | 0.037 | 0.052 |

technology with 20% default switching activities. It consumed 12.35mW and 0.167mm$^2$ in power and area at 333MHz respectively. We also demonstrated it on ML605 FPGA board [131] at 25MHz with 3% LUT used.

Fig. 77 illustrates this demonstration on the board. For the verification of the DSP accelerator logic, we generated reference data using Matlab, compared them with the results obtained on the FPGA board. For instance, we generated input $x$=$0.3cos(2\pi f_1 n/N)+0.3sin(2\pi f_2 n/N)+0.3cos(2\pi f_3 n/N)$ with 16-bit 256 points samples where $f_1$=16, $f_2$=64, $f_3$=100, $N$=256, $n$=[0, ..., $N$-1] for the demonstration of FIR and FFT operations. The low-pass FIR filter was configured by $12th$ order and 1/4 normalized cut-off frequency. In the first step we programmed and configured commands and control register fields for computing FFT and FIR via micro-controller ($\mu$C) and AHB bus. In the second step the DSP accelerator loaded commands when they were available in FIFOs. The input samples (x) were transformed to X, frequency domain information of x, by the FFT operation. In the figure, the FFT result (X) shows three frequency components at $f_1$=16, $f_2$=64, $f_3$=100. Input samples (x) were filtered by the low-pass FIR filter with the 1/4 cut-off frequency. The result of FIR was denoted as y. In the last step, the FIR result (y) was transformed to Y by FFT to show the FIR results in frequency domain. The Y contained only low-frequency bin at $f_1$=16 due to the low-pass filtering.

## C. Conclusions

This section presents data processing units for low-power embedded SoCs. The first design introduces a radiation sensor system and its implementation. The design can compress massive sensor data, while providing high throughput, low-power, and significant compression rate. The proposed architecture is implemented with a router design for on-chip/wafer network. In the synthesis result, the data processing logic consumes 112.17mW@1GHz, and router consumes 79mW@1GHz with 128-bit data width.

In the second design we propose a DSP accelerator for sensor hub SoCs. The DSP accelerator provides both low-power consumption and programmable functionality, enabling energy-efficient digital signal processing. The DSP accelerator does not include general purpose registers demanding large area and power. The DSP accelerator loads and stores data from and to external SRAM through AHB bus for low area and power consumption. We also propose FFT, CP, FIR accelerating schemes using a pipeline architecture and auxiliary accumulators for high throughput and parallel execution. The design consumed 12.35mW power on 0.167mm$^2$ area at 333MHz. We also demonstrated it on ML605 FPGA board for verification. The result shows the proposed DSP accelerator outperforms general purpose DSPs in throughput performance, close to that of a hardware accelerator with programmable control and low area and power consumption.

# CHAPTER V

# CONCLUSIONS

Reducing system-on-chip power consumption has become a critical challenge for the nanotechnology era. Problems relating to power consumption are not only applicable to the battery powered, handheld and mobile applications where the power influences designs not only in terms of time to market, but also for cost and reliability.

We address three solutions for the low-power SoC design. First, this thesis presents a low-power embedded LDPC-H.264 JSCD architecture to lower the baseband energy consumption of a channel decoder using joint source decoding and DVFS. With the tremendous increase in the capabilities of portable multimedia devices and services, the demand to improve the energy efficiency and error robustness of such systems motivates the interest in low-power joint source-channel decoding with UEP. A novel configuration search scheme based on UEP to trade-off power and performance on power sensitive mobile devices is also presented. The proposed low-power MIMO and H.264 video joint detector/decoder design using the proposed scheme minimizes energy for portable, wireless embedded systems.

Second, we address a link-level QoS methodology using UEP for the low-power NoC and low latency on-chip network designs for MPSoCs. The QoS NoC study has been extended to develop minimum latency on-chip networks. We also contribute two NoC designs, WaveSync, an low-latency focused network-on-chip architecture for GALS designs and the SDPR scheme utilizing path diversity present in typical mesh topology network-on-chips. SDPR is akin to having a higher link width but without the significant hardware overhead associated with simple bus width scaling.

Third, we explore data processing accelerator designs for embedded SoCs. The demand for embedded SoCs containing sensors for sensing and acceleration has been increased significantly. We propose a solution for a data processing and control logic design that initiates a new sensor SoC system for radiation detection generating data at or above Peta-bits-per-second level. We also contribute to develop a DSP accelerator supporting DSP centric operations such as FFT, FIR, and 3-D cross product operations for low-power and high performance embedded SoCs. These all give benefits on SoC design for low-power sensor network systems.

Looking to the future, there are significant opportunities in emerging portable and sensor networked systems and hardware support for the systems. Leveraging the expertise in the previous established works will open a new challenge and need innovative idea to redesign the entire system. The future research directions can include:

- Study of our JSCD schemes with run-time configuration searching approach in real world environment

- Exploration of path diversity on new routing schemes for on-chip network, minimizing traversal latency along with reliability variation.

- Design of ultra low-power DSP accelerator for battery restricted sensor systems such as solar powered nano-sensor networks.

# REFERENCES

[1] International Technology Roadmap for Semiconductors (ITRS) Working Group, "International Technology Roadmap for Semiconductors (ITRS), 2009 Edition." `http://www.itrs.net/Links/2009ITRS/Home2009.htm`, accessed on May 20, 2012.

[2] M. Bedford Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar Operand Networks: On-Chip Interconnect for ILP in Partitioned Architectures," in *The Ninth International Symposium on High-Performance Computer Architecture (HPCA-9)*, pp. 341 – 353, 2003.

[3] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. Keckler, and C. Moore, "Exploiting ILP, TLP, and DLP with The Polymorphous TRIPS Architecture," in *30th Annual International Symposium on Computer Architecture (ISCA)*, pp. 422 – 433, 2003.

[4] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51 – 61, 2007.

[5] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. Brown, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro*, vol. 27, no. 5, pp. 15 – 31, 2007.

[6] W. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *Design Automation Conference (DAC)*, pp. 684 – 689, 2001.

[7] S. Xiao, C. Wu, J. Du, and Y. Yang, "Reliable Transmission of H.264 Video over Wireless Network," in *Advanced Information Networking and Applications (AINA)*, vol. 2, p. 5 pp., 2006.

[8] M. Stoufs, A. Munteanu, P. Schelkens, and J. Cornelis, “Optimal Joint Source-Channel Coding using Unequal Error Protection for the Scalable Extension of H.264/MPEG-4 AVC,” *IEEE International Conference on Image Processing (ICIP)*, vol. 6, pp. VI –517–VI –520, 2007.

[9] B. Parrein, F. Boulos, P. Le Callet, and J. Guedon, “Priority Image and Video Encoding Transmission Based on a Discrete Radon Transform,” *Packet Video 2007*, pp. 105–112, 2007.

[10] P. Yip, J. Malcolm, W. Fernando, K. Loo, and H. Arachchi, “Joint Source and Channel Coding for H.264 Compliant Stereoscopic Video Transmission,” *Canadian Conference on Electrical and Computer Engineering*, pp. 188–191, 2005.

[11] T. Kuroda, K. Suzuki, S. Mita, T. Fujita, F. Yamane, F. Sano, A. Chiba, Y. Watanabe, K. Matsuda, T. Maeda, T. Sakurai, and T. Furuyama, “Variable supply-voltage scheme for low-power high-speed CMOS digital design,” *Solid-State Circuits, IEEE Journal of*, vol. 33, no. 3, pp. 454–462, 1998.

[12] P. Gratz, C. Kim, R. McDonald, S. Keckler, and D. Burger, “Implementation and Evaluation of On-Chip Network Architectures,” in *International Conference on Computer Design (ICCD)*, pp. 477 – 484, 2006.

[13] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. W. Keckler, and D. Burger, “Implementation and Evaluation of a Dynamically Routed Processor Operand Network,” in *The 1st ACM/IEEE Int’l Symposium on Networks-on-Chip*, pp. 7–17, 2007.

[14] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger, “On-Chip Interconnection Networks of the TRIPS Chip,” *IEEE Micro*, vol. 27, pp. 41–50, 2007.

[15] N. Thomos, S. Argyropoulos, N. Boulgouris, and M. Strintzis, “Robust Transmission of H.264/AVC Video using Adaptive Slice Grouping and Unequal Error Protection,” *IEEE International Conference on Multimedia and Expo*, pp. 593–596, 2006.

[16] T. Stockhammer and M. Bystrom, “H.264/AVC Data Partitioning for Mobile Video Communication,” *International Conference on Image Processing (ICIP)*, vol. 1, pp. 545–548, 2004.

[17] M. M. Ghandi, *Layered Video Coding for Wireless Communications*. PhD thesis, University of Essex, 2006.

[18] R. Guo, L. Wang, and X. Jiang, “Stereo Video Transmission using LDPC Code,” in *International Journal of Communications, Network and System Sciences*, pp. 254–259, 2008.

[19] Y. Wang, S. Yu, and X. Yang, “Error Robustness Scheme for H.264 Based on LDPC Code,” *International Multi-Media Modelling Conference*, pp. 4 pp.–, 2006.

[20] L. Qi, L. Yang, W. Wensheng, C. Huijuan, and T. Kun, “Robust Video Transmission Scheme using Dynamic Rate Selection LDPC and RS codes,” *IMACS Multiconference on Computational Engineering in Systems Applications*, vol. 2, pp. 1673–1679, 2006.

[21] V. Kumar and O. Milenkovic, “On Unequal Error Protection LDPC Codes Based on Plotkin-type Constructions,” *Global Telecommunications Conference (GLOBECOM)*, vol. 1, pp. 493–497 Vol.1, 2004.

[22] Y. S. Yang and G. Choi, “Low-Power Embedded LDPC-H.264 Joint Decoding Architecture Based on Unequal Error Protection,” in *Future Information Technology (FutureTech)*, pp. 1–6, 2010.

[23] Y. Wang, S. Yu, and X. Yang, “Unequal Iterative Decoding for Power Efficient Video Transmission,” *IEEE International Conference on Multimedia and Expo*, pp. 613–616, 2006.

[24] X. Lu, E. Erkip, Y. Wang, and D. Goodman, “Power Efficient Multimedia Communication over Wireless Channels,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 10, pp. 1738–1751, 2003.

[25] Q. Zhang, Z. Ji, W. Zhu, and Y.-Q. Zhang, “Power-minimized Bit Allocation for Video Communication over Wireless Channels,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 398–410, 2002.

[26] Y. Eisenberg, C. Luna, T. Pappas, R. Berry, and A. Katsaggelos, “Joint Source Coding and Transmission Power Management for Energy Efficient Wireless Video Communications,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 6, pp. 411–424, 2002.

[27] J. Dielissen, A. Hekstra, and V. Berg, “Low Cost LDPC Decoder for DVB-S2,” in *Design, Automation and Test in Europe Conference Exhibition (DATE)*, vol. 2, pp. 1 –6, 2006.

[28] W. Wang and G. Choi, “Minimum-Energy LDPC Decoder for Real-Time Mobile Application,” *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1–6, 2007.

[29] W. Wang and G. Choi, “Speculative Energy Scheduling for LDPC Decoding,” *International Symposium on Quality Electronic Design (ISQED)*, pp. 79–84, 2007.

[30] W. Wang, G. Choi, and K. K. Gunnam, “Low-power VLSI Design of LDPC Decoder using DVFS for AWGN Channels,” in *International Conference on VLSI Design*,

pp. 51–56, 2009.

[31] P. Moberg, A. Osseiran, and P. Skillermark, “Cost Comparison between SISO and MIMO Deployments in Future Wide Area Cellular Networks,” in *Vehicular Tech. Conf.*, pp. 1–5, 2009.

[32] G.-H. Yang, D. Shen, and V. Li, “Unequal Error Protection for MIMO Systems with a Hybrid Structure,” *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 4 pp.–685, 2006.

[33] Y. Liu, Q. song Tong, A. dong Men, Z. yi Quan, and B. Yang, “A Joint Source-Channel Coding Scheme Focused on Unequal Error Protection for H.264 Transmission over MIMO-OFDM System,” *Computing, Communication, Control, and Management (CCCM)*, vol. 2, pp. 491–495, 2008.

[34] X. Li, S. Yang, Y. Wang, and Z. Li, “Performance of UEP based MIMO scheme for LDPC codes,” in *International Conf. on Computer Engineering and Technology (ICCET)*, vol. 6, pp. 216–220, 2010.

[35] Y. S. Yang, P. Bhagawat, and G. Choi, “Energy-efficient MIMO detection using unequal error protection for embedded joint decoding system,” in *Design Automation Conference (DAC)*, pp. 579–584, 2011.

[36] Z. Cao, B. Foo, L. He, and M. van der Schaar, “Optimality and Improvement of Dynamic Voltage Scaling Algorithms for Multimedia Applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 3, pp. 681 –690, 2010.

[37] J. Kim, S. Yoo, and C.-M. Kyung, “Program Phase and Runtime Distribution-Aware

Online DVFS for Combined Vdd/Vbb Scaling," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 417 –422, 2009.

[38] Y. S. Yang and G. Choi, "Low-power Baseband Processing for Wireless Multimedia Systems using Unequal Error Protection," in *Wireless Telecommunications Symposium (WTS)*, pp. 1 –6, 2010.

[39] S. Kumar, L. Xu, M. K. Mandal, and S. Panchanathan, "Error Resiliency Schemes in H.264/AVC Standard," *Journal of Visual Communication and Image Representation*, vol. 17, no. 2, pp. 425 – 450, 2006.

[40] F. Zhai, Y. Eisenberg, T. N. Pappas, R. Berry, and A. K. Katsaggelos, "Joint Source-Channel Coding and Power Adaptation for Energy Efficient Wireless Video Communications," *Signal Processing: Image Communication*, vol. 20, no. 4, pp. 371 – 387, 2005.

[41] JVT, "International Standard of Joint Video Specification," *ITU-T Rec. H.264, ISO/IEC 14 496-10 AVC*, 2003.

[42] VCEG, "H.264/AVC JVT JM-18.3." `http://iphome.hhi.de/suehring/tml/download/`, accessed on May 20, 2012.

[43] R. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962.

[44] D. MacKay and R. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics Letters*, vol. 33, no. 6, pp. 457–458, 1997.

[45] R. Tanner, "A Recursive Approach to Low Complexity Codes," *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 533–547, 1981.

[46] F. Kienle and N. Wehn, “Low Complexity Stopping Criterion for LDPC Code Decoders,” *Vehicular Technology Conference (VTC)*, vol. 1, pp. 606–609, 2005.

[47] G. Glikiotis and V. Paliouras, “A Low-power Termination Criterion for Iterative LDPC Code Decoders,” *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 122–127, 2005.

[48] J. Chen and M. Fossorier, “Near Optimum Universal Belief Propagation Based Decoding of Low-Density Parity Check Codes,” *IEEE Transactions on Communications*, vol. 50, no. 3, pp. 406 –414, 2002.

[49] M. Mansour and N. Shanbhag, “High-throughput LDPC Decoders,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 6, pp. 976–996, 2003.

[50] M. Mansour and N. Shanbhag, “A 640-Mb/s 2048-bit Programmable LDPC Decoder Chip,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 684–698, 2006.

[51] D. Hocevar, “A Reduced Complexity Decoder Architecture via Layered Decoding of LDPC Codes,” in *Signal Processing Systems (SIPS)*, pp. 107 – 112, 2004.

[52] K. Gunnam, G. Choi, and M. Yeary, “A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes,” *International Conference on VLSI Design*, pp. 738–743, 2007.

[53] K. Xu, “H.264/AVC Baseline Decoder.” `http://www.opencores.org/projects.cgi/web/nova/overview`, accessed on May 20, 2012.

[54] K. Xu and C. S. Choy, “Low-power H.264/AVC Baseline Decoder for Portable Applications,” in *International symposium on Low power electronics and design (ISLPED)*, pp. 256–261, 2007.

[55] Synopsys, "Design Compiler." `http://www.synopsys.com`, accessed on May 20, 2012.

[56] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms, year = 2009,*. MIT Press, 3rd ed.

[57] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *Solid-State Circuits Conference, 1990. Digest of Technical Papers. 37th ISSCC., 1990 IEEE International*, pp. 238–239, 1990.

[58] P. Wolniansky, G. Foschini, G. Golden, and R. Valenzuela, "V-BLAST: An architecture for realizing very high data rates over the rich-scattering wireless channel," *International Symposium on Signals, Systems, and Electronics*, pp. 295–300, 1998.

[59] Y. Wu, T. Cui, and C. Tellambura, "Optimal low-complexity detection for space division multiple access wireless systems," *IEEE Communications Letters*, vol. 10, no. 3, pp. 156–158, 2006.

[60] P. Bhagawat, S. Ekambavanan, S. Das, G. Choi, and Khatri.S, "VLSI Implementation of a Staggered Sphere Decoder Design for MIMO Detection," *Forty-Fifth Annual Allerton Conf.*, pp. 228–235, 2007.

[61] B. Hassibi and H. Vikalo, "On the expected complexity of sphere decoding," in *Thirty-Fifth Asilomar Conf. on Signals, Systems and Computers*, vol. 2, pp. 1051–1055, 2001.

[62] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "VLSI implementation of MIMO detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566 – 1577, 2005.

[63] C. Hess, M. Wenk, A. Burg, P. Luethi, C. Studer, N. Felber, and W. Fichtner, "Reduced-complexity mimo detector with close-to ml error rate performance," in *Great Lakes symposium on VLSI (GLSVLSI)*, pp. 200–203, 2007.

[64] "FreePDK: An open-source variation-aware design kit,"

[65] C. Alpert, A. Devgan, and S. Quay, "Buffer Insertion for Noise and Delay Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 11, pp. 1633 –1645, 1999.

[66] H. Zhou and D. Wong, "Global Routing with Crosstalk Constraints," in *DAC*, pp. 374 –377, 1998.

[67] R. Arunachalam, E. Acar, and S. Nassif, "Optimal Shielding/Spacing Metrics for Low Power Design," in *VLSI*, pp. 167 – 172, 2003.

[68] B. Victor and K. Keutzer, "Bus Encoding to Prevent Crosstalk Delay," in *ICCAD*, pp. 57 –63, 2001.

[69] P. Pande, H. Zhu, A. Ganguly, and C. Grecu, "Crosstalk-Aware Energy Reduction in NoC Communication Fabrics," in *SOC Conference*, pp. 225 –228, 2006.

[70] R. Hegde and N. Shanbhag, "Toward Achieving Energy Efficiency in Presence of Deep Submicron Noise," *IEEE Transactions on VLSI Systems,*, vol. 8, no. 4, pp. 379 –391, 2000.

[71] S. Sridhara and N. Shanbhag, "Coding for Reliable On-Chip Buses: A Class of Fundamental Bounds and Practical Codes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 5, pp. 977 –982, 2007.

[72] P. P. Pande, A. Ganguly, B. Feero, B. Belzer, and C. Grecu, "Design of Low power Reliable Networks on Chip through Joint Crosstalk Avoidance and Forward Error

Correction Coding," in *Defect and Fault Tolerance in VLSI Systems*, pp. 466 –476, 2006.

[73] A. Ganguly, P. P. Pande, B. Belzer, and C. Grecu, "Addressing Signal Integrity in Networks on Chip Interconnects through Crosstalk-Aware Double Error Correction Coding," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 317 –324, 2007.

[74] S. E. Lee, Y. S. Yang, G. Choi, W. Wu, and R. Iyer, "Low-Power, Resilient Interconnection with Orthogonal Latin Squares," *IEEE Design Test of Computers*, pp. 30–39, 2011.

[75] S. Sridhara and N. Shanbhag, "Coding for System-on-Chip Networks: A Unified Framework," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 6, pp. 655 –667, 2005.

[76] P.-T. Huang, W.-L. Fang, Y.-L. Wang, and W. Hwang, "Low Power and Reliable Interconnection with Self-Corrected Green Coding Scheme for Network-on-Chip," in *ACM/IEEE International Symposium on Networks-on-Chip (NoCS)*, pp. 77 –83, 2008.

[77] R. Kumar, Y. S. Yang, and G. Choi, "Intra-Flit Skew Reduction for Asynchronous Bypass Channel in NoCs," in *International Conference on VLSI Design (VLSI Design)*, pp. 238 –243, 2011.

[78] W. Dally and J. Poulton, *Digital Systems Engineering*. New York, NY: Cambridge Univ. Press, 1998.

[79] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist, "Lowering Power Consumption in Clock by Using

Globally Asynchronous Locally Synchronous Design Style," in *Design Automation Conference, 1999. Proceedings. 36th*, pp. 873 –878, 1999.

[80] G. Gill, S. Attarde, G. Lacourba, and S. Nowick, "A low-latency adaptive asynchronous interconnection network using bi-modal router nodes," in *NoCS*, pp. 193 –200, 2011.

[81] M. Horak, S. Nowick, M. Carlberg, and U. Vishkin, "A Low-Overhead Asynchronous Interconnection Network for GALS Chip Multiprocessors," *IEEE Transactions on CAD*, vol. 30, no. 4, pp. 494 –507, 2011.

[82] M. Donno, E. Macii, and L. Mazzoni, "Power-Aware Clock Tree Planning," in *Int'l Symposium on Physical Design*, pp. 138–147, 2004.

[83] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," in *Solid-State Circuits Conference (ISSCC)l*, pp. 98 –589, 2007.

[84] M. Amde, T. Felicijan, A. Efthymiou, D. Edwards, and L. Lavagno, "Asynchronous on-chip networks," *IEE Proceedings Computers and Digital Techniques*, vol. 152, no. 2, pp. 273 – 283, 2005.

[85] T. Jain, P. Gratz, A. Sprintson, and G. Choi, "Asynchronous Bypass Channels: Improving Performance for Multi-synchronous NoCs," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pp. 51 – 58, 2010.

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

[87] W. Dally, “Express Cubes: Improving The Performance of K-ary N-cube Interconnection Networks,” *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016 – 1023, 1991.

[88] U. Ogras and R. Marculescu, “Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion,” in *Int’l Conf. on Computer-Aided Design*, pp. 246 – 253, 2005.

[89] T. Krishna, A. Kumar, P. Chiang, M. Erez, and L. Peh, “NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication,” in *Symposium on High Performance Interconnects*, pp. 11 –20, 2008.

[90] W. Burleson, M. Ciesielski, F. Klass, and W. Liu, “Wave-Pipelining: a Tutorial and Research Survey,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 3, pp. 464 –474, 1998.

[91] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. Das, “A Low Latency Router Supporting Adaptivity for On-Chip Interconnects,” in *DAC*, pp. 559 – 564, 2005.

[92] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in *Annual Int’l Symposium on Computer Architecture*, 1995.

[93] P. Gratz and S. W. Keclker, “Realistic Workload Characterization and Analysis for Networks-on-Chip Design,” in *The 4th Workshop on Chip Multiprocessor Memory Systems and Interconnects*, 2010.

[94] XILINX, “Power Consumption at 40 and 45nm,” in *White Paper: Spartan-6 and Virtex-6 Devices*, 2009.

[95] G. Desoli and E. Filippi, "An Outlook on The Evolution of Mobile Terminals: from Monolithic to Modular Multiradio, Multiapplication Platforms," *IEEE Circuits and Systems Magazine*, vol. 6, no. 2, pp. 17 – 29, 2006.

[96] J. Xu, W. Wolf, J. Henkel, S. Chakradhar, and T. Lv, "A Case Study in Networks-On-Chip Design for Embedded Video," in *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, vol. 2, pp. 770 – 775, 2004.

[97] H. G. Lee, U. Ogras, R. Marculescu, and N. Chang, "Design Space Exploration and Prototyping for On-Chip Multimedia Applications," in *DAC*, pp. 137 – 142, 2006.

[98] T. Xu, A. Yin, P. Liljeberg, and H. Tenhunen, "A Study of 3D Network-on-Chip Design for Data Parallel H.264 Coding," in *NORCHIP*, pp. 1 – 6, 2009.

[99] L. Xin and C. sing Choy, "A Low-Latency NoC Router with Lookahead Bypass," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 3981 – 3984, 2010.

[100] X. Wang and L. Bandi, "A Low-Area and Low-Latency Network-On-Chip," in *23rd Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1 – 5, 2010.

[101] E. Carvalho, N. Calazans, and F. Moraes, "Congestion-Aware Task Mapping in NoC-based MPSoCs with Dynamic Workload," in *IEEE Computer Society Annual Symposium on VLSI*, pp. 459 – 460, 2007.

[102] L. S. Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," in *High-Performance Computer Architecture (HPCA)*, pp. 255 – 266, 2001.

[103] K. Lee, S.-J. Lee, and H.-J. Yoo, “SILENT: Serialized Low Energy Transmission Coding for On-Chip Interconnection Networks,” in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 448 – 451, 2004.

[104] D. Seo, A. Ali, W.-T. Lim, and N. Rafique, “Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks,” in *ISCA*, pp. 432 – 443, 2005.

[105] C. Izu, J. Miguel-Alonso, and J. Gregorio, “Effects of Injection Pressure on Network Throughput,” in *Parallel, Distributed, and Network-Based Processing (PDP)*, p. 8 pp., 2006.

[106] Y. Wang and Z. Wang, “Explicit Routing Algorithms for Internet Traffic Engineering,” in *Eight International Conference on Computer Communications and Networks*, pp. 582 – 588, 1999.

[107] A. Elwalid, C. Jin, S. Low, and I. Widjaja, “MATE: MPLS Adaptive Traffic Engineering,” in *INFOCOM*, vol. 3, pp. 1300 – 1309, 2001.

[108] R. Banner and A. Orda, “Multipath Routing Algorithms for Congestion Minimization,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 2, pp. 413 – 424, 2007.

[109] L. G. Valiant and G. J. Brebner, “Universal Schemes for Parallel Communication,” in *ACM Symposium on Theory of Computing*, pp. 263 – 277, 1981.

[110] T. Nesson and S. L. Johnsson, “ROMM Routing: A Class of Efficient Minimal Routing Algorithms,” in *The International Workshop on Parallel Computer Routing and Communication*, pp. 185 – 199, 1994.

[111] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, and S. Devadas, “Path-based, Randomized, Oblivious, Minimal routing,” in *The 2nd International Workshop on Network on Chip Architectures (NoCArc)*, pp. 23 – 28, 2009.

[112] S. Murali, D. Atienza, L. Benini, and G. De Micheli, "A Multi-Path Routing Strategy with Guaranteed In-Order Packet Delivery and Fault-Tolerance for Networks on Chip," in *DAC*, pp. 845 – 848, 2006.

[113] G. Michelogiannakis, D. Sanchez, W. Dally, and C. Kozyrakis, "Evaluating Bufferless Flow Control for On-chip Networks," in *Fourth ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pp. 9 – 16, 2010.

[114] G.-M. Chiu, "The Odd-Even Turn Model for Adaptive Routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729 –738, 2000.

[115] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547 – 553, 1987.

[116] N. Kamaci and Y. Altunbasak, "Performance Comparison of The Emerging H.264 Video Coding Standard with The Existing Standards," in *International Conference on Multimedia and Expo (ICME)*, vol. 1, pp. 345 – 348, 2003.

[117] "Video Traces." `http://trace.eas.asu.edu/yuv/index.html`, accessed on May 20, 2012.

[118] S. Brennan, A. Mielke, D. Torney, and A. Maccabe, "Radiation detection with distributed sensor networks," *Computer*, vol. 37, no. 8, pp. 57 – 59, 2004.

[119] Y. Yang, Y. Ju, H. Xia, W. Zhao, and Y. Zhen, "A Network Protocol Stack Based Radiation Sensor Network For Emergency System," *IJCSNS*, vol. 8, no. 8, pp. 312 – 318, 2008.

[120] E. Culurciello and P. Weerakoon, "Three-Dimensional Photodetectors in 3-D Silicon-On-Insulator Technology," *Electron Device Letters, IEEE*, vol. 28, no. 2,

pp. 117 –119, 2007.

[121] A. Ephremides, “Book Review [review of Algorithmic Information Theory: Mathematics of Digital Information Processing (Seibt, P.; 2006)],” *Signal Processing Magazine, IEEE*, vol. 24, pp. 128 –129, july 2007.

[122] Z. Yu and B. Baas, “High Performance, Energy Efficiency, and Scalability With GALS Chip Multiprocessors,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 1, pp. 66 –79, 2009.

[123] S. Sridhara and N. Shanbhag, “Coding for system-on-chip networks: a unified framework,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, no. 6, pp. 655 –667, 2005.

[124] J. W. Cooley and J. W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[125] ARM, “AMBA Open Specification.” `http://www.arm.com/products/system-ip/amba/amba-open-specifications.php`, accessed on May 20, 2012.

[126] T. Instruments, “TI TMS320C62x DSPs C62x Core Benchmarks.” `http://www.ti.com/lsds/ti/dsp/home.page`, accessed on May 20, 2012.

[127] T. Instruments, “TI TMS320C67x Floating Point DSPs C67x Core Benchmarks.” `http://www.ti.com/lsds/ti/dsp/home.page`, accessed on May 20, 2012.

[128] T. Instruments, “TI FFT hardware accelerator (HWAFFT) .” `http://www.ti.com/lit/an/sprabb6a/sprabb6a.pdf`, accessed on May 20, 2012.

[129] Tensilica, “ConnX D2 DSP Engine.” http://www.tensilica.com/uploads/pdf/connx_d2_pb.pdf, accessed on May 20, 2012.

[130] T. Instruments, “TI TMS320C55x DSP Benchmarks.” http://www.ti.com/lsds/ti/dsp/home.page, accessed on May 20, 2012.

[131] Xilinx, “ML605 Hardware User Guide.” http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf, accessed on May 20, 2012.

# VITA

Yoon Seok Yang received his B.S. and M.S. degree from Hanyang University, Korea, in February 1998 and 2000. He also received his M.S. degree in Electrical Engineering and Computer Science from University of California, Irvine, USA, in March 2008. He received his Ph.D. degree in Electrical and Computer Engineering from Texas A&M University in August 2012. His research interests include a unified architecture for communication and multimedia systems and high-performance, low-power interconnection design for on-chip networks. He is a student member of IEEE.

Yoon Seok Yang may be reached in the Department of Electrical and Computer Engineering at Texas A&M University, 214 Zachry Engineering Center TAMU 3128, Texas 77843-3128 USA. His email address is yoonseoky@gmail.com.