UNIVERSITY OF CALIFORNIA

Los Angeles

Flexibility, Scalability, and Efficiency in Next-Generation Digital Signal Processors

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Electrical and Computer Engineering

by

Uneeb Yaqub Rathore

2022

ABSTRACT OF THE DISSERTATION

Flexibility, Scalability, and Efficiency in Next-Generation
Digital Signal Processors

by

Uneeb Yaqub Rathore
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2022
Professor Dejan Marković, Chair

Despite advancements in transistor density, the last decade has seen the slowing down of Moore's law, an increasing silicon area cost, and an increasing number of dedicated accelerators in modern System on Chips (SoCs) and System in Packages (SiPs), leading to dark silicon. In trying to find alternate ways to fit more compute on a package in a cost effective way, leading chip manufacturers are adopting designs with more flexible hardware and their integration on silicon interposer based multi-chip platform technologies.

Flexible chips can reuse hardware resources shared across algorithms, increasing active utilization of silicon and reducing required chip area. Additionally they can accommodate frequent design changes for constantly evolving standards such as 5G, which would otherwise require costly chip re-designs and re-spins. However, existing flexible architectures such as coarse-grain DSPs and CGRA significantly lag behind their dedicated accelerator counterparts in terms of throughput and energy and area efficiencies (10×-25×). There is a significant need today for flexible designs that are re-usable, have high throughput, and are also efficient enough for the strict energy and cost requirements of mobile and edge devices, in addition to ensuring compliance with the evolving protocols. Multi-chip scaling and het-

erogeneous integration can significantly lower manufacturing costs and time-to-market due to higher chip yields and IP-design reuse across multiple nodes. However, large interposer bump pitch, bulky inter-chip communication links, individual custom timing circuity, and lower channel bandwidths stand in the way of widespread adoption. Moreover, in energy- and cost-sensitive mobile applications, high channel efficiency coupled with low channel area serve as additional constraints.

To address these challenges, this dissertation presents a flexible, domain-specific, 784-Core, Universal Digital Signal Processor (UDSP) array, targeting DSP applications (such as FIR, IIR, FFT and Vector-Dot-Product), achieving a 4.2× energy-efficiency gap and 6.4× area-efficiency gap from their ASIC counterparts, with high throughput (1.1GHz). The UDSP is realized with a course-grain domain-specific core that balances granularity and utilization, interconnected via a network tailored to DSP kernels with the "right" amount of connectivity. In addition, the trade-off between silicon area and compile flexibility is explored for multi-layer sparse switchbox designs resulting in an area- and time-efficient hardware-compiler co-optimized switchbox to further enhance design productivity. For advancing multi-chip scaling, this dissertation presents the 1^{st} functional, 2×2 UDSP processor on a 2-layer Silicon Interconnect Fabric (Si-IF) with 10-µm pitch I/O bumps. Utilizing the proposed Streaming Near Range - 10-µm (SNR-10) channel, the inter-chip links archive 0.38pJ/bit efficiency at 1.1GHz, and the highest bandwidth density per layer at 149Gbps/mm/layer. In an effort to further increase SNR-10 bandwidth without sacrificing technology portability, a 2.1mW, very wide range, 0.0032mm^2, fully synthesizable DLL is developed. The DLL uses a ring oscillator and counter based coarse delay line to reduce area and increase frequency range. It uses an active-preemptive fine delay line switching scheme to reduce DNL, and uses independent dual-edge delays to allow duty cycle tracking, enabling high-speed DDR links in future revisions of SNR-10.

The dissertation of Uneeb Yaqub Rathore is approved.

Anthony John Nowatzki

Danijela Cabric

Subramanian Srikantes Iyer

Dejan Marković, Committee Chair

University of California, Los Angeles

2022

To my parents.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AIB	Advanced Interface BusThis is a bus
ASIC	Application Specific Integrated Circuit
ACaS	ACcelerator as a Service
BER	Bit Error Rate
BW	Binary-weighted
CDF	Cumulative Distribution Function
CGRA	Coarse Grain Reconfigurable Array
CORDIC	Coordinate Rotation Digital Computer
DCC	Duty Cycle Correction
DCD	Duty Cycle Distortion
DCPD	Duty Cycle Pre-Distortion
DDR	Double Data Rate
DFG	Data Flow Graph
DIP	Dual In-line Package
DLL	Delay Locked Loop

DNL Differential Non-Linearity

DSE Design Space Exploration

DSP Digital Signal Processing

ESD Electro-Static Discharge

FFT Fast Fourier Transform

FIFO First In First Out

FIR Finite Impulse Response (filter)

FoM Figure of Merit

FPGA Field Programmable Gate Array

GAAFET Gate All Around Field Effect Transistor

GF Global Foundries

GPIO General Purpose Input Output

GUI Graphical User Interface

HVCC Hyper Vector Cross Correlation

HWC Hardware Cost

IC Integrated Circuit

ICI Inter-Channel Interference

IFFT Inverse Fast Fourier Transform

IIR Infinite Impulse Response (filter)

ILP Integer Linear Programming

INL Integral Non-Linearity

IP Intellectual Property

ISI Inter-Symbol Interference

LDO Linear Drop-Out (regulator)

LIPINCON Low-voltage-In-Package-INterCONnect

LSI Local Silicon Interconnect

MCBF Mean Connections Before Failure

NP Non-Polynomial

PCB Printed Circuit Board

PDN Power Distribution Network

PE Processing Element

PHY Physical (layer from the OSI model)

PRNG Pseudo-Random Number Generator

PVT Process Voltage and Temperature

RAM Random Access Memory

RDL Re-Distribution Layer

SDR Single Data Rate

SerDes Serializer De-Serializer

Si-IF Silicon Interconnect Fabric

SiP System in Package

SNR-10 Streaming Near Range 10µm protocol

SoC System on Chip

SoA State of Art

TCB Thermal Compression Bonding

TSMC Taiwan Semiconductor Manufacturing Corporation

Tx/Rx Transmitter/Receiver

UCLA University of California Los Angeles

UDSP Universal Digital Signal Processor

VS Vertical Stack

Acknowledgments

I am forever grateful to my parents, grandparents, brothers Haad and Arham, and aunt Huma, for their loving support and kindness through out the ups and downs of my doctoral years at UCLA.

I would like to thank my advisor, Professor Dejan Marković, for his vital support and advice over my time spent at UCLA. He gave me the freedom to experiment in a wide variety of topics and carve out my niche in the circuits space. I am indebted to him for the opportunities he opened and the chip implementations he assisted with. I would also like to thank Professor Subu for his advice and support with the chip implementations. I am appreciative of his career advice and insights into the future of the circuits and packaging world and how to best navigate it. I thank Professor Tony for showing me intuitive ways to look at architecture design spaces and Professor Danijela for showing me the larger impact of algorithm design on realized performance. Their guidance helped me in the finer details of the presented work and subsequent chip implementations.

I am thankful to the immeasurable number of colleagues and friends that helped me along my technical journey. For the brainstorming sessions, and for getting me out of local design minimums, I am grateful to Sumeet Nagi, Sina Basir-Kazeruni, Shahroze Kabir, Krutikesh Sahoo, Chenkai Ling, Siva Chandra Jangam, Sida Lee, Steven Moran, Qaiser Nehal, Wenhao, Tonmoy Monsoor, Manoj Reddy, and Mahmoud Rashad. I am also thankful to Kyle, Katie,

Ryo, Deeona and William without whom this work wouldn't have been (in the slightest) possible. I am indebted to Hariprasad Chandrakumar, Dejan Rozgić, Vahagn Hokhikyan, Trevor Black, Alireza Yousefi, Farhana Sheikh, Somnath Kundu, David Kehlet and Tim Hoang for the lively discussions.

I would also like to acknowledge the support of Boeing, Intel, UCLA CHIPS, and DARPA for their guidance, thorough verification, and funding in the CRAFT, CHIPS, and DRBE programs.

Vita

2015 B.Sc., Electrical Engineering (Minor Computer Science)
Lahore University of Management Sciences, Lahore, Pakistan

2015 NFM Gold Medal
Lahore University of Management Sciences, Lahore, Pakistan

2015-2016 UCLA Graduate Division Fellowship
University of California, Los Angeles

2018 M.S., Electrical and Computer Engineering
University of California, Los Angeles

2018 Analog Devices Outstanding Student Designer Award
Analog Devices Inc.

2019 Graduate Student Intern
Intel, San Jose, CA

2017-2022 Teaching Assistant, ECE Department
University of California, Los Angeles

2016-2022 Graduate Student Research Assistant, ECE Department
University of California, Los Angeles

Patents & Publications

U. Rathore*, S. S. Nagi*, S. Iyer, D. Marković, "A 16nm 785GMACs/J 784-Core Digital Signal Processor Array with a Multilayer Switch Box Interconnect, Assembled as a 2×2 Dielet with 10µm-Pitch Inter-Dielet I/O for Runtime Multi-Program Reconfiguration," Proceedings ISSCC '22 - International Solid-State Circuits Conference, pp. 52-53, 2022.

K. Sahoo, **U. Rathore**, S. Nagi, S. Jangam, D. Marković, and S. Iyer, "Functional demonstration of < 0.4 pJ/bit, 9.8µm fine-pitch dielet-to-dielet links for advanced packaging using Silicon Interconnect Fabric," 2022 IEEE 72th Electronic Components and Technology Conference (ECTC), 2022. *In press.*

S. Jangam, **U. Rathore**, S. Nagi, D. Marković and S. S. Iyer, "Demonstration of a Low Latency (20 ps) Fine-pitch (10 um) Assembly on the Silicon Interconnect Fabric," 2020 IEEE 70th Electronic Components and Technology Conference (ECTC), pp. 1801-1805, 2020.

S. Nagi, **U. Rathore**, F. Sheikh, S. Weber, "Reconfigurable Digital Signal Processing (DSP) Vector Engine," Patent US20200225947, 2020.

CHAPTER 1

Introduction

1.1 Motivation for This Work

In recent years it is becoming harder to keep up the pace of Moore's law [1, 2, 3]. With increasing computational demands and higher energy-efficiency requirements, an increasing use of dedicated accelerators have become the dominant method to bridge this gap of stagnating transistor budgets [4]. This phenomenon is exacerbated in power-limited mobile SoCs (System of Chips) [5] as well as high-performance data center processors [6, 7]. Additionally, not being able to proportionally scale down the voltage to feature-size, has ended Dennard scaling leading to increasing chip power densities. This has resulted in power-limited SoCs to throttle their on-board accelerators leading to dark silicon where the majority of the chip is under utilized [8, 9, 10]. Although dedicated blocks on SoCs increase operational-efficiency, frequent changes in algorithms, feature set requirements, and standards makes these blocks quickly obsolete, requiring costly redesign changes and chip re-spins [11, 12]. With the constantly evolving communication standards such as *5G* (shown in Figure 1.1, based on [13, 14]), these redesigns are becoming more frequent. *5G*'s fragmented market and diverse set of requirements demand solutions that can rapidly adapt to its needs [13].

Compounding the situation are the rising fabrication costs in advanced nodes, which are making chip redesigns prohibitively expensive [15] as shown in Figure 1.2 (for average

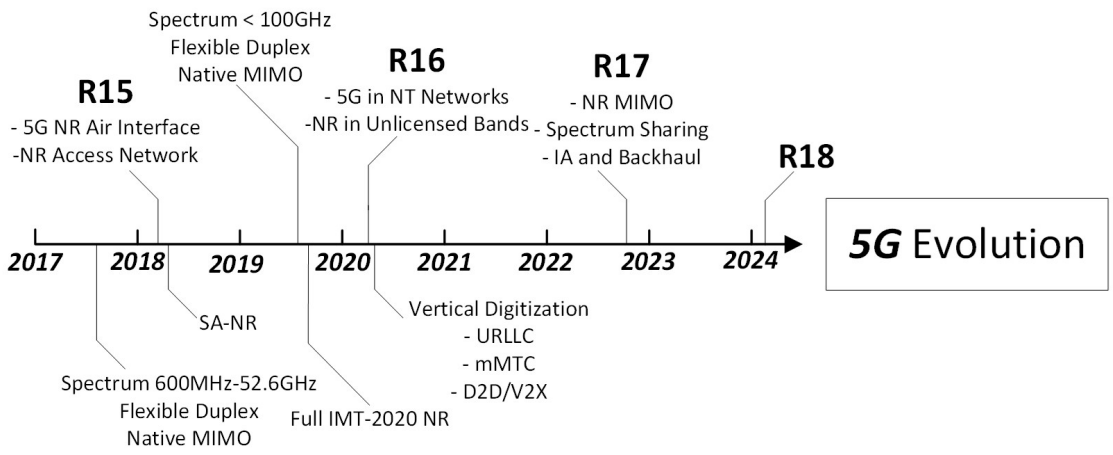


Figure 1.1: Frequent design changes in the 5G communication standards (based on [13, 14]).

commercial chips). Advanced nodes have to rely on new non-planer transistor technologies like FinFETs and GAAFETs, in addition to requiring expensive EUV based lithography processes. This, together with the rising demand of chips from industries like automobile and home automation, have caused chip costs to drastically increase [16]. Although FPGAs are flexible, their fine-grain nature makes for slower operating speed and lower efficiencies. In the past two decades, specialized high-performance course-grain re-configurable arrays (CGRAs) and very large instruction word processors (VLIWs) have re-garnered interest and shown promise to address the programability and efficiency aspects simultaneously [17]. The template-based CGRA architecture, ADRES, has the first row of cores connected to the VLIW processor that handles execution [18]. TRIPS aims to exploit parallelism by replacing the super-scalar pipeline with a large mesh-based CGRA [19]. Versat is a co-processor mainly targeting DSP functions like FFT, FIR, DCT etc., focusing on a small number of cores with a shared data-bus, resulting in smaller area [20]. More modern architectures like Plasticine incorporate address-pattern-generators to produce patterns to access data from internal and external storage [17, 21]. However, these architectures are still more than 10× away in terms of energy- and area-efficiency from functionally equivalent dedicated hardware.

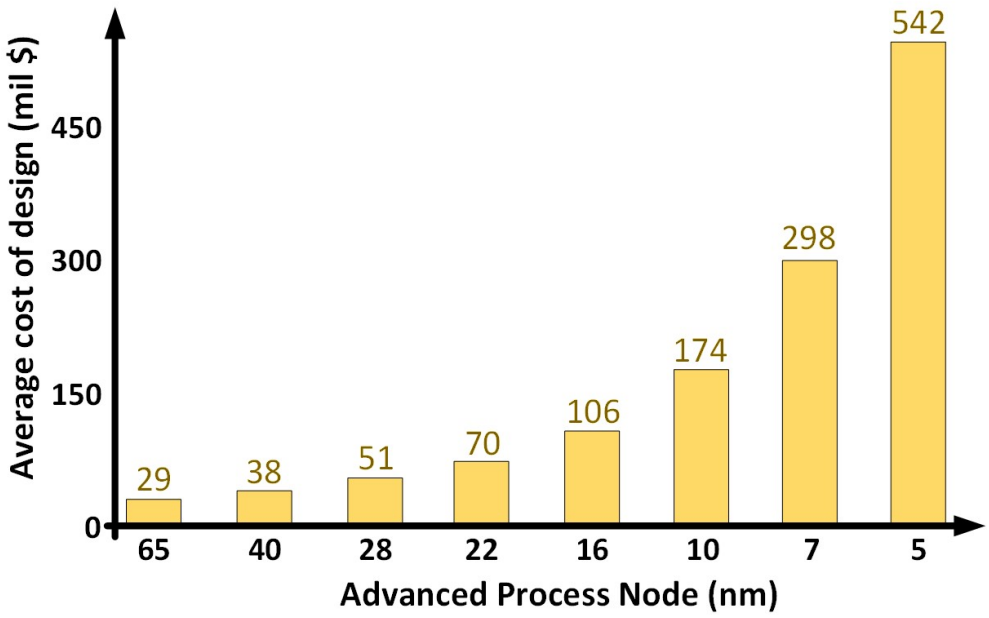


Figure 1.2: Increasing fabrication costs in advanced nodes [15].

To investigate this further, design efficiency needs to be adequately defined. For efficiency of silicon designs, there are two important metrics. First is the *energy efficiency* which is the number of (normalized) operations that a design can perform in a Joule (measured in $GOPS/mW$). Second is the *area efficiency* which is the number of operations that can be performed in a given silicon area (measured in $GOPS/mm^2$). Higher energy efficiencies lead to longer battery life in mobile devices and higher area efficiencies lead to lower developmental costs of silicon. Figure 1.3, adapted from [22, 23, 24], plots the average efficiencies of a decade worth of designs. **DSPs* include CGRAs and FPGA-DSPs. Not surprisingly, there is an inverse relationship between efficiency and flexibility. While temporally flexible CPUs and spatially flexible FPGAs can easily adapt to algorithm changes, these architectures are highly energy and area inefficient with low throughput, leading to higher cost of development and lower battery life for mobile SoCs. Dedicated hardware, while most efficient, is inflexible and incapable of adapting to the rapidly changing requirements. Although both CGRAs and FPGA-DSPs are much more efficient than general-purpose CPUs, they are still a decade

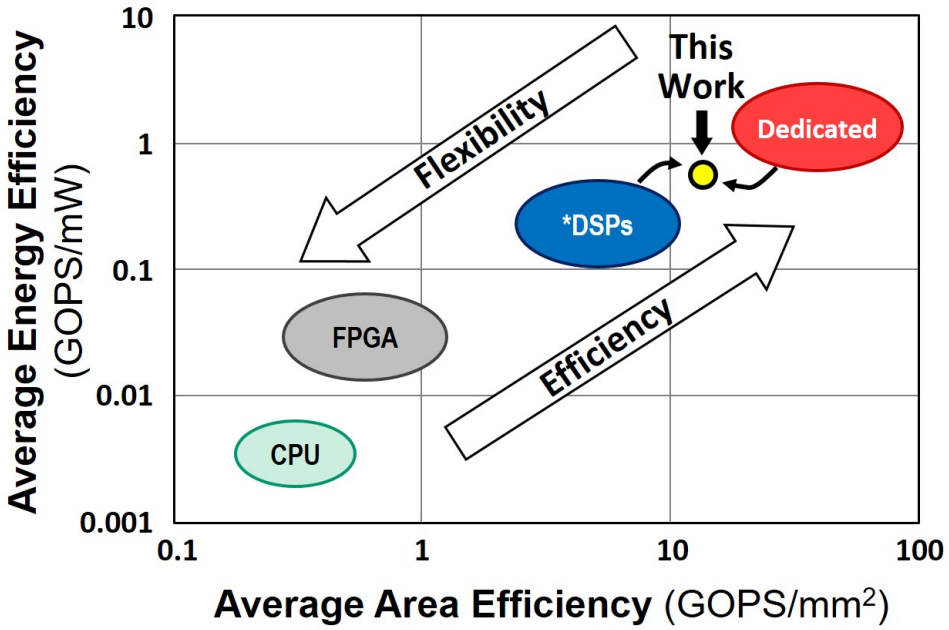


Figure 1.3: Energy- and area-efficiency of architectures with differing flexibility.

away in terms of efficiency from their dedicated ASIC counterparts. There is a significant need today for flexible designs that are re-usable and that are efficient enough for the strict energy requirements of mobile and edge devices to ensure compliance with the evolving protocol and algorithm changes. In addition, these designs should not compromise on speed and have high throughput.

The work presented in this dissertation addresses the gap between existing solutions and dedicated accelerators by building a scalable, domain-specific, coarse-grain processor array that aims to be within 5× efficiency of its ASIC counterpart in-terms of energy and area. The approach presented in this dissertation can be summarized as two key methods. First, the right amount of flexibility is added to the underlying ASIC functions by preserving common data paths. Second, domain specificity is incorporated into the design of the interconnects in addition to hardware-compiler co-optimization of internal switch matrices, resulting in higher efficiencies than current FPGA-DSPs and CGRAs. Achieving close to ASIC efficiencies with a reconfigurable architecture can reduce design costs and lower the time-to-market.

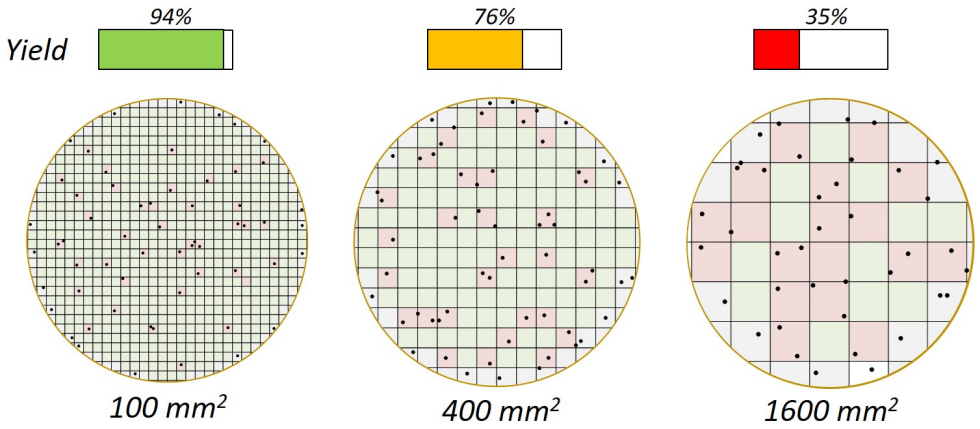


Figure 1.4: Example chip yield with increasing chip area.

Multi-chip module (MCM) scaling is another important aspect in reducing cost and time-to-market for designs. Large SoCs with many fixed-function accelerators occupy larger chip area. Since the (random) defect rate is constant in a process, there is a greater chance of defect in a bigger chip [25] (example shown in Figure 1.4). This leads to higher fabrication costs due to lower manufacturing yields which are then passed onto consumers. MCM scaling can combat this by integrating smaller heterogeneous chips in a single package, significantly lowering costs due to higher-yields and IP-design-reuse across multiple nodes [26]. MCM scaling was used by AMD to connect 8 high-performance Zen (and later Zen2) processors using longer distance SerDes to successfully commercialize the technology [27, 28]. In contrast Nvidia opted for a monolithic V100 GPU [29] that had roughly the same total silicon compute area as AMD's processor yet its cost was 4× that of AMD's CPU. This is because GPU applications are more sensitive to memory, cache and inter-PE bandwidth and latency than CPUs, which is a major concern in MCM designs. For applications such as DSPs on mobile devices, that require large on chip bandwidth, MCM scaling has an additional requirement of high bandwidth densities at high transfer efficiencies.

The recent interest in advanced packaging technologies has offered promising candidates for MCM scaling. Intel's EMIB [30] uses a silicon bridge embedded inside a cavity in an

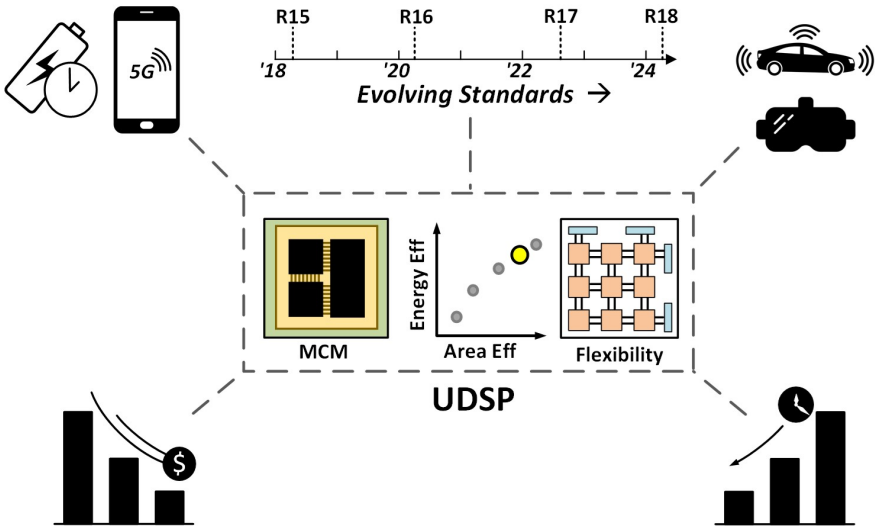


Figure 1.5: Flexible, high efficiency designs that support MCM scaling can tackle the challenges of emerging markets.

organic package. The silicon bridge allows EMIB to achieve high connection density at 55-µm bump pitch. However, higher pitch and integration with an organic substrate prove to be costly. TSMC's CoWoS [31] uses a silicon based redistribution layer (RDL) as a chip-to-chip interconnection platform for integration of chips with demonstration of up to 2× the reticle limit [32]. Such technologies however can get costly due to their high cost of multi-layer interposer development (up to 15 layers in-case of CoWoS) and packaging and integration (in case of Intel). The work presented in this dissertation targets high bandwidth (per interposer layer) with high transfer efficiency and low latency by developing a low-area, fine-pitch communication channels to enable MCM scaling for a DSP processor array. The key approach is to develop area and energy efficient communication I/Os, timing correction circuits and soft protocols for use with a 10-µm pitch emerging wafer-scale silicon integration technology, the Silicon Interconnect Fabric (Si-IF) [33]. The designs presented target low area design overhead and full synthesizability for quick portability to other technologies.

Constantly evolving needs of standards in edge devices and mobile computing devices need efficiency and flexibility in a single package (Figure 1.5). Enabling energy efficient MCM

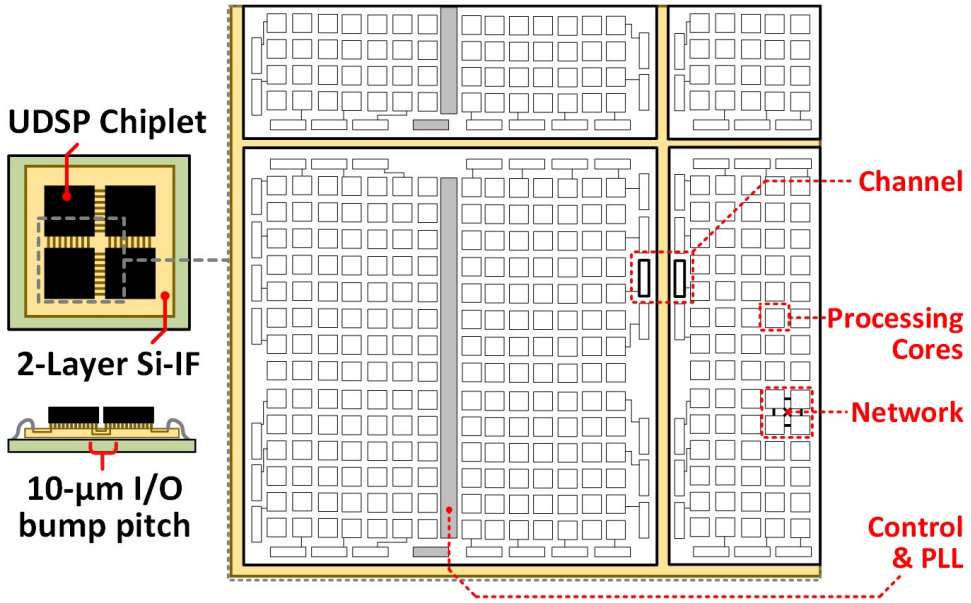


Figure 1.6: The reconfigurable processor (UDSP) MCM with processing cores, network, and channels that can stitch two dielets to form a larger processing array.

scaling on such designs hastens the time-to-market and lowers the cost for the consumer. In an effort to achieve this goal, this dissertation presents a flexible Universal Digital Signal processor (UDSP) chip with multilayered switchboxes and interconnect, in addition to multi-chip module scaling on 10-µm pitch Si-IF, and synthesizable DLL based timing correction circuits. The three highlights of this work are efficiency, scalability & flexibility. These three pillars of design are essential for the next generation of 5G, mobile and edge computing devices (Figure 1.6). 4 efficient and flexible UDSP dielets are scaled on a 2×2 assembly using area and energy efficient channels at the boundary of each chip. Coupled with a custom compiler, the assembly is treated as a large unified processor array with 784 processing elements (PEs).

1.2 Organization of This Dissertation

This dissertation is organized in the following way. The architecture of the UDSP is discussed in Chapter 2 with the details of the Core described in Subsection 2.2, the details of the Interconnect in Subsection 2.3 and the Switchbox in Subsection 2.4. Chapter 3 describes the Streaming Near Range 10µm link and protocol that allows multi-chip module scaling on the UDSP. To program the UDSP, the static and dynamic compiler tool-flow is described in Chapter 4. The UDSP assemblies and test results are outlined in Chapter 5. Chapter 6 describes a synthesizable DLL for the next generation of SNR-10 channels. Finally, Chapter 7 summarizes the contributions of this work along with suggested future research directions and topics. Figure 1.7 shows the relative hierarchy of implementations and their respective chapters and sections (in parenthesis).

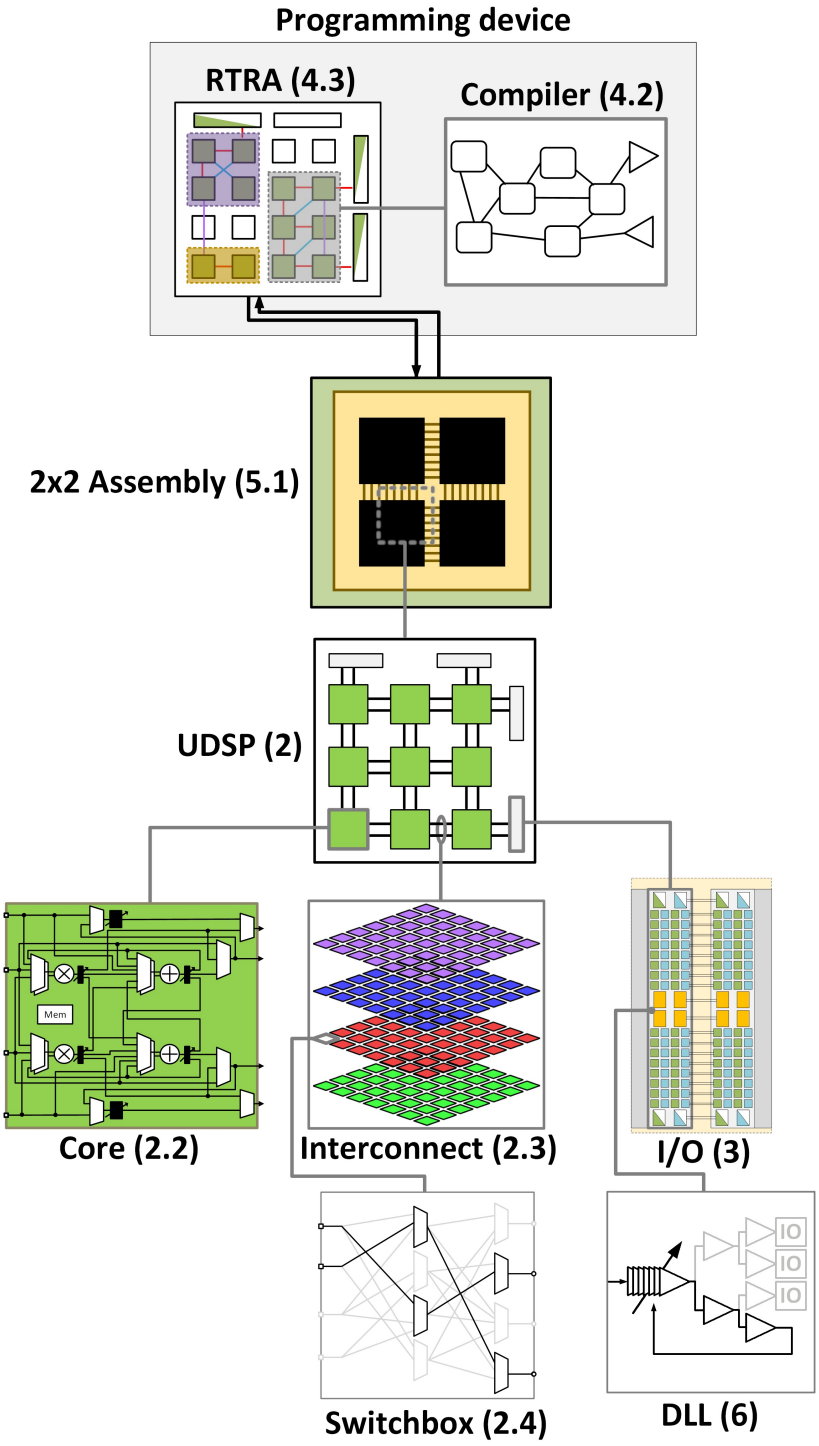


Figure 1.7: Relative implementation hierarchy and dissertation reference.

CHAPTER 2

Universal Digital Signal Processor: A Scalable, Efficient, Flexible, and Runtime-Configurable Digital Signal Processor in 16nm Technology

2.1 Synopsis of the UDSP Compute Fabric

The universal digital signal processor (UDSP) is composed of re-configurable processing elements that allow for the flexibility to map several algorithms that belong to a domain. UDSP's compute fabric has two components, the core and the interconnect. The interconnect is further made up of switchboxes. Domain-specific algorithms, or their abstractions known as data flow graphs (DFG), are mapped to the compute fabric by a compiler (Figure 2.1). In the following sections the design of each of these elements is explored and presented in detail. The design focus is on making an optimal trade-off between flexibility and efficiency. This trade-off is made by making the core and interconnect domain-specific. Improving flexibility helps in the mapping of several (similar domain) algorithms in addition to lowering compile-times. Increasing efficiency helps in getting the UDSP to near ASIC performance in terms of silicon area cost and power draw.

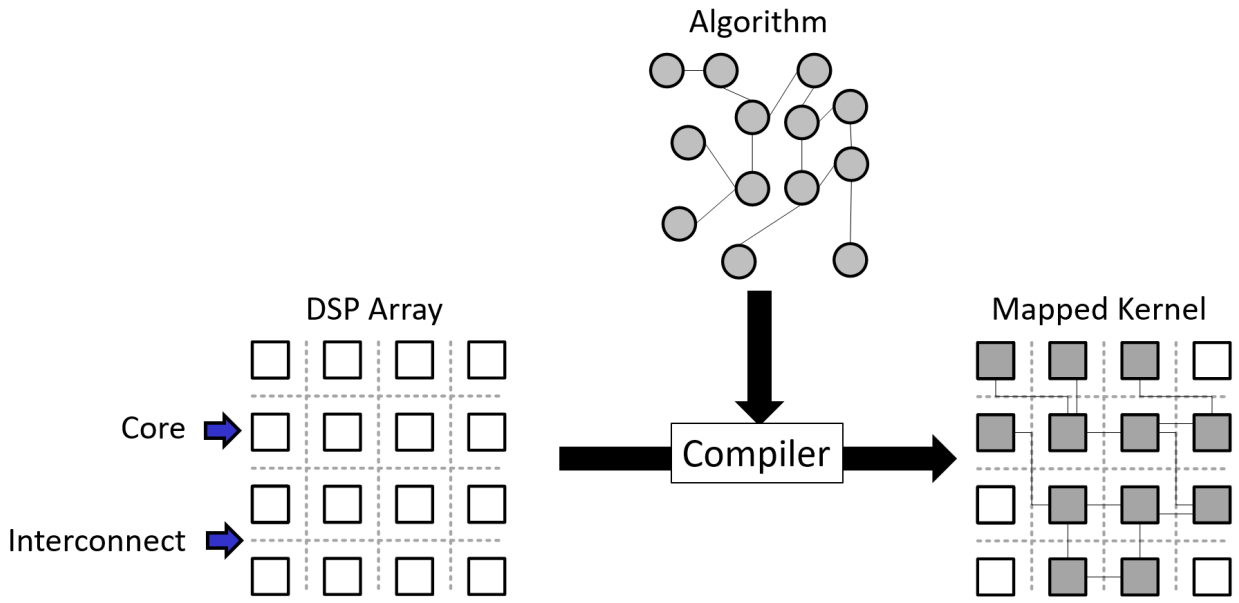


Figure 2.1: A compute fabric consists of *cores* and *interconnects* which are used by the compiler to map algorithm DFGs.

2.2 Design of the Core

The basic unit of the compute fabric is the processing element called the *core*. The core is built to be domain-specific and to map a chosen canonical set of kernels in the DSP domain. The core contains compute resources such as accumulators, routing resources to reconfigure data paths, instruction memory to store programs, and data memory to store data. It is designed iteratively by mapping DFGs on its elements and pruning the connections and elements that are not needed. Desirable characteristics of the core include high-speed operation, high energy- and area-efficiency, as well as good spatial and temporal flexibility by supporting a large number of connections between elements and instructions for spatio-temporal dynamics. To achieve these constraints the core exploits domain specificity.

2.2.1 Algorithmic and Kernel Requirements

In order to make a domain-specific core, an analysis of a particular domain is performed. For this processor, the digital signal processing (DSP) domain is chosen as it has key applications in the emerging communications processing and machine learning space. A number of common DSP algorithms and their corresponding kernels are selected, as shown in Table 2.1 and Figure 2.2. These set of algorithms and kernels serve as a guide to understanding the compute elements and the connection statistics of the DSP domain. Much of this set relies on accumulators and multipliers in equal part such as MM, MAC, CMAC, and FIR kernels. Kernels such as CORDIC, require additional temporal dynamics. Kernels such as ED require computation of quadratic terms. Extending this to general non-linear Taylor series expansions, support for a looped multiplier, or cascaded multipliers is also desired. MM, FIR, and VDP kernel based algorithms often have one input of their multipliers as fixed coefficients for a large chunks of data. As such, a constant memory bank is needed that is directly connected to the multipliers and can be reprogrammed. Kernels such as FIR need programmable number of pipeline delays, and kernels such as IIR need tight loop bounds to operate at high-speed. Most kernels have denser local I/O and sparser global I/O, which translates to dense internal connections in the core compared to its sparser I/O. Finally, to lower the core's compile-time, having adequate flexibility is another important requirement.

2.2.2 Proposed Core Architecture

To meet the above requirements, common graph sub-structures from the kernels in Figure 2.2 are extracted manually to create the connection and compute elements in a core. The process is done iteratively as shown in Figure 2.3. In each iteration additional kernels are

Table 2.1: DSP domain-specific algorithms and kernels.

Algorithms	Kernels
Digital Up Conversion	Finite Impulse Response (FIR) filter
Digital Down Conversion	Infinite Impulse Response (IIR) filter
MIMO Beam Forming	Direct, Transpose, and Lattice FIR forms
Adaptive Filtering	CORDIC
Convolution Neural Networks	Multiply Accumulate (MAC)
Spectrum Sensing	Complex MAC
Fast Fourier Transform (FFT)	Radix-2 butterfly
Inverse FFT (IFFT)	Vector Dot Product (VDP)
	Matrix-Multiply (MM)
	Complex MM (CMM)
	Zero Forcing function (ZF)
	Euclidean Distance (ED)
	3×3, 5×5, 7×7 Convolution
	Minimum Mean Squared Error (MMSE)

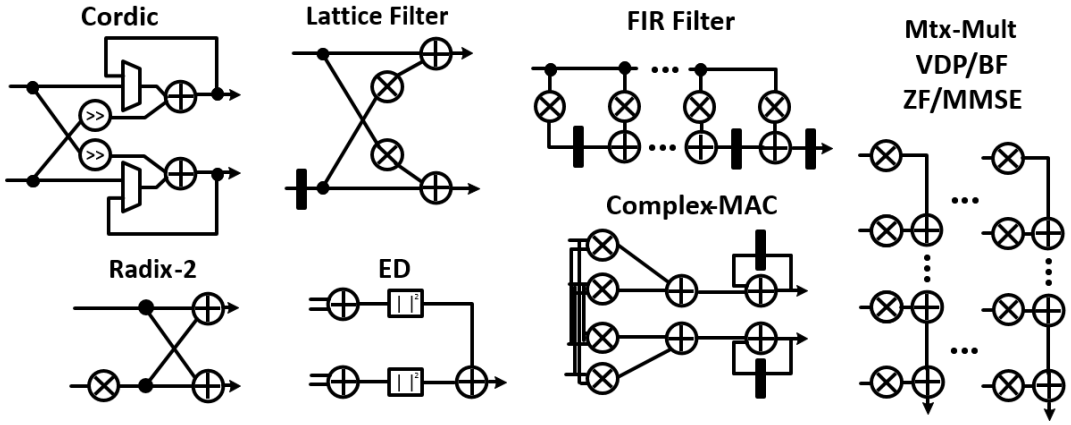


Figure 2.2: Common DSP kernels and their DFGs.

introduced and the core's coverage and utilization are checked w.r.t. the new kernels. If required connections or compute elements are missing, they are added. Extra resources are pruned. The algorithm can be split into two or more cores, having made the assumption that the routing network has infinite capacity. Emphasis is placed in balancing core granularity and utilization to maximize efficiency. Larger cores result in confinement of dense connections inside the core which leads to more area efficient inter-core routing networks.

The final core version, Core v4.2, is shown in Figure 2.3 and 2.4 with some connections

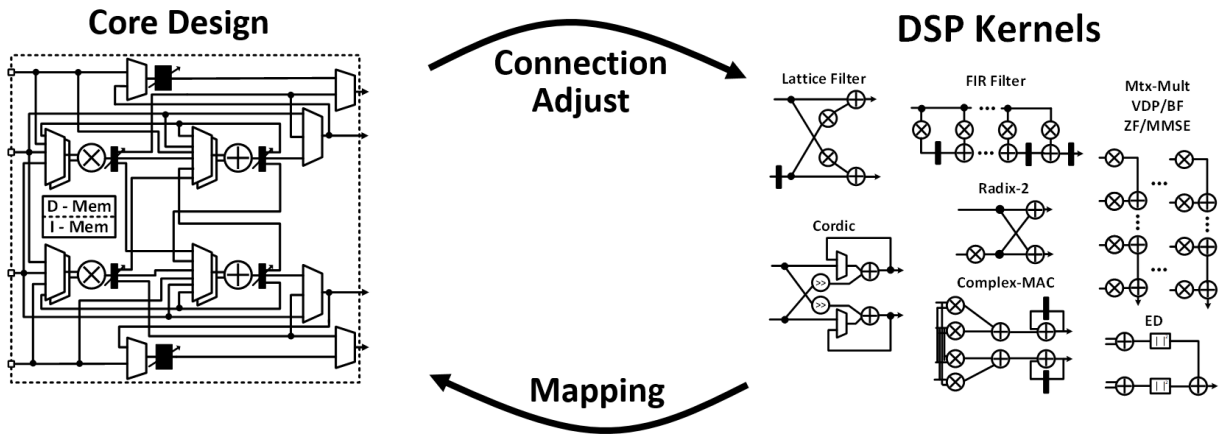


Figure 2.3: Iterative process of core design by checking utilization of mapped kernels.

omitted for clarity. It is a 16-bit fixed point architecture with a 256-bit data memory and 384-bits per instruction. The core has 4 inputs and 4 outputs, 2 adders and 2 multipliers. The adders are in loop back to allow for single cycle MACs, and support maximum precision (31-bits) during looped addition. Each of the signal sources inside the core is pipelined in order to achieve an operating frequency of 1.1 GHz. The pipeline depth is programmable to help with retiming during compile-time. For longer buffer delays, 2 delay-lines are included as well. Unlike the PE design in [23], all connections inside the UDSP core are forced to be 1-hop, which means that each element's output has no more than 1 multiplexer before being routed to an elements input. Though this increases area reduces the efficiency of the core, it increases flexibility resulting in valuable savings in compile-times as well as mild improvements in operating speed. The core has 2 types of resets; the hard reset is used to wipe the instruction and data registers, and the soft reset is used to wipe only the data. This is useful when the user does not want to change the program, rather flush the pipeline for a new set of input data.

The connections between the elements of the core preserve the common sub-structure of the domain. Figure 2.4 shows how supposedly different kernels can share the same underlying connection structure, and which combinations of connections are used to adapt the core to

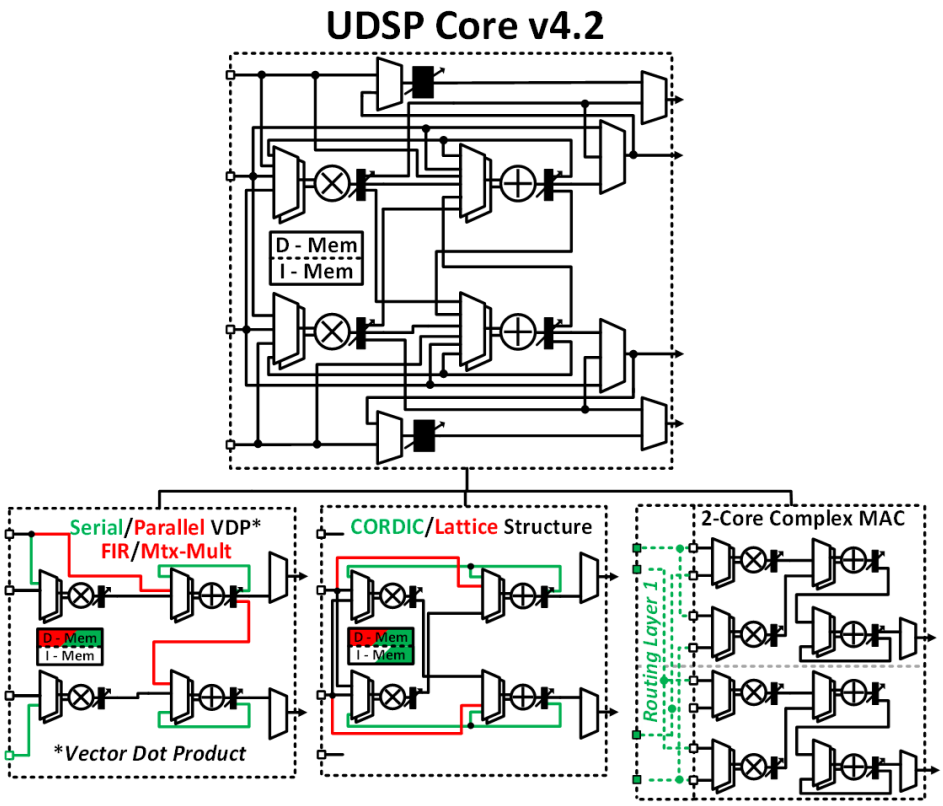


Figure 2.4: UDSP core version 4.2 with basic kernel support examples.

those particular kernels. Larger kernels which require more resources than a single core can provide, such as the complex MAC, can be mapped using two adjacent cores. Figure 2.5 shows more examples of mapped DSP domain kernels. Examples such as serial VDP and Partial Euclidean Distance (PED) can have their repetitive unit mapped onto a single core, and the radix-2 butterfly (which makes the FFT) can be mapped onto multiple cores. Some connections have been omitted for clarity as the core's connections are dense. A better way to represent the core is through its connectivity and delay matrices. In these matrices, shown in Table 2.2 and 2.3 respectively, the sources are lined up on the left and the sinks are on the top. Entries in the connectivity matrix represent the presence of connections and the element port to which they are mappable. Entries in the delay matrix represent the configurable number of delays from a source to a sink. The compiler uses these abstraction layers to quickly bind resources to the core (due to its 1-hop nature).

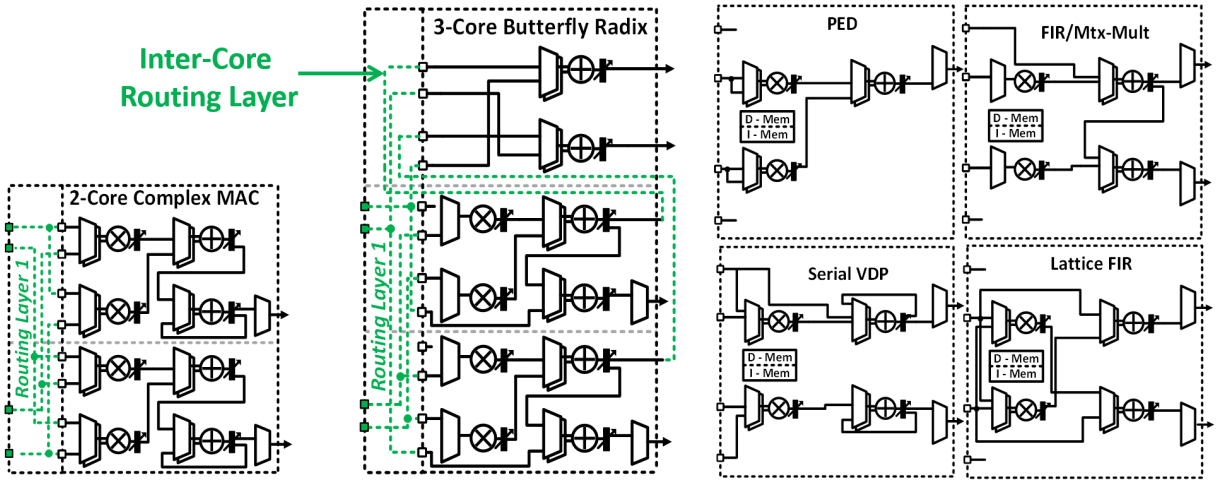


Figure 2.5: Single core and larger multi-core kernel support and mappings.

Table 2.2: Core v4.2 connectivity matrix.

	Mult0	Mult1	Add0	Add1	Cost0	Const1	Out0	Out1	Out2	Out3
In0	[2]	[2]	[1]				[1]			
In1	[1]	[2]		[2]	[1]			[1]		
In2	[2]	[1]	[2]			[1]			[1]	
In3	[2]	[2]		[1]						[1]
Mult0			[1]	[2]			[1]	[1]		
Mult1			[2]	[1]					[1]	[1]
Add0	[1]	[1]	[1]	[2]				[1]	[1]	
Add1	[1]	[1]	[2]	[1]				[1]	[1]	
Cost0	[2]		[2]	[1]						
Cost1		[2]	[1]	[2]						
Zero	[1]	[1]								

Table 2.3: Core v4.2 delay matrix.

	Mult0	Mult1	Add0	Add1	Cost0	Const1	Out0	Out1	Out2	Out3
In0	[1 2]	[1 2]	[1 2]				[2 3 4 5 6 7 8 9]			
In1	[1 2]	[1 2]		[1 2]	[1]			[2 3]		
In2	[1 2]	[1 2]	[1 2]			[1]			[2 3]	
In3	[1 2]	[1 2]		[1 2]						[2 3 4 5 6 7 8 9]
Mult0			[1]	[1]			[1 2]	[1 2]		
Mult1			[1]	[1]					[1 2]	[1 2]
Add0	[1 2]	[1 2]	[1]	[1 2]				[1 2]	[1 2]	
Add1	[1 2]	[1 2]	[1 2]	[1]				[1 2]	[1 2]	
Cost0	[0]		[0]	[0]						
Cost1		[0]	[0]	[0]						
Zero	[0]	[0]								

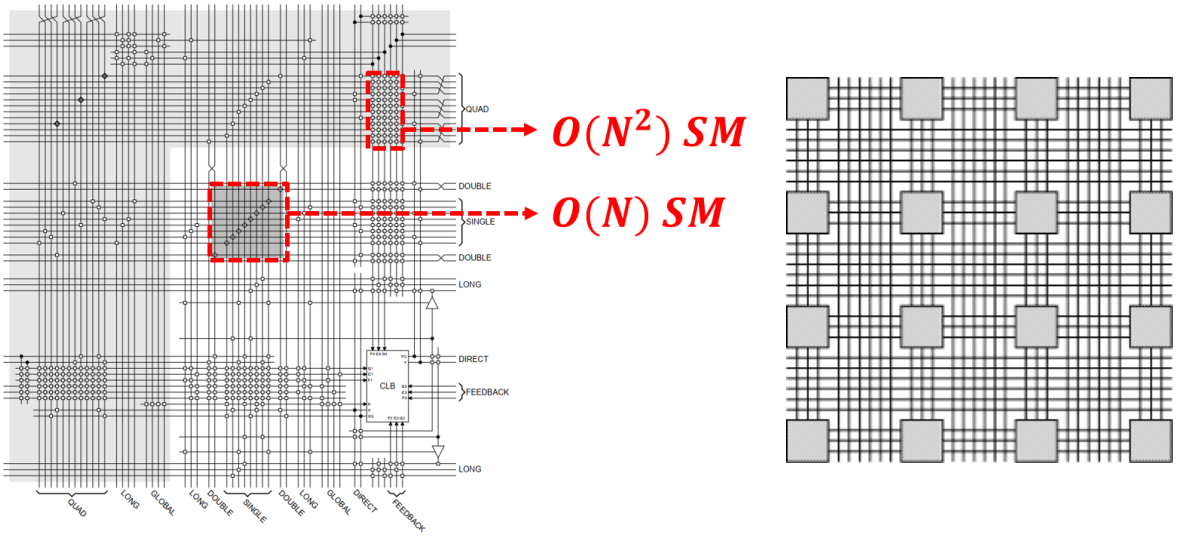
2.3 Design of the Multi-layer Interconnect

To cater to larger kernels, a domain-specific inter-core routing architecture is needed which has the *right* amount of flexibility and efficiency. Interconnect design is, in general, a balancing act between scalability, speed, efficiency and connectivity (or flexibility). In the following sections, a brief analysis of existing routing networks is presented, followed by the requirements of a domain-specific network. Finally an interconnect architecture catered to the DSP domain is presented including its I/O architecture.

2.3.1 Analysis of Routing Network Topologies

To understand how to achieve the right balance, it is instructive to analyze prior network topologies and their trade-offs. FPGA networks, such as that shown in Figure 2.6 from [34], have high fine-grain connectivity and are scalable with order *O(N)*. Often these are mesh networks (Figure 2.6a) that are repeatable allowing for large and small FPGA designs in the same family, without a lot of redesign effort. Internally, the FPGA network has fully connected (order $O(N^2)$) as well as sparse (order *O(N)*) switch and connection blocks (Figure 2.6b). Since FPGAs cater to a wide array of algorithms, in addition to being fine-grain, their interconnects are dense and dominate their area, power and delay. FPGA interconnects can occupy up to 80% of the chip area, contribute to 75% of the delay and 60% of the power [35]. According to [36], these inefficiencies can lead to energy- and area-efficiency gaps as large 60× and 140× respectively, with respect to dedicated accelerators.

Networks that employ coarse-grain processing elements (PEs) such as those shown in Figure 2.6a (from [37]) and Figure 2.6b (from [38]), can use a grid of routers. Such networks have limited single hop connectivity where each router can be static, or can be dynamic with



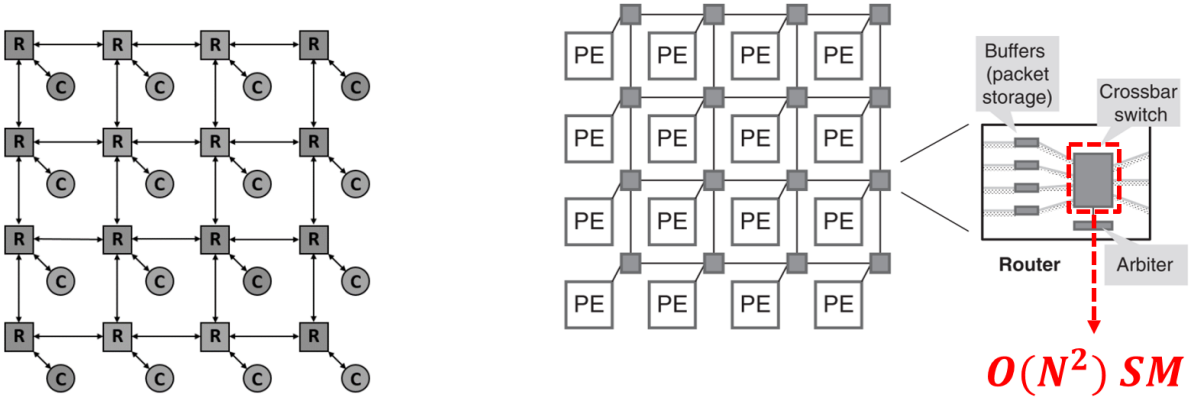
(a) FPGA network internal connections.

(b) FPGA scalable mesh network.

Figure 2.6: Xilinx FPGA network example from [34].

an arbiter and a crossbar switchbox. Statically configured networks are deterministic but due to their limited connectivity, can be slow, especially when it comes to multi-core loop bounds. Dynamic routers with arbiters can have non-deterministic latency, which can be detrimental to DSP-type flows, which require exact delay matching of a mapped flow graph. Forcing matching can result is data stalls, which is not acceptable in streaming type flows. Routing crossbars often have $O(N^2)$ complexity as they are fully connected. Despite their lack of dense connectivity, these networks are scalable and often don't occupy much area compared to compute logic, which makes them an attractive option for efficient design.

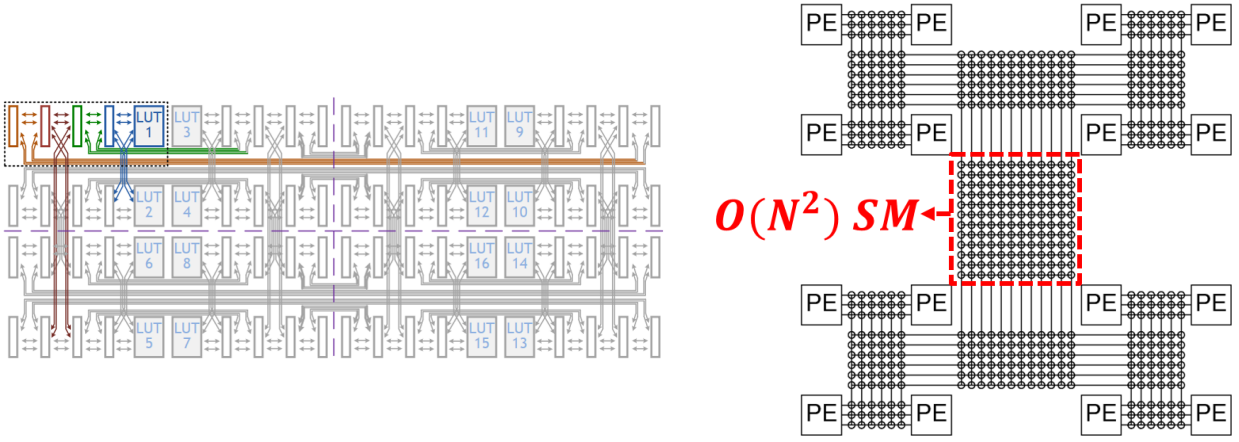
Hierarchical networks such as [39] and [40], can be used for coarse-grain as well as fine-grain routing. These architectures are more densely connected and have a longer reach than one-hop or mesh connections. In addition they do not compromise much on speed as the hierarchical nature reduces the logical distance between nodes. The network in [39] shown in Figure 2.8a is of order $O(N \cdot log_2(N))$ and the one in [40] shown in 2.8b is of order $O(N \cdot log_4(N))$. These networks often rely on fully connected $O(N^2)$ internal connection blocks which occupy a large area, in return for good connectivity and reach. Designs based



(a) Router grid network [37].

(b) Router arbiter [38].

Figure 2.7: Router grid networks for coarse-grain PEs [35].



(a) FPGA hierarchical network [39].

(b) Coarse-grain hierarchical network [40].

Figure 2.8: Hierarchical network examples.

on these networks are not scalable and are limited to a few PEs, adequate for smaller scale deployments.

2.3.2 Domain-Specific Network Requirements

An ideal network design would be fast, scalable, energy and area efficient and well connected for good routability. These are conflicting constraints, and as seen in prior art, networks often need to sacrifice one aspect to meet the requirements in other aspects. FPGAs sacrifice

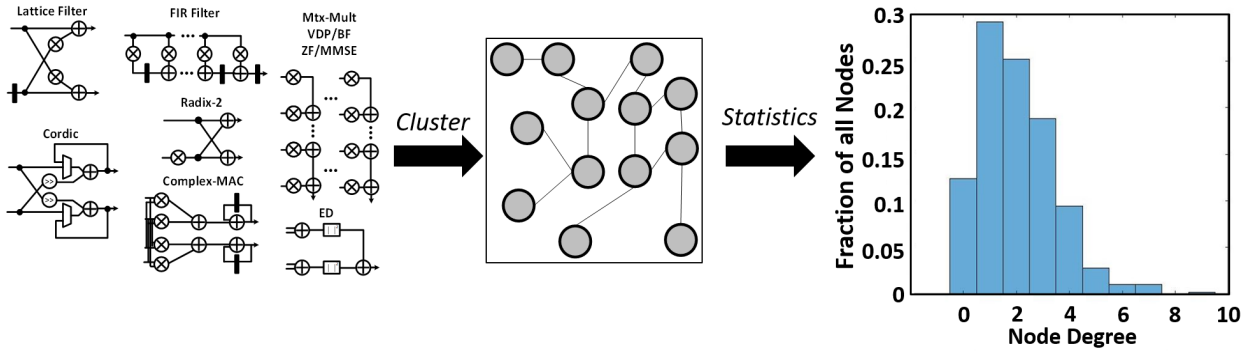


Figure 2.9: Analyzing statistics of the DSP kernels.

efficiency and speed to meet connectivity, router based network sacrifice speed and flexibility to meet efficiency, and scalability, and hierarchical networks sacrifice scalability to meet speed and connectivity. To get a routing network without trading off any of the desired properties, an extra dimension needs to be exploited. That dimension is domain specificity. In the context of PEs, the concept of domain specificity is well understood to mean sub-graphs with *good* algorithm coverage (including by an objective measure). However, the same measure is not a good or intuitive fit for connection networks, due to the diversity of connections (graph edges) and the uncertainty associated with post compile routing paths. A new definition, that accommodates this diversity is needed. In [41], that metric is derived from the statistics of post compiled route distributions. Figure 2.9 shows the method used to derive these statistics. The domain-specific algorithms and kernels used to design the core are clustered into cores using a basic compiler. The connections between the clustered cores are then analyzed for their statistics such as node degree distribution and hop separation. These statistical measures are then defined as a representative measure of the *specificity* of a domain. Since the network under design should be able to accommodate the underlying kernels with different sizes, as well as *similar* kernels not in the original design set, the derived statistics are used to replicate and reproduce graphs which have similar statistics. This allows for a way to measure and ensure the robustness of the network to variation in

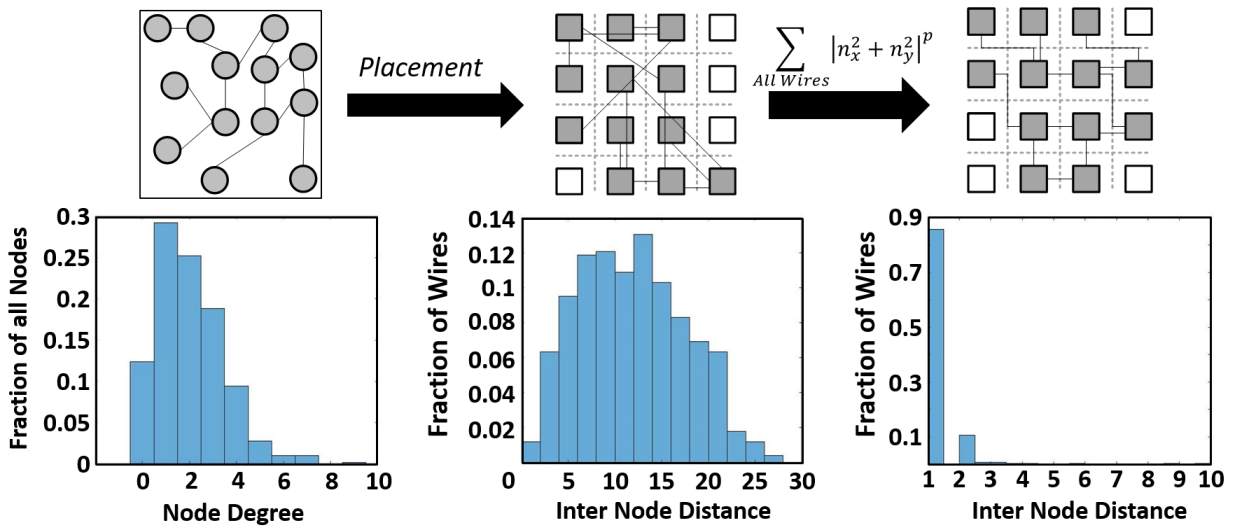
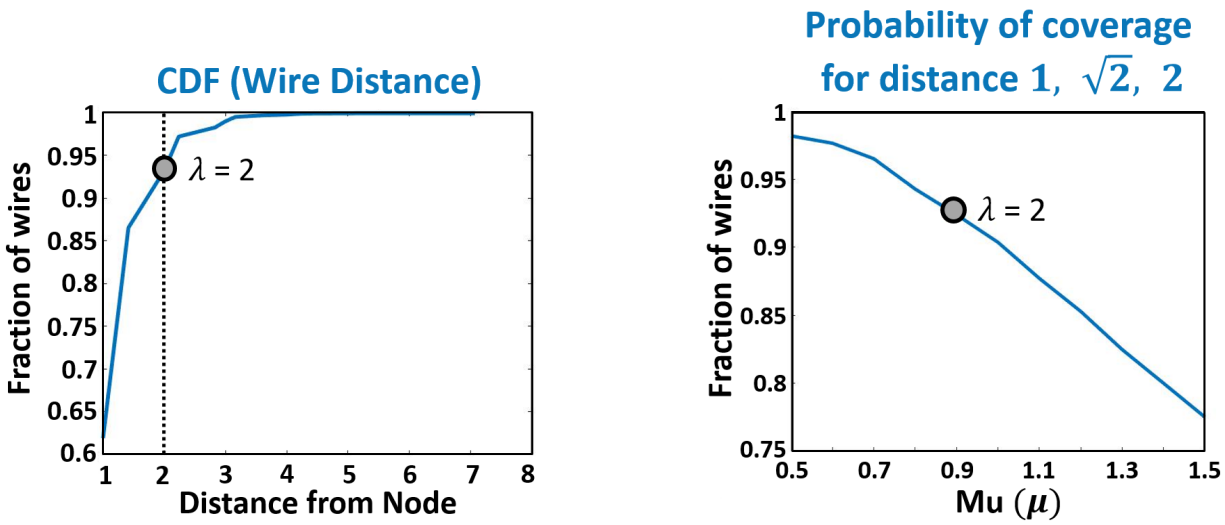


Figure 2.10: Graph models and their sorted inter-core connection distance distribution.

domain statistics.

Following these domain statistics, similar Bernoulli random graphs are generated and are randomly assigned to a virtual manhattan grid of cores. A mimic of the final compile-time placement algorithm 'simulated annealing' is run over the placed cores that minimizes routing distance. The process is illustrated in Figure 2.10. After the algorithm fixes placement, the statistics of the mapped inter-core distances are analyzed. The sorted inter-node distances follow a squared negative exponential distribution. For the DSP kernels chosen for the core, it is observed that ∼90% of the energy in the distance distribution is found in cores that are 1, $\sqrt{2}$, or 2 hops away from a reference core. A small portion of connections (5%-10%) exceed these distances. Though minimizing absolute distance is the main criteria, directions of those connections can be weighted differently. This allows for the co-optimization of the direction weights with the distance. This is used for fine optimization of bus widths.

Figure 2.11a shows the cumulative probability distribution function (CDF) w.r.t. wire distance. Distance-2 connections are marked by a dotted line and the underlying Poisson distribution with mean $\lambda = 2$ is also marked. Here λ only indicates the presence of a con-



(a) Cumulative distribution function for sorted wire distances.

(b) Variation of 2-hop coverage with underlying exponential distribution mean μ.

Figure 2.11: Analysis of domain connection statistics.

nection set, and not the number of connections per set. In Figure 2.11b, the variation of the distance-2 wire coverage probability is shown w.r.t. the underlying mean of the exponential distribution μ. It is observed that for even higher $\mu = 1.3$, the coverage probability remains above 80% indicating the robustness of the distance-2 design to kernel-space variation. The underlying Poisson distribution with mean $\lambda = 2$ is also marked. Though in this example the coverage is ∼93%, for the general set of DSP kernels the coverage is ∼85% as the μ is higher at 1.1. More detail on the methodology and comparison with FPGA can be found in [41].

2.3.3 Proposed Interconnect Architecture

With the constrains of scalability, efficiency and flexibility in mind, and insights into connection statistics of a domain, a three layered routing network is proposed in Figure 2.12 that is capable of covering all distance-2 (and lower) connections that may arise. The three

layers are logical (not physical). At the bottom are the layer of cores (Figure 2.13a) that connect through 8 I/Os to the layer-1 switchbox directly above. A network of switchboxes that connect 1-hop away in the north, south, east and west directions make up layer 1. Since the 1-hop connections outnumber the other distance connections, number of available 16-bit word lines in layer 1 are 4 in each direction (Figure 2.13b). The second layer which logically sits on top of layer 1 directly connects the core to its diagonal neighbours, in the north-east south-east, north-west, and south-west directions. The diagonal neighbours are each $\sqrt{2}$ distance away from the reference core (Figure 2.13c). The number of word lines going to these directions are half of that of layer 1 since the probability is much lower. Layer 1 connects to Layer 2 with 2 input and 2 output connections. Layer 3 sits on top of layer 2 and connects to cores that are a distance 2 away, hopping over the layer 1 cores to directly connect longer distance connections (Figure 2.13d). The probability of distance 2, rounded up, dictates that 2 words be used to connect these cores. The switchboxes that make up each layer are vertically bundled together into a stack of cores and 3 switchboxes. This is the repeatable vertical stack unit. A scalable array of these vertical stack units is built to produce the UDSP since the connection density is not a function of the number of cores.

Each of the connections in every layer are delay-less. This is critical in lowering compilation times as retiming is not required for placements that conform to distances < 2 between connected cores. To ensure sustained 1.1GHz operation regardless of core mapping, the critical path between any two directly connected cores is kept to $<$ 900ps during layout (Figure 2.14). The clock speed being independent of mapping coupled with a scalable delay-less network, allows for large algorithms (that span many cores) to have ASIC-like throughput and performance efficiencies if mapped onto the UDSP. The design methodology of constructing the layered network can be used to add additional layers on top of layer 3, if the cumulative distribution function (from Figure 2.11a) is skewed to the right for a particular domain. In

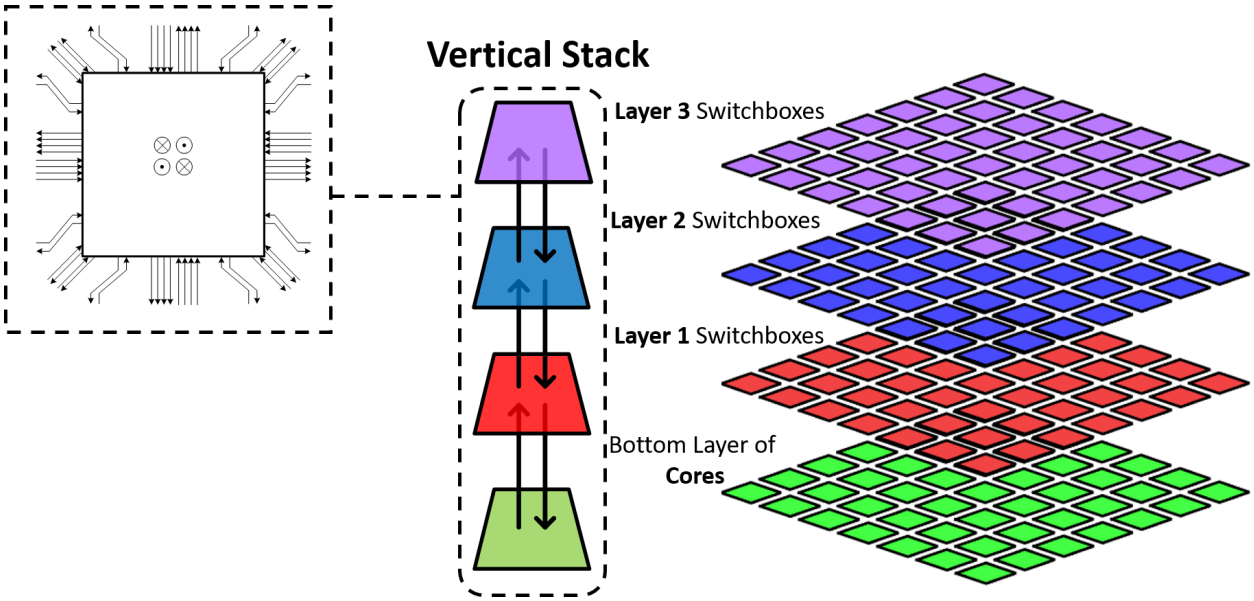
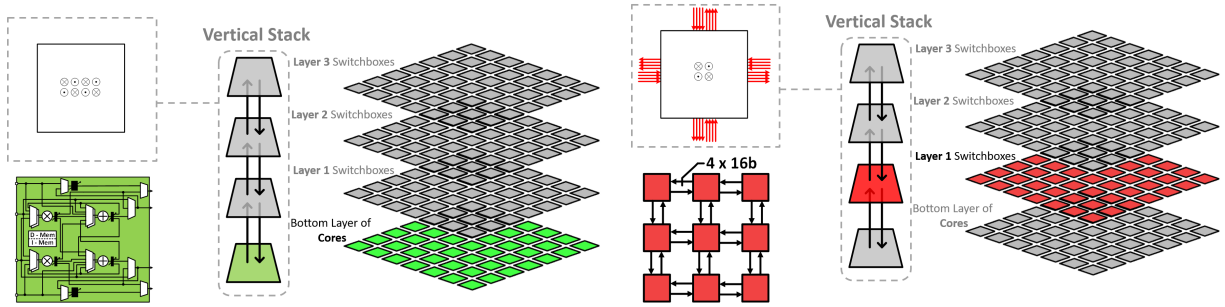
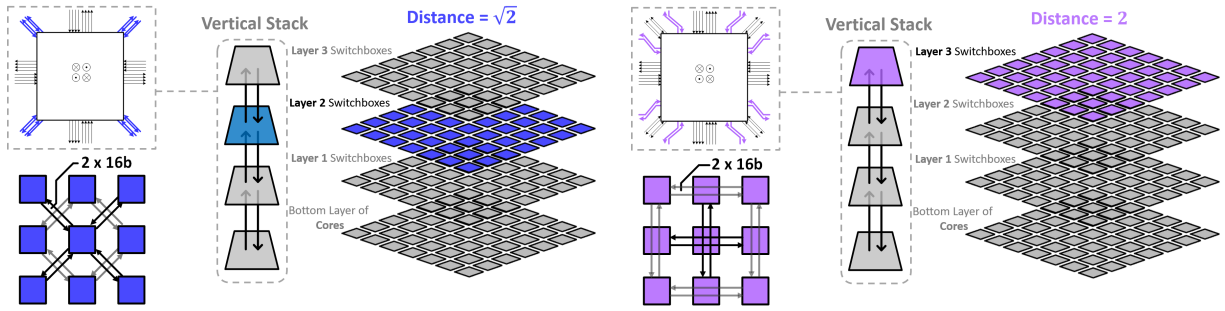


Figure 2.12: Proposed 3-layered routing network.



(a) Core layer.

(b) Layer 1, with 1-hop support.



(c) Layer 2, with $\sqrt{2}$-hop support

(d) Layer 3, with 2-hop support

Figure 2.13: Individual layers in the routing layer stack.

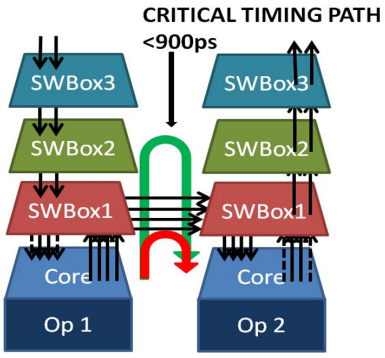
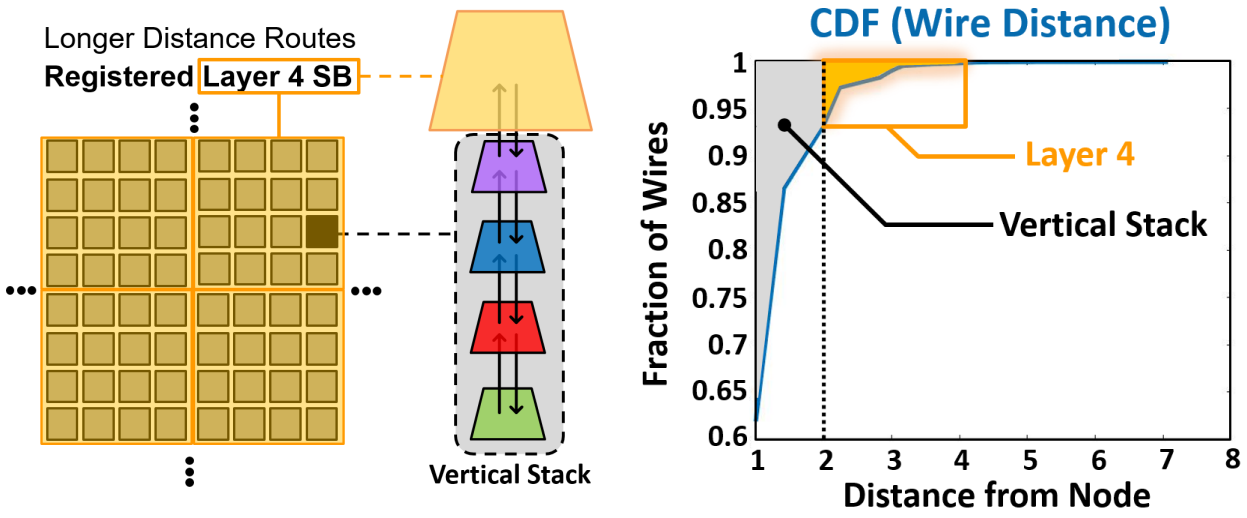


Figure 2.14: Critical path delay in each layer of the delay-less network.

that case, the underlying domain's μ value would be higher (Figure 2.11b), and increasing the layers would increase the coverage probability.

2.3.4 Design of the I/O Network

For the three layered interconnect architecture, the coverage probability reaches 85% for the DSP domain. This leaves about 15% of the connections that have distance > 2 and which cannot be accommodated by the delay-less network. For these connections there are two options. In the first option, the compiler routes these connections using multiple hops of the interconnect and cores, by registering each connection in a core after each hop. This method uses up resources of the switchboxes, cores, and routing network and puts a strain on the compiler to find these resources, reducing flexibility and increasing compile-time. To avoid this, provisions for a second method are introduced, which uses dedicated hardware resources to achieve longer distance routes. An additional layer 4 sparse switchbox is added on top of layer 3, that is registered and synthesized for 1.1GHz operation. Each switchbox in this layer connects to 16 vertical stacks underneath it (logically). The architecture of the layer is 1-hop mesh shown in Figure 2.15a. This allows for a low area overhead interconnect network that can accommodate the small number of longer distance connections needed by the DSP



(a) Layer 4 switchbox mesh network encapsulating 16 cores each.

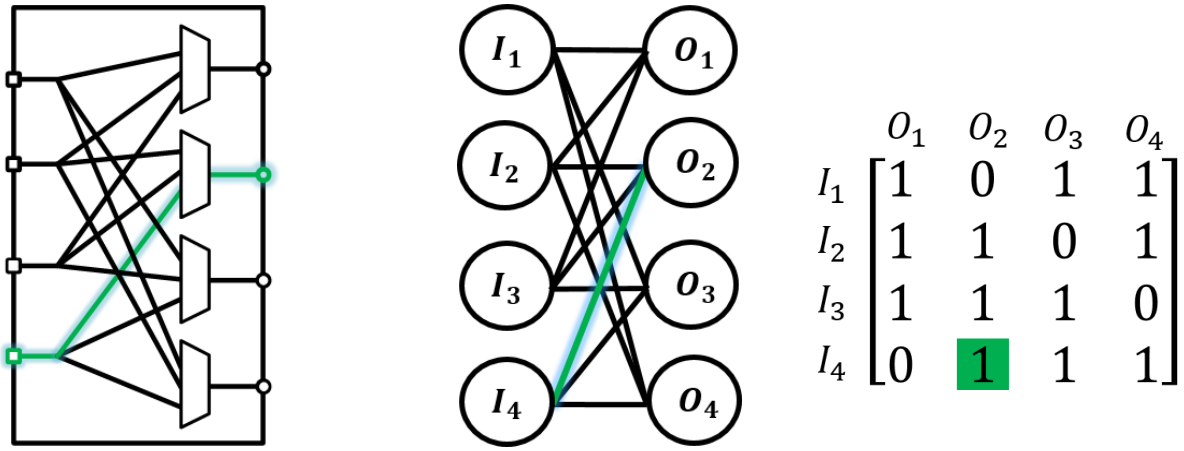
(b) Longer distance connection coverage with layer 4.

Figure 2.15: Additional hierarchical layer 4 network for longer distance routes.

domain (Figure 2.15b). This layer is not part of the vertical stack and so is not programmed in the same frame. It requires a separate program schedule and it can be independently configured. Layer 4 extends to the boundary of the design and can be tied directly to I/Os. Since the layer is registered, routing connections through it requires retiming on the part of the compiler. A hop within the same 16 core region takes 2 clock cycles, and a hop across region boundaries takes 4 clock cycles in the current design.

2.4 Multi-layer Switchbox Design and Methodology

Seen as a black box, a switchbox is responsible for connecting together input and output ports by providing possible routes between them. Conventionally, switchboxes are used in several areas of network architecture, both logical and physical. Logical switchboxes are used in packet networks and are referred to as routing matrices. Inputs and outputs are referred to as *Nodes* and each node can serve as an input or output or both. The paths



(a) Physical hardware view. (b) Abstract graph view. (c) Matrix view.

Figure 2.16: Representations of a 2-layered switchbox.

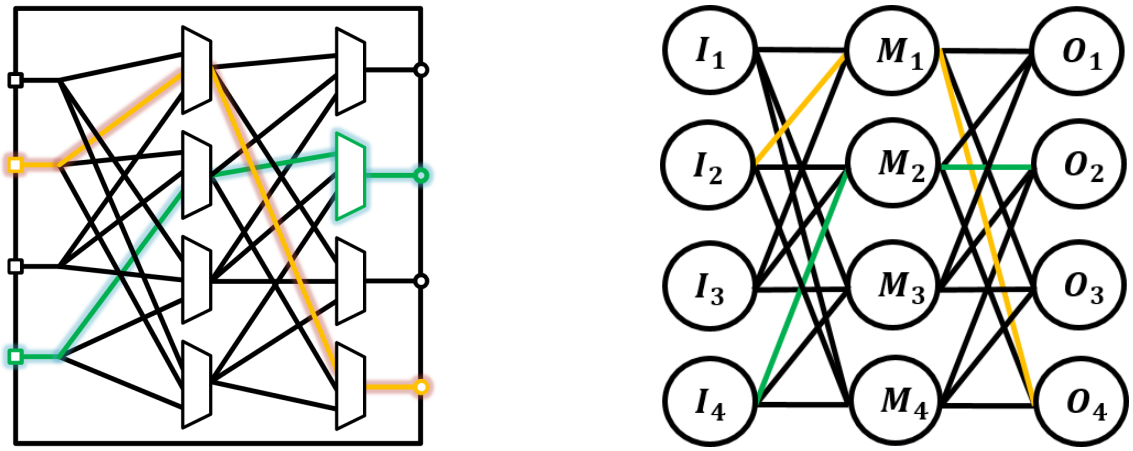
between the nodes can be weighted to signify the amount of information a connection can carry. Contrasting this to physical switchboxes such as those used in FPGAs, inputs and outputs are physical pins and the paths between them are connecting wires. Path capacity and utilization is binary, where if a connection is utilizing a path, that wire is unavailable for other connections. Physical switchboxes are referred to as switch matrices and fundamentally carry the same information as routing matrices, the difference being, the routing matrix is a capacity generalization of a switch matrix. In the following sections, the challenge of designing physical switchboxes is explored, a novel representation of a physical switchbox is described, and an experimental verification method is presented. With these tools a design space exploration (DSE) methodology is proposed, and using this methodology switchboxes for the UDSP are designed and evaluated.

2.4.1 Challenges in Switchbox Design

Figure 2.16a shows the physical hardware view of a 2-layered switchbox using muxes as switching elements and wires as connecting elements. Its corresponding graph view is shown

in 2.16b with nodes and edges replacing inputs and multiplexer outputs. Figure 2.16c shows the matrix view with the inputs on the row side and outputs on the column side. A common path is highlighted to visualize the forms it can take in each view. Since the switch matrix is not full (or not fully connected), this example switchbox is considered *sparse*. Finding a possible route between an input and output in a sparse 2-layered switchbox such as that shown is equivalent to finding the corresponding entry in its switch matrix. However, because the switchbox only consists of 2 layers and is sparse, there are guaranteed to exist I/O connections that cannot be made. For example the connection between input 3 and output 4 is missing in Figure 2.16. To resolve this, one of two approaches can be taken, 1) fully connect the switchbox, or, 2) add sparse middle layers to allow for full connectivity. The first approach has the benefit of simplicity and quick compile-times since each connection has a dedicated route and matrix entry which the compiler can quickly check at runtime. However this convenience comes at a cost of a potentially large $O(N^2)$ hardware cost that is not scalable and is prohibitive for large switchboxes. Using this approach for the UDSP would incur large switchboxes with high area overhead, due to the large (> 64) possible I/O requirement per vertical stack. Most of the extra silicon will go un-utilized since at any given time a vertical stack can at maximum use 8 of those connections. This erodes away any area- and energy-efficiency benefits that the UDSP may otherwise achieve. Therefore the second approach is used, with a sparse multi-layer switchbox providing more routing flexibility at lower hardware cost.

A multi-layer switchbox or a multi-layer network of nodes can be theoretically constructed by serially stacking two or more 2-layer switchboxes. Figure 2.17 shows a 3-layer switchbox formed by concatenating the 2-layered switchbox from Figure 2.16 with a copy of itself. Two paths are highlighted to visualize the forms they can take in each view. A compiler trying to map I/O pairs (such as the ones highlighted) on this sparse multi-layered switchbox



(a) Physical hardware view.

(b) Abstract graph view.

Figure 2.17: Representations of a 3-layered switchbox by concatenating two 2-layered switchboxes.

will face a challenge: namely it would need to solve for the map-ability of disjoint input-output path tuple sets. For example in Figure 2.17b, after the compiler maps the first path (green) through the center M node (mux M_2), it cannot map the second path through that middle layer node although there exists a path through it. This is because once a resource is utilized it cannot be shared. In addition to finding disjoint paths, a compiler will also strive to find the shortest path (in-case of unbalanced paths). This can be done by either a brute-force search or an iterative modified bellmen-ford algorithm that takes into account the disjoint nature of the paths. Both methods of routing on a multi-layered switchbox have the potential for large compile-times. Designing a switchbox that lowers these compile-times while remaining sparse and area efficient is a significant design challenge.

In a sense, the compile-time and the possibility of compile can be taken as a measure of routability or flexibility of a switchbox. The more random disjoint paths a switchbox can map, the higher its routability, at the cost of higher silicon area. Intuitively, the routability should increase if more silicon area is dedicated to the switchbox or in other words, if the sparsity of the switchbox is lowered. This is shown in Equation 2.1.

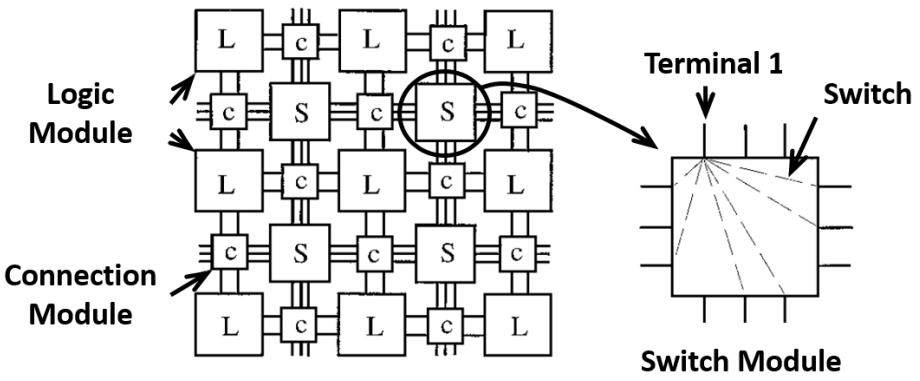


Figure 2.18: Flexibility in an FPGA switchbox, example taken from [42].

$$Area \propto \frac{1}{Connection\,Sparsity} \propto Routability \tag{2.1}$$

A fully connected switchbox therefore by definition has the highest silicon area, and the highest flexibility. It is challenging to design a switchbox that is sparse but in the right way, such that for a particular silicon area, it maximizes routability. The problem statement for switchbox design can be summarized as: Given a particular **Area**, how can the maximum **Flexibility** be extracted from a switch matrix.

Finding the right connections and the right locations for sparsity in a switchbox is an NP hard problem. This problem is most common in FPGAs where the routing network consists of an array of dense and sparse switchboxes. The switchboxes are usually bi-directional may have a different physical representation than the ones shown above, but share the same graph representation. The techniques presented ahead apply to both uni-directional and bi-directional switchboxes. Rose et al. [42] presents Figure 2.18 which is a common representation of the routing fabric in FPGAs. It highlights two kinds of switchboxes called the connection module (which interfaces with the LUTs) and the switch module (which interfaces between the connection modules). From [42] flexibility of a switch module, represented

by F_S, is defined as the number of programming switches between one terminal and others, for example, the switch module in the Figure 2.18 has $F_S = 6$. Similarly the flexibility of a connection module is represented by F_C. Alone, these two metrics define some measure of the connectivity of the switchbox, however they do not encapsulate the inter-dependencies of multi-switchbox (multi-layer) global routes. As a consequence, to ensure multi-hop global route fit, F_S, F_C, and equivalent (local connectivity) metrics are determined experimentally by running several possible longer distance connection sets, mimicking the iterations of a routing compiler. The local connectivity parameters such as F_S, are adjusted till 100% of the routing sets are satisfied. This results in NP time complexity since the aforementioned compiler mimicking step can be reduced to a subgraph isomorphism problem [43]. Therefore the challenge in the design of a multi-layered switchbox is to have a feed forward methodology of design, and not solve the generalized N-dimensional matching problem which is NP-complete as shown by Karp [44] for N=3.

2.4.2 Hyper-Matrix Representation for Multi-layer Switchboxes

Coming up with a fast and effective heuristic to solve for global and (or) multi-layer switchboxes requires an effective mathematical model of the underlying switch matrix. To develop this model, it is instructive to observe a simple 2-layer fully connected example switchbox shown in Figure 2.19. The entries in the matrix represent the capacity (number) of paths from a certain input to a certain output. For this switch matrix (because of being 2-layer and single-connections only), the capacity is limited to 1 per input-output route pair.

Coincidentally, each '1' entry in the matrix (besides the first in every column) also represents an additional hardware multiplexer, therefore the sum of the matrix can be taken as a measure of the silicon area which is a useful property. In an effort to reduce the hard-

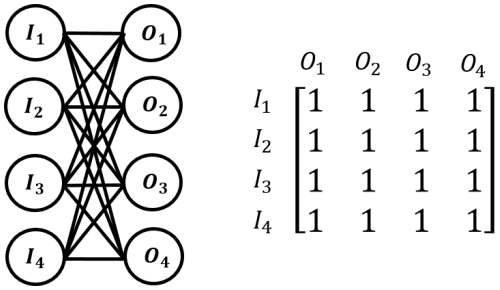
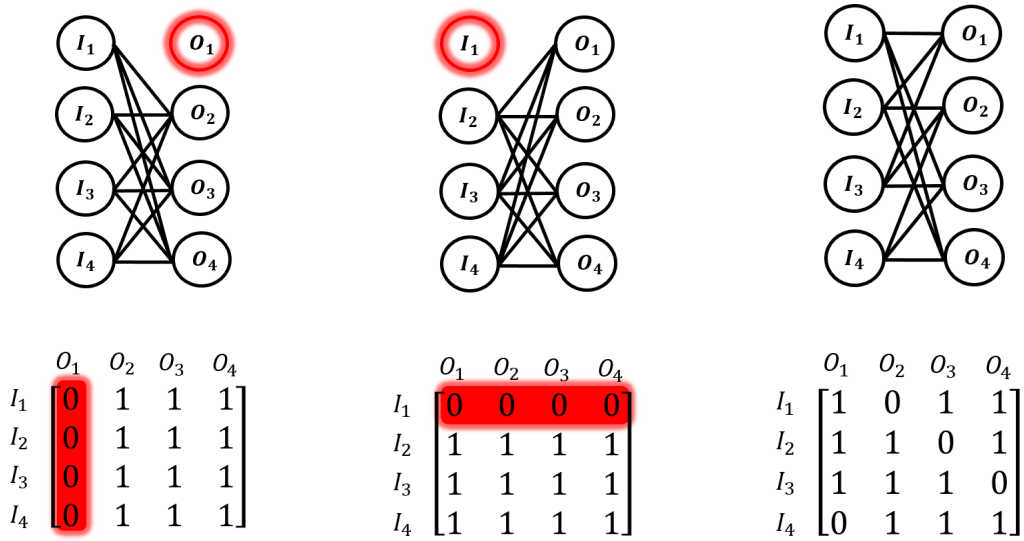


Figure 2.19: 2-layered 4 I/O fully connected switchbox graph (left) and matrix (right).



(a) Removal from one output. (b) Removal from one input. (c) Balanced edge removal.

Figure 2.20: Different ways to remove 4 connections (reducing equivalent hardware (Mux) cost).

ware cost, 4 connections (muxes) are taken out of the switchbox as shown in Figure 2.20. Different approaches can be used to take out connections that yield the same final hardware cost. Figure 2.20a erases connections from a single output which causes that node to have *zero capacity to receive information*. Figure 2.20b erases connections from a single input which causes that node to have *zero capacity to send information*. Intuitively, from the compiler's perspective, both methods make it impossible for the compiler to route to a particular output or route from a particular input. Removing the connections this way, does a poor job at maximizing routability whilst lowering silicon area. In contrast, an equal distributed removal of connections across nodes shown in Figure 2.20c demonstrates a better

strategy to improve routability, allowing higher maximum information to flow through the switchbox. This routability can be mathematically formalized as the measure of connection decorrelation among the rows (inputs) of a matrix, as well as the connection decorrelation among the columns (outputs) of the matrix [41]. To compute this the individual correlations of each combination of row is computed as in Equation 2.2, followed by summing up all the correlation contributions in Equation 2.3 and finally summing contributions from the input and output side and inverting the result to give routability shown in Equation 2.4.

$$Input\ Cross\ Correlation_{ij} \triangleq \vec{\boldsymbol{Row}}_i \cdot \vec{\boldsymbol{Row}}_j \tag{2.2a}$$

$$Output\ Cross\ Correlation_{ij} \triangleq \vec{\boldsymbol{Col}}_i \cdot \vec{\boldsymbol{Col}}_j \tag{2.2b}$$

$$Input_{CC} \triangleq \sum_{i=1}^{N_{Rows}} \sum_{j=i+1}^{N_{Rows}} \vec{\boldsymbol{Row}}_i \cdot \vec{\boldsymbol{Row}}_j \tag{2.3a}$$

$$Output_{CC} \triangleq \sum_{i=1}^{N_{Cols}} \sum_{j=i+1}^{N_{Cols}} \vec{\boldsymbol{Col}}_i \cdot \vec{\boldsymbol{Col}}_j \tag{2.3b}$$

$$Routability \triangleq (Input_{CC} + Output_{CC})^{-1} \tag{2.4}$$

$N_{Rows} \triangleq Number\ of\ Matrix\ Rows$ $\quad$ $N_{Cols} \triangleq Number\ of\ Columns\ Matrix$

$Input_{CC} \triangleq Input\ Cross\ Correlation$ $\quad$ $Output_{CC} \triangleq Output\ Cross\ Correlation$

These equations give a way to directly compare two (or more) I/O and hardware matched switchboxes to discern which has the most efficient use of a given silicon area without the need to run multiple iterations of routing algorithms and increase design time complexity. Running the example in Figure 2.20 through this method, it can be observed that Figure

2.20c is indeed the most flexible of the 3 choices. Intuitively, since the input-output paths are maximally decorrelated for a given hardware cost, the compiler should have an easier time routing through the switchbox because using one route has (by definition of minimized correlation) the least chance to affect the next route (in a set of randomly chosen routes). The paths are maximally disjoint.

Extending this analysis to multi-layered switchboxes, first requires the extension of the matrix representation. As stated earlier, a multi-layered switchbox can be thought of as multiple serially concatenated (port for port) 2-layer switchboxes. As an example, the switchbox from Figure 2.20c is concatenated with itself to give a 3-layer switchbox in Figure 2.21. A common way to extend, the matrix view, in terms of capacity of an I/O link, is to perform matrix-multiplication on the underlying 2 matrices. The resultant matrix is shown below the multi-layered switchbox in Figure 2.21. The entries in this matrix indicate the 'capacity' or the number of distinct paths from a particular input to a particular output. For example, from I_1 to O_1 the capacity is 2, and from the abstract graph view above the matrix, we can see that the two distinct paths are $I_1 \rightarrow M_1 \rightarrow O_1$ and $I_1 \rightarrow M_3 \rightarrow O_1$. A useful property for such a representation is that it can be extended for an arbitrary (N) number of layers by a simple matrix multiplication operation.

Although such a matrix is good for keeping track of connection counts between I/Os, it does not give any information on the path dependencies. In fact the matrix-multiply operation provably destroys that information. There are several underlying connection configurations (of multi-layered switchboxes) that can exists for the same resultant matrix. The representation is compressed which is why repeating the analysis from Equations 2.3a, 2.3b, and 2.4 on this multi-layer matrix does not give definitive results. In addition the sum of the entries in the matrix, in general, does not correspond to the number of muxes.

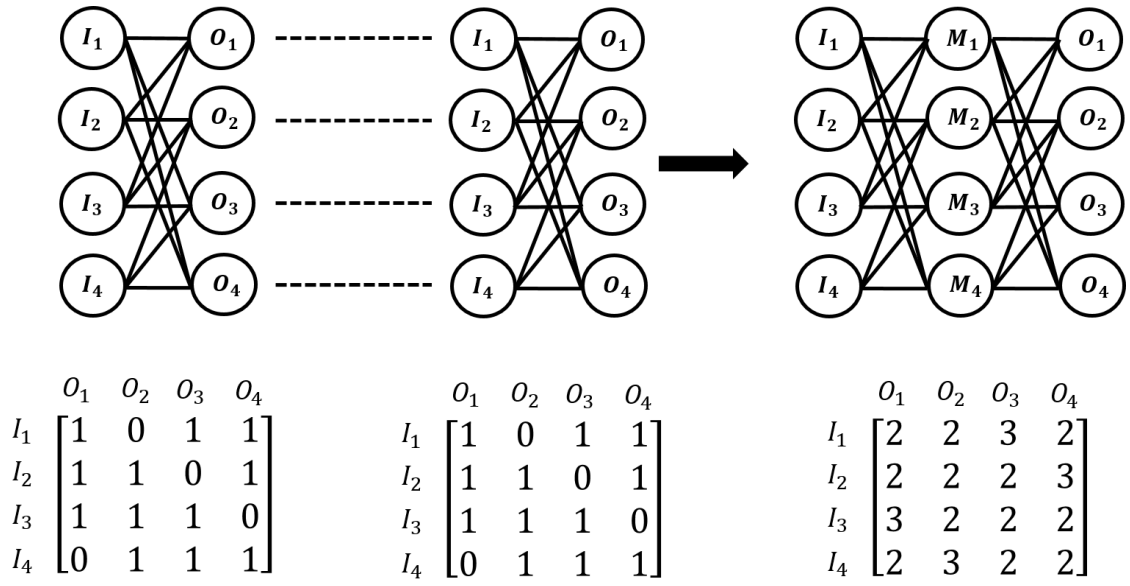


Figure 2.21: Formation of a multi-layer switchbox through serial concatenation of two 2-layer switchboxes.

To solve this, a new multi-dimensional 'hyper-matrix' representation is proposed that preserves the connectivity information, previously lost to compression. In this representation, an N-layer switchbox is represented by an N-dimensional hyper-matrix, where each entry in the hyper-matrix corresponds to a unique path in the underlying switchbox. Figure 2.22 shows an example 3-layer switchbox hyper matrix with its physical view as well as its hyper-matrix view. For visualization, '0' entries are colored in grey and '1' entries are colored in white. Two paths are highlighted in the physical view and their corresponding hyper-matrix entries are also highlighted. This representation is analogous to the 2-D representation in Figure 2.16, and is the proper generalization to N-D space. It shares similar properties to its 2-layer matrix counter part, such as finding the availability of a particular path is a simple look up operation. Similar matrices can be constructed for ≥ 3 dimensions however visualizing it on paper is challenging. A 4 layer switchbox would result in a 4-D hyper-matrix, which can be visualized as a 3-D hyper-matrix with each entry being a 1-D vector.

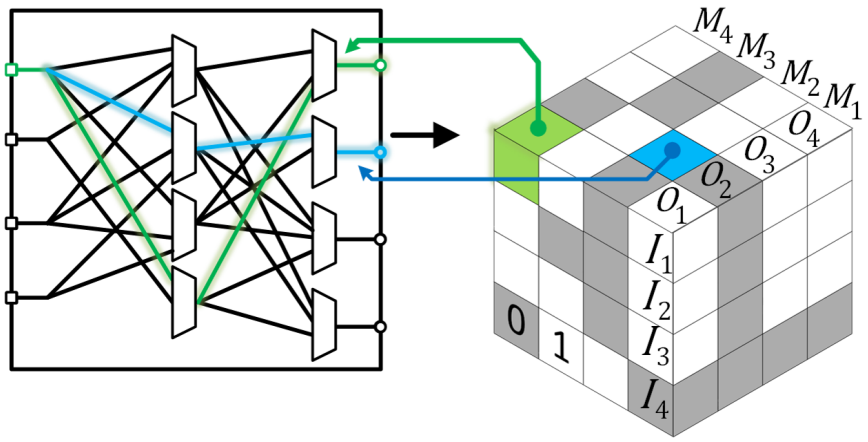


Figure 2.22: Hyper-matrix representation of a multi-layered switchbox.

To construct a multi-layered switchbox representation by concatenating two smaller (port for port) switchboxes as before, a new tensor operand is needed. This operand should be able to produce a high dimensional hyper-matrix from concatenation of 2 lower dimension hyper-matrices. For this, a new hyper-matrix tensor product is proposed called the *Rathore product* with operator symbol $\textcircled{\mathbf{R}}$. The $\textcircled{\mathbf{R}}$ product is defined formally in Equation 2.5 as operating on two arbitrary dimensional hyper-matrices $\mathbf{X}$ and $\mathbf{Y}$.

$$\mathbf{X} \textcircled{\mathbf{R}} \mathbf{Y} = \mathbf{Z} \;\; where, \;\; \mathbf{X} \in \mathbb{R}^{J_1 \times J_2 \ldots \times J_N}, \mathbf{Y} \in \mathbb{R}^{K_1 \times K_2 \ldots \times K_M}, \mathbf{Z} \in \mathbb{R}^{I_1 \times I_2 \ldots \times I_P} \tag{2.5a}$$

$$then, \;\; z_{<i_1,i_2,\ldots i_P>} \triangleq z_{<j_1,\ldots,j_N,k_1 \ldots k_M>} \triangleq x_{<j_1,j_2,\ldots j_N>} * y_{<k_1,k_2,\ldots k_M>} \tag{2.5b}$$

$$N \triangleq Dimension\ of\ \mathbf{X} \qquad M \triangleq Dimension\ of\ \mathbf{Y} \qquad N, M \geq 2$$

$$P \triangleq Dimension\ of\ \mathbf{Z}\ , \qquad P = N + M - 1\ , \qquad J_N = K_1$$

$$J_1, K_1, I_1 \triangleq Number\ of\ input\ nodes \qquad J_N, K_M, I_P \triangleq Number\ of\ output\ nodes$$

$$J_{i,(i\neq 1,N)}, K_{i,(i\neq 1,M)}, I_{i,(i\neq 1,P)} \triangleq Number\ of\ intermediate\ routing\ nodes$$

$$z_{<i_1,i_2,\ldots i_P>}, x_{<j_1,j_2,\ldots j_N>}, y_{<k_1,k_2,\ldots k_M>} \triangleq Individual\ elements\ in\ \mathbf{Z}\ ,\ \mathbf{X}\ and\ \mathbf{Y}\ matrices$$

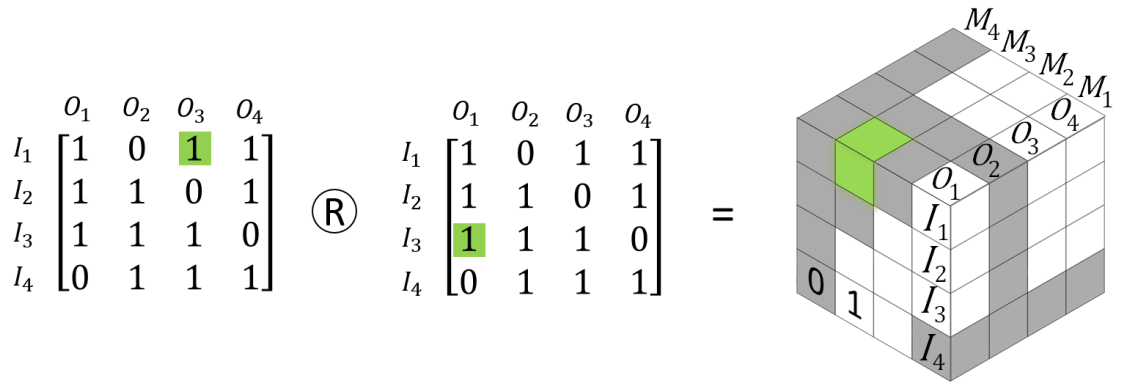


Figure 2.23: Formation of the hyper-matrix by using the (**R**) (Rathore) tensor product.

The (**R**) tensor product is not commutative, however, it is associative. It also requires that the connecting port dimensions match up port for port, i.e. $J_N = K_1$. The resultant hyper-matrix **Z** has one less dimension than the sum of the two underlying hyper-matrix dimensions. Equation 2.5 allows us to build representations of multi-layered switchboxes from simple 2-layered switchboxes. As an example, the hyper-matrix form of the 3-layered switchbox from Figure 2.21 is constructed in Figure 2.23 using the proposed tensor product. The highlighted path (in green) shows how the two underlying 2-D switchbox elements multiply to give the resultant 3-D switchbox element. The output layer of the first (**X**) matrix becomes the middle layer of the multi-layered switchbox. It should be noted that despite their helpful properties, hyper-matrix representation of switchboxes cannot directly derive the silicon cost of the underlying switchbox. To derive this cost, the underlying 2-D matrices must be used. Interestingly, collapsing (by summing together elements in) any hyper-matrix into just the input and output 2-D matrix yields the same compressed capacity matrix that is derived from direct matrix multiplication of the underlying 2-D switch matrices (as the one shown in Figure 2.21).

2.4.3 MCBF: A Switchbox Evaluation Methodology

Before proceeding to use the hyper-matrix for design, a base line design and evaluation methodology must be established to compare against. In order to experimentally verify, the routability of a multi-layered switchbox, a 'Mean connections before failure' (MCBF) metric is proposed. MCBF experimentally measures the average number of random I/O connections through a switchbox before the first routing conflict. It gives a meaningful way to compare the routability of two switchboxes with the same number of inputs and outputs regardless of the internal hardware cost. I/O pairs are randomly chosen (without repetition) from the set of all possible combination of I/Os. A fully connected 2-layer switchbox yields an MCBF equal to the number of outputs, in a directional switchbox. In a directional switchbox, a single input can connect to multiple outputs, however each middle layer node and output node can only support a maximum of one path. In a bidirectional switchbox, the MCBF increases to the sum of the number of input and output nodes. The bidirectional case can be derived from the unidirectional analysis presented. A higher MCBF typically correlates with better routing times from the compiler. In order to illustrate the MCBF of multi-layered switchboxes, examples of three 3-layered switchboxes are plotted in Figure 2.24 with varying layer densities as well as varying number of intermediate nodes. Since the switchbox has three layers, the number of nodes must be chosen for each layer.

The number of input and output nodes are both kept at 22 each for this example, and the intermediate node number is varied from 6 - 10. On the X-axis is the layer 1 density of connections which is the average number of input to middle layer connections as a fraction of a fully connected layer set. On the Y-axis is the layer 2 density of connections which is the average number of middle to output layer connections as a fraction of a fully connected layer set. A density of 1 represents a fully connected layer. For each layer density pair, and

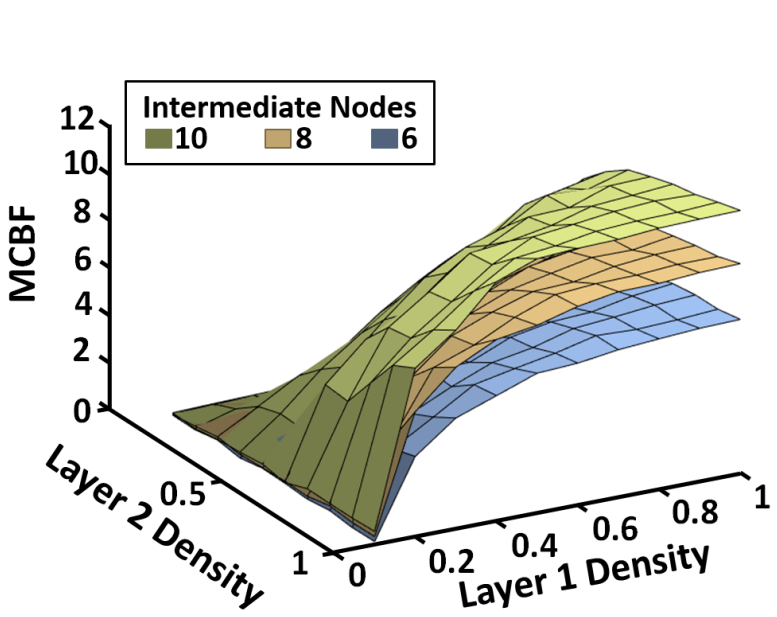


Figure 2.24: MCBF plot for 3-layered directional switchboxes, with 22 I/Os and varying densities and number of intermediate nodes.

intermediate node tuple, $\geq$ 10 million switchboxes are generated through brute-force, and their MCBF, and variance of MCBF calculated. The single best design is subsequently chosen based on the highest objective function: MCBF - 2*Variance(MCBF) and plotted. This produces a bounding surface over layer densities that shows the nearly best MCBF possible for this arrangement of nodes and node-per-layer numbers. Changing the intermediate nodes-per-layer between 10, 8 and 6, results in different surfaces in the MCBF plane. It can be observed that as the switchboxes get more dense, their MCBF increases, rapidly increasing in densities around 0.4 - 0.7. In addition, adding more intermediate nodes not only results in better MCBF but more maximum capacity of the switchbox. This is expected as higher densities and more layers are essentially adding extra hardware cost (in terms of multiplexers or switches) which increases routability from Equation 2.1.

To expose the trade-off between silicon cost and the MCBF routability metric, the MCBF per hardware cost (MCBF/HWC) is plotted in Figure 2.25. The surfaces in the plot are for

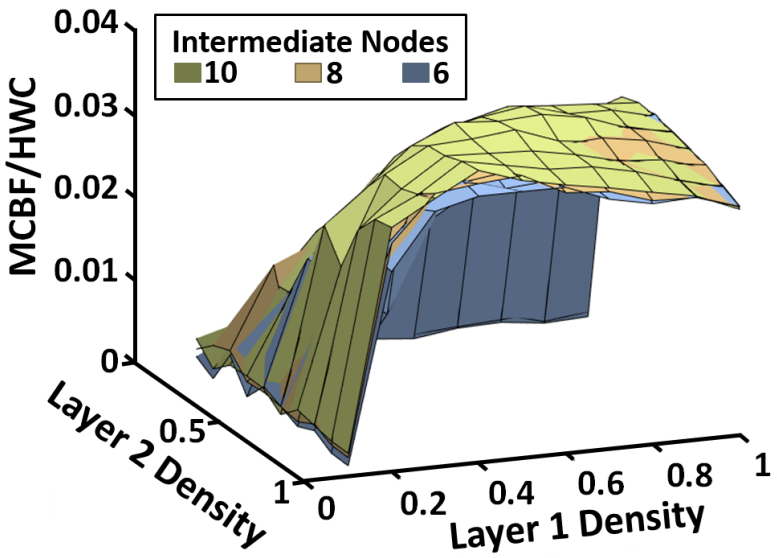


Figure 2.25: MCBF per hardware cost (MCBF/HWC) for 3-layered directional switchboxes, with 22 I/Os and varying densities and number of intermediate nodes.

the same 'best' switchboxes shown in Figure 2.24. It is noted that the MCBF/HWC does not change much with the number of intermediate nodes (though 6 and lower tend to give lower MCBFs). Also interesting is that the peak of all the graphs is near the 0.6 - 0.7 density range, meaning at these densities the silicon efficiency is maximized. Noting carefully, the curves slightly favour layer 2 densities over layer 1, which is due to the directional nature of the switchboxes which causes output side layer densities to have more impact. The figure also highlights that a fully connected switchbox may not be the most silicon efficient in terms of routability. Although the MCBF/HWC surface plots present an effective tool to trade-off between routability and silicon area, the method to produce them consumes a large amount of time. The possibility of generation of random switchboxes in a $\mathbb{R}^{22\times8+8\times22}$ dimensional space is large, and can grow exponentially larger for a larger number of P layers: $\mathbb{R}^{I_1\times I_2+\ldots I_{P-1}\times I_P}$. In addition, the MCBF measurement requires running routing experiments which is the equivalent to employing heuristics to solve the N-dimensional NP matching problem. Using the MCBF as the main design methodology would therefore be akin to the closed loop design

methodology used in FPGAs which has NP time complexity.

2.4.4 HVCC based Design Space Exploration

A better and faster open loop methodology for 2-layer switchboxes is presented in Equations 2.2, 2.3, and 2.4. However, the same operations can not be applied to multi-layer switchboxes due to the path dependency information compression observed in conventional (multi-layer switchbox) matrix representations such as that shown in Figure 2.21. To alleviate this problem, a hyper-matrix representation for multi-layered switchboxes is proposed and a method to derive it is shown in Equation 2.5. The new representation stores each path as an individualized entry. In order to maximize the disjoint nature of paths and avoid routing conflicts (in a sparse switchbox, for a given silicon area), the paths need to be decorrelated. For a 3-dimensional hyper-matrix, this translates to minimizing the row (I, input), column (O, output), and aisle (M, middle) dimension vector cross-correlations. These hyper-vectors are visualized in Figure 2.26. A general hyper vector is a 1-D array of numbers indexed along a particular dimension in a multi-dimensional tensor as shown in Equation 2.6a.

The concept can be extended to N-dimensions for an N-dimensional hyper-matrix. For such a matrix, the hyper-vector cross-correlations (HVCC) in each dimension is calculated by taking the dot product of the (non repetitive) combination of vector pairs pertaining to that dimension as shown in Equation 2.6b. Intuitively, the HVCC of the k^{th} dimension measures the *global* inter-dependencies of paths that flow through the nodes of the k^{th} layer of the underlying switchbox. Since the HVCC of each dimension needs to be *individually* minimized, the routability is defined as the inverse of the *sum* of all the HVCCs summarized in Equation 2.6c. This means that for high routability, the underlying hyper-matrix should have low path correlation amongst all the nodes in all of its layers. It should be noted that

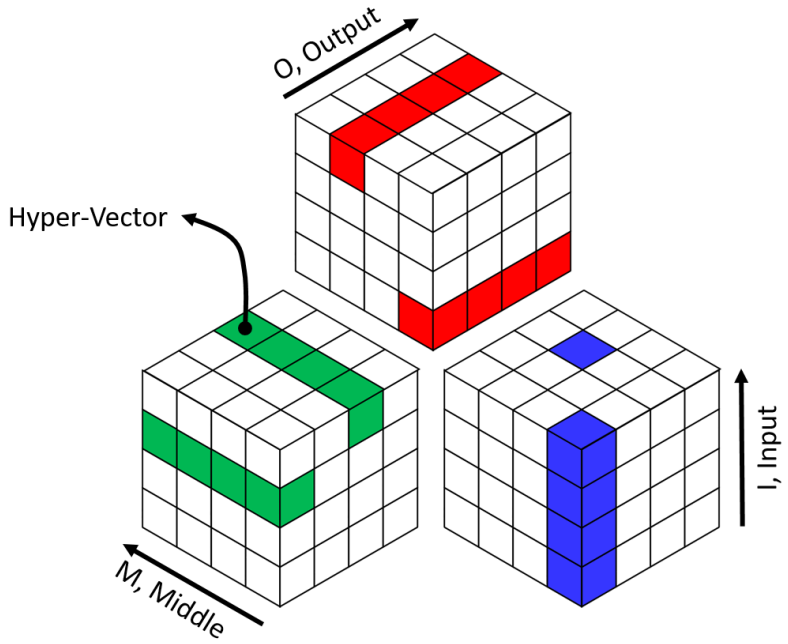


Figure 2.26: Hyper-vectors in each dimension, for a 3-layered switch matrix with 4-4-4 configuration.

the hyper-matrix cannot denote resource cost, and therefore, in order to change connections in the hyper-matrix, the changes need to be made to the underlying 2-D switch matrices. Then through the $\textcircled{\mathbf{R}}$ tensor product, the new hyper-matrix is calculated.

$$\vec{\boldsymbol{x}}^{k}_{<i_1,...,;,...i_N>} \triangleq [\boldsymbol{X}(i_1,...,1,...i_N), \boldsymbol{X}(i_1,...,2,...i_N), ...\boldsymbol{X}(i_1,...,S_K,...i_N)]' \tag{2.6a}$$

$$HV^{k}_{CC} \triangleq \sum_{\substack{All\ combinations\ of \\ <i_1,...,;,...i_N> and <j_1,...,;,...j_N> \\ 1\le i_i, j_i \le S_I}} \vec{\boldsymbol{x}}^{k}_{<i_1,...,;,...i_N>} \cdot \vec{\boldsymbol{x}}^{k}_{<j_1,...,;,...j_N>} \tag{2.6b}$$

$$Routability \triangleq (\sum_{k=1}^{N} HV^{k}_{CC})^{-1} \tag{2.6c}$$

$$\vec{\boldsymbol{x}}^{k}_{<i_1,...,;,...i_N>} \triangleq Hyper\ Vector\ in\ the\ k^{th}\ dimension \qquad S_K \triangleq size\ of\ the\ k^{th}\ dimension$$

$$\boldsymbol{X} \triangleq N\text{-}D\ Hyper\ Matrix \qquad \boldsymbol{X}(i_1,...,i_k,...i_N) \triangleq Hyper\ Matrix\ indexed\ element$$

$$HV^{k}_{CC} \triangleq k^{th}\ dimension\ Hyper\text{-}Vector\ cross\ correlation$$

With a method to represent a multi-layer switchbox (hyper-matrix), a proposed method to objectively calculate its routability (HVCC), and an evaluation metric (MCBF) to compare against baseline brute-force exploration, a design space exploration methodology is devised, as shown in Figure 2.27, and described below.

- For a particular node-per-layer set switchbox architecture, a fully connected switchbox (across all layers) is taken as a starting point. Its hyper-matrix representation is computed, and subsequently a HVCC score given to its state.
- Next, a single connection from the underlying switchbox layers is removed (more than one connection can be removed for speed up reasons, at the cost of optimality)
- The resultant switchbox's hyper-matrix is computed by using the $\textcircled{\mathbf{R}}$ tensor product described in Equation 2.5.
- The HVCC is computed over that hyper matrix to give it a routability score as described in Equation 2.5. This score shows the reduction in routability for the removal of the particular connection (chosen earlier).
- Since there are a large number of connections that can be removed from a switchbox, the above procedure is repeated for every possible connection removal option giving $O(N \times S_K)$ computational complexity. Intuitively each connection removal corresponds

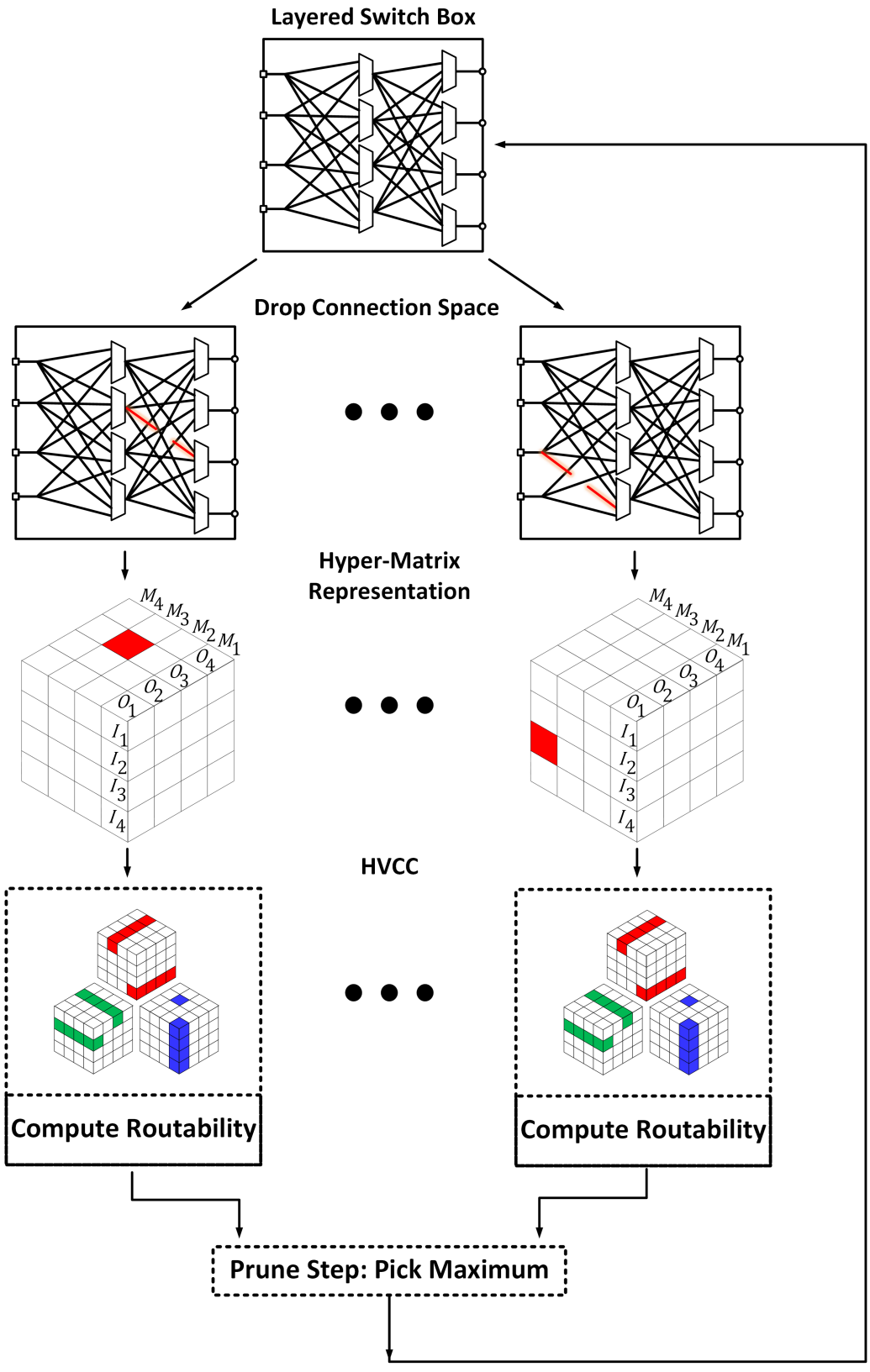


Figure 2.27: Switchbox design space exploration using HVCC based pruning method. The pruning process creates a trajectory in the *switchbox-configuration* space.

to a unique local direction of movement in the *switchbox-configuration* space (a high dimensional discrete space that consists of all possible switchbox configurations).

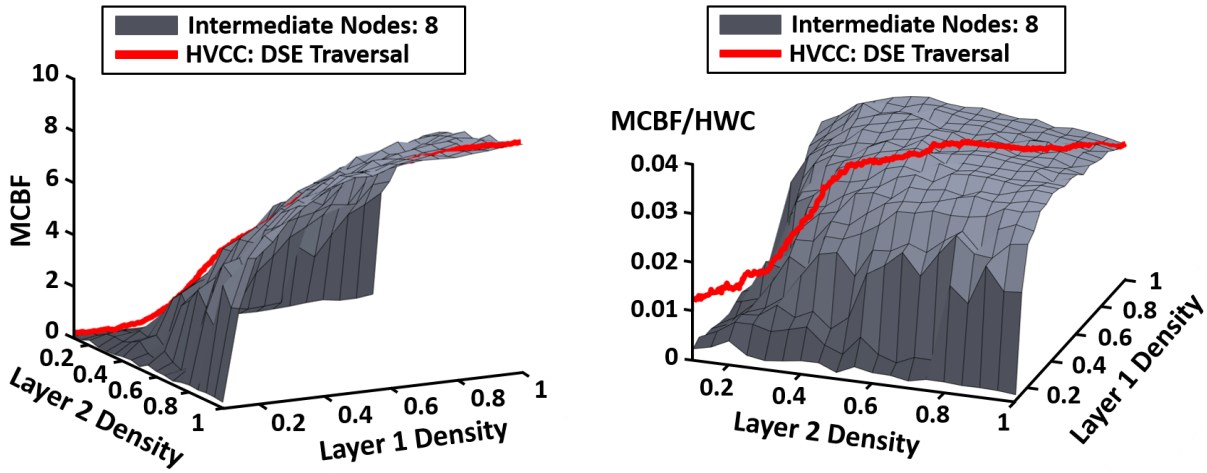
- From all possible routability scores the best one is chosen, and the original switchbox pruned accordingly. This is akin to moving in the locally optimal direction in the *switchbox-configuration* space.
- The pruning process is continued till the switchbox is empty.

The HVCC based pruning process creates a trajectory in the *switchbox-configuration* space. At each point in the trajectory, the underlying **hardware cost** as well as the **MCBF** experimental evaluation metric can be calculated. Consequently the hardware efficiency (MCBF/HWC) can be calculated at every point in the trajectory. An optimal trajectory of the *switchbox-configuration* space can then be defined as the one containing the maximum hardware efficiency point. It should be noted that there is no guarantee the *switchbox-configuration* space trajectory calculated by starting from a fully connected switchbox is globally optimal because the algorithm above is a discrete optimization heuristic for an underlying integer liner problem. Techniques such as random initialization can give closer to optimal trajectories, but are not explored as part of this dissertation and are left for future work.

2.4.5 Proposed SB Design and Evaluation

The HVCC based DSE methodology is evaluated using MCBF and MCBF/HWC experimental metrics (from Figure 2.24 and 2.25) against baseline brute-force exploration of switchbox architectures. For an example 3-layer switchbox with 22-8-22 configuration (of nodes-per-layer), the brute-force baseline MCBF is plotted in Figure 2.28a. It can be observed that the

HVCC based traversal trajectory nearly mimics (and occasionally exceeds) the best of the brute-force exploration surface. Assuming a reasonably large (and representative) portion of the search space is sampled by brute-force exploration, this plot shows that HVCC can arrive at similar results in a systematic way in polynomial time $O(N \times S_K)$ making it a faster way to do DSE. Figure 2.28b shows the silicon or hardware efficiency metric for the same designs (orientation rotated for better visibility). It can be observed that the same HVCC based traversal results in a better MCBF/HWC for all connection sparsities that fall under its trajectory. Intuitively, this shows that de-correlating the paths in a switchbox results in better route mapping utilization of the underlying silicon which gives better silicon efficiency.



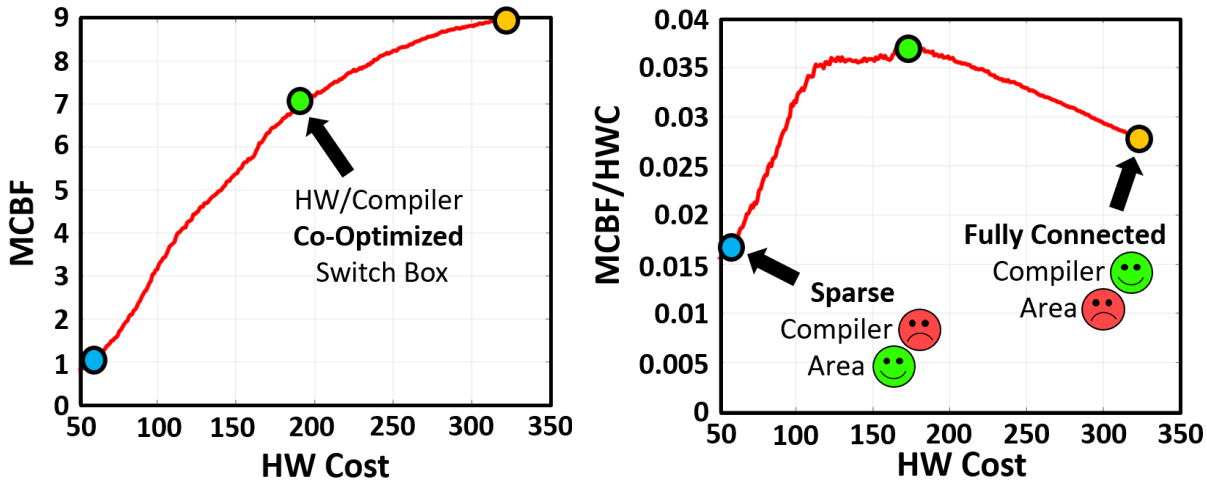
(a) MCBF comparison.

(b) MCBF per hardware cost comparison.

Figure 2.28: HVCC based DSE vs. brute-force exploration on an example 3-layered switchbox.

In order to further understand the implications of this result from a hardware design perspective, the 2-dimensional density axes (one for each local layer pair), in Figures 2.24 2.25 and 2.28 are compressed into one number of 'hardware cost'. The compression can be done for an N-layer switchbox as well, with (N-1)-dimensional density axes. This allows easy 2-D visualization (of the compressed form of these axes in terms) of total hardware cost against MCBF (experimental routability) and MCBF/HWC (experimental silicon efficiency).

For the HVCC traversal of the 22-8-22 switchbox example (from Figure 2.28), the MCBF and MCBF/HWC metrics are plotted against (compressed) hardware cost in Figure 2.29a and Figure 2.29b respectively.



(a) MCBF comparison.

(b) MCBF per hardware cost comparison.

Figure 2.29: Experimental routability and experimental silicon efficiency vs. silicon hardware cost for an example 3-layered switchbox.

The orange point (dot) to the right in each figure represents the maximum hardware cost, indicating the underlying multi-layer switchbox is fully connected. Intuitively, such an architecture will have quick compile-times as the compiler will always be able to find a path between two I/Os. In contrast the blue point (dot) on the left of each graph represents a very sparse multi-layer switchbox which intuitively has a small silicon area, at the cost of greater compile-times and reduced routability as indicated by the low MCBF. The compiler will have a hard time trying to map disjoint I/O routes through a small number of switch resources. The HVCC trajectory gives a way to traverse between the two extremes in a near optimal way that minimizes path correlations. At the the peak of the MCBF/HWC graph (green point), the silicon efficiency is maximized. Intuitively, at this architecture point, the marginal reduction in compile effort for the marginal increase in silicon area in maximized (for this trajectory); the implication being that the switchbox is *Hardware-Compiler* co-optimized.

From an analysis perspective, the co-optimized architecture would, at first sight, be the ideal design point. However, it should be noted that the optimization is over MCBF, which is a 1-dimensional compressed experimental representation of the routability and by extension compile-times. A more expansive test should take into account the variance of the compile-time as well, because in a real compile scenario, the worst case routing time will limit the overall compile-time. MCBF is not sufficient of a representation to take into account that variance. A better representation is a direct plot of the fraction of 'concurrent random I/O route pairs' that are mappable through a switchbox architecture vs. the 'I/O pair set-size'. Such a representation of a switchbox would intuitively show that for a randomly selected I/O route set, *what is the the probability that the compiler will be able to satisfy the set?* A higher probability would not only imply lower compile-times but would also give a measure of compile-time variance. Since the plot is against 'I/O pair set-size', the effort of the compiler can be fine tuned for the maximum set size of interest. As an example of such a plot, the proposed UDSP's layer-1 switchbox is plotted in Figure 2.30.

For the UDSP, set sizes of I/O pairs rarely exceed 8 due to the core I/O count being 8. This means that I/O-pair-set sizes greater than 8 do not require high route mapping probability. Figure 2.30 shows that for up to 8 concurrent I/O routes the mapping probability is >95%. Intuitively, getting higher set routing probabilities amounts to de-correlating the underlying paths (making them more disjoint so a larger set is mappable). This is an important result, because it directly ties the minimization of HVCC (measuring the disjoint nature of paths) to the probability that the compiler would be successful in mapping path-sets. It signifies that the *switchbox-configuration* space trajectory plotted earlier is, by definition, maximizing the resultant I/O mapping probability. The last remaining step in the DSE process is to tune the switchbox selection point (from the HVCC trajectory) that maximizes the desired I/O set's mapping probability as shown in Figure 2.30, as opposed to

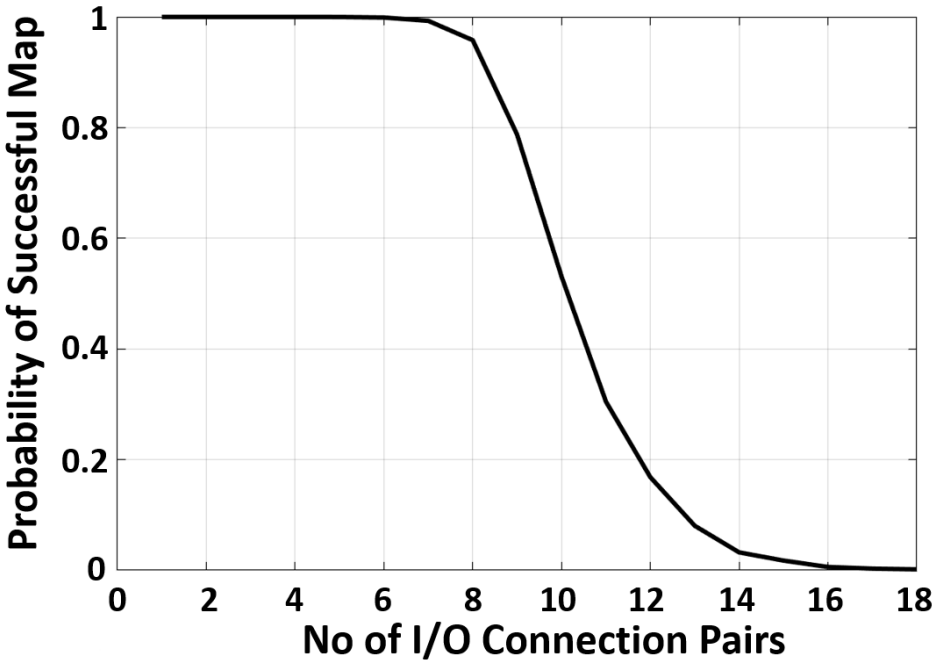


Figure 2.30: Connection mapping probability for varying I/O set size of randomly selected I/O pairs.

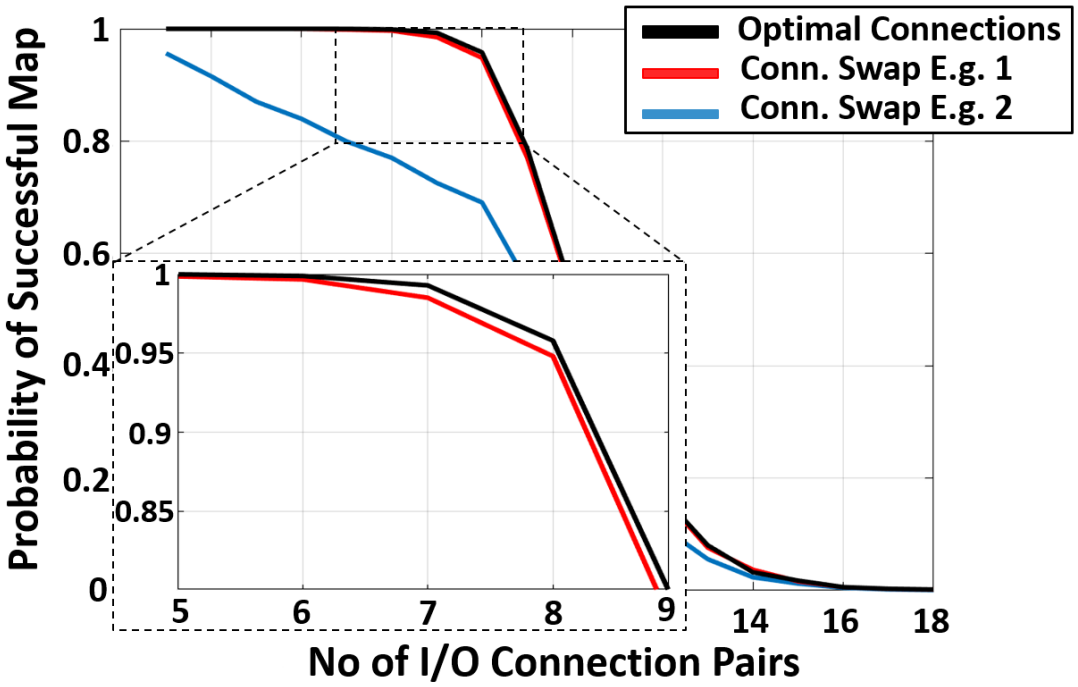


Figure 2.31: Connection mapping probability sensitivity with connection configuration.

choosing the maximum MCBF/HWC point from Figure 2.29b. For the UDSP the desired mapping set size is 8. To illustrate the optimality of this design point in terms of routing probabilities, 1 and 2 connections are swapped from the underlying switchbox respectively, with the resultant switchbox's routing probabilities shown in Figure 2.31. With 1 changed connection the routing probability drops slightly, however with 2 changed connections, the probability drops significantly. The design is sensitive to slight changes in architecture (given constant area), indicating that the DSE is indeed finding a near optimal 8 I/O set point.

CHAPTER 3

Streaming Near Range 10µm: A 10-µm Pitch, 0.38pJ per bit, Inter-dielet I/O and Communication Protocol for Multi-chip Module Scaling

The UDSP interconnect is efficient, flexible and scalable, allowing it to scale to a large number of cores on a single silicon chip. However it is not economical to fabricate large chips due to higher silicon area cost. The cost rise is not proportional to silicon area because of the defects that occur during the manufacturing process. Spot defects that dominate mature processes are random and hard to get rid off [45]. This leads to larger chips having lower yield, the cost of which is eventually passed on to the lesser quantity of fully functional chips, and ultimately to the consumer. It is therefore more cost effective to build smaller chips, as larger chips have exponential rise in cost w.r.t. silicon area. An effective way to economically increase the core count with linearly increasing costs is to employ multi-chip module (MCM) scaling, where multiple smaller chips are packaged together to create a larger system.

In addition to higher yields and cost benefits, MCM scaling also allows for IP reuse by utilizing the same dielet, either multiple times in a single SoC, or across different SoCs. MCM integration allows for multi-technology dielets to be integrated in a single package. The allowance for MCMs for such flexible heterogeneous integration results in quicker time-to-market and lower SoC IP-design costs translating into lower product costs for consumers.

3.1 Prior Work on MCM Scaling

IBM had initially been at the fore-front of MCM design for decades. With initial exploration and design in ceramic based MCMs [46] to commercial and widely successful products like 4381 MCMs [47], IBM laid down many of the foundations of package integration technologies. However, in the decades that followed, the room for aggressive transistor scaling perpetuated monolithic designs as easier and cost effective solutions causing MCM integration to be mostly neglected except for select high-performance servers and academic research. More recently, with the rising costs of advanced-node monolithic designs, and future uncertainties in the pace of Moore's law, the market has seen a resurgent and growing interest in MCM integration. Platforms like Intel EMIB, Intel Foveros, TSMC InFO, and TSMC CoWoS have emerged as leading contenders for chip integration and die-to-die communication.

Intel's approach with EMIB (Figure 3.1) is to embed small standardized passive silicon based interconnect layers between dielet boundaries [30]. The dielets use the open-source Advanced Interface Bus (AIB) communication protocol to communicate between dies [48]. The AIB/EMIB interface has 55-µm bump pitch. The EMIB die is embedded inside an organic package which handles the longer distance connections as well as power delivery to the dies. EMIB uses solder joints from the dielet to the bridge die. TSMC uses the CoWoS

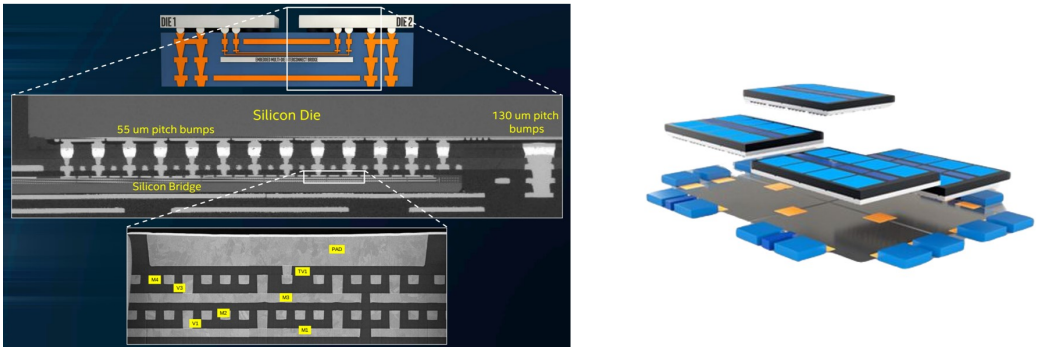


Figure 3.1: Intel EMIB platform [30].

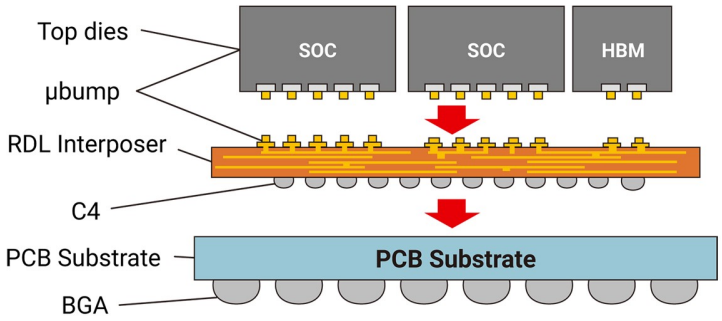
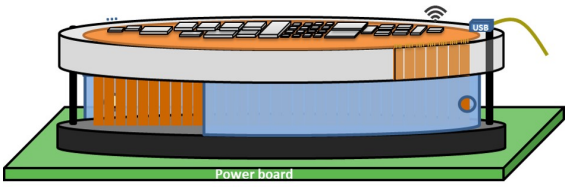


Figure 3.2: TSMC CoWoS-R platform [31].

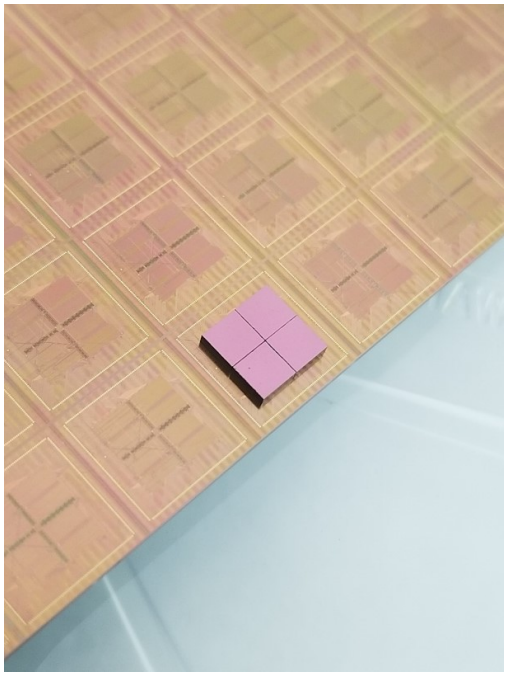
family of technologies [31, 32], which mainly comprises of a redistribution layer (RDL) (with support for embedded decoupling capacitors) assembled on top of the dielets as seen in Figure 3.2. The RDL is made from silicon and can sport a very high number of internal copper layers leading to high cross-sectional bandwidth. It carries power as well as local silicon interconnects (LSIs) between dies. CoWoS also uses solder joints to connect dielets to the RDL and has a 40-μm bump pitch and its implementations use TSMC's proprietary LIPINCON communication protocol to communicate between dies. Though modern MCMs show incredible leaps in integration, their larger solder-based bump pitch requires them to compromise on either bandwidth, energy-efficiency, or number of copper layers in the interposer leading to higher cost.

3.2 Silicon Interconnect Fabric

The Silicon Interconnect Fabric (Si-IF) [33] is a set of technology solutions surrounding a 10-μm pitch wafer-scale silicon interposer developed at UCLA. It achieves an aggressive integration bump pitch buy direct copper-to-copper thermal compression bonding (TCB). The fine-pitch of the Si-IF allows for high cross-sectional bandwidth support without compromising on efficiency or increasing the number of layers (and cost). The idea is to replace the organic packages, and PCBs which traditionally bridge memory and multiple forms of



(a) The Si-IF concept.



(b) An Si-IF wafer with a 2×2 UDSP assembly.

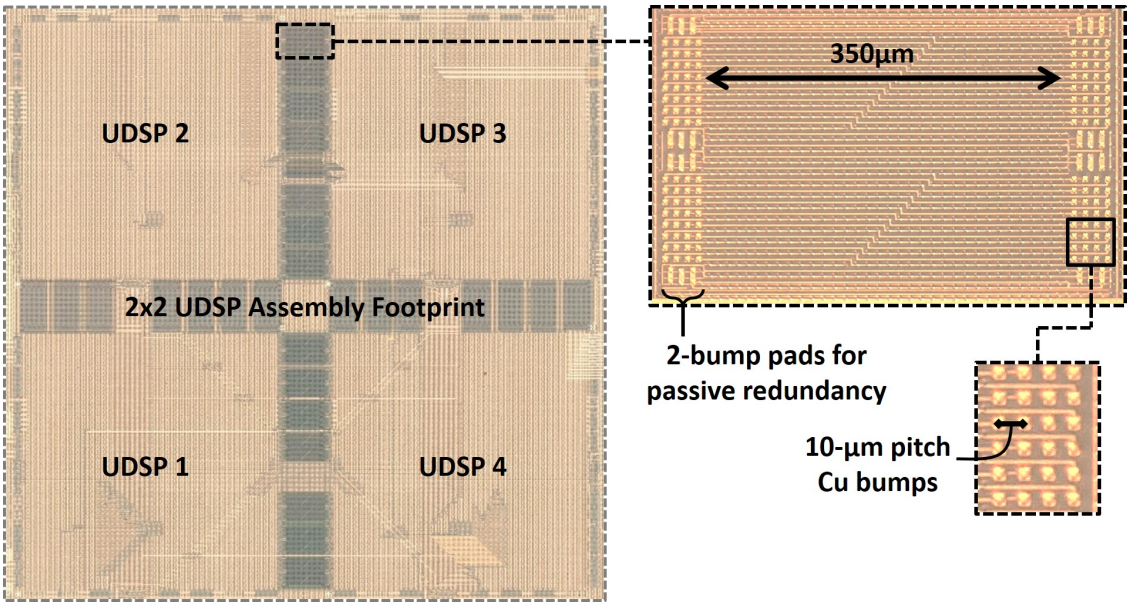
Figure 3.3: The 10-µm pitch, Silicon Interconnect Fabric paradigm.

compute (CPU, GPU, TPU, etc.) with a single large wafer-level fan-out layer, which sports power delivery, cooling, embedded decaps and network of wafer. The Si-IF is capable of heterogeneous integration as shown in the concept Figure 3.3a, with more detail provided in [49]. Figure 3.3b shows a 2×2 UDSP assembly on an Si-IF wafer (this work).

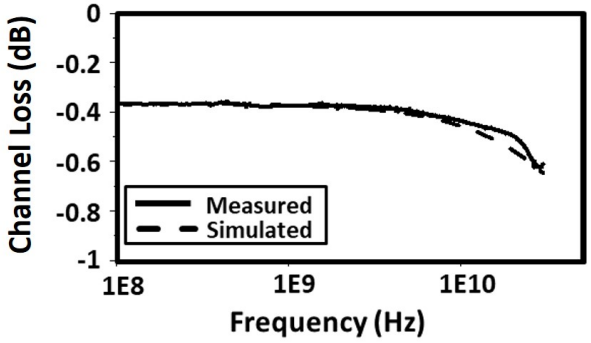
3.2.1 Si-IF Design Characteristics

Figure 3.4a shows the footprint of the 2×2 UDSP dielet assembly on the Si-IF. The imprint of the 4 dielets and their inter-die communication links can be seen. The dielets are to be placed in close proximity of each other, with a link length of 350µm. The links are bundled in a channel. Data links have 10-µm pitch bumps and corresponding 10-µm pitch pad terminations on the dielets. Critical control and clock links however have passive 2-bump redundancy. The two 10µm bumps are shorted on the Si-IF. Figure 3.4b shows the simulated

and measured response of the passive link on the Si-IF [49]. It is observed that due to the short channel length, the link has very low insertion loss even at high frequencies (> 10 GHz). The short length also contributes to lower signal round trip time (< 2ps) translating to negligible self inter-symbol interference (ISI) and cross inter-channel interference (ICI) at frequencies of interest (0.5GHz - 4GHz). This implies link drivers can be small and power efficient, without any need for pre-emphasis, and the link receivers can be simple inverters and need not have equalizers.



(a) 2×2 UDSP assembly footprint on the Si-IF with a link zoomed in.



(b) Frequency characteristics of Si-IF links [49].

Figure 3.4: The 10-µm pitch Si-IF characteristics for channel design.

3.2.2 Requirements and Challenges in Si-IF Link Design

Since the Si-IF sports ultra-fine-pitch interconnects, the main design challenge is the density of the I/Os that connect to those interconnects. The high density of I/Os is not only an active transistor density problem, it is also difficult to manufacture pads on the top metal layer that conform to sizes of 7×7 µm without violating DRC rules. In order to solve the latter, a wafer-pull after top copper manufacture is done, and coated with a film of thin nitride to prevent oxidation. The top copper metal layer, in most processes, can handle much finer details, such as 7µm dimensions, unlike the redistribution layer (RDL). Figure 3.5 shows the relative silicon area available per I/O while using the Si-IF vs. while using state-of-art interposer protocols such as [50] and MCM protocols such as [51]. There is a 74.2× per-I/O-area gap requirement compared to [51] and a 3.6× area reduction requirement compared to [50]. In addition to the smaller area, the communication protocol is required to support data rates of up to 1.1Gbps per pin conforming to the maximum speed of the UDSP to allow for seamless communication across dielet boundaries. In addition, due to the new nature of the process, the protocol is required to have redundancy and repair support to improve packaging yield and recover from random one-off manufacturing defects during TCB. ESD is another main concern and the internal dielet to dielet I/O's are required to have adequate resilience. With two layers of the Si-IF available, each supporting 1.5µm wire features, the link should support maximum utility of the layers, making the pad depth count to 4 (2 wires per layer). Placing all these features inside a tight area means silicon is at a premium in the design of the link. The resultant communication IP block needs to have self contained active and pad area.

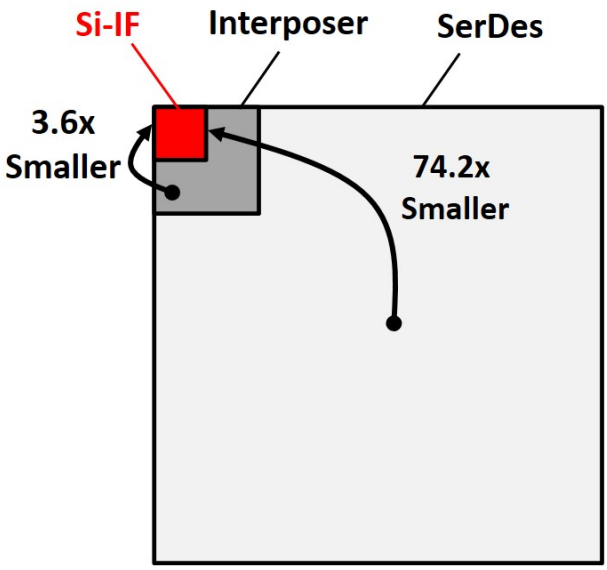


Figure 3.5: Relative area-per-I/O comparison.

3.3 Proposed Streaming Near Range 10µm Protocol Architecture

In order to meet the challenging requirements of designing a fine-pitch I/O for the Si-IF, the Streaming Near Range -10µm (SNR-10) protocol is proposed that focuses on minimizing the area per I/O. Several trade-offs are made to shrink the size of each I/O including unidirectional I/Os over bidirectional I/Os, a clock forwarded architecture over clock data recovery, a common clock correction scheme, a minimal handshake at boot, FIFO based clock domain transfer, a minimal redundancy repair mechanic, amortization of common resources by using a 64 bits per channel, using a lower amount of electro-static discharge (ESD) protection, and having support for a limited number of modes, namely synchronous and asynchronous modes. By optimizing each feature for area, the total I/O area is bought down to fit inside the physical boundaries of the footprint of the channel. The SNR-10 channel is a physical (PHY) layer protocol that is built to communicate data between dielets on the Si-IF and other such fine-pitch interposer technologies. It is built to be fully synthesizable for quick portability between technologies. The protocol is built to communicate with its counterpart

channel across heterogeneous technology dielet deployments. The clock speed of the channel is determined by the host, and the maximum speed is determined by the technology.

3.3.1 Interface Specifications

The channel is designed to have minimal interface from the perspective of chip as well as minimal pad overhead from the perspective of the fine-pitch interposer such as the Si-IF. From the side of the chip, the interface pins along with their functions is detailed in Table 3.1. The chip-side interface consists of two 16-bit data inputs and two 16-bit data outputs from the channel. The direction of the data inputs and outputs is not configurable as in other protocols like AIB [48]. The depth of the circular FIFO is programmable, as is the mode of the channel. It is also expected that the host chip is responsible for soft reset as well as reset signals. The active low reset is typically set high after a post stable power on cycle. The soft reset is active high and is used to halt and reset the movement of data paths. It does not affect the state registers. This is useful in flushing out the channel in case of a cross-chip algorithm change. The channel can indicate to the host chip if it has successfully locked to an adjacent channel before the host chip decides to send data. The channel lock does not engage if the hard faults are more than what the built-in redundancy-repair can handle.

Table 3.2 shows the Si-IF side interface pins along with their functions. The Si-IF side contains 4 control pins (two Tx and two Rx) to communicate channel states across chip boundaries to the adjacent channel on the other chip. Since the architecture is a clock forwarded one, 4 differential clock pins are also present, two for transmit and two for receive. The majority of the pad area is dedicated to regular data movement as this keeps the overhead of the channel area small. Figure 3.6 shows the standard as well as the uniform

Table 3.1: SNR-10 chip-side interface specification.

Chip side pins	Function
Mode	Synchronous mode with clocked data or Asynchronous mode with digital pass-through
Reset	Active low reset to reset the state machine as well as the data pins of the channel
Soft Reset	Active high reset that only resets data paths
Data 1 CI [0:15]	16-bit chip side input data stream 1
Data 2 CI [0:15]	16-bit chip side input data stream 2
Clk	Clock pin
FIFO depth [0:2]	4-position programmable circular FIFO depth
Control In	JTAG style input pin to reset redundancy-repair or override its registers
Control Out	JTAG style output pin to read out state register values
Data 1 CO [0:15]	16-bit chip side output data stream 1
Data 2 CO [0:15]	16-bit chip side output data stream 2
Channel Locked	A bit to signify the successful completion of the handshake process

Table 3.2: SNR-10 Si-IF-side interface specification.

Si-IF side pins	Function
Control Rx [0:1]	Control and redundancy-repair pads input from Si-IF
Data 1 Rx [0:15]	16-bit Si-IF side input data stream 1
Data 2 Rx [0:15]	16-bit Si-IF side input data stream 2
Clk Rx [0:1]	Differential clock input
Clk Tx [0:1]	Differential clock output
Data 1 Tx [0:15]	16-bit Si-IF side output data stream 1
Data 2 Tx [0:15]	16-bit Si-IF side output data stream 2
Control Tx [0:1]	Control and redundancy-repair pads output from Si-IF

pad layout for the SNR-10 channel. The data pads on both layouts have 10-µm pitch, and both layouts measure 37µm × 237µm. The actual pad size is 7µm × 7µm and the Si-IF bump diameter is 5µm. This gives a 3µm inter-pad distance and a 1µm bump alignment margin. The inter-channel distance specification is set to a maximum of 500µm due to the fine integration capability of the Si-IF bonding process [49].

The difference between the two pads is the amount of passive redundancy available for the control pads as well the clock pads. The uniform pad layout has 4-bump passive redundancy and 2-pin logical redundancy, for both control and clock. The standard pad layout has 2-bump passive redundancy. Passive redundancy on the clock and control pads is required as these are considered critical for the operation of the handshake protocol. Passive redundancy requires 2 or more bumps to be connected to the same logical pad to make the channel immune to the case of one of them having a manufacturing defect. The standard pad layout is used in the UDSP dielet due to better short channel (350µm) yields. For packaging technologies with worse yields the uniform pad layout is recommended.

3.3.2 Unidirectional I/Os

The SNR-10 implements a minimalist I/O design to reduce area and fit as many I/O's as possible. Each I/O is unidirectional with the channel containing an equal number of Rx and Tx I/Os arranged in vertical columns. A single connected I/O is shown in Figure 3.7. There are two modes of operation, asynchronous, where the data is passed as feed through, and synchronous, where the channel operates in a clocked single data rate (SDR) mode. Figure 3.7 highlights the SDR mode data path. The Tx pad terminates in a large buffer driver sufficient to drive the signals in the particular technology. Several buffer sizes were experimented with in TSMC 16nm ULVT and GF 22nm SLVT technologies, and to keep

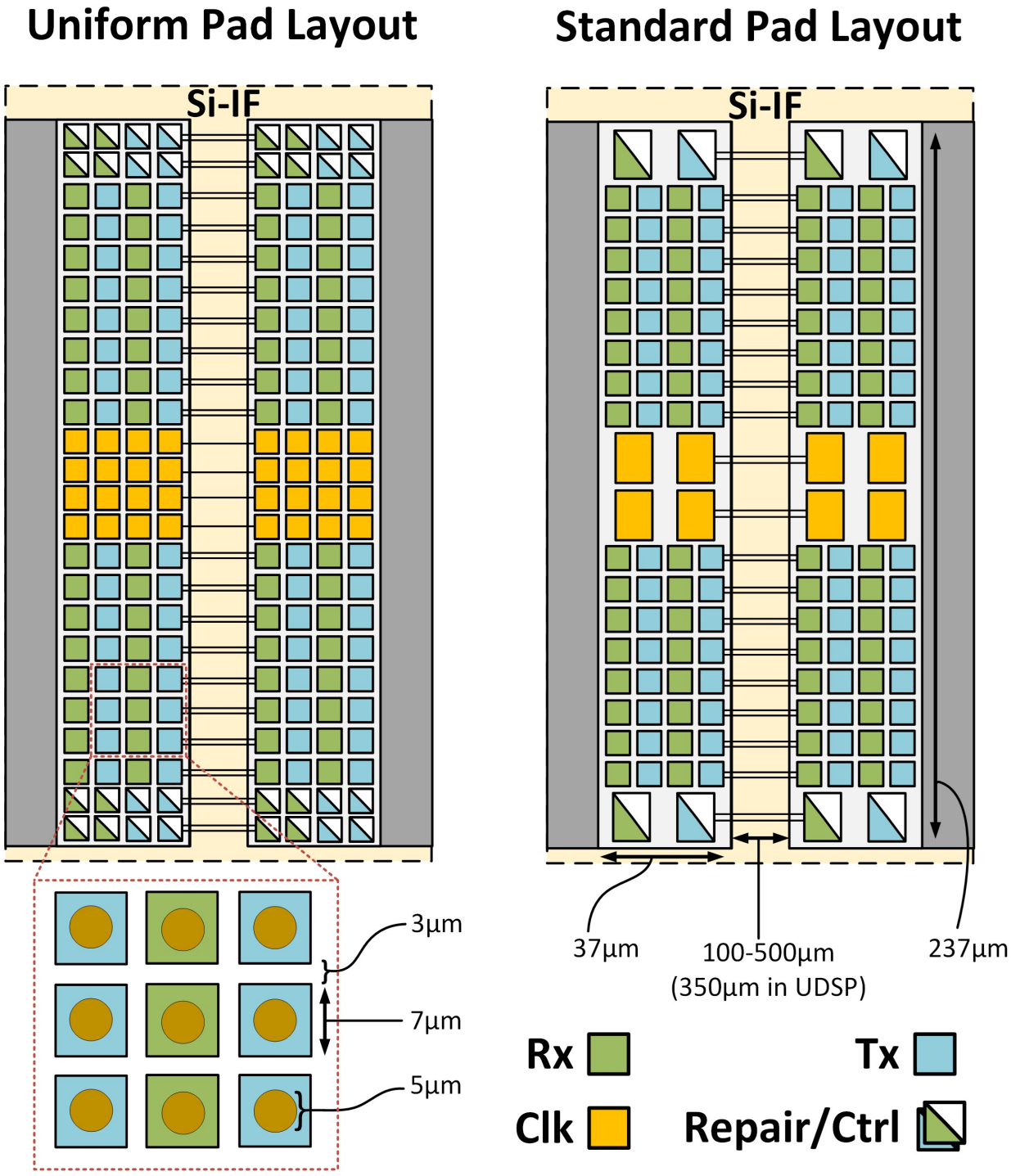


Figure 3.6: SNR-10 channel bump pad layout options.

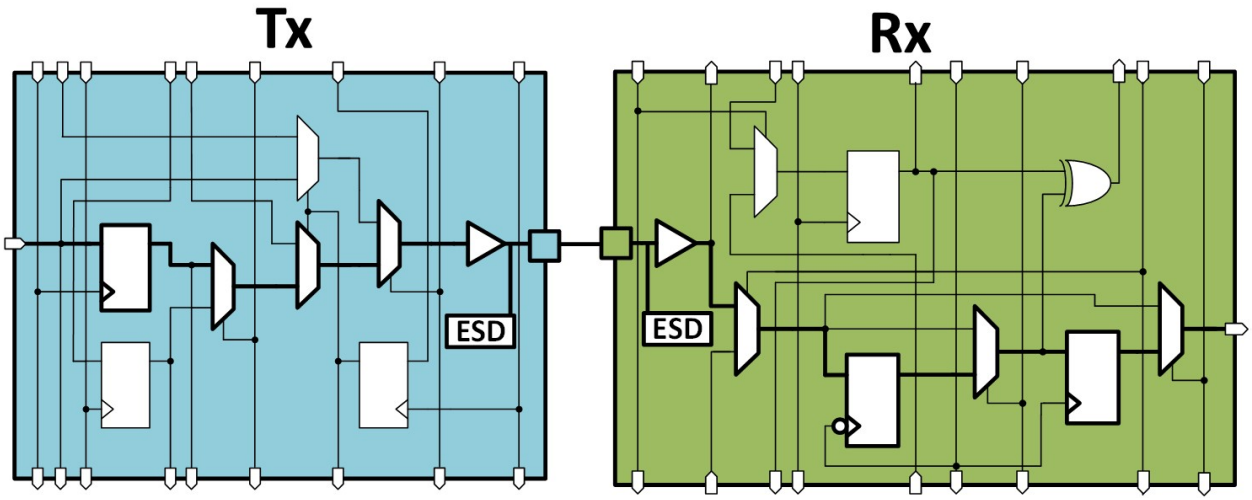


Figure 3.7: A unidirectional SNR-10 I/O pair.

the I/O synthesizable, only buffers from standard cell libraries were chosen. It was found that the 16× and 20× buffer sizes in the respective technologies were sufficient to drive the load of the Si-IF connection reliably. For the UDSP implementation, an over designed buffer of 64× was chosen to allow for different inter-channel distances to be tested during assembly. Lowering the buffer size has the potential to reduce run-time power and recover a small amount of area. Clock correction and repair mechanisms are amortized and shared amongst the I/Os as are the control mechanisms. This makes the incremental cost of adding additional I/Os lower which makes the area per I/O lower as well.

3.3.3 Initialization and Handshake Protocol

Two SNR-10 channels across heterogeneous dielets must be able to establish a communication channel along with verification for their respective dielets that the channel is working properly. SNR-10 achieves this via an initial handshake process that is performed once per boot up cycle as shown in Figure 3.8. This is because certain aspects of the channel are assumed to be constant throughout the post boot up life-cycle of the channel. Among them

are the broken and repaired pins, that are caused by physical defects, either through ESD events while manufacturing and thermal compression bonding (TCB), or by chip Rx and Tx defects caused during manufacturing. Bumps on the Si-IF may not be level due to process variation which may cause them to randomly be disconnected from the dielet during TCB. The synchronous or asynchronous mode of the channel also needs to be verified as well as shared across the two dielets, and is not bound to change during the course of the channel deployment. Lastly the clock domain transfer, local clock drift, and sampling edge selection also needs a one-time adjustment as it is also not expected to vary over the use of the channel as the channel size is small and the internal clock tree drift of the receiver side is small as well. If the clock is expected to vary over long periods of time (in a certain technology implementation), the channel states need to be reset in order to perform the handshake protocol again.

The unidirectional nature of the channel simplifies the handshake and no predetermined master and slave process is needed. From Figure 3.8, after channel powers on, it measures and adjusts the local and received clock synchronization and which edge to sample using a phase detector and adequate delay based sampling margins. This also establishes presence. If presence is not detected then the local clock is used to sample till a presence is detected. The local state of the channel (redundancy and FIFO) is reset and a connection request is sent through the dedicated respective side control pads. The connection request is acknowledged by the Rx and its internal state is reset as well. The local Tx channel then activates its redundancy-repair mechanic which is described in more detail in the following section. After the completion of the repair, and sharing of repair data, the appropriate adjustments are made on either side of the channel. The mode from the Tx end is selected and conveyed to the Rx and confirmation of adjustment made is received. Notice that even for asynchronous mode, the channel must first synchronize in synchronous mode if the redundancy repair

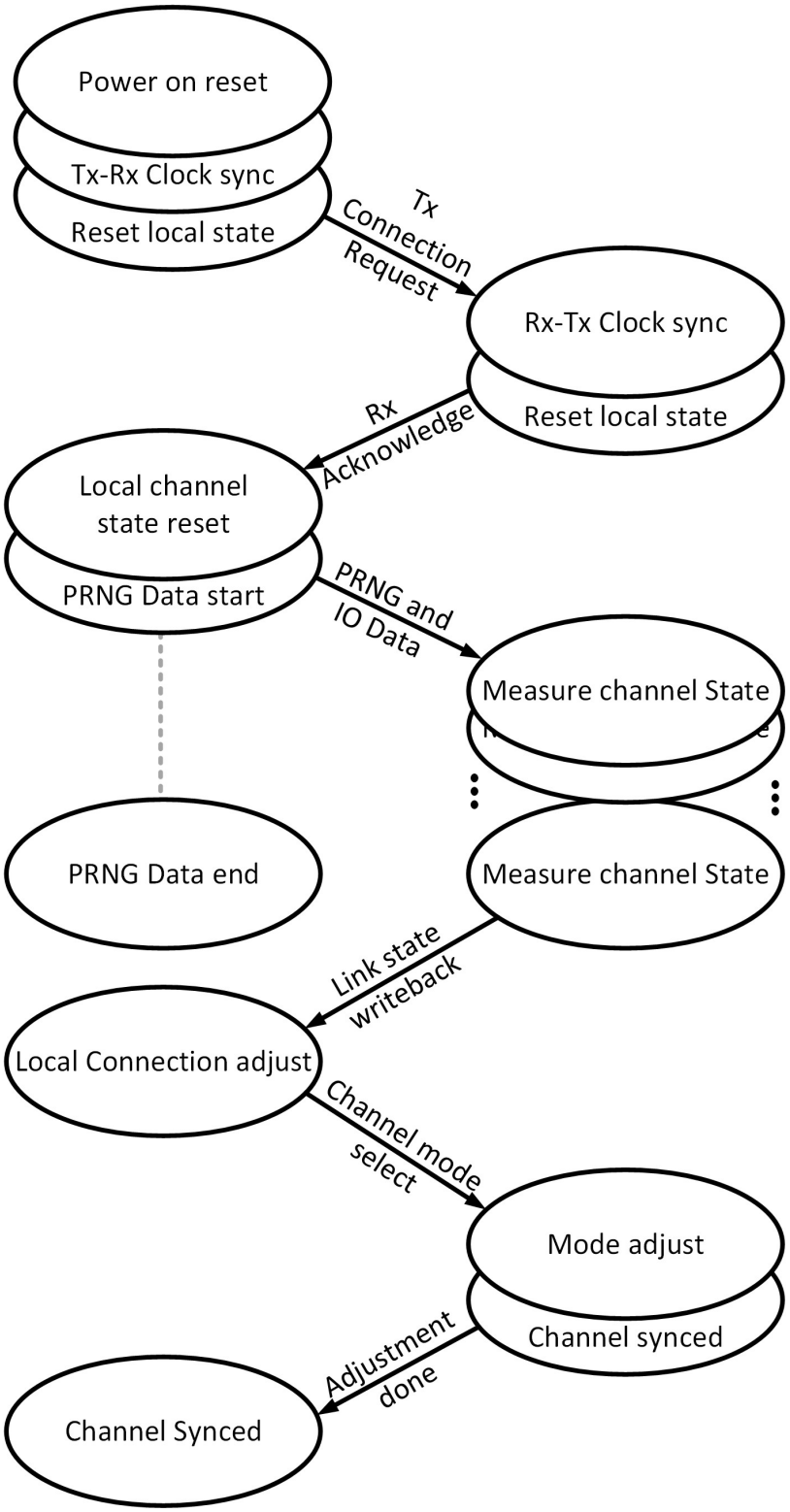


Figure 3.8: Handshake protocol.

functionality is to be utilized. If the channel is used without clocks in pure asynchronous mode, then the state registers need to be manually overwritten from the chip side as the channel cannot check for connection errors. Finally, the channels are synced and the 'Channel Locked' bit goes high if all channels successfully complete the handshake. If, however, the repair reports more errors that can be handled, the channel does not reach the locked state. The state data can be pulled out via the chip side control interface to check the particular errors in more detail. Using the same interface the channel states can be forced into a certain position to allow for debug.

3.3.4 Redundancy and Repair Procedure

Since, at the time of design, the Si-IF is a new, advanced, fine-pitch packaging process, the yield of the short channels is unknown. With SNR-10 being the first demonstration of a working communication channel on the Si-IF, it is prudent to assume and account for defects in the manufacture. Such defects can take on the form of logical defects in the I/O mainly due to ESD and physical defects in the interposer wires due to layer shorts, disconnects or open wires. Large, passive redundancy pads that connect to two bumps and two wires on the Si-IF are chosen for critical signals like clock and control data. To take full advantage of fine-pitch, and to reduce the area per I/O to increase bandwidth density, it is essential to not have passive or logical redundancy of each I/O individually. Therefore for the most numerous I/Os, namely the data I/Os, there is a redundancy and repair mechanic built into the initial handshake process.

During the repair phase, the transmitter and receiver share a singular source (Tx side) of a pseudo-random number generator (PRNG) built into each channel. The PRNG sends data to either side in a scan chain fashion as highlighted in Figure 3.9, where the receiver

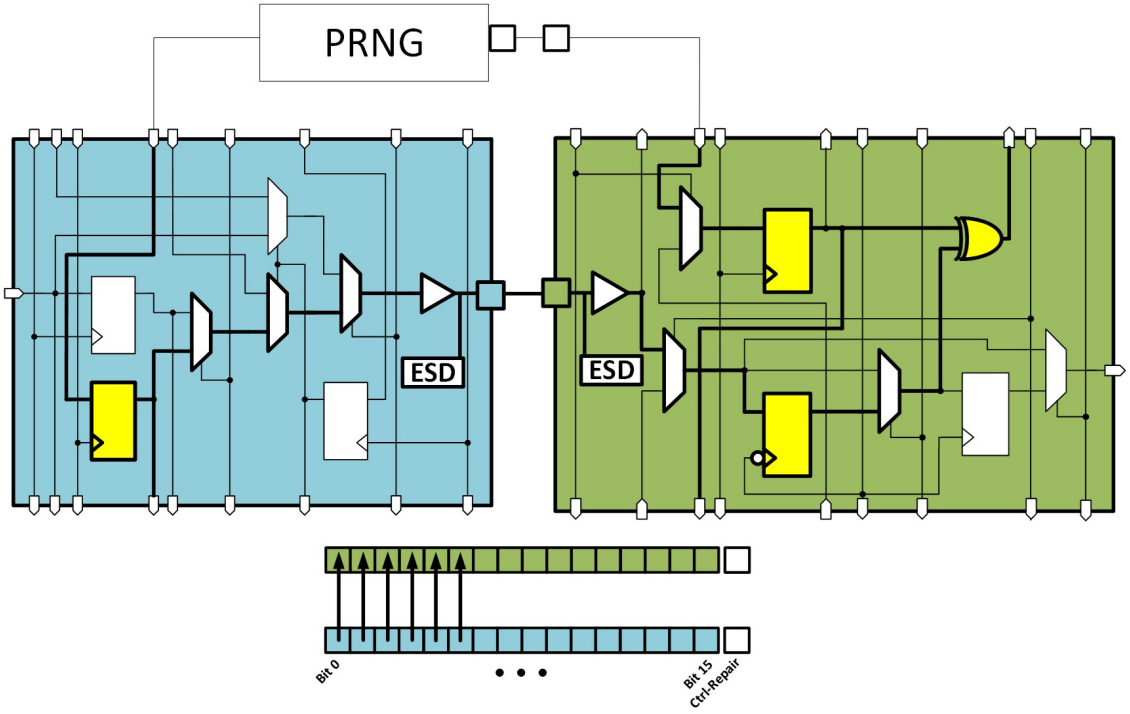


Figure 3.9: Redundancy and repair - check phase.

I/O registers the incoming data from the transmitter as well as directly from the scan chain. The PRNG along with the connected scan chain ensures that the data sent and received is spatially and temporally decorrelated. This allows for the detection of adjacent bit-short (spatial) errors and stuck-at-1 or stuck-at-0 (temporal) errors, among others. An exclusive OR gate on the Rx side is used to compare the received data to the ground truth, and if a mismatch is detected, as in Figure 3.10, the particular bit is recorded in a local state register on the Rx side. Care is taken that the circuit resources required per I/O are minimal to minimize area, with the majority of area being occupied by the flip-flops.

Once the entire process completes, lasting 80 clock cycles, the final link state is processed on the Rx side, whereupon detection of the first error in the row, from the MSB to the LSB, shifts all the remaining bits towards the LSB side by one, as shown in Figure 3.11. This is done only for the first error, giving a logical redundancy of 1 per 16 bits for this design. The last bit is scheduled to be rerouted through the control pad, which doubles as

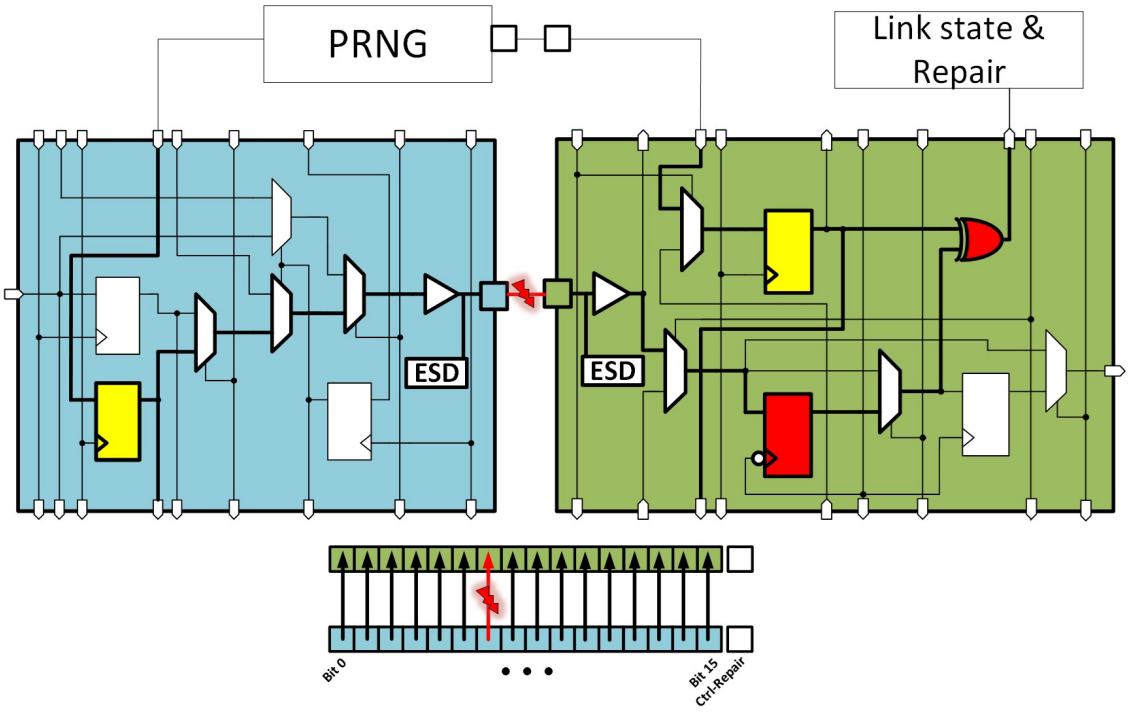


Figure 3.10: Redundancy and repair - detect phase.

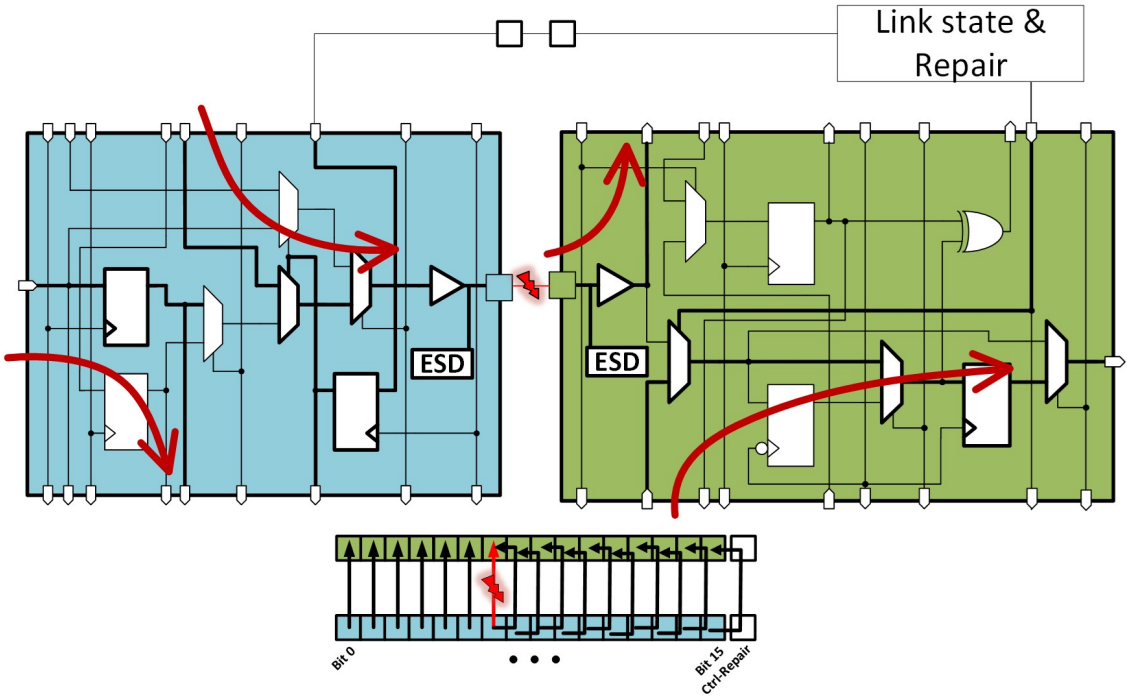


Figure 3.11: Redundancy and repair - repair phase.

a redundancy pad, once the handshake process is completed. Vitally important is that the repaired pins experience the same data delay as the regular pins so that the receiver latch timing is not affected, especially at high-speed. This is also exemplified in the judicious use and order of the muxes and flip-flops, accounting for the rerouting time on longer connection lines in the channel. There are only two pairs of control (redundancy) pads per channel which means only one of the 16-bit lanes can use it at a given time. Preference is given to Data-1 lanes over Data-2. Once the Rx side is adjusted, the repair data is sent back to the Tx side to adequately adjust the Tx pads as well, replicating the Rx adjustments, as shown in Figure 3.11. The dual use of the control pins as redundancy pins avoids extra channel area contingent on the fact that control pins are not used during regular operation.

3.3.5 ESD Protection and Mitigation

From Figure 3.7 in the Tx pad, minimal additional ESD is used as the large driver transistor's reverse body diodes are expected to sink any expected ESD currents that may arise from the process. The Rx pad, on the other hand, requires a larger ESD protection diode. Several smaller standard cell diodes are used in parallel on that node to mitigate ESD risks. The process of thermal compression bonding (TCB) also runs the risk of causing ESD. To reduce this risk special precautions are taken with the ionizer and cleaning fluids to keep the potential of the pumps and Si-IF neutral. Post TCB cleaning using plasma etching is also skipped in order to ensure an ESD charge doesn't build up on the longer wires such as those running to the wire bond pads. Skipping this step requires gold to be pre-deposited onto the copper wire-bonding pads on the periphery which adds an extra step to the process. The act of wire-bonding to the PCB can also cause ESD. To mitigate this risk, the wire-bonding is performed on a special PCB socket that shorts all the pins to ground.

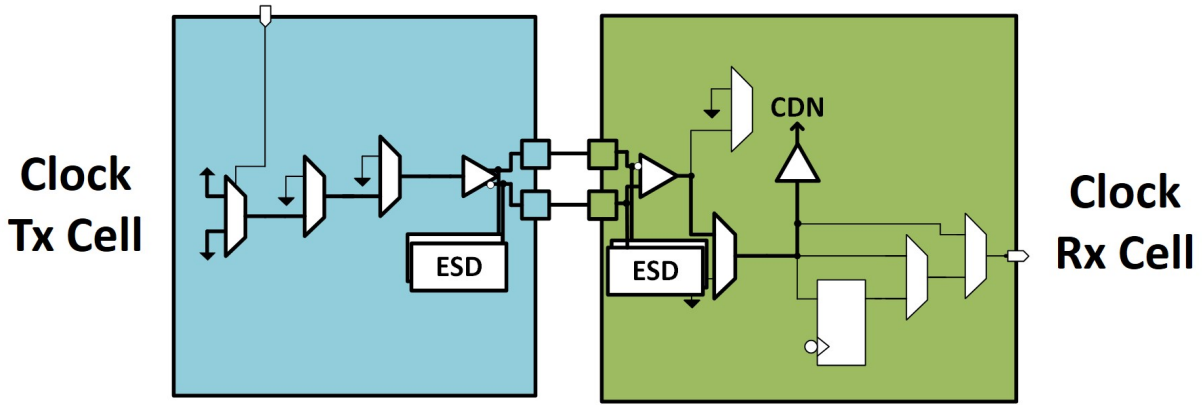


Figure 3.12: Clock path in SNR-10.

The ESD mitigation mechanisms are tested using two PCB design runs, one using the precautions and the other without. It is found that the inner channels (the ones connected to a $\leq$ 500µm adjacent channel) are free from ESD regardless of technique used and technology used, however external channels (the ones with longer routes and connected to the external world using wire-bonding pads) are highly sensitive to the protection techniques employed. In TSMC 16nm, without any protection the failure rate of the Tx pads is found to be 15% and that for the Rx pads to be 44%. In GF 22nm, the rates drop to 0% and 3% respectively. An ESD event failure comprises of a bit either stuck at 0 or 1, or being inverted (indicating a gate punch-though). With the protection and mitigation techniques employed, all rates across both technologies drop to 0% (tested with 7,168 links).

3.3.6 Clock Correction and CDT

In order for the data to correctly be sampled by the receiver (Rx) in a clock forwarded architecture, the clock provided by the sender (Tx) should be aligned to the center of the data eye. For a single data rate channel, assuming a near 50% duty cycle, if the sender's data transitions and clock rising edges are aligned then the clock falling edge naturally falls in the center of the eye. For duty cycles that are not 50% the sampling edge needs to be

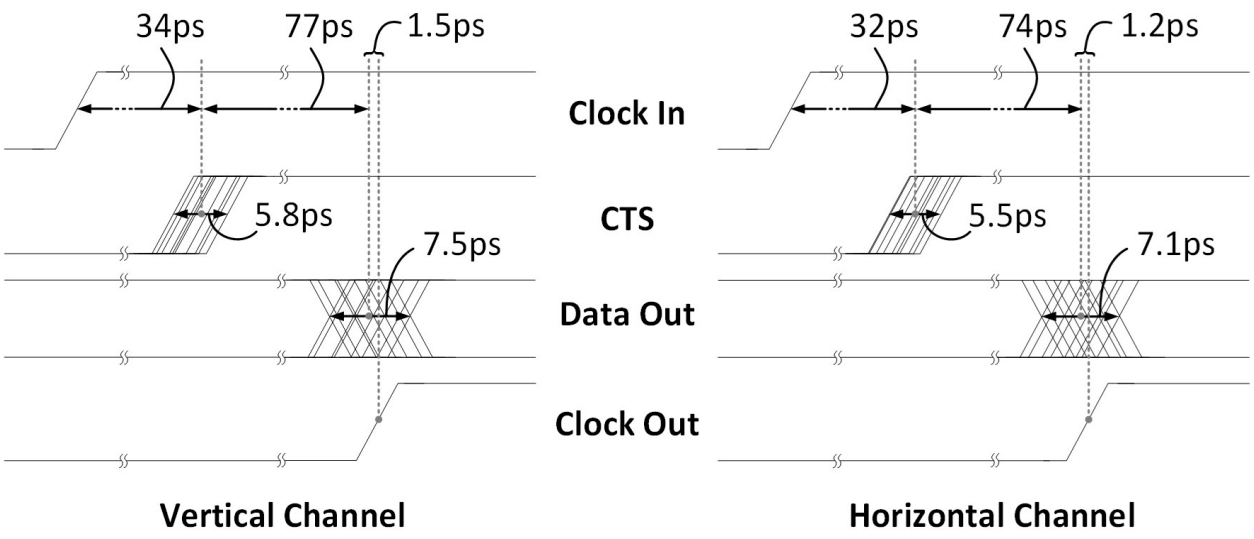
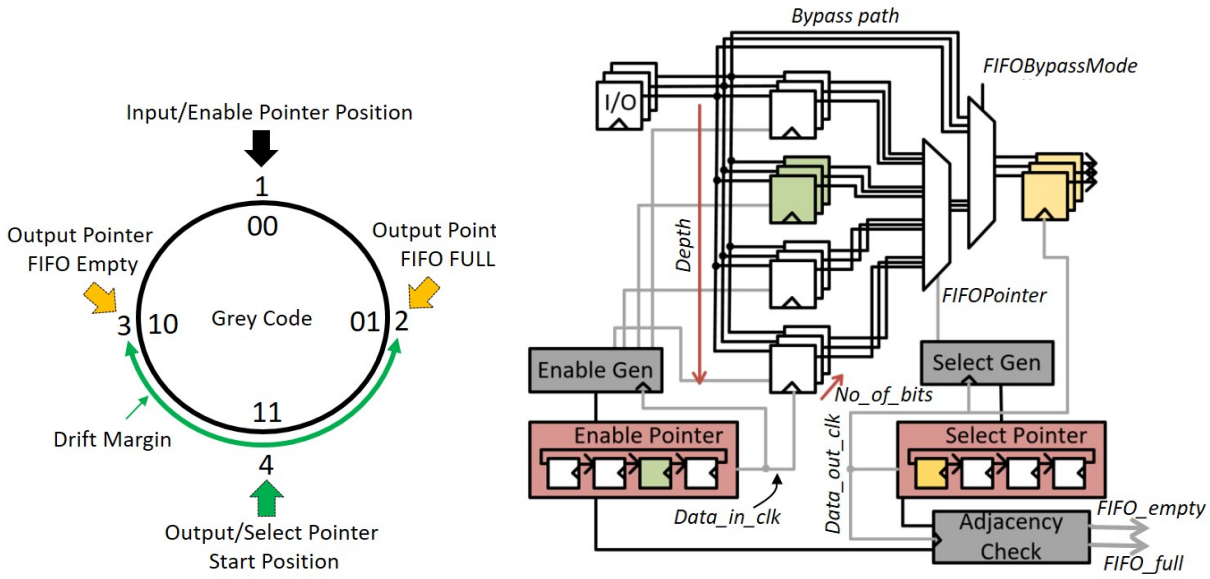


Figure 3.13: Relative clock and data path timing in SNR-10.

more carefully chosen so as to have enough margin while registering data on the receiver side. To align the clock perfectly with the data, the clock shares a near identical logic path to the data as shown in Figure 3.12. To verify the egde matching of the pad, post-layout simulation results are graphically plotted in Figure 3.13, one for the horizontal channel and one for the vertical channel. From the figure, it can be seen that the synthesized clock tree (CTS) has an average delay of 32-34ps with a $\leq$ 6ps spread on the leaf node registers. The small spread is attributed to the smaller size of the channel due to the smaller 10-µm pitch. The synthesized I/O paths add an additional 1-2ps of spread causing the output spread to be $\leq$ 7.5ps. The nearly identical clock path allows for the clock output to have $\leq$ 2ps matching error w.r.t. the average of the data path outputs.

With the sender clock well matched to its data, the onus falls on the receiver to match the sampling time to the data. Since the SNR-10 channel is synthesizable and can be easily ported across technologies, the technology based delay variation must be taken into account. For this reason it can be seen (from Figure 3.7) that each I/O has the capability to latch incoming data at the negative edge as well as the positive edge. For a receiver side clock delay



(a) Relative position of pointers.

(b) Schematic of circular FIFO.

Figure 3.14: FIFO architecture and pointers.

that exceeds half a cycle, the opposite edge needs to be used to sample the data. This check is performed by a simple bang-bang phase detector at the beginning of the handshake process. Due to the smaller chip size, symmetric arrangement of dielets on the Si-IF, and smaller data skews relative to the clock cycle period of 909ps (at 1.1GHz), the SNR-10 channels (in the UDSP 2×2) are observed to latch on the negative edge, and have not indicated any sampling margin violation. However, the possibility exists to latch data on the positive edge for ports to technologies that have larger clock tree delays or higher frequencies (in SDR mode) or even duty cycle distortion. By default, if a receiver side clock is not detected, the local clock is used to sample the data.

After the data is sampled it is sent to a circular gray-code vector FIFO along with the sampling clock to perform clock domain transfer (CDT) as shown in Figure 3.14. This is essential if the data is to reliably transfer between the sender's clock domain and the receiver's clock domain. The CDT mechanism needs to account and correct for relative clock drift (over time) between adjacent dielets due to manufacturing and run-time PVT variations. Larger

chips with deeper clock trees are more susceptible to this variation. It is therefore essential for a PHY layer protocol to have a programmable FIFO. SNR-10 has a gray-code FIFO to help its timing stability for vector data capture of multiple bits while retaining synthesizability. The FIFO also has programmable initial depth which can be programmed by the host chip via the chip-side interface (see Table 3.1). The depth is designed with 4 positions. In the UDSP version of the SNR-10, the bypass path is taken as the clocks are well matched with sufficient margin and low run-time drift due to the smaller size of the dielets. This leads to a 3 clock cycle end-to-end latency in the worst case. At 1.1GHz the delay comes out to be 2.8ns. It should be noted that larger more diverse clock trees may require more drift margins and a larger FIFO which can increase the size of the channel, the power consumption, as well as the delay. To keep the FIFO circular and gray-coded, an even number of flops are chosen in the circle and the code is generated using the method described in [52].

3.4 SNR-10 Measurement Results and Comparison

The SNR-10 channel is designed in TSMC 16nm and has a channel footprint of 8769µm^2 with 64 I/Os resulting in an I/O area density of 137µm^2 per bit. The channel is part of the larger UDSP chip implementation [53]. The same channel is designed in GF 22nm at a much smaller scale to check synthesizability and portability. However, the GF 22nm implementation works at 700MHz due to the reduced frequency of the underlying UDSP. The following results are from the TSMC 16nm implementation. The SNR-10 is tested for its maximum power draw by sending a continuous stream of '1' - '0' oscillations with 100% switching activity factor across 56 connected channels at 1.1GHz, using the UDSP 2×2 assembly. The assembly uses a 2-layer Si-IF with a wire reach of 350um as shown in the footprint in Figure 3.4a. At nominal operational frequency of 1.1GHz the voltage is kept at 0.8V. The channel is

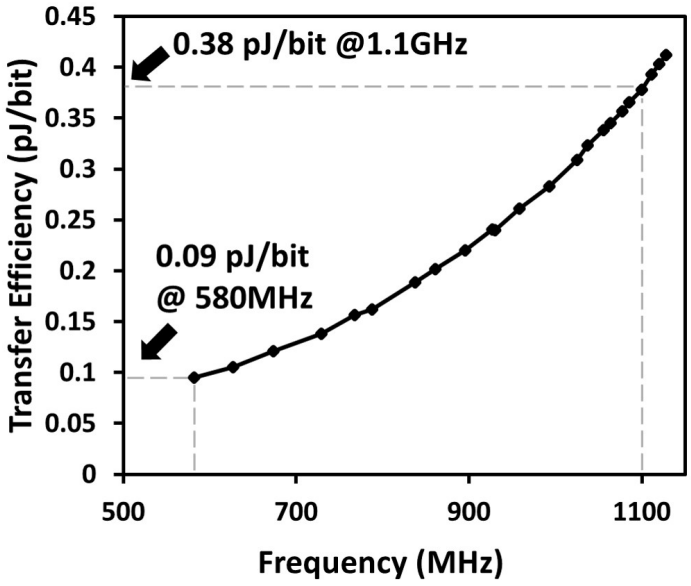


Figure 3.15: Data transfer efficiency vs. voltage-frequency scaling.

observed to average a 0.38pJ per bit energy-efficiency in this scenario. Reducing the voltage and frequency of the design lowers the power draw substantially, reducing it to 0.09pJ per bit at 580MHz SDR. The tested frequency range and transfer efficiency is summarized in Figure 3.15.

To compare the performance of SNR-10 to similar fine-pitch MCM protocols for interposers, 3 state-of-art works [54, 51, 50] are chosen which, at the time of writing, have good I/O area density as well as good efficiency. All protocols, including SNR-10, are targeted towards parallel digital I/Os and are assumed to operate at their maximum data-rate. The comparison is summarized in Table 3.3. With just 2-layers of Si-IF, and the overhead of large passive redundant control and clock pads, the peak shoreline bandwidth of the SNR-10 channel is measured to be 297Gbps per mm. In contrast, [52] and [50] use a larger number of layers yet achieve comparable shoreline bandwidth densities. LIPINCON [51] achieves a 5.4× bandwidth density of SNR-10 with the use of 7.5× more layers. Larger number of Si-IF layers would allow the channel to scale to higher bandwidth densities due to more pin stacks

Table 3.3: Comparison of proposed SNR-10 with SotA MCM protocols.

Spec	[This work]	[54] CICC'21	[51] JSSC'20	[50] JSSC'19
Techology	16nm	16nm	7nm	16nm
Package Substrate	**2-Layer Si-IF**	4-Layer EMIB	15-Layer CoWoS	MCM/ PCB
Bump Pitch (µm)	**10**	55	40	150
Reach (µm)	350	3,000	500	80,000
Data Rate (Gbps/pin)	1.1	2	8	25
Voltage (V)	0.8	0.9	0.3	0.95
Energy Effc. (pJ/bit)	**0.38**	0.83	0.56	1.17
I/O Area Density (μm^2/bit)	**137**	203	500	10,175
Peak Shoreline BW Density (Gbps/mm)	297	256	1600	292
Layer BW Density (Gbps/mm/layer)	**149**	64	107	25
Delay (ns)	**2.8**	4	N/A	N/A

lined up deeper in the chip. For two layers, and with 1.5-µm track widths on the Si-IF, the stacks can at most extend to 4 rows. Normalizing to the number of layers, SNR-10 achieves a bandwidth density per layer of 149Gbps/mm/layer which is much higher than any of the competition. The latency is kept at 3 clock cycles for this implementation, comparable to other protocols, as the FIFO is bypassed due to smaller chip size resulting in lower PVT drift. To date, SNR-10 is the first and only sub 10-µm pitch protocol demonstrated to work on a fine-pitch wafer-scale interposer. In addition it has the lowest area per I/O of 137μm^2.

The performance of all aspects of the channel (such as latency, energy-efficiency, area per I/O etc.), can be combined into a figure of merit (FoM). The UCLA FoM [49] is used to compare a large number of protocols [48, 50, 51, 55, 56, 57, 58, 59, 60, 61, 62, 63] plotted against link length in Figure 3.16. The dotted line represents a simple driver-plus-register synchronous link on the Si-IF [64] as the upper bound bare-bones maximum. Such an

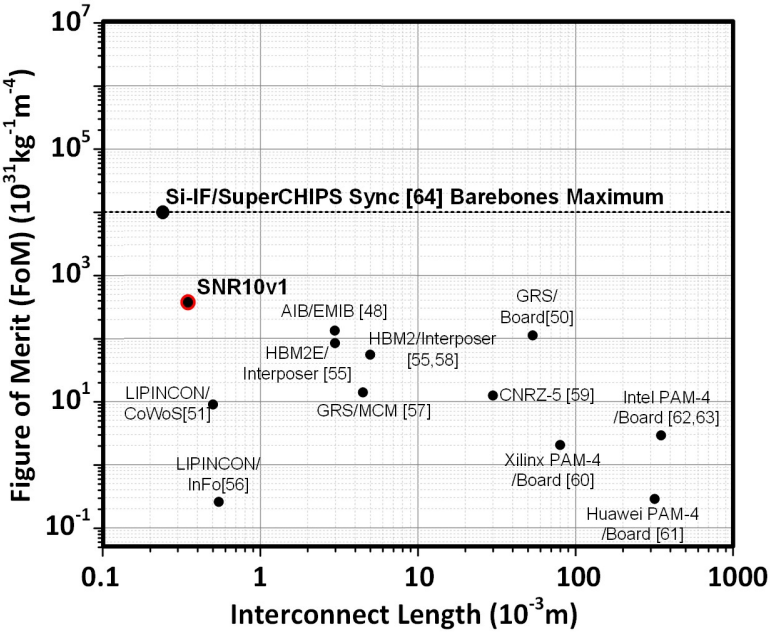


Figure 3.16: UCLA FoM for the SNR-10 channel.

arrangement would have the least amount of area, power, delay and speed (assuming perfect clocks, ideal matching, and zero manufacturing defects). SNR-10 sits well above the the other protocols, however, it still has much room for improvement. Part of the reason for the large improvement gap is the amount of amortization of common elements, like the FIFO and redundancy-repair mechanic, is spread over a smaller number of (64) data I/Os. The other reason is that the channel is not nearly operating at its full speed due to lack of DDR and DCC schemes. Adding these mechanics leads to the design of SNR-10 version 2, a larger channel with 4 times as many I/Os and DDR capability. However the design of such a channel comes with its own challenges which are further discussed in Chapter 6.

CHAPTER 4

UDSP Control Logic and Compiler Tool-flow

The architecture of the UDSP consists of several configurable parts; from the different configurations of the cores, to the switchboxes in the vertical stack, to the inter-core network and the I/O network. In addition, UDSP sports the division and coordination of larger kernels on MCM modules. To achieve these properties, accompanying the architecture are an offline compiler, a real-time compiler, and programming interfaces. The offline compiler provides the main interface for the user and is responsible for creating soft programmable binaries. The real-time compiler maintains the current state of the UDSP MCM and it funnels the soft program binaries into hard program data frames which are sent to the dielets via the programming interfaces. The task of the compiler is to make the experience for the user as quick and effortless as possible. Several key architecture and design aspects described in Chapters 2 and 3 enable fast compile-times, orders of magnitude faster than baseline FPGAs. This section describes the tool-flow and how it takes advantage of the UDSP architecture to give compile-times of $<$ 1s for moderate sized kernels. Taking advantage of the control capabilities of the UDSP, a real-time resource management unit and interface is proposed to increase active utilization and speed-up total execution time.

4.1 Programming Interface and UDSP Control Logic

Figure 4.1: 2×2 UDSP compute fabric with run-time control support.

The MCM control architecture is designed to support run-time multi-program compile and configuration as shown in Figure 4.1. At the center of the UDSP is the control module and PLL. The control module ensures support for several key characteristics of programming the UDSP. It allows for kernels (or programs) to be written to each vertical stack individually while the array is running other programs. Kernels can be kept in soft (data) reset once written, to be executed at a later time. This can speed-up instruction dependent execution bottlenecks in a bandwidth constrained pipeline. The control module can help read and remove programs in a critically resource limited array, transferring back a partially executed program for re-execution at a later time. In addition, the control can simultaneously execute multiple kernels on the array as long as they don't over lap in their vertical stack (VS) assignments. The compiler can coordinate with the control of four UDSPs to execute a cross-dielet program using the SNR-10 channels. The 3-cycle communication latency between

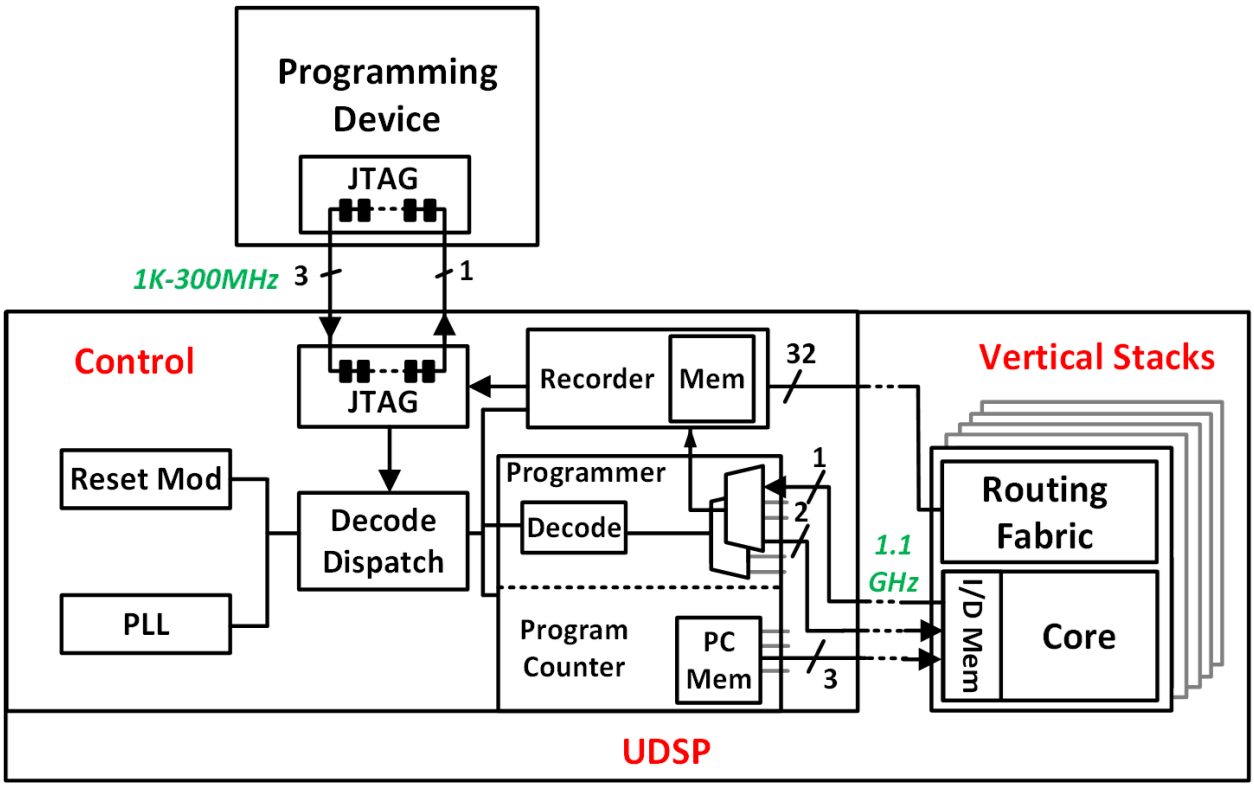


Figure 4.2: UDSP control logic.

dielets is accommodated during the retiming step in the compiler. Shared JTAG write ensures that the instruction reaches all UDSPs at the same time so their execution starting time is synchronized.

Figure 4.2 shows the internal blocks of the control logic. The compiler and subsequent binaries of the program are hosted by an external Linux based programming device. This device uses a JTAG interface to send instructions over to the UDSPs custom JTAG block. The interface frequency is at maximum $1/3^{rd}$ the UDSP's operational frequency (up to 300MHz). The data is sent in fixed size frames with the Linux device acting as the master and the UDSP device acting as the slave. On the die-side, the received instruction frame goes through a decode process. Flags in the instruction determine which modules to send the payload of the instruction to. The control hosts the reset module that can soft-reset or hard-reset each VS individually. In soft-reset, only the data paths in vertical stacks are placed

in reset. This is useful in flushing the pipeline and resetting the kernel. In hard-reset, in addition to data registers ,the VS instruction registers are placed in reset as well, erasing any programmed kernels. Part of the instruction code contains clock-select instructions which are sent to a module that interfaces with the PLL. The control hosts a programmer module which takes the relevant bits from the received frame, decodes the address and programs the appropriate vertical stack. The program module sends the instructions serially over a 1-bit 1.1GHz line to each VS. This module also contains program counter memory which can be pre-assigned a sequence of instruction pointer values for temporal dynamics of kernels. Each VS individually interfaces with the program counter using a 3-bit interface. The presence of a recorder (or observer) module allows for instruction loop-back to the programming device, as well as 16kb of contiguous data recording. The recording interface is connected to the interconnect network and signals of interest can be routed to the recorders ports. The recorder is used for self-test, debug and instruction write-back.

4.2 Static Compiler Tool-flow

The static compiler comprises of the back-end tools developed for the UDSP to do retiming, clustering, placement, and routing of the kernels. The steps are shown in Figure 4.3. The compiler uses the Simulink interface from MATLAB [65], taking advantage of its user friendly and powerful GUI for building data flow graphs (DFGs). A data flow graph (DFG) is an efficient high level representation for analyzing data movement and dependency properties in directed compute graph, where the nodes of the graph represent computation elements and the edges represent data flows. DFGs act as a bridge between algorithmic representation and architectural implementations [66]. Computational graphs are constructed in Simulink's GUI and fed to the compiler as '.mdl' model files. The tool interprets and forms the internal DFG

representation of the kernel. The DFG is checked to only contain elements that the UDSP supports and to break apart complex blocks, if they exist, into basic adder and multiplier based flow graphs.

4.2.1 Retiming, Resource Binding, and Place & Route

Following the interpretation step, the DFG is retimed according to the compute fabric constraints, namely each multiplier and each adder carry a delay of 1. For feed-forward graphs this process is simpler due to lack of loops and clear levels of dependency. Delay cut-sets are made at each dependency level, and delay insertions are made as required. Most paths in the core contain programmable delays as shown by the delay matrix in Table 2.3. The core contains 2 delay lines as well to support higher number of delays. In case of loops in the DFG, the loop is cut and mapped first without adding any more delay to the loop. If the mapping result fails then the loop's delay is doubled (effective frequency of the kernel is halved) and the mapping process repeated.

Though described as an independent first step, retiming is iteratively performed in conjunction with clustering and placement, improving mapping and core utilization with each iteration. After retiming (in each iteration) the compiler proceeds to perform resource binding (clustering) during which the DFG compute nodes are bound to physical core elements. Staring off with one element, each cluster grows by absorbing its neighbouring elements, each time checking core mappability support by comparing the subgraph to the connectivity matrix from Table 2.2. Comparison is an O(1) operation since the core is small and all connections with it are 1-hop. This lowers the compile-time contribution of resource binding. If the cluster cannot absorb a map, a new cluster (and core) is formed at the boundary and the process continues. I/Os are bound to SNR-10 channels.

Once a valid clustered graph is built, each cluster is assigned a core, at random, on a virtual manhattan grid of vertical stacks. Simulated annealing is performed to lower the inter-node distance. The minimization cost function is non-linear and only optimizes for the first valid map. This means that if a core's connection falls above distance 2, then the cost assigned to it is disproportionately higher. The non-linearity of the placement cost function allows simulated annealing to trade-off connection length with validity of mapping due to the 2-distance direct connection support of the interconnect network. Since the interconnect is delay-less, no further retiming optimization is needed in most cases, where the 2-distance mapping is successful. In cases where the placement tool has longer distance connections left over, the I/O layer is used, with the cluster graph going through partial retiming. If partial retiming is unsuccessful, the placement process is repeated.

After a valid placement is achieved the relevant connections are given to the router which performs a depth first search (DFS) operation for randomly initialized connection-sets through a switchbox. Though this process is a heuristic for an NP-complete problem, the solution is often found in the first attempt (rarely needing a second attempt), due to the high I/O mapping probability of the switchbox as shown in Figure 2.30. Like the matrices of the core, the switchbox's hyper-matrix is used to quickly check connection maps in an O(1) operation. The contribution of routing to compile-time is therefore low. The I/O router is invoked last to route delay-balanced connection paths from the SNR-10 channels to the vertical stacks via layer 4.

After a valid compile has been achieved, bits streams are generated from instruction look-up matrices for the core, the switchboxes and the I/O layer. These instructions are bundled into one of two types of frames (Figure 4.4a). Type-1 frames carry one vertical stack's instruction and type-2 frames carry instructions for the I/O network. Each frame also contains flags to activate other portions of the control logic, such as reset and the

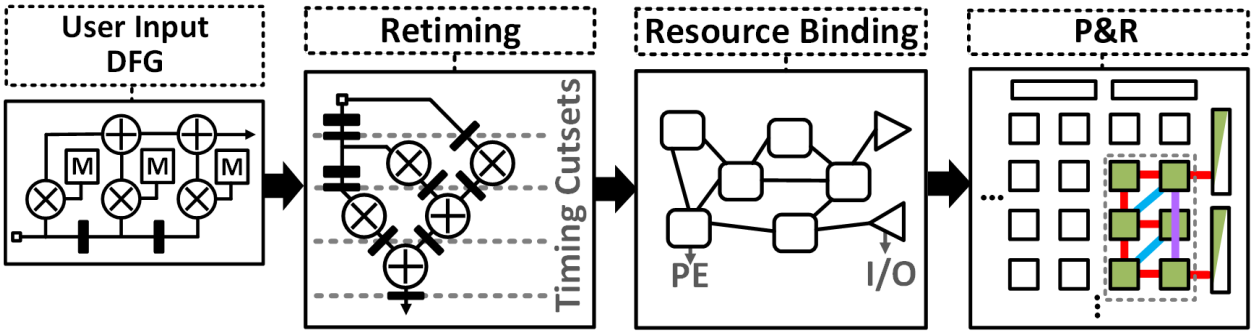
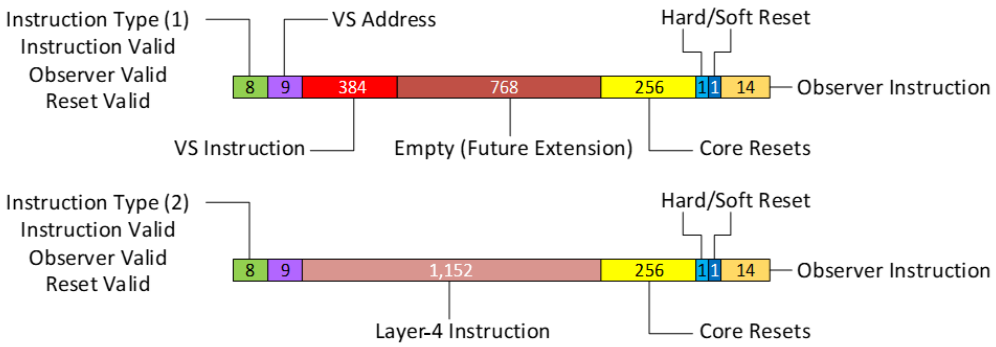
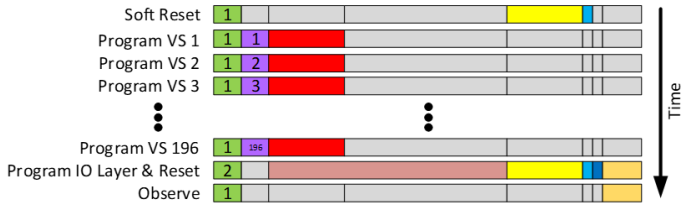


Figure 4.3: Compile flow: retiming, resource binding, and place & route.

recorder. A typical kernel programming flow is shown in Figure 4.4b. First a reset frame is sent placing particular cores in hard-reset to erase any prior instructions. Subsequently each VS involved is programmed, up to the full size of the array. The I/O network is programmed next, and as a last step, the relevant cores are placed out of soft-reset to start executing on data. External data is assumed to be streaming in from the UDSP I/O at this point. Above is a brief description of each of the steps involved in getting a valid compile with more detail available in [41].



(a) Instruction frame type and bit-field breakdown.



(b) Programming multiple vertical stacks.

Figure 4.4: Instruction frame from compiler to the UDSP.

4.2.2 Static Compile Results

Figure 4.5 shows the compile-time for the static compile flow for increasing kernel sizes. For reference, a 20-core kernel can accommodate a 40-tap FIR filter. The results were generated by averaging 50 compile runs. Moderate size kernels can compile in less than 500ms due to several optimizations during hardware design (described in Chapter 2). Time taken to bind DFGs to core resources scales linearly with core count as the internal connections of the core are 1-hop and checking a valid configuration is an O(1) operation. This makes the clustering heuristic an O(N) operation for the entire graph kernel and contributes to lower compile-times. Placements for small kernels ($<$ 15 cores) are quick as well because each core is directly connected to 12 of its neighbours using the 3-layered interconnect. For larger kernels, the time complexity of simulated annealing takes over and becomes the major compile-time contributor. Interestingly, route times scale linearly as well. This is because of the high route mapping probability of switchboxes as shown in Figure 2.30 which comes from maximizing the disjoint I/O paths by minimizing their path correlations. In summary, each of the hardware design choices made in the core, switchboxes, and interconnects (together) contribute to the fast ($<$ 1s) compile-times for moderate sized kernels.

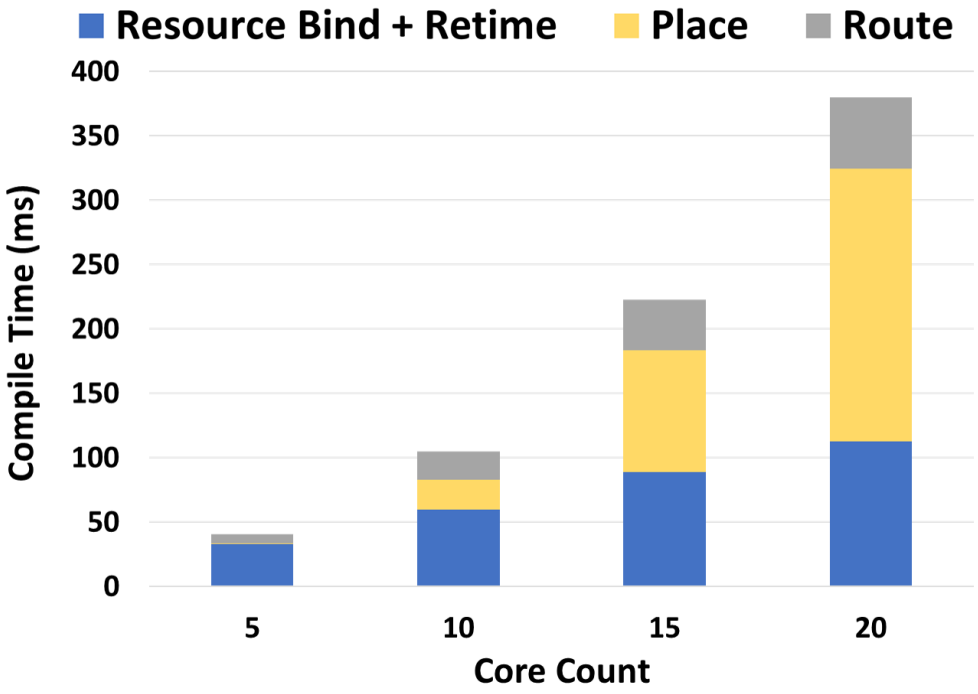


Figure 4.5: UDSP compiler performance w.r.t. kernel size.

4.3 Dynamic Compiler for Run-time Configuration

Hosted on the Linux programming device is a dynamic run-time compiler which assists in the management of the array's resources. This last layer compiler is what interfaces with the 2×2 UDSP MCM (Figure 4.6). The dynamic compiler consists of a resource monitor to track array utilization, a real-time (RT) macro memory to store the kernels operating on the array, a garbage collection module to handle core resets and instruction write-backs, and a frame exchange module to hosts the custom JTAG interface. For a single kernel, the output frames from the static compiler correspond to a bounding polygon of vertical stacks on the array. These frames can be directly sent to the frame exchange module of the dynamic compiler in case the user intends to bypass the resource management functionality and treat the array as statically compiled.

The dynamic compiler can assist in higher array utilization. This is done by taking the

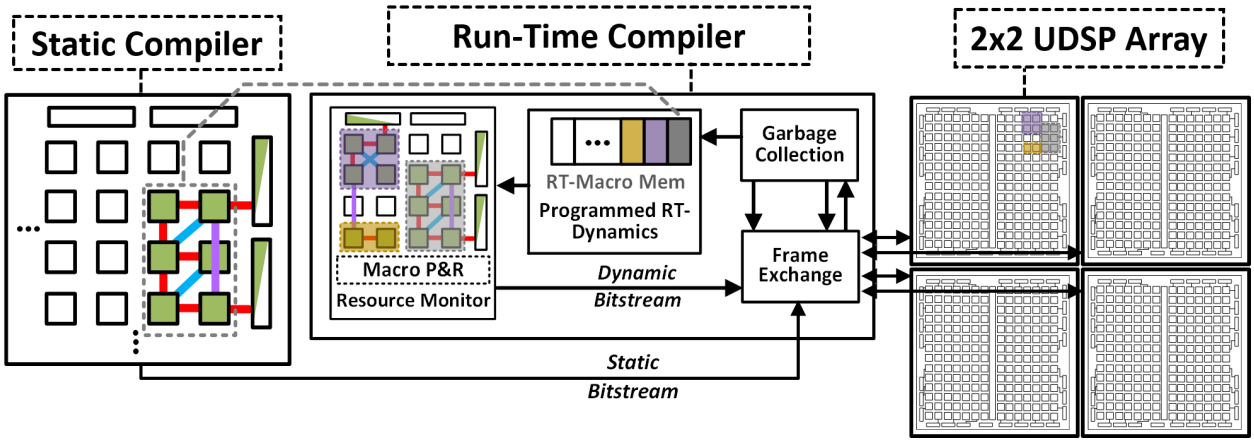


Figure 4.6: Run-time scheduler in the loop for real-time resource monitoring and reconfiguration.

frames of the bounding polygon of the VSs (from the static compiler) as a soft binary and storing it in the macro memory. The resource monitor tracks the currently active programs on the array and the in active VSs. When the resource monitor determines sufficient availability of space on the MCM array, it takes the next-in-line kernel and dynamically generates its (address modified) bit-stream. Quick address modification is possible in the UDSP because the underlying interconnect network has translational symmetry. The frame exchange module contacts the appropriate UDSP for instruction write. The resource monitor can, in addition, force the eviction of a lower priority kernel as well. This is done by first placing the currently executing kernel in soft-reset, then using the observer module in the UDSP to pull out the instruction via JTAG for storage in a queue in the RT macro memory. This sequence of operations is handled by the garbage collection module. Since the I/O cannot be translated on the array, the relevant part of the I/O network has to be reprogrammed and recompiled every time a dynamic program is placed. The pipelined nature and adequate connectivity of the I/O network ensures that moderate I/O kernels (with 8-16 I/Os per 20 VSs) are routed quickly by the dynamic compiler. The frame for the I/O network is then sent over to the relevant UDSP.

4.4 RTRA: An Active Resource Management Engine

Though the dynamic compiler can increase the active utilization of the array, it has its limitations such as lack of rotational symmetry, longer I/O network compile-times for high I/O kernels (like FFT), lack of data buffering for time sensitive kernels etc. These limitations can cause the array to experience lower active utilization and longer kernel execution times. Adequately addressing these issues at runtime requires fast, dedicated hardware. Additionally, adding a service layer on top of the RT reconfiguration model requires support for extra interfaces and memory. The resultant set of technologies that can enable a fully autonomous, real-time, active resource management engine is referred to as Real-Time Reconfigurable Array (RTRA) and is shown in Figure 4.7. In this section the workings and provisions for such a system are described briefly with more detail present in [67].

In the RTRA architecture, the UDSP array (an example of a CGRA) is the main compute engine. The I/O network layer is redesigned to have equal latency connections to all cores and to maximize the disjoint paths from each I/O to each core, maximizing the network's silicon efficiency. Essentially the I/O network is treated as a big switchbox, with chip I/Os (SNR-10) on one side, and chip PEs (vertical stacks) on the other side, with an optimized multi-layer switchbox design (Chapter 2.4) between them. Scaling up the array size inevitably leads to Rents rule [68] taking a toll via either an increase in the I/O network's area or a decrease in the connectivity between the I/Os and the compute cores. To facilitate kernels with pre-mapped data arrivals, a data memory bank (RAM) is added which can store large input and output data blocks. Additionally a multi-bank memory and memory controller, with support for programmable access patterns, is added as a mid-level cache for sustaining high data throughput from the RAM to the array. A dedicated RT compiler and scheduler are added. The compiler keeps a resource map and places the soft polygons from program memory

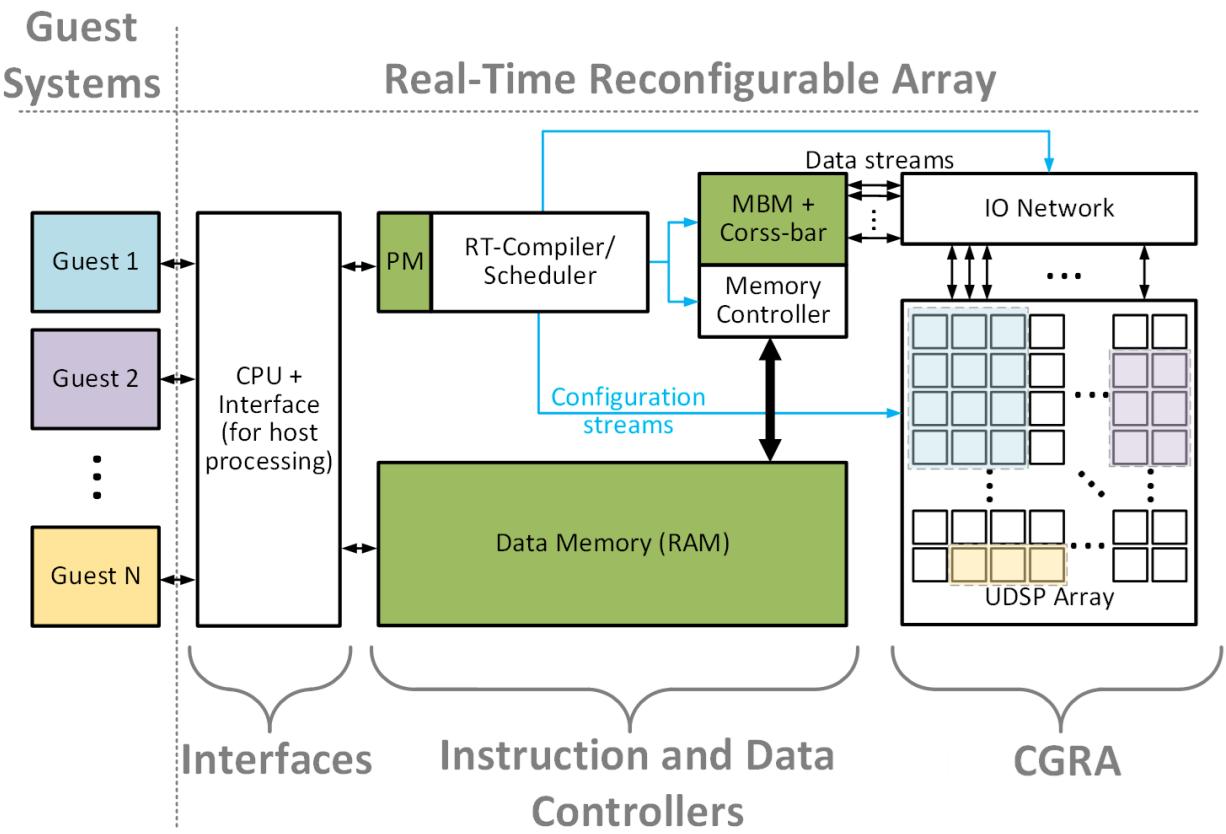


Figure 4.7: RTRA architecture building blocks.

to the array using a fast (< 50 cycles) edge-overlap-detection heuristic. Additionally it communicates the appropriate I/O network configuration and multi-bank memory patterns.

Active utilization can suffer due to higher chance of overlap of multiple large kernels. A multi-size compile scheme is proposed that allows for different kernel sizes for the same program, effectively providing several space-time configuration options for a program. Since the UDSP is not symmetric under rotation, multi-size compile can additionally provide rotated versions of the polygons. Due to the fast compile-time of the UDSP's static compiler, several multi-size kernel versions can be generated quickly. The scheduler takes the meta data (polygon shapes and execution time) from the output of multi-size compile of multiple programs and plays *space-time tetris* of sort, on the array, minimizing the overall execution

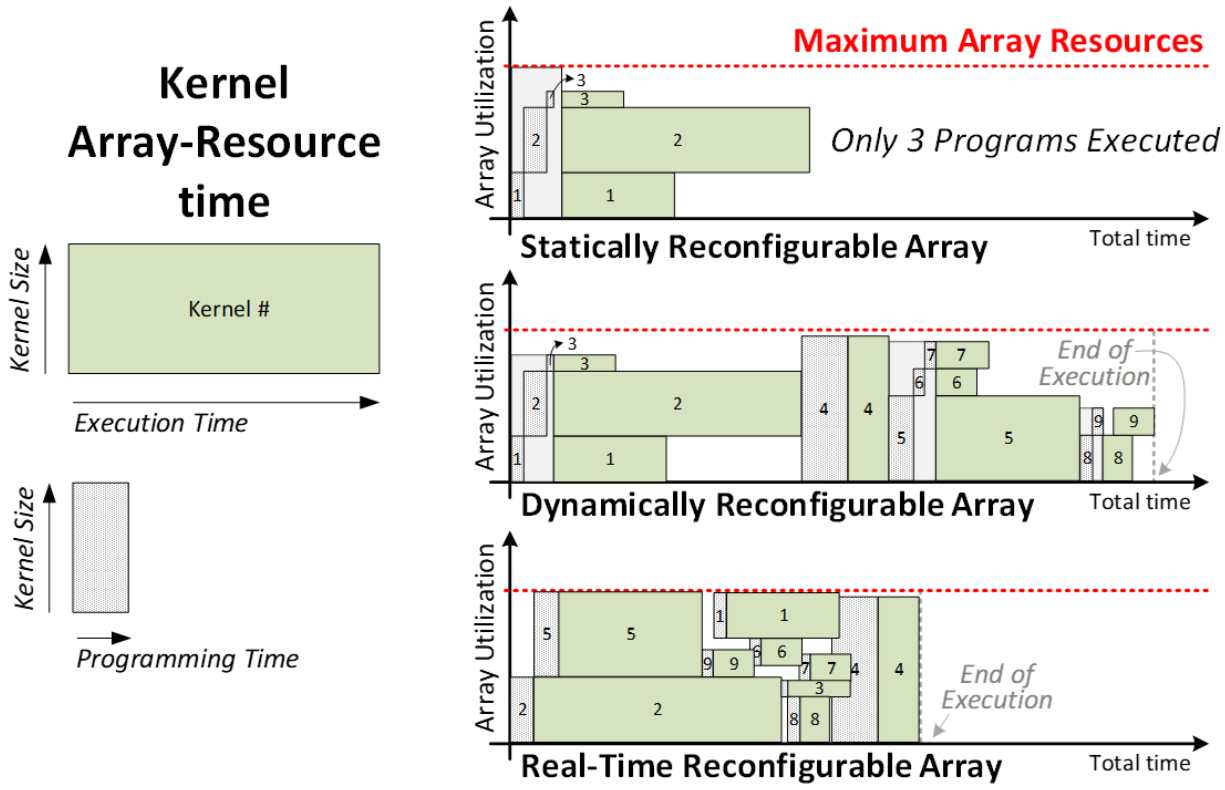


Figure 4.8: Comparison between statically, dynamically, and real-time configured arrays.

time of the cumulative sum of programs. The scheduler additionally handles the priority based eviction (and subsequent re-run) of programs. Once a program's size and timing are decided by the scheduler, relevant instructions are sent to the UDSP array, the I/O network and the memory. As an example, the total execution time for 9 random-size programs is compared between a statically configured array, a dynamically configured array, and an RTRA, in Figure 4.8. It can be seen that RTRA ensures high array (space-time) utilization allowing for faster execution of programs.

To allow the underlying CGRA to be used by more than one system at a time (for yet greater utilization), the scheduler and data memory are interfaced with a local CPU and program ports (e.g. PCI-E) for several guest systems. The local CPU is responsible for data security of the array, and accepting and redirecting program requests from different hosts to

the scheduler. The memory, dynamic scheduler, and CGRA make up the real-time resource management system called RTRA. Where as the additional CPU and interfaces allow for the CGRA to be used as an accelerator-as-a-service (ACaS) for several guest systems.

CHAPTER 5

UDSP Assemblies, Test Setups and Measurements

This Chapter discusses the implementation and results of the UDSP architecture. The architecture is implemented in two different technologies, GF 22nm and TSMC 16nm. Both implementations contain SNR-10 channels at the edges of the DSP and are therefore accompanied by the 10-µm pitch Si-IF. The dielets are bonded to the Si-IF through thermal compression bonding (TCB). The Si-IF is then wire-bonded to test PCBs. Following single die assemblies, an MCM assembly of 4 TSMC 16nm dies is done on the Si-IF using the SNR-10 channels (for internal communication) to scale up the processor array. Together, the three designs demonstrate portability and scalability of the UDSP compute fabric (Figure 5.1). Depending on the design, a different PCB architecture is chosen. Programs are compiled by a compiler running on a small Linux device and transferred over via the JTAG interface. An FPGA is used in the loop during testing of the hardware-assisted, real-time reconfigurable array (RTRA). Power and efficiency measurements under several conditions for several algorithms are performed and the results are compared to baseline ASICs and FPGAs.

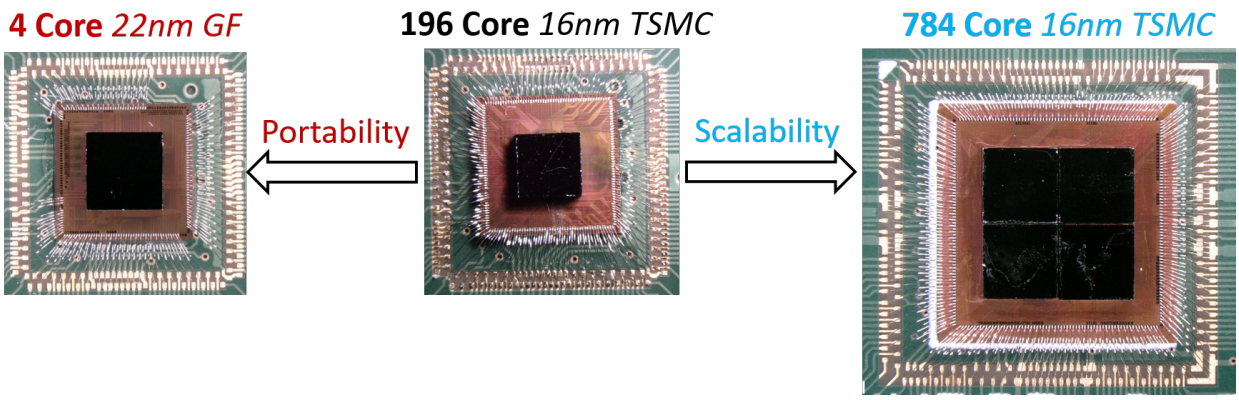


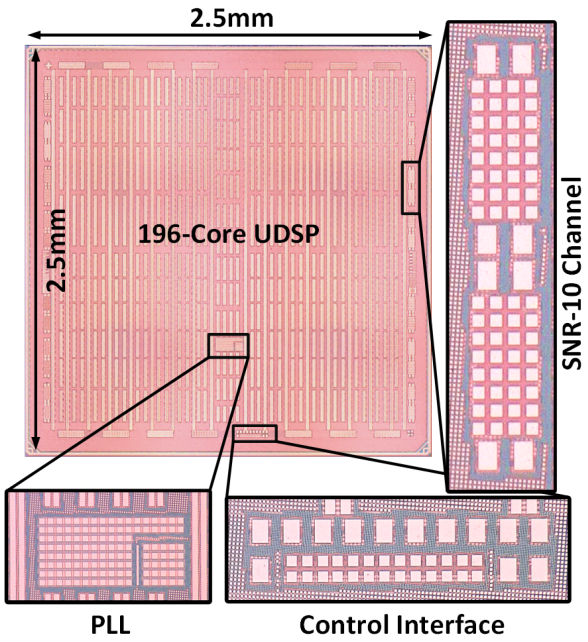
Figure 5.1: Portability and scalability of the UDSP compute fabric.

5.1 Assemblies and Test Setups

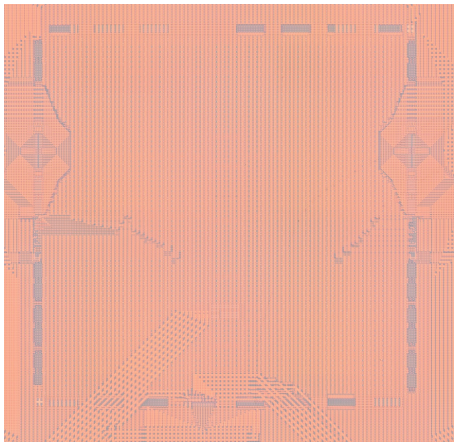
5.1.1 196-Core and 28-Channel UDSP in TSMC 16nm on Si-IF

The main implementation of the UDSP is done in TSMC 16nm, with the dielet containing 196 cores in a 14×14 grid (Figure 5.2a). The die size, including the seal ring, is 2.5mm × 2.5mm. The core area is 2.1mm × 2.1mm. Each die contains 28 SNR-10 channels at its periphery to allow it to scale to MCM implementations. At the bottom is the control interface. The larger pads on the control interface are for bonding two bumps in parallel which gives passive redundancy to critical connections. Towards the center of the design is the PLL that takes a 10MHz reference clock as input and outputs 50MHz - 1.1GHz as needed for testing. The design frequency is 1.1GHz at an operating voltage of 0.8V. The long vertical copper lines that run across the die are V_{DD} and V_{SS} rails. The top-left corner has a '+' marker with its negative on the bottom-right. THese markers are used in die alignment on the Si-IF. In order to have the dielet in assembly form, a wafer pull at the 9^{th} metal layer is performed and covered with a layer of nitride for transport. Figure 5.2b shows the footprint on the Si-IF which uses 2 wiring layers and one bump layer and is completely

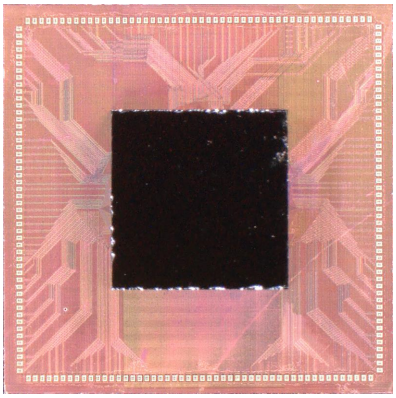
passive. At UCLA, the nitride layer is dissolved and TCB is used to bond the underside of the UDSP to the Si-IF with direct copper-to-copper bonding (Figure 5.2c). The PLL power is provided by an independent power domain that is spaced out from the digital domain to reduce coupled switching noise.



(a) 196-core UDSP with 28 SNR-10 channels.



(b) Footprint on the Si-IF.



(c) 1×1 dielet assembly on Si-IF using TCB.

Figure 5.2: UDSP implementation and assembly in TSMC 16nm.

The assembled die is then wire bonded onto a PCB shown in Figure 5.3. The PCB contains a power delivery circuit, a clocking circuit and the Tx and Rx interface for external

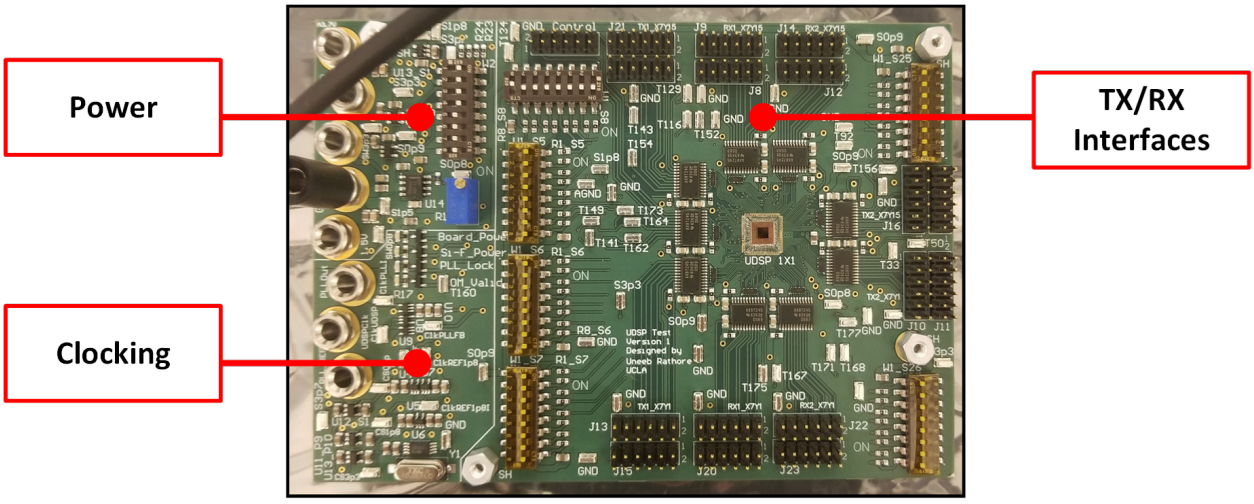


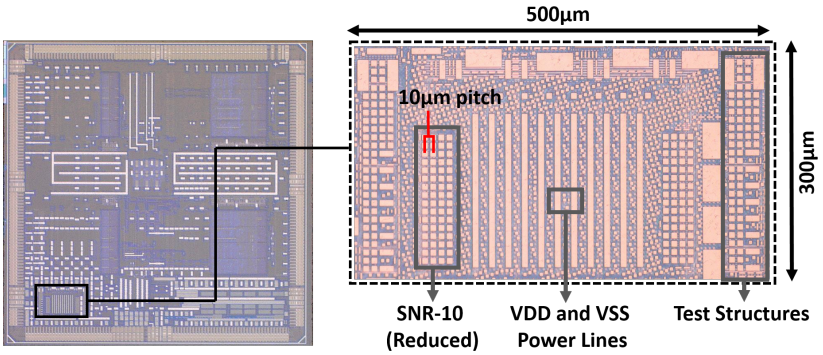
Figure 5.3: 1×1 UDSP assembly PCB test setup.

data communication through level-shifters. External testing of the UDSP is limited to 50MHz due to the limitation of the level-shifters. After low-speed functional testing of the UDSP, internal high-speed differential testing is done to verify full-speed operation at 1.1GHz. The speed of the control interface is limited by the speed of the programming device. An FPGA based programming device is tested to achieve $1/3^{rd}$ the operational frequency of the UDSP to a maximum of 25MHz (limited by level-shifters on the PCB). Switching to a Linux based programming device (Raspberry Pi) gives much slower programming speed of ∼10kHz due to the limitation of its output GPIO pins. However, the benefit gained is the quick debug time of the compiler (which takes much longer on the FPGA due to its reprogram time). The inputs to the PLL pins are programmable by fixed DIP-switches. The PLL voltage regulator and power routing is isolated (weakly coupled by diodes) from the digital power delivery to reduce coupled switching noise. ESD events are observed on 10µm pads that interface externally. 30% of the Rx pads and 5% of the Tx pads, on average, experience some form of ESD, either gate punch through, stuck-at-0, or stuck-at-1. This issue is addressed later in the 2×2 test PCB.

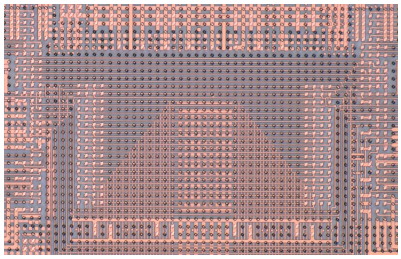
5.1.2 4-Core and 2-Channel UDSP in GF 22nm on Si-IF

The GF 22nm chip contains a small 4-core implementation of the UDSP. Since the UDSP design was scripted and the network and core implementation was automated, the development of the chip took less than 2 weeks time. This implementation does not contain the PLL, rather it contains a ring oscillator as its primary clock source. The design frequency is 500MHz at an operating voltage of 0.8V. It contains 2 SNR-10 (reduced) channels at the edges. The channels are reduced, with redundancy-repair disabled and extra pads removed, to fit in the allocated space on the tape-out. The quick turn-over time of the design in a new technology indicates the portability of the UDSP. The scaling down of cores to 4 shows the modularity of the compute fabric. This design is used for functional testing and assembly, not for performance measurements. Figure 5.4a shows the underside of the die. The implementation contains test-structures (not part of the UDSP) as well which are used solely in characterization of the Si-IF, more detail of which can be found in [69]. The two SNR-10 channels (reduced) with 64 10-µm pitch pads each can be seen as well. The vertical copper lines underneath the UDSP are for V_{DD} and V_{SS}. On the top of the vertical lines are (spaced out) control pads. A similar wafer pull is performed at the 9^{th} metal layer and covered with a layer of nitride which is later on removed at UCLA for copper-to-copper TCB. Figure 5.4b shows the footprint on the Si-IF and Figure 5.4c shows the assembled die on Si-IF. At the periphery of the Si-IF sample are gold plated 60µm × 100µm copper pads used for wire-bonding to PCB. The same PCB from the TSMC 16nm 1×1 UDSP implementation is reused (Figure 5.3) to save on design time.

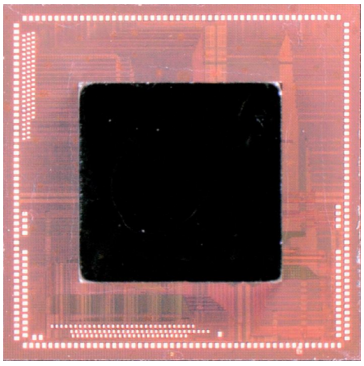
Functional testing shows that all internal modules are working as expected. The SNR-10 channels are connected to each other and no ESD event was observed between the links on the Si-IF. The channel works at 500MHz as well. Clock forwarding was not performed as



(a) 4-core UDSP with 2 SNR-10 channels.



(b) Footprint on the Si-IF.



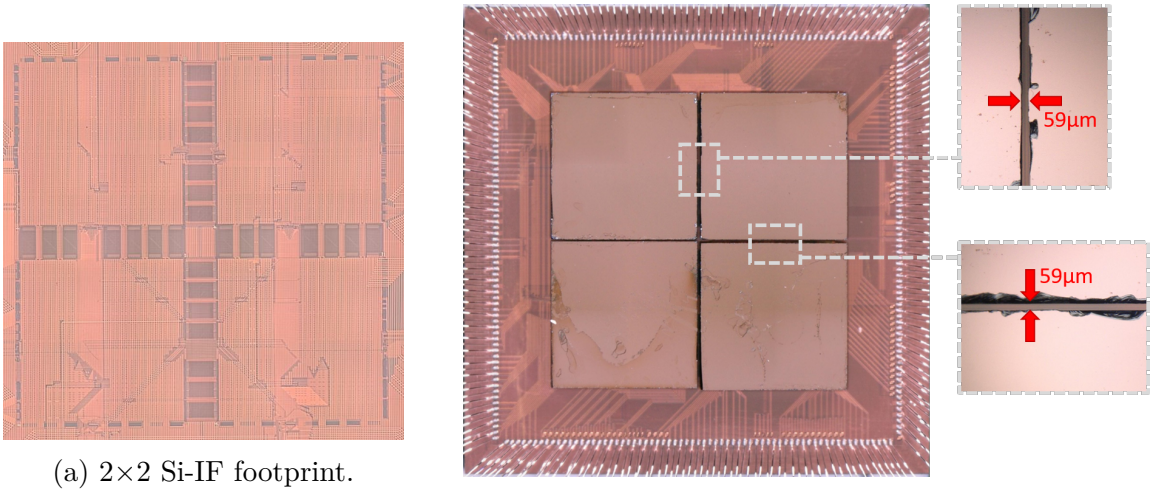
(c) 1×1 dielet assembly on Si-IF using TCB.

Figure 5.4: UDSP implementation and assembly in GF 22nm.

the channel was communicating to its counterpart on the same chip which had the same leaf node skew derived from the same clock tree.

5.1.3 784-Core and 56-Channel 2×2 UDSP MCM in TSMC 16nm on Si-IF

The UDSP dielet in TSMC 16nm is designed to be scalable to multi-chip modules. Four UDSP dies from Figure 5.2a are bonded in a 2×2 assembly on the Si-IF via TCB. This assembly contains a total of 784 cores, 1792 I/Os and 56 connected SNR-10 channels. The footprint of the 2×2 Si-IF is shown in Figure 5.5a (zoomed version in Figure 3.4a). The copper links used by the SNR-10 channel can be seen on the Si-IF footprint. For the clock



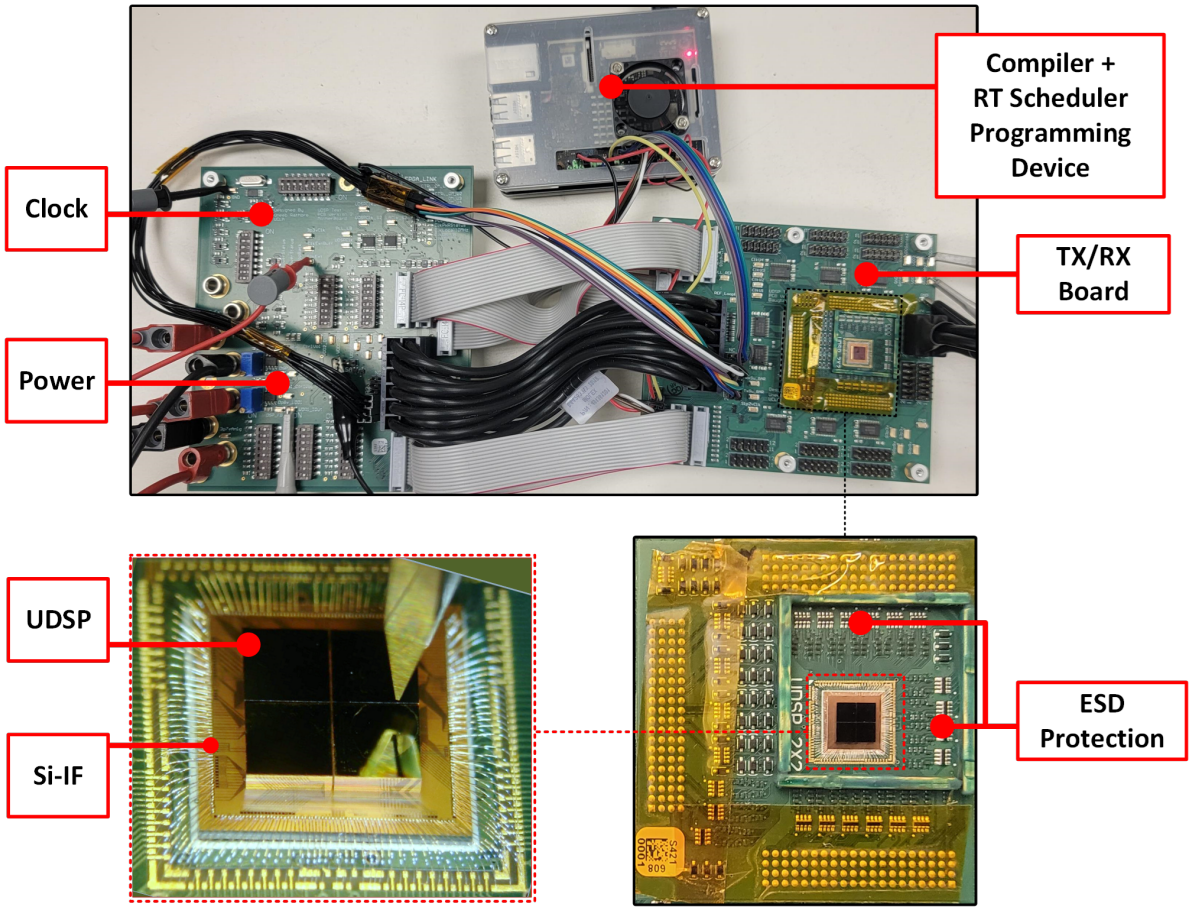
(a) 2×2 Si-IF footprint.

(b) 2×2 dielet assembly on Si-IF using TCB.

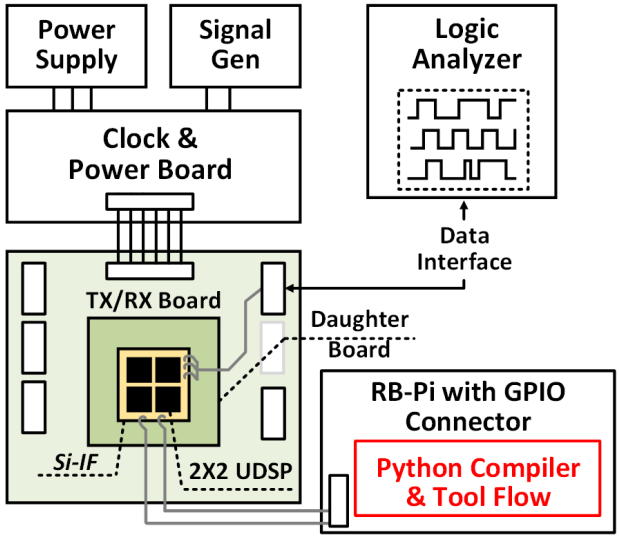
Figure 5.5: UDSP MCM implementation and assembly in TSMC 16nm.

distribution, a single (external 10MHz) clock source is sourced to the center and then fanned out in an 'X' *cross* wiring pattern for skew balancing. Though each UDSP sports its own PLL, UDSP-1's (bottom-left) PLL's 1.1GHz output clock is chosen as the main clock for the MCM system, and its output is again bought to the center and fanned out to all UDSPs (including back to UDSP-1 as well). This ensures negligible reference clock skew at 1.1GHz between the dielets. The clock's 'X' *cross* wiring pattern can also be seen in the footprint. Figure 5.5b shows the assembled dies. The distance between adjacent dies is less than 60µm. The periphery of the Si-IF has over 220 wire bonds. The 2×2 assembly has 2 voltage domains, one for the two left side DSPs and one for the two right side DSPs. This is to ease the current requirement on the 2 LDOs used for power delivery, one for each domain. Like its 1×1 counterpart, the PLL's analog power routing is spaced out from the digital power domain for reduced noise.

Figure 5.6a shows the test setup for the 2×2 assembly. It consists of 3 PBCs. The motherboard contains the power and clocking as well as static DIP-switches for the 4 UDSP dies. These switches control the PLL, resets, as well as clock delivery and power delivery configurations. The Tx and Rx board contains level-shifter interfaces to connect the assembly



(a) Physical view of the test setup with 3 PCBs and the 2×2 zoomed in.



(b) Logical view of the test setup.

Figure 5.6: 2×2 UDSP assembly PCB test setup.

to external logic and signal analyzers. A smaller daughter board contains the wire bonded test sample in addition to ESD protection circuits. With these circuits no external ESD events are observed for over 250 pins tested across 2 samples. For the internal SNR-10 die-to-die links, no internal ESD events are observed for over 6,000 links tested across two samples. The UDSPs communicate directly with the Linux device via a custom JTAG interface. Figure 5.6b shows the logical view of the setup. A logic analyzer is used to externally provide 16-bit streaming data. The data frequency is limited to 50MHz due to the level-shifter drivers. For high-speed testing, the data is generated internally in the cores and passed around through the interconnect network and SNR-10 channels.

5.2 Measurement Results

The measurements presented in this section are taken from the 2×2 UDSP assembly inclusive of all active SNR-10 inter-die communication channels, PLLs and additional sources of power draw. On the Si-IF there are only 2 available layers for power delivery as well as data links. Most data links are short range and leave room for the Si-IF power distribution network (PDN) which is constructed as a mesh. In order to measure the resistance of the Si-IF and the voltage droop at different stages in the power delivery network, dedicated measurement lines are routed from the output of the LDO, from the daughter board, and from the center of the Si-IF to PCB test points. Figure 5.7 shows the voltage droop measurements at every stage for one voltage domain in the MCM. It is observed that at the daughter board voltage tracks the LDO voltage much closer than that the UDSP voltage tracks the PCB. This indicates a relatively higher wire bond and Si-IF resistance. Adjusting the output voltage (to track the UDSP's required 0.8V) at the center of the Si-IF results in a large LDO output voltage increase. Though the voltage droop is high, differential frequency failure testing

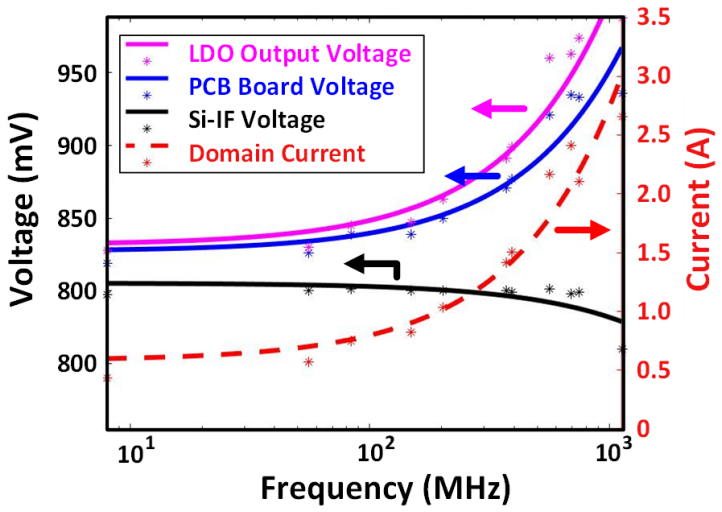


Figure 5.7: Voltage drop comparison for a single voltage domain in the 2×2 UDSP assembly.

of several cores at different edges of the 2×2 assembly reveals no significant relative droop across the Si-IF. This indicates that the resistance is majorly a result of the wire bonds. The corresponding current draw is shown on the right in Figure 5.7, and increases exponentially with frequency. Even though the current load is divided across two LDOs (one for each voltage domain), the small size of the LDO IC causes the device to heat up to 80°C. Heat sinks are used to cool the LDO. All power calculations are made with the appropriate voltage measured at the center of the Si-IF, and the output current from the LDO.

The inter-die communication on the MCM is tested as well. At 1.1GHz, the 56 SNR-10 channels across 28 Si-IF links are toggled each cycle for maximum power draw. The channels achieve an energy-efficiency of 0.38pJ/bit. At that high-speed, the dense 10-µm pitch allows for a maximum bandwidth density of 297 Gbps/mm with the total inter-die bandwidth reaching 493Gbps (Figure 5.8). Total cross-sectional bandwidth on the MCM reaches ∼2Tbps across all 4 dies.

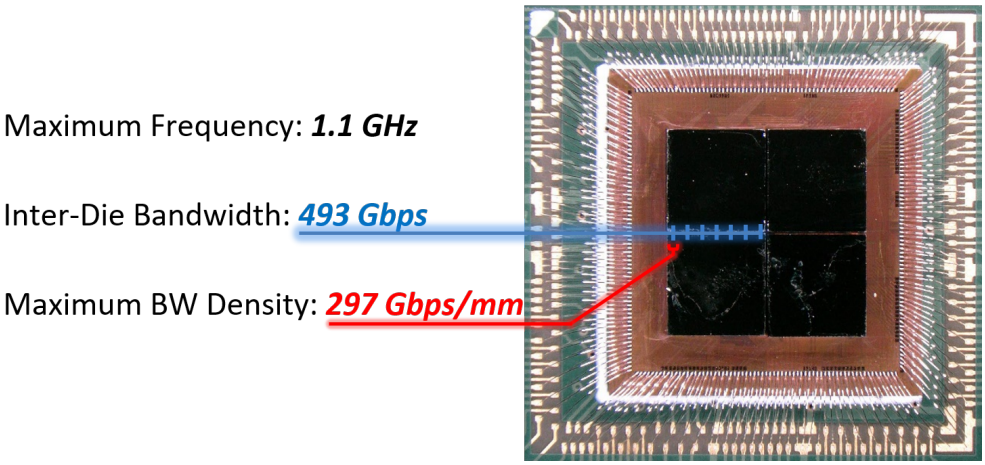
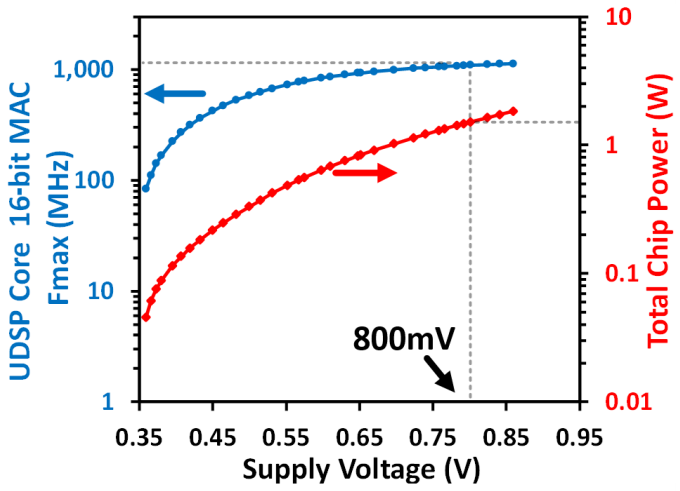


Figure 5.8: Dense communication bandwidth using SNR-10 on the 2×2 UDSP assembly.

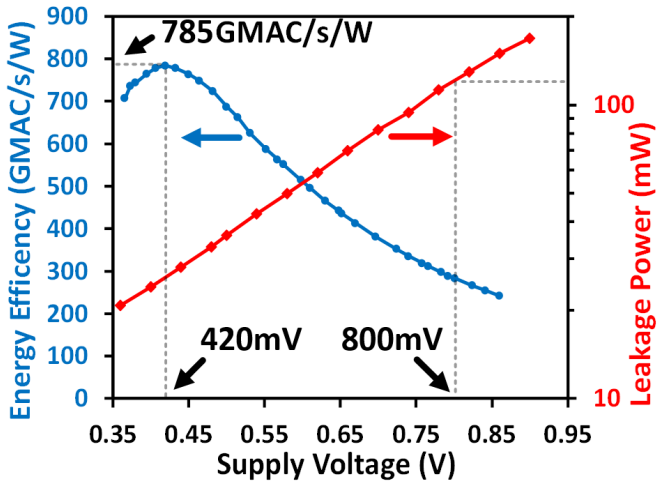
5.2.1 Voltage and Frequency Scaling

To test the performance of the MCM with varying voltage conditions, the supply voltage and frequency are adjusted till all cores pass functional testing. All cores on all dies are programmed to execute MAC instructions and the SNR-10 channels set to toggle every cycle (100% switching probability). The total power draw for the 4 dies is noted. Figure 5.9a shows the measured results. At the design voltage of 0.8V, the UDSP can operate at 1.1GHz drawing ∼1.35W power. In Figure 5.9b, the leakage power of the design is plotted and can reach > 100mW per die at the nominal voltage. Scaling down the voltage to 0.42V (and correspondingly scaling the frequency) increases the compute efficiency of the UDSP, reaching a peak of 785GMACs/s/W. The compute efficiency is measured in terms of GMACs because the core's atomic operation is a MAC (2 MACs per core per cycle).

To measure the performance for algorithms other than a MAC, four moderate size kernels are chosen. These kernels consist of 16-tap FIR (real), 8×8 matrix multiplication (real), 16-point FFT (complex), and 8×8 beam forming (complex) (Table 5.1). The larger kernel sizes averages out power measurements and in addition, exemplifies the mapping capability of the



(a) UDSP frequency scaling and total power scaling with voltage.



(b) UDSP energy-efficiency scaling and leakage power scaling with voltage.

Figure 5.9: UDSP voltage-frequency scaling measurement results for the TSMC 16nm die.

interconnect network. The high throughput of all kernels, operating at the maximum UDSP frequency, indicates the speed-agnostic routing support of the compute fabric. Table 5.1 shows the energy used per atomic operation at the maximum throughput point as well as the maximum efficiency point. For low-power device applications, the UDSP can increase its efficiency at the cost of speed. For high throughput applications, the UDSP can operate at 1.1GHz at reduced efficiency (by 2×). The core utilization is $> 90\%$ for all kernels, except for the FFT kernel. The FFT achieves a lower utilization due to its intrinsic multiplier-to-adder

Table 5.1: Algorithm performance and efficiency of the UDSP.

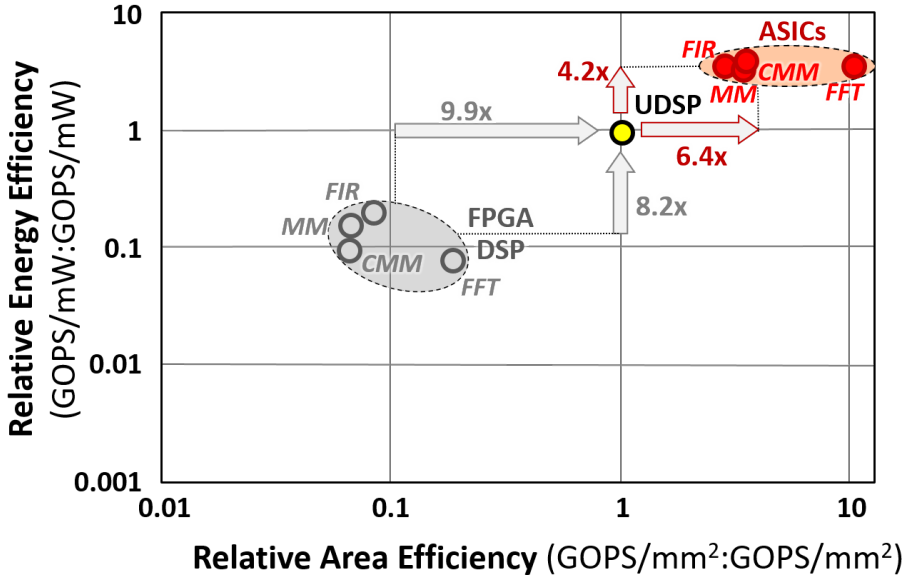
OpCond	0.8V, 40°C				0.42V, 40°C
Algorithm	Throughput (GS/s)	E_{total} (pJ)	Utilization		E_{total} (pJ)
			Cores	%	
16-tap FIR	1.1 Real	**3.54** /Tap	8	99	**1.29** /Tap
16-pt FFT	17.6 Cplx	**11.2** /C-Radix	86	53	**4.09** /C-Radix
4x4 Beam-forming	4.4 Cplx	**14.2** /C-MAC	34	92	**5.15** /C-MAC
8x8 Matrix Multiply	8.8 Real	**3.50** /MAC	32	98	**1.27** /MAC

ratio of ∼0.6 (for large FFT sizes), as compared to the ratio of 1.0 inside the core. The energy-efficiency, however, is kept high by keeping the unused elements in soft-reset.

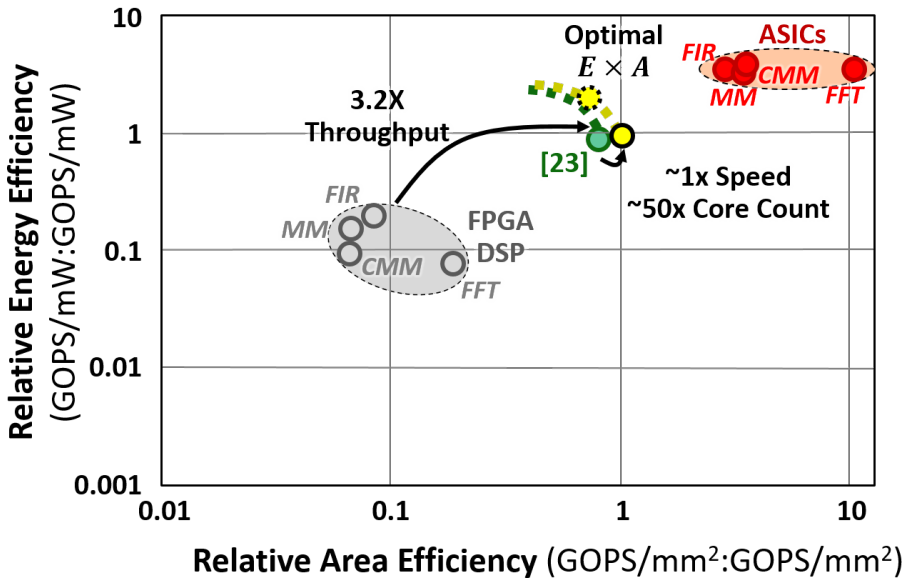
5.2.2 Comparison with FPGA-DSPs and ASICs

In order to present the relative strength of the UDSP architecture, plotted in Figure 5.10a are the relative efficiencies of the same kernels across FPGA-DSPs and ASICs in same node equivalent technology. These include Matirx Multiplication (MM), Complex Matrix Multiplication (CMM), FIR filters and FFTs. Measurements are made using a commercial 16nm Xilinx FPGA, utilizing its internal coarse-grain DSP blocks with results generated from its ISE tool. When mapping for area measurements, benefit is given to the FPGA by running its area-optimized compile flow. A similar advantage is given for energy measurements as well. The leakage energy of the FPGA is normalized to only the utilized area, with the same done for the UDSP. The ASIC measurements are for sign-off ready 16nm kernel blocks in the same technology as the UDSP. For each kernel, efficiencies are normalized to that of the UDSP operating at 1.1GHz and 0.8V. The UDSP is, on average, a decade away from FPGA-DSPs in terms of its area- and energy-efficiency (geometric means), and is 4.2× and 6.4× away from its corresponding ASIC counterparts in terms of energy- and area-efficiency

respectively (geometric means). In the FPGA, FFT is better implemented in terms of area-efficiency compared to the UDSP due to more fine-grain control over the multiplier to adder ratio.



(a) Efficiency gaps compared to ASIC and FPGA.



(b) Throughput gap and comparison with prior work [23].

Figure 5.10: Performance comparison in terms of relative energy and area efficiencies and throughput.

Running at its nominal voltage, the UDSP is 3.2× faster than its corresponding FPGA-DSP counterpart as shown in Figure 5.10b. The comparison is measured for each kernel with the throughput-optimized flow of the FPGA compiler. The outlier delays (especially those from the I/O connections) are dropped from the comparison to give the FPGA the advantage. The geometric mean of the resulting throughput numbers (kernel-to-kernel) is taken for comparison. Lowering the voltage and frequency, the UDSP can see a 2× improvement in energy-efficiency at the cost of roughly 30% throughput loss at its energy-area optimal point. At this point the product of both efficiencies is maximized. Comparing this architecture to prior work from [23] (scaled to 16nm using [71]), the UDSP achieves nearly the same speed and slightly better energy- and area-efficiency. The voltage-frequency scaling curves for both designs are plotted in Figure 5.10b. However, where the UDSP excels above its counterpart is that it manages to achieve these high efficiencies with a scalable interconnect and MCM design. Where the design in [23] only has 16 PEs due to its non-scalable interconnect architecture, and with power and area calculated without I/O overhead, the UDSP sports 786 cores connected across a 4-dielet MCM, with power and area inclusive of the PLL, I/O, control and seal-ring. To gain this efficiency advantage, the UDSP architecture leverages efficient core design, domain specificity in the interconnect design, compiler-hardware optimized switchbox design, and energy-efficient I/O design.

Comparing compile time of the UDSP to that of the FPGA-DSP, the UDSP takes on average 121× lower time to compile for the same kernel (Figure 5.11). The FPGA-DSP times are measured using an 8-core CPU system in addition to optimizing the flow for compile-time. The bulk of the the FPGA's compile time is spent on routing and placement. ASICs are inflexible therefore their compile time is not shown. The UDSP compiler is hosted on a single core machine. The lower compile times of the UDSP are attributed to quick placement due to longer reach of cores (2-hop connectivity) and quick routing due to switchboxes with

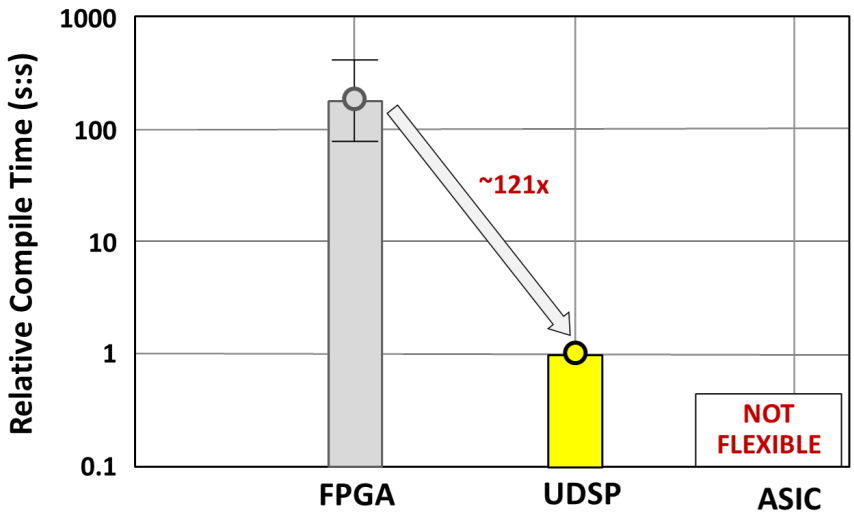


Figure 5.11: Average compile time comparison of UDSP and FPGA.

high routing probabilities.

Table 5.2 shows the summary of the UDSP MCM implementation. The compute fabric's data format is 16-bit fixed point, with an energy-efficiency of 284GMAC/J at 1.1GHz and 785GMAC/J at 315MHz. 4 dies totaling 784 cores make up the MCM. The inter-die communication uses 14 SNR-10 channels across 7 links on each side, totalling 493Gbps bandwidth at 0.38pJ/bit efficiency and 2.8ns latency across two adjacent dies. The entire assembly can reach a peak performance of 1725GMAC/s. For moderate kernel sizes, the UDSP achieves fast sub-second compile-times. In addition, the 2×2 UDSP is the first functional processor (to-date) on Si-IF with 10-µm pitch I/O bumps.

5.2.3 Example Application: DUC and DDC

The large number of cores can be used to map large kernels that need to operate at fast line-rate frequencies. A digital up converter (DUC) and a digital down converter (DDC) is implemented as a surrogate mapping onto the UDSP (since the external I/Os cannot

Table 5.2: Performance summary of UDSP 2×2 MCM.

Technology		16nm CMOS on 2-Layer Si-IF
Dielet Size		2.5mm x 2.5mm
Data Type		16-bit Fixed
Compute Energy Efficiency	Nominal	0.8V, 1.1GHz, 284 GMAC/J
	Peak	0.42V, 315MHz, 785 GMAC/J
Number of Cores		196 per Dielet 784 Total
Dielet I/O		1,792 pins
Inter-Dielet BW (0.8V, 1.1GHz)		492.8 Gbps @ 0.38 pJ/bit
Peak Performance of 2x2 UDSP		1,724.8 GMAC/s

run at high enough frequencies due to the level-shifter speed limitation). The mapping is shown in Figure 5.12. It is assumed that an analog front end (AFE) can interface with the SNR-10 channels and so can an FPGA (or memory controller). The AFE produces samples at 3.3GS/s with a Nyquist rate of 1.65GHz. Since this is higher than what the UDSP can absorb, the samples are 3-way parallelized and fed into the digital mixer. The digital mixers occupy a small number of cores, with the majority of the cores being used by the the 3-way parallel interpolation and decimation filters. The cutoff is kept at a normalized frequency of 0.3 with transition bandwidth at 0.06 and stop band attenuation at 54dB resulting in a 58-coefficient equiripple filter. Both in-phase (I) and quadrature (Q) samples need separate filters. The underlying signal is assumed to have a bandwidth of 1.2GS/s. At the other end, the FPGA absorbs the data samples at the decimated rate.

Other applications such as blind signal classification can also be mapped to the UDSP. However certain applications such neural networks or large matrix multiplication can become inter-die bandwidth limited due to their high reliance on the UDSP's internal dense interconnect. Even with the high bandwidth SNR-10 communication channel, the inter-die shoreline

Figure 5.12: An example mapping of a digital up converter (DUC) and digital down converter (DDC) on the 2×2 UDSP array, using digital mixers and 3-way parallel interpolation and decimation filters.

can only carry $1/8^{th}$ internal cross-sectional bandwidth of the UDSP interconnect. To alleviate this, the shore-line bandwidth of the SNR-10 channel must be increased. However significant timing correction challenges during data hand-off limit that increase.

CHAPTER 6

A Fully Synthesizable, < 1 LSB DNL, 0.0032mm^2 Delay Locked Loop with DCC and Large Frequency Tuning Range

In order to scale up the shoreline bandwidth density (Gbps/mm) of SNR-10, the number of data pin stacks (dimension of the layout of pads perpendicular to the chip edge) need to be increased as does the data rate per pad. This increase needs to supported by the corresponding increase in the number of layers on the Si-IF. Switching to a DDR scheme (vs. SDR) would effectively double the data rate per pad increasing the bandwidth. Increasing the number of shoreline pads (dimension of pads parallel to the chip edge) per channel will also increase the shoreline bandwidth as it will allow for further amortization of common circuits such as clock correction and redundancy check and repair. In order to realize all three changes, the channel pad count is increased from 64 to 512 DDR, as shown in Figure 6.1, effectively increasing the total number flip-flops on the transmitter and receiver side to 512 each. This increase in the number of flip-flops results in a large internal clock tree on either side of the channel where the leaf nodes of the tree are the data flops in the I/Os. Such a high fan-out clock tree can reduce the margin in sampling time especially at higher speed data transfer rates. This is because the clock tree adds skew to the otherwise ideal received clock in a clock-forwarded architecture as shown in Figure 6.2a. In addition, synthesized

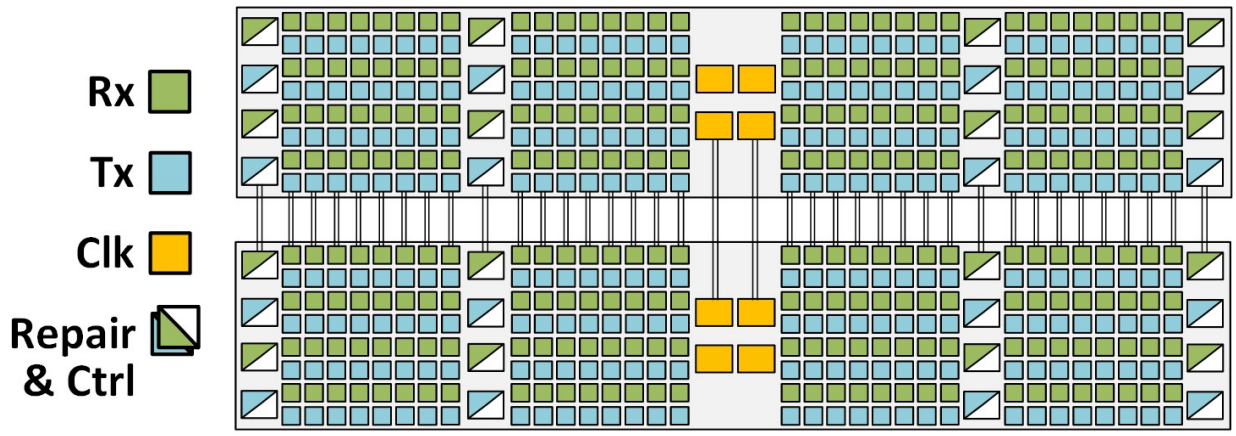
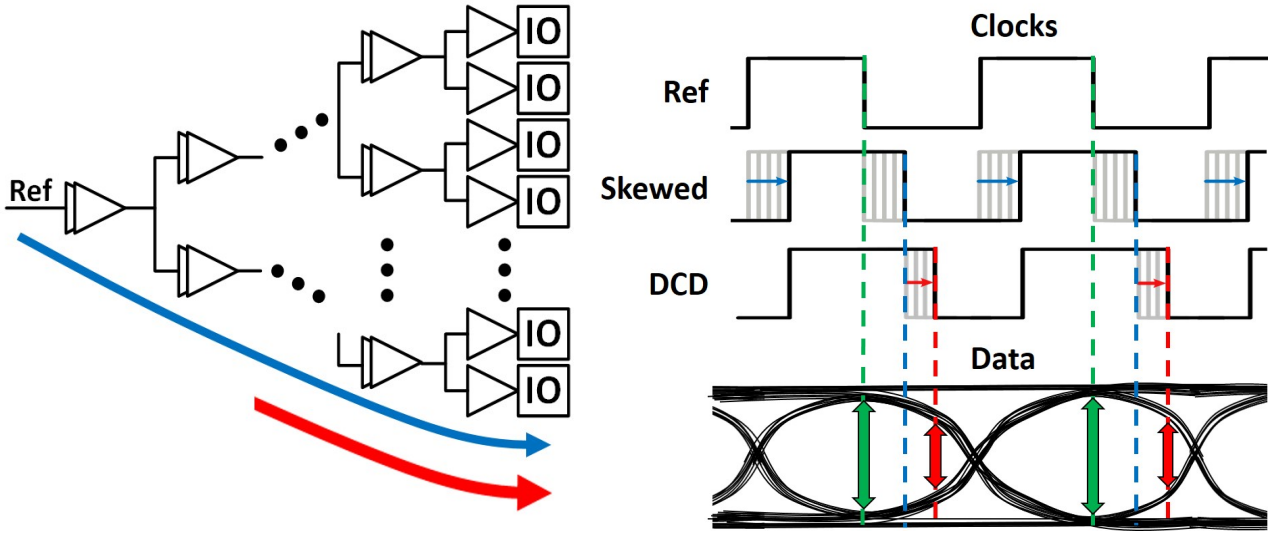


Figure 6.1: Two connected SNR-10 channels with 512 I/Os each.



(a) Internal clock tree

(b) Clock tree skew and duty cycle distortion

Figure 6.2: Clock tree effects on sampling margin.

clock trees result in duty cycle distortion due to imbalanced clock buffers which can further reduce sampling margins as depicted in Figure 6.2b. This phenomenon is exacerbated in DDR mode where both edges are used for sampling.

6.1 Delay Locked Loop for Clock Correction

To mitigate the clock skew and duty cycle distortion, a compensation delay needs to be inserted in series with the clock tree (CTS) such that the the cumulative phase shift of the clock tree ($\phi_{CTS+DCD}$) plus the added compensation delay (ϕ_{comp}) amounts to a shift by an entire cycle ($T_{clk} = \phi_{CTS+DCD} + \phi_{comp}$). Since the clock skew and duty cycle distortion is a function of PVT variation, which includes manufacturing as well as run-time variations, a robust solution entails the design of the compensation mechanism in a feedback loop. The feedback loop, referred to as the 'delay locked loop' (DLL), consists of the clock tree, a phase detector to detect the difference in phases of the original and distorted clocks, an adjustable delay line element, and a controller to actively adjust the delay line to compensate for the current phase mismatch. Figure 6.4 shows an example DLL setup. At the heart of the loop is the delay line which is responsible for adjusting the compensation delay (ϕ_{comp}). Equation 6.1 describes phase budgets assigned to typical sources of errors and mismatches. Errors in determining correct sampling times from a DLL can arise from the phase detector's input mismatch, the resolution of the delay line, the variations of the leaf nodes of the clock tree after synthesis, the noise of the CTS and DLL, and lastly the clock tree skew and duty cycle distortion ($\phi_{CTS+DCD}$). To illustrate the importance of $\phi_{CTS+DCD}$ relative to the other sources of error, simulated numbers from GF 22nm process are used for the architecture of the delay line (presented in the following sections). It can be seen from Equation 6.2 that $\phi_{CTS+DCD}$ is the largest contributor to the minimum cycle time, increasing it to ~800ps, limiting the design to 1.25GHz. The delay line mitigates $\phi_{CTS+DCD}$ as in Equation 6.3, allowing for higher frequency, lower voltage, and more robust data transfer timing. Equation 6.2 and 6.3 are shown graphically in Figure 6.3, showing a 3.2× potential increase in speed.

$$\frac{T_{clk}}{4} \geq Err_{PD} + Res_{DL} + \Delta_{CTS} + 6(\sqrt{\delta_{DL}^2 + \delta_{CTS}^2}) + \phi_{CTS+DCD} \tag{6.1}$$

$$\frac{800ps}{4} \geq 2 * 17ps + \sim 7 * 2.5ps + \sim 151ps \tag{6.2}$$

$$\frac{256ps}{4} \geq \sim 3ps + \sim 6ps + \sim 2 * 17ps + \sim 7 * 3.5ps \tag{6.3}$$

$$T_{clk} \triangleq Clock\ Period \qquad Err_{PD} \triangleq Phase\ Detector\ Error$$

$$Res_{DL} \triangleq Delay\ Line\ Resolution$$

$$\delta_{DL} \triangleq RMS\ Phase\ Noise\ Delay\ Line \qquad \delta_{CTS} \triangleq RMS\ Phase\ Noise\ Clock\ Tree$$

$$\phi_{CTS+DCD} \triangleq Clock\ Tree\ Skew\ and\ Duty\ Cycle\ Distortion$$

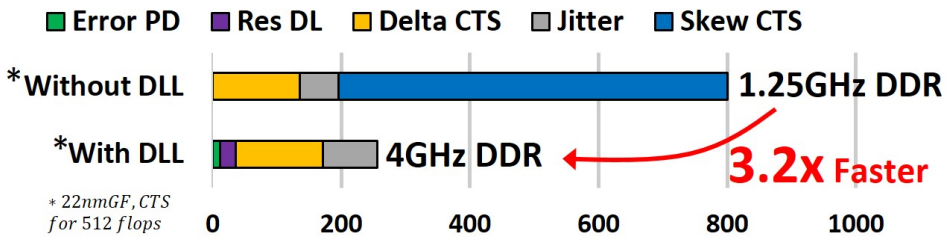


Figure 6.3: Clock sampling margin breakdown with and without a DLL.

All-digital delay line implementations where the delay is controlled by a binary word (as opposed to an analogue voltage or current) have an associated resolution (Res_{DL}), which is the incremental delay added with each subsequent binary word increase. Due to PVT variations, each increment will incur a small error during run-time adjustment. The incremental variation (error) with delay step is the differential non-linearity (DNL) of the delay line and it is essential that the DNL error is kept as low as possible ($<$ 1 LSB or $< Res_{DL}$) to give the most control over the final phase and keep the delay line monotonic. Large DNL

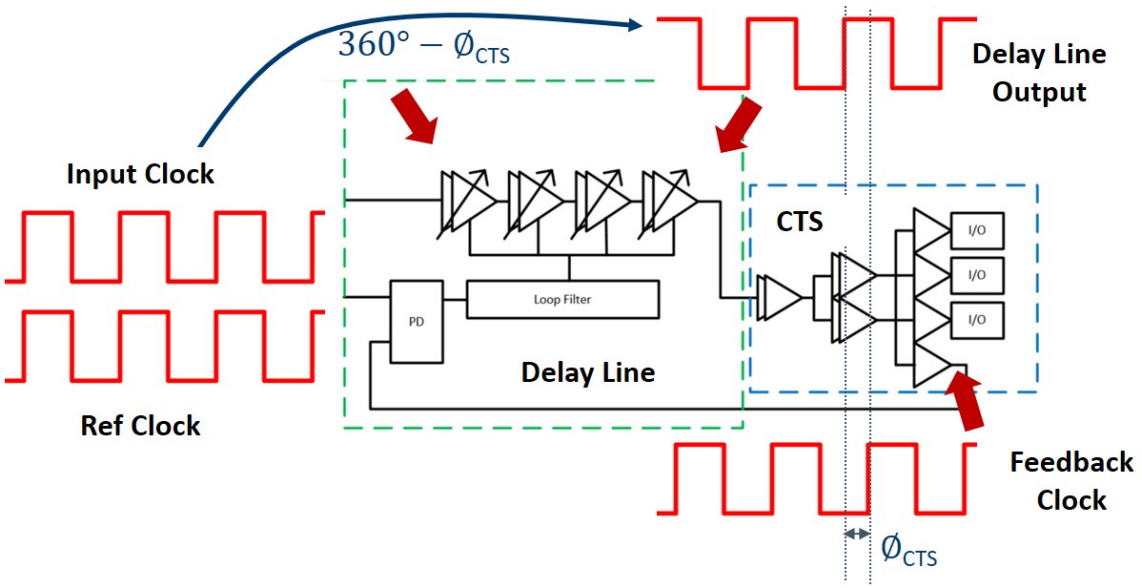


Figure 6.4: An example delay locked loop (DLL).

errors in a DLL appear as sudden phase jumps which can require more complex control and cause momentary data sampling errors. The integration of the DNL errors is the Integral Non-Linearity (INL). The INL is a measure of the deviation from the ideal delay (defined as a straight line between the end points) over a range of input words.

6.2 Delay Line Requirements for SNR-10

Since SNR-10 is a fine-pitch protocol, much emphasis is placed on it being area-efficient and energy-efficient. This places a strict area constraint and power constraint on any delay line deskew mechanism that the channel may utilize. In addition, the portability of SNR-10 requires the delay line (of its internal DLL), have a large frequency tuning range to be compatible across a wide array of dielet types and dielet-to-dielet boundaries. The adjustment mechanism should have a DNL and INL of < 1.5 LSB across the entire delay range for low bit error rates (BER) during runtime tuning, as different chip IPs may have their own delay adjustment requirements. The DNL and INL should not require any manual calibration as

several of these channels will line dielets' shorelines and manually calibrating each channel would consume time and resources. The delay line should support duty cycle tracking and correction mechanisms to correct for timing distortions caused by either the clock distribution network or the delay line itself. And lastly, the entire delay locked loop architecture should be fully-synthesizable, without the need for manual layout (e.g. for P-Cells in AD-DLLs), for quick and rapid deployment across several process technologies. This implies that the delay line should be inherently immune to process variations that may arise from digital design flows and automatic place-and-route. Since the DLL is meant to go in an always-on channel, the lock-time is not of focus.

6.3 Review of Prior Art Delay Lines and DLLs

DLLs and delay lines have a rich design literature. DLLs can be classified in to two categories, analog DLLs such as [72, 73] , and digital DLLs such as [51, 74, 75, 76, 77]. Analog DLLs generally use one or more charge pumps connected to voltage controlled delay lines to control the delay. In contrast digital DLLs use a binary word to control the delay of the delay line. Digital DLLs (and more recently All Digital DLLs, ADDLLs) are increasingly preferred due to their smaller area, ease of technology migration, lower power, and quicker locking times. The DLL in [51] uses an 8 phase coarse delay line (that can fit a single cycle), followed by a 16-bit phase-interpolation based fine delay line. This gives the DLL fine-grain control over the output phase allowing a communication speed-up of up to 8Gbps per trace. Both the coarse and the fine delay line architectures are sensitive to matching errors and require careful layout. Another DLL targeting lower area is [74] which uses two identical half-delay cells, each one containing a current-starved inverter followed by a thermometric coarse-delay loading capacitors, and thermometric fine-delay loading capacitors. This DLL uses a simple

binary control scheme and a (successive approximation register) SAR for fast locking. The DLL lacks the capability to track the duty cycle and the delay cells are highly prone to mismatch errors and need careful layout to keep a low DNL. The DLL in [75] proposes a looped coarse-delay, one for each edge, followed by a common fine-delay, which allows for a very wide frequency tuning range and low area. However, the architecture has limited runtime range of delay adjustment due to potential DNL while switching between coarse-delays. The coarse-delay and the fine-delay need to be carefully matched in layout to lower this DNL. In addition the DLL has limited duty-cycle correction capability due to a unified fine delay line. In [76], the delay line comes close to being synthesizable (except for the layout of the fine TDC), however this comes at the cost of higher silicon area (due to the large number of shift registers), as well as low frequency tuning range (*f-ratio*). The DLL in [77] focuses on making a dead-zone free phase detector and achieves an impressively low area. It uses an MSAR to achieve fast locking. However, this architecture has the lowest *f-ratio* which is prohibitive to portability. Even though ADDLLs present technology migration as a feature, most implementations such as [74, 51, 75, 77] use custom designed blocks (not robust to process variation) to convert the digital word to an analog delay, reducing their portability. In order to address these limitations simultaneously, a new, variation-robust, fully-synthesizable DLL architecture is needed, which targets low area, low power and wide frequency range for use in rapid deployment of fine-pitch I/O channels. Such an architecture should only use standard cells and the standard digital flow without the need for custom layout and matching.

6.4 Proposed Synthesizable Delay Line

6.4.1 Synthesizable Building Blocks and Trade-offs

For the delay line to be synthesizable, each of its sub-blocks needs to be synthesizable. Each sub-block should therefore be a product of common standard cells found in all digital synthesis libraries. If constructed using only standard cells, the digital place-and-route (P&R) process can be used to do their layout. However, digital P&R tool-flows do not take into account considerations such as delay-matching or capacitance load-matching (to within picosecond margins) which are critical to the operation of the delay line. Such critical operational sub-blocks, that are most susceptible to individual layout variation, can be dealt with using hierarchical layout. One such block is the differential delay element (DDE) which is used to create the basic thermometric delay adjustment unit. It comprises of 4 standard cells, 3 of which are inverters and 1 is an inverting multiplexer (with output drive). The DDE has an inherent delay through both of its paths, with one path being slightly capacitivly loaded by an appropriately sized inverter to create an extra delay. The extra delay is small and is the resolution of the DDE. In order for this arrangement to work all gates should be in close proximity to each other with minimal capacitive loading of the lines. To achieve this close proximity, such sub-blocks have their P&R done first, with higher layout hierarchies instantiating copies of such sub-modules. For the DDE, this approach allows for extremely consistent differential delays with very low variation amongst several instances. Instance-to-instance variation with delay step is the differential non-linearity (DNL) of the delay line. Higher level P&R placement of DDE instances don't matter as much, as the mismatches due to layout are shared amongst both paths (slow and fast). The trade-off of this approach is that the base delay of the delay line (constructed solely through such

DDE instances) is slightly larger resulting is lower operational frequencies or lower frequency tuning ranges - a problem that would be dealt with next.

Another critical block that is susceptible to layout is the phase detector (PD), in which the mismatch in the clock and data sampling times can be exacerbated by the additive capacitance due to layout. The same approach as before is applied here and the PD P&R is done hierarchically as well. In the flip-flops used in the PD, the clock's sampling point is often not centered w.r.t. data with a few picoseconds of error. This error is minimized by adding inverter gates to serve as added capacitance's for delay correction. For the current implementation, this adjustment has allowed for the PD error to be $<$ 3ps after layout. Though analog layout implementations get much better matching across all corners, they sacrifice synthesizability and by extension, portability. With better matching, analog implementations can target much higher frequencies in the 10's of GHz ranges. However for this delay line design, such a trade-off is acceptable since it can be seen from Equation 6.3 that the PD error is an insignificant part in the total phase error as the design is made to operate at sub 4GHz ranges.

Another hierarchical P&R block is a fast, 2-bit, graycode counter. Since this counter needs to operate at $\sim$10GHz (or even higher for advanced node implementations), it's standard cells need to be in close proximity to each other during layout, which is again done by running P&R on this block first before declaring it's instance in hierarchical modules. Analog implementations of high-speed counters can run at much higher frequencies, however for this application, even a synthesized 2-bit counter has plenty of frequency headroom.

To make a programmable delay line out of DDEs, two structures are commonly used. Figure 6.5 shows Binary-weighted (BW) and Thermometric Delay Line (TDL) configurations. BW delay lines have a simpler control scheme due to lack of a decoder, and they can

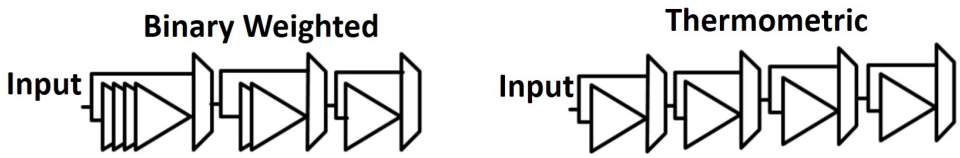


Figure 6.5: Binary-weighted vs Thermometric delay line configurations.

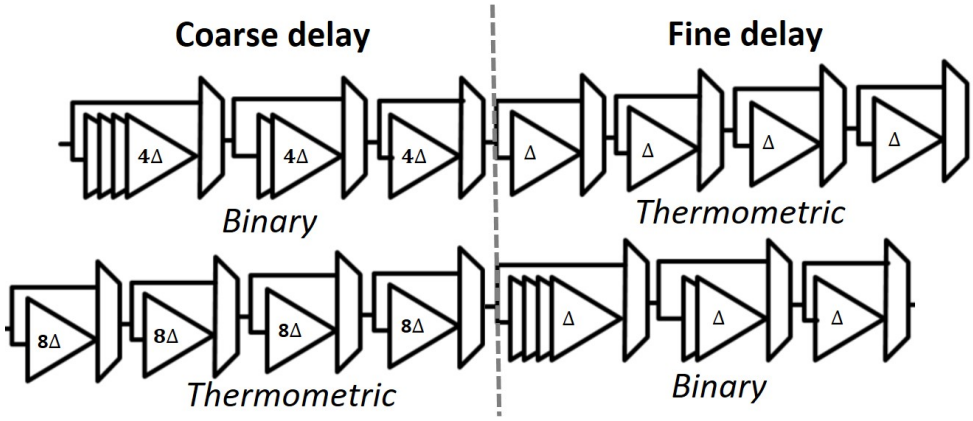


Figure 6.6: Combination implementations for full delay lines.

accommodate higher frequencies due to their low (minimum) base delay. However BW delay lines suffer from large DNL due to PVT variation based mismatches between DDEs. Larger the number DDE delays, larger the mismatch. Reducing the mismatch requires careful hand layout of the DDEs as well as process control. In contrast, TDLs have low DNL < 1 LSB since only one successive LSB element is switching state per adjustment. This means that the DNL is limited to element-to-element mismatch. TDLs are susceptible to large INLs and suffer from larger area due to decoders. In order to reduce INL as well as decoder area, the decoder is often divided into a row decoder and a column decoder, accompanied with careful layout. Since in both configurations the total delay is controlled by the number of elements rather than voltages or currents, both architectures suffer from large area as the frequencies get smaller and the delays get larger. It should be noted that this limitation is present in digital delay lines where the number of discrete elements determine the range of delay tuning.

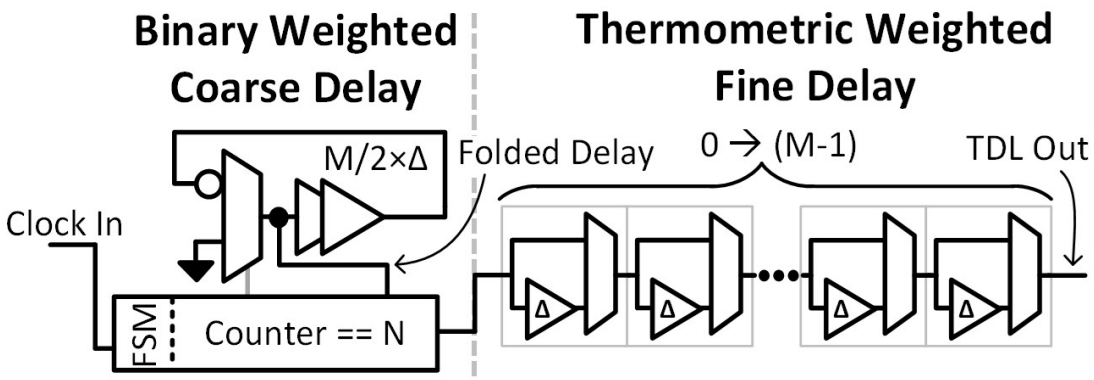
Combined implementations of the two configurations shown in Figure 6.6, help to alleviate some of the effects associated with each individual type of delay line. Here the delay line is divided into coarse and fine units, where the entire sum of fine delays is carefully matched (in design and P&R) to a single coarse delay unit. By carefully selecting the number of elements and limiting the frequency range of operation, the INL and DNL can be kept reasonably small. In cases this is not enough, the delay line is calibrated before operation to reduce INL and DNL even further. To support larger delay ranges, often the coarse delay line is set during initialization, and only the fine delay line is used at run-time to minimize DNL. This effectively divides delay ranges into discrete sections of continuous adjustment intervals, and jumping between intervals is not done at run-time due to the high DNL cost associated with it. A combination of the above techniques is used to keep INL and DNL low during operation. However the combination architectures, similar to their individual counterparts, suffer from large area as the frequencies get lower and the delays get larger.

6.4.2 An Oscillator based Folded Delay Line with Low INL and Small Area

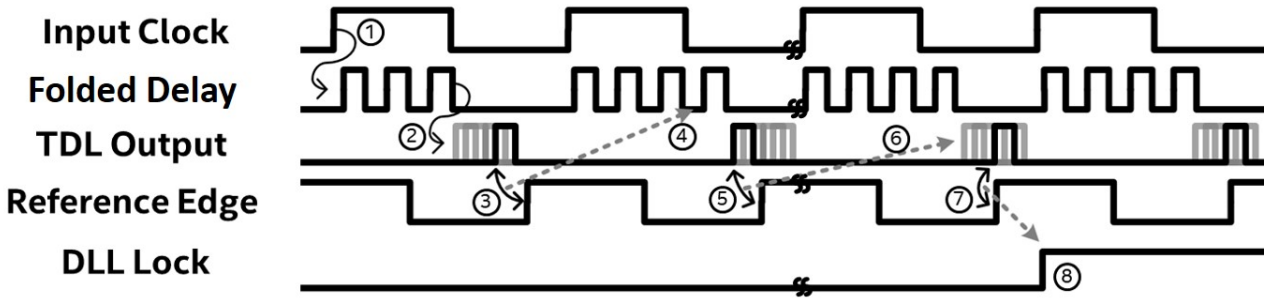
INL arises from PVT mismatches in the different coarse delay elements. The ideal way to solve this would be to some how reuse the same coarse delay element to create delays. This reuse of the same element would eliminate PVT effects over the produced delay, given the reuse time is short enough for the voltage and temperature to be stable. In addition, reuse of the same coarse delay element would drastically reduce the area of the delay line since only one delay element would need to be synthesized. The way to achieve this reuse is through making the delay element fold back onto itself effectively making it into an oscillator with a period twice that of the underlying delay element. After the desired number of oscillations

(akin to the desired number of coarse delay elements) are finished, the delay is passed onto the fine delay line stage, which is constructed from a TDL to keep the DNL low as shown in Figure 6.7a. A TDL made from DDEs makes the DNL robust to placement variation of the individual DDEs since the time-of-flight delay between two DDEs is shared for all TDL delay words. Assuming the TDL's tuning range is perfectly matched to the coarse delay period, large delays can be traversed without DNL or INL. The delay adjustment mechanism of a DLL built from this delay line can be seen from the timing diagram in Figure 6.7b. At (1) the input edge is detected and the folded delay oscillator starts oscillating. At (2) when the accompanying counter reaches the desired count, the oscillator is turned off, and the edge is passed to the TDL. At (3) the TDL delay is incrementally increased as the final delay does not match up to the required reference edge, as measured by the PD. If the TDL saturates, the coarse delay count is increase by one and the TDL is reset to zero delay at (4). The process of checking the phase (5) and adjusting the TDL (6) continues till (7) when the output phase matches the reference, at which point the DLL assumes a locked stance (8). A similar mechanic is used for decreasing delays where the coarse count is decremented.

Though this mechanism solves the area as well as the coarse-range INL problems, its downsides include the inability to process multi-cycle edges at the same time as well as the capability to process only one of the two edges of the clock (positive or negative) but not both. In addition, a synthesized version of this delay line would be subject to large PVT variations between the folded delay period and the effective range of the TDL. Erring on the side of caution the TDL range would need to be, by design, larger than the oscillator's period to make the design functionally able to lock. This mismatch-by-design would introduce large non-monotonic ($>$ 1 LSB) DNL errors to prevent PVT based large ($>$ 1 LSB) monotonic DNL errors at the coarse and fine switching boundary - a catch-22. Note that monotonic DNL errors $>$ 1 LSB can cause delay regions where no lock can be achieved rendering the



(a) Coarse folded delay with fine TDL architecture.



(b) Timing diagram for the proposed delay line.

Figure 6.7: Proposed oscillator based folded delay line.

DLL non-functional.

6.4.3 Eliminating DNL by an Additional TDL

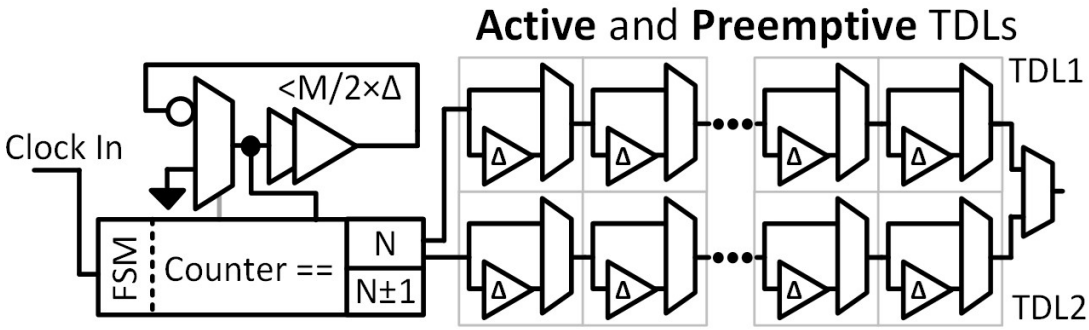
In order to tackle the DNL problem in the proposed architecture, a modification is proposed to the second stage, where an additional preemptive TDL is introduced along side the main active TDL. The counter and control is set to route the N^{th} edge to one TDL and depending on the adjustment direction, the $N\pm1^{st}$ edge to the other TDL. Each TDL has an over-designed range w.r.t. the folded delay period. Figure 6.8a shows the proposed modification. One of the the two TDLs is connected to the output at a time, determined by an output Mux. The idea is to use the second TDL to preemptively lock to the first TDL before a

coarse-delay switch from N to N±1 is made.

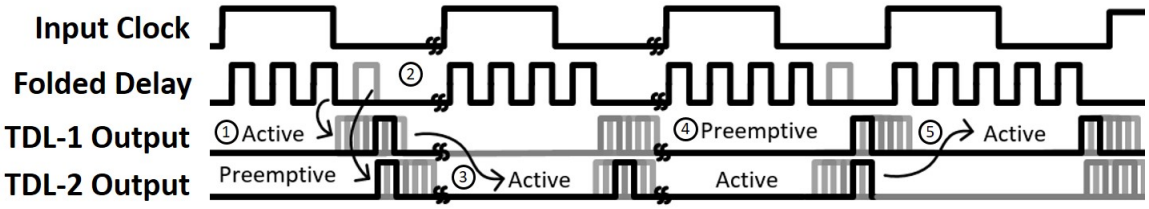
The mechanism is shown in the timing diagram in Figure 6.8b. Assuming that TDL-1 is the currently active TDL and the delay needs to be increased to achieve a lock on the DLL, the control increases the delay on TDL-1 derived from the N^{th} edge at (1). As TDL-1 gets close to being saturated, the control preemptively locks TDL-2 with the $N+1^{st}$ (coarse-delay) edge at its input, to TDL-1 which has the N^{th} (coarse-delay) edge at its input (2). This ensures that both TDLs have the same output phase to within 1 LSB of each other. Since TDL-2 has more headroom in the increasing direction, the control switches to TDL-2 at the output making it the active delay line (3), while TDL-1 is retired to an inactive state at (4). When the switch is made the folded delay counter is increased as well. As the delay is increased further, at some point TDL-2 saturates as well. At this point (5), TDL-1 is preemptively locked to TDL-2's output before the switch is made again. This allows for seem-less ($<$ 1 LSB DNL) switching between the coarse folded delay and fine TDLs. The mechanism also grants inherent immunity (at a marginal cost of slew rate) to mismatches in the folded delay period verses the TDL's total range. In addition, it provides immunity against the TDL-1 and TDL-2 mismatches to within 1 LSB. This means that the TDL layout can be decoupled from the folded delay line layout, making the architecture robust to the digital P&R tool-flow.

6.4.4 Increasing Frequency Tuning Range

In order to keep the phase detector control mechanism small and simple, harmonic locking detection is not supported and the design assumes the absence of multi-cycle edges in the delay line. This limits the maximum frequency range of the DLL to the (minimum) base delay of the delay line (the phase detector can be made to be more complex to account for



(a) Common folded delay with Active and Preemptive TDLs.



(b) Timing diagram for the 2-TDL delay line.

Figure 6.8: Additional preemptive TDL to reduce folded delay crossover DNL.

multi-cycle phases in the TDLs if desirable). At minimum, the base delay of the architecture in Figure 6.8 occurs with a zero count in the coarse stage, and the sum of the minimum delay of the DDEs from one of the TDLs. This delay is highly dependent on technology and will scale down if the DDE is implemented in advanced nodes.

For an implementation in GF 22nm at nominal voltage (0.8V) with 31-TDL stages, the minimum delay from the input of the delay line to the output is 454ps corresponding to a maximum operational frequency of 2.2GHz. The frequency can be improved to 2.8GHz with a 10% increase in voltage. The minimum frequency supported by the delay line comes from its highest supported delay. For this architecture, since the coarse delay stage is a high-speed oscillator attached to a counter, the maximum delay is $2^N * \phi_{Occi}$ where 'N' is the number of counter bits. Increasing the size of the counter exponentially increases the size of

the lowest achievable frequency. However the size and the speed of the counter are closely linked quantities, and increasing the size results in a lower speed counter, which translates to higher oscillator delay and a correspondingly large TDL. This leads to an overall larger area delay line due to its synthesizability requirements. In addition, a larger resultant TDL with more stages lowers the maximum operational frequency. In order to achieve a high frequency tuning range, the counter needs to be made independent of the speed of the oscillator.

To make the counter independent of the underlying oscillator's speed, two modifications are made to it. First, the counter is bifurcated into two parts, 1) a high-speed 2-bit grey code counter and 2) a cascade ripple counter (shown in Figure 6.9). Second the counter is made to count *down* from N to 1 and 0, instead of counting *up* from 0 to N,N$\pm$1. From the first modification, the grey code counter can be made to operate at very high frequencies since its length is fixed to 2 bits. The grey code counter is hierarchically synthesized and laid out first (with the oscillator) so as to keep its components in close proximity. The 2-bit grey code counter is needed to near-perfectly detect the count to 0 or 1 (in time w.r.t. the oscillator's edge). The MSB of this counter is $1/4^{th}$ the oscillator clock rate and is provided as a clock to the first stage of the ripple counter. Subsequent ripple counter stages take as their clock the output provided by the previous ripple counter stage. In order to make a larger counter more ripple counter, stages can be added with each stage doubling the frequency range. The higher ripple counter bits take a longer time to propagate their state to the grey code counter (weather their count has hit zero or not). The grey code counter only detects a count of '1' or a '0' if the ripple counter bits have all gone to zero. This check is facilitated by an OR-gate ripple tree running in the opposite direction to the counter. The total delay for the MSB to influence the gray code counter from the moment of oscillation of the oscillator is $N*t_{Clk\rightarrow Q}+N*t_{OR}$. However due to the counting down nature of the counter as seen in Figure 6.9, the higher order bits go to zero first, and as such have more time to

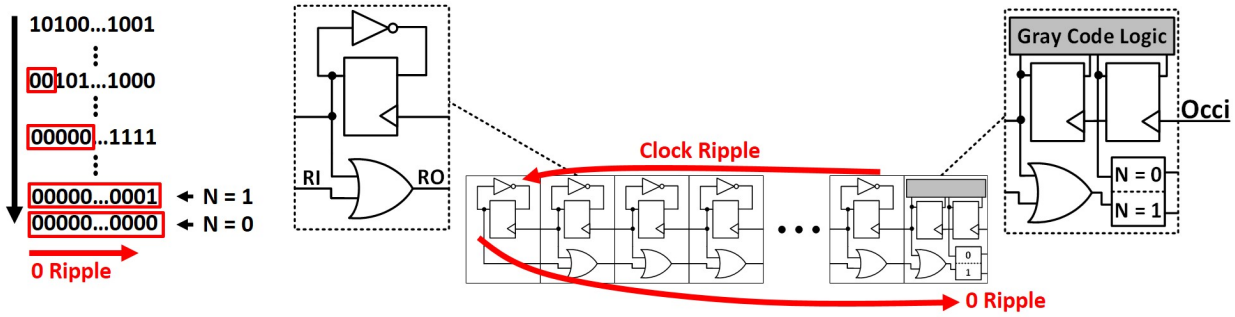


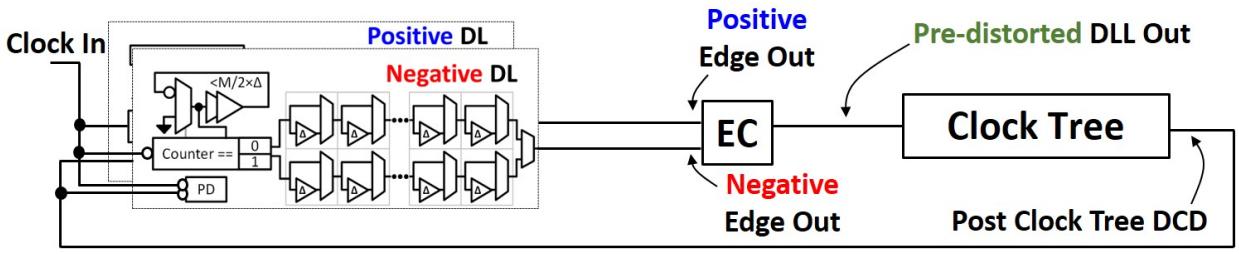
Figure 6.9: Folded delay oscillator counter architecture.

propagate their effect to the grey code counter. This allows the ripple counter to be extended indefinitely and completely decouples the counter size to its speed, as the controller is only interested in a count of '0' or '1'. With a decoupled counter, the coarse delay can support a very large number of folded delays corresponding to very low frequencies, increasing the frequency tuning range independent of oscillator speed.

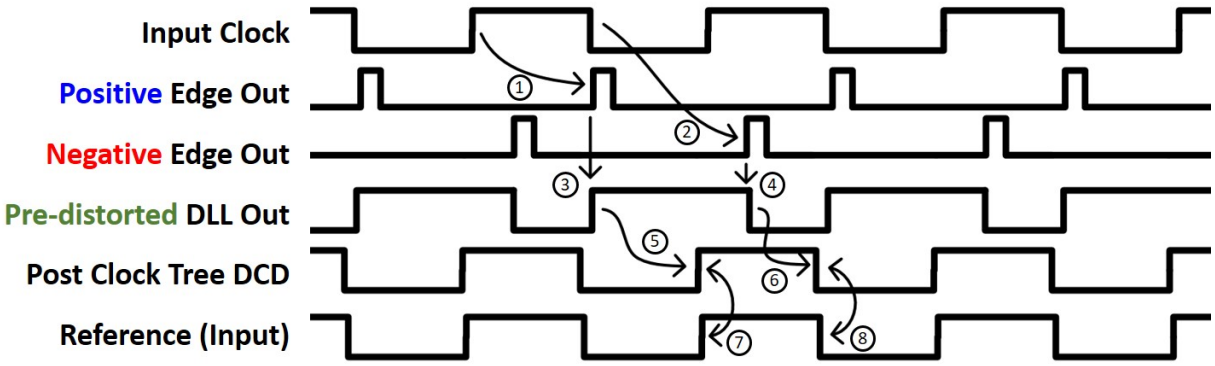
6.4.5 Duty Cycle Correction by Dual-Edge Control

A drawback of using a folded delay line architecture is that an edge renders the delay line as occupied - in the worst case, for the entire duration of the cycle. One way to solve the problem is to implement an identical circuit that works on the negative clock edge. This results in two independent edge tracking delay lines working simultaneously on the same input clock. The solution increases the area of the design, however because of the folded coarse delay, there is enough area headroom to accommodate another delay line. Figure 6.10a shows the 2 delay lines working in tandem in a clock-deskew DLL. The output of each delay line is a pulse in time which the edge combiner (EC) combines to produce a full clock waveform. The negative delay line's PD takes an inverted reference clock as well as an inverted feedback clock w.r.t. the positive delay line's PD.

The timing diagram in Figure 6.10b assumes the delay line is connected in a delay locked loop with a 50% duty cycle reference input clock. It can be seen that at (1) and (2), the individual positive edge and negative edge delay lines are independently delaying their respective edges by the appropriate amount. The edge combiner at (3) and (4) is converting the pulses form the delay line into a clock. Because of the independent, feedback, delay-lock of each edge, this output clock is not going to be at 50% duty cycle. Instead, the clock will have a distorted duty cycle. At (5) and (6), the distorted duty cycle clock goes through the clock tree that adds its own duty cycle distortion. The added distortion however, cancels out the pre-distorted DLL's output. This happens naturally due to the feedback nature of tracking two independent edges. The feedback phase detector and control forces the positive (7) and negative (8) edges to match the reference. In this way the delay line can perform duty cycle pre-distortion (DCPD) on the clock to cancel out any further distortions in the loop till the feedback point.



(a) Independent dual-edge delay line implementation.



(b) Timing diagram for duty cycle pre-distortion and DCC.

Figure 6.10: DCC using dual-edge duty cycle pre-distortion.

6.5 Complete Implementation of the Proposed Delay Line

The resultant complete architecture is shown in Figure 6.11. Here the synthesizable DDE is also shown as implemented in the fabricated version, using a hierarchically synthesized inverter pair, with one branch loaded by the gate of an appropriately sized inverter's gate capacitance to give a differential delay. The relative sizes of the inverters can be used to control the resolution of the delay line. For the implementation in GF 22nm, the resolution is kept at 5.4ps. 31-element TDLs are used with a total of 4 TDLs per delay line implementation. The number of DDEs in the TDLs are a function of the coarse folded delay and the resolution as shown in Equation 6.4. With advanced technology node implementations, the ϕ_{Occi} can be reduced as can the delay resolution. The folded coarse delay line is kept at

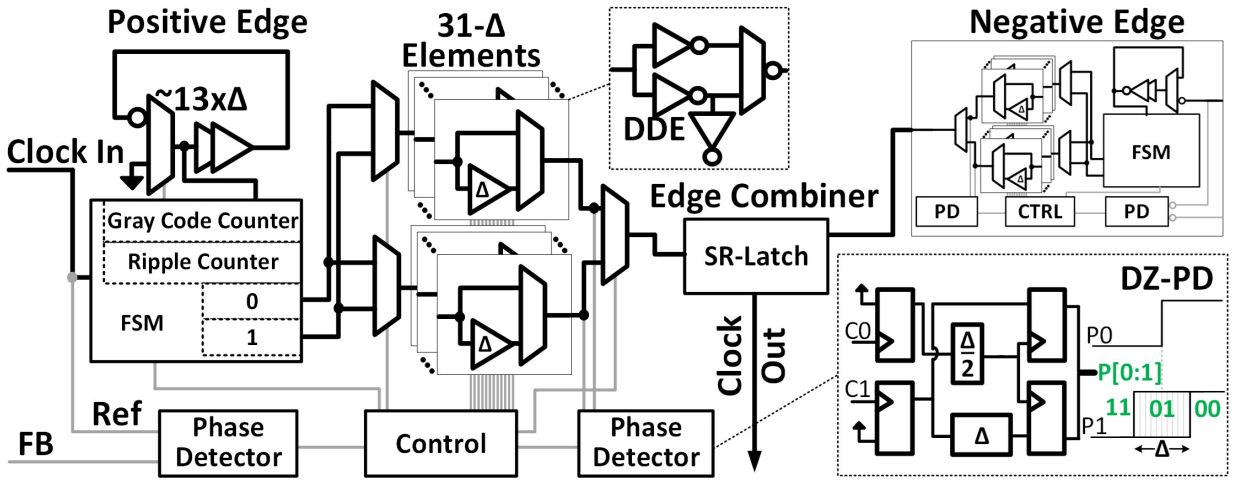


Figure 6.11: Simplified dual-edge version with control details omitted for clarity.

roughly 26× the resolution (Res_{DL}) to give headroom for PVT variation due to synthesis. Synthesizing the folded delay line can result in higher than expected delay as well as synthesizing the TDL's can result in TDL mismatches or lower than expected delay, hence the extra headroom. A 3-Res_{DL} margin is found to be sufficient for this technology over PVT and several synthesis runs.

$$Stages_{TDL} \geq \frac{2 * \phi_{Occi}}{Res_{DL}} \tag{6.4}$$

$$Stages_{TDL} \triangleq Number\ of\ TDL\ Stages \qquad \phi_{Occi} \triangleq Oscillator\ Delay$$

$$Res_{DL} \triangleq TDL\ Resolution$$

The edge of the input clock triggers the oscillator and counter. The counter counts down from N, where N is specified by the controller. Each TDL can select from either a count of '0' or a count of '1' at the end of the counting mechanism. At this point the counter is also

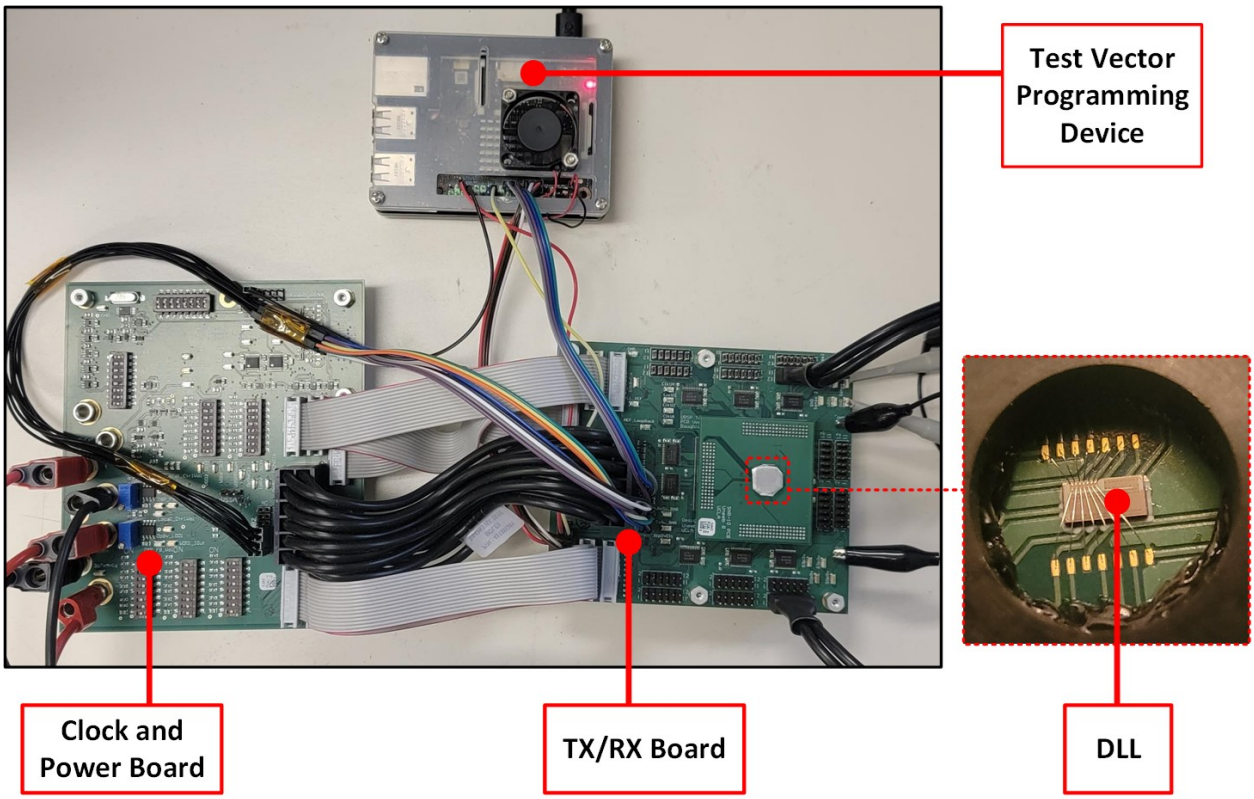
pre-loaded, ready for the next cycle. The counter has 14 usable bits allowing a coarse delay tuning range of 16,384 counts. This leads to a minimum frequency of 471kHz. The design has two phase detectors - one PD is used internally to lock the output of the two TDLs and the other PD is used to lock the reference to the feedback clock. The PD has a dead-zone of 1 LSB which allows the DLL to save power (11%) by reducing the usage of the control circuit. The control circuit contributes most of the power at higher frequencies where the delay is lowest. For known noisy environments, the locking and de-locking threshold 'K' is programmable. This allows for the DLL to stay locked up until 'K' consecutive de-locks in the same direction are detected by the PD.

The control is responsible for keeping track of the coarse delay count N and seem-less switching between the coarse delay elements using TDL pre-locks. The bandwidth of the control is programmable from $\frac{1}{2} \rightarrow \frac{1}{20}\times$ the clock rate, and can help reduce power consumption drastically at higher frequencies. The bandwidth is kept at $\frac{1}{3}\times$ for the duration of testing. Although the negative edge control can be thought of as independent, it allows for the shared control of both delay lines by a single control word. This is useful in cases where the duty cycle at the output is not of importance and can save excess control power buy turning off the negative edge control. Though not necessary for the operation of the delay line, the DNL discrepancy between the coarse delay and TDLs can also be programmed and calibrated in the control to allow for the largest slew-rate. To achieve a fast first-lock in the delay line, the control performs a two step search. In step one, the coarse delay counter counts the number of folded delays required to exceed the reference edge. This step lasts 8 clock cycles. In step two, the control performs a binary search on TDL-1 making it the active line. This step lasts for 18 clock cycles (6 control iterations) due to the artificially reduced bandwidth. Two more control iterations (6 clock cycles) are use to switch the TDL control back to its regular adjustment state and verifying the DLL is locked. In total the

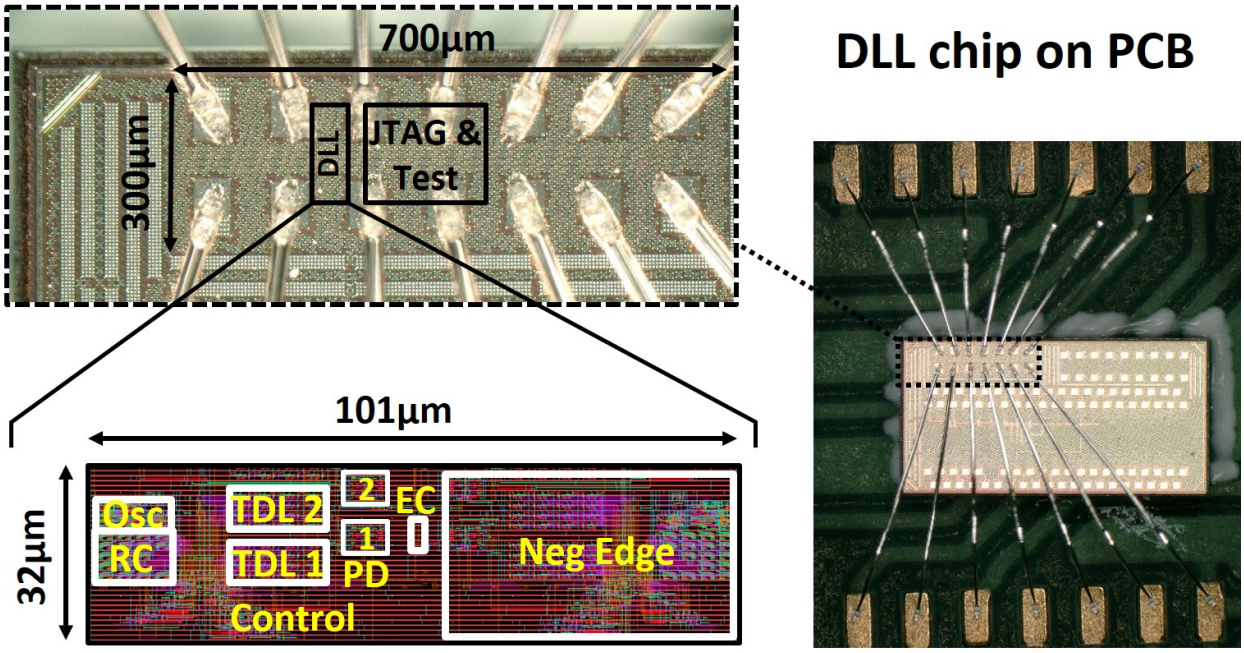
DLL is able to lock in 32 clock cycles.

The edge combiner is implemented as an S-R latch and converts incoming positive and negative edge pulses to clock edges (as shown in Figure 6.10b). The negative edge implementation is identical to the positive edge implementation except for its inverted clock and feedback inputs. Although implementing the negative edge as a separate delay line does take up extra area, that area is far less than the area gained by switching to a folded delay line architecture (that requires the implementation of 2 folded delays). The dual-edge duty cycle correction was tested and verified up until 10%-90% duty cycle distortion in the GF 22nm implementation. In general, the minimum correctable duty-cycle is determined by the mismatch in the absolute delay of the delay lines while it is operating at the highest frequency. This mismatch is technology dependent.

Figure 6.12 shows the test setup as well as delay line block breakdown. The testing circuits on board are accessible through JTAG. The JTAG frames are sent over from a Linux based programming device for automated device testing. Zooming into the DLL blocks in Figure 6.12b, it can be seen that the majority of the area is dedicated to control and decoupling capacitors, and the main delay line components occupy a small area. Particularly interesting is the area occupied by the coarse-delay mechanism of the oscillator and ripple counter (combined) to the fine-delay mechanisms of the TDLs. Due to the folded nature of the delay, the coarse delay section occupies less than half the area of the TDLs. The area saving is significant enough to bring down the total area of the design to 0.0032mm^2.



(a) DLL test setup.



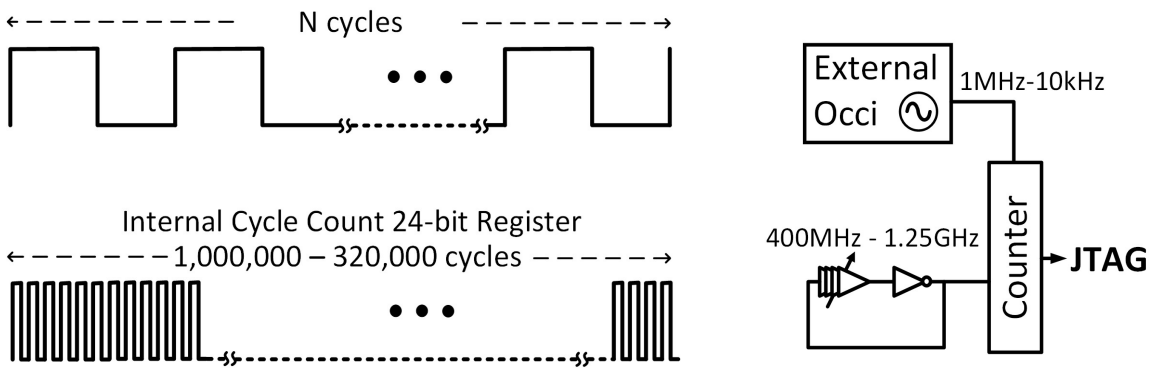
(b) Chip photo and delay line block breakdown.

Figure 6.12: Test setup and chip photo of the proposed delay line.

6.6 Measurement and Results of the Delay Line and DLL

6.6.1 On-chip Measurement Setup

In order to measure the differential delay of the individual cells in a reliable and precise way, a measurement circuit was build on the chip. The measurement circuit was built to measure picosecond level accuracy and automate delay measurements. Most delays on the chip are designed to be connected as an oscillator in self feedback including the individual TDLs and the complete delay line. From Figure 6.13 it can be seen that connecting the delay under measurement as an oscillator allows for the temporal amplification of a small delta in that delay. Over a million oscillations, even picosecond measurements get reliably amplified. An external clean reference is used as the start and stop mechanic for an internal high-speed counter responsible for counting the internal looped delay's oscillations. The internal counter is a 24-bit whereas the external start and stop counter is programmable with its count ranging form $1 \rightarrow 32$ cycles. The setup can reach a measurement precision as low as 0.03ps. The final count of the counter after measurement is pulled to the programming device via JTAG. The oscillation based measurement approach acts as a windowed-averaging low-pass filter for the phase noise in the the measurement setup and suppresses the high frequency thermal and voltage jitter. In order to further reduce the noise, specifically targeting low frequency noise, the measurements are repeated in an automated fashion every 5 minutes over a day and the results averaged out.



(a) External vs. internal count comparison.

(b) Measurement counter.

Figure 6.13: On chip measurement setup.

6.6.2 DNL and INL Results

Figure 6.14 shows the measured DNL and INL of a test chip. The LSB is 5.4ps and both DNL and INL are measured as a fraction of the LSB. In total there are 4 TDLs, 2 for the positive edge control and 2 for the negative edge control. All measured INLs are within 0.5 LSB. The DNL is even lower due to the hierarchical layout of the differential delay element. The measurement spans 32 delay steps which is the length of the delay line.

Figure 6.15 shows the DNL and INL measurements over a larger contiguous delay step interval of 200 steps. During this measurement only one TDL is used and the preemptive locking mechanism as well the second delay line is disabled. Figure 6.15a has overlaid on top (in red) the total delay measured as the delay is increased. It can be seen that without the DNL correction mechanism, the mismatch in the TDL range and the coarse folded delay is large (~10 LSB). This is due to mismatch by design as well as layout based PVT variation due to the design being synthesizable. For the INL, in Figure 6.15b, the deviation is measured from the average ideal (linear) line connecting the end points of the delay steps and delays. The INL can be seen to be ~5 LSB. It should be noted that even though the INL is not

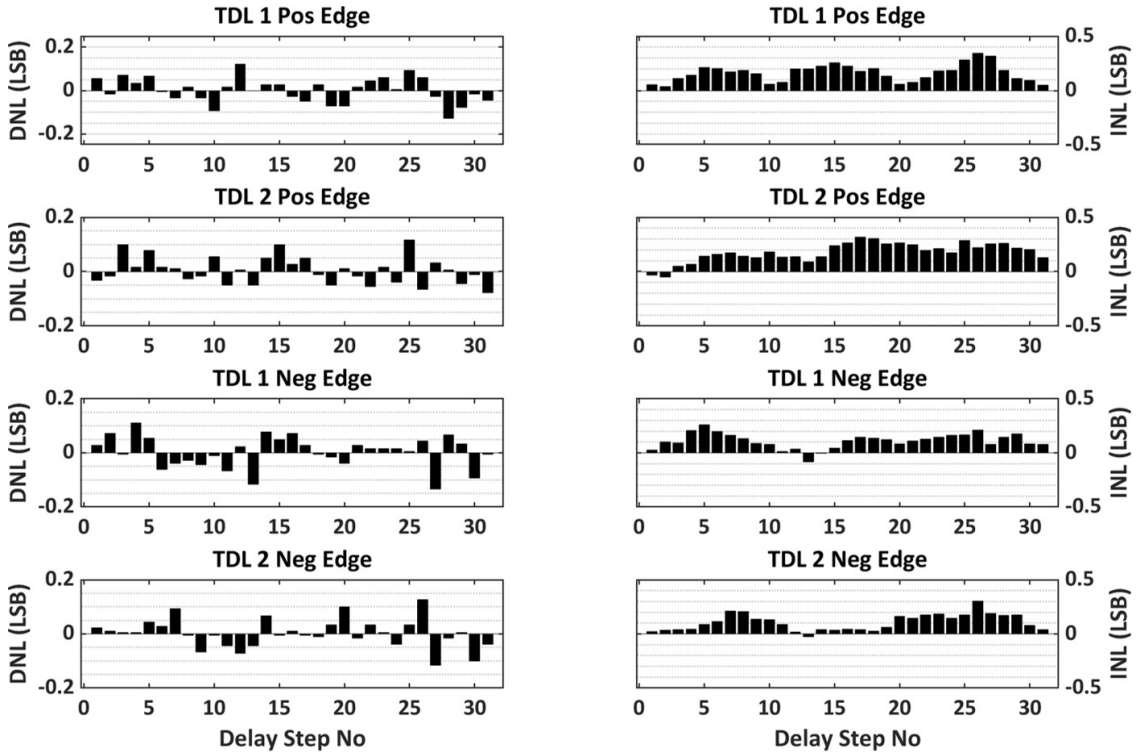
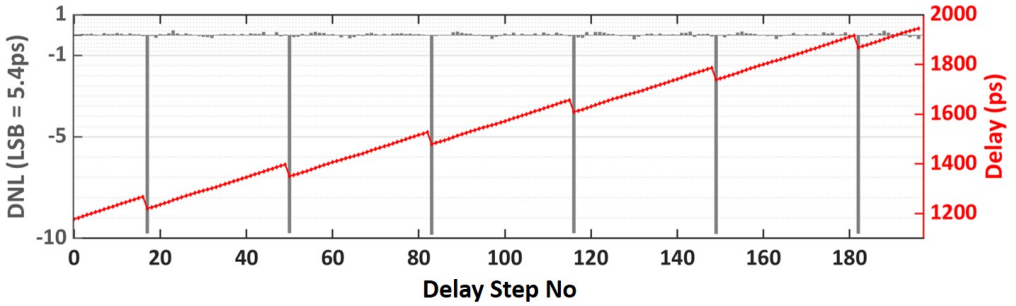
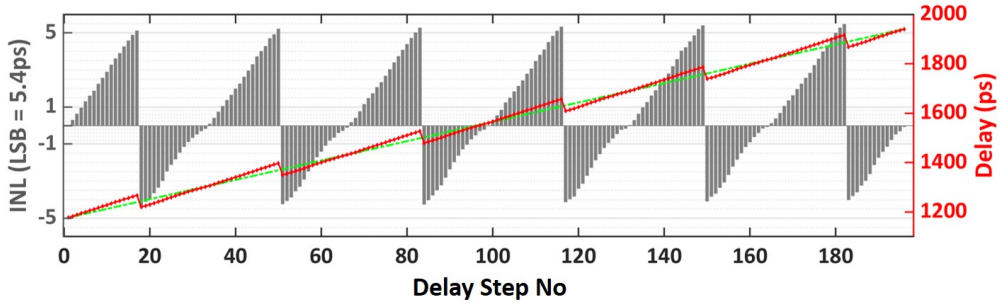


Figure 6.14: Measured DNL and INL in TDLs.

small, it is bounded by the maximum mismatch in the folded delay and TDLs. This is because long delays are achieved by folding them in the coarse stage which has no process variation w.r.t. to itself during different iterations of the fold. This ensures that the ideal line (green) is no more than than a fixed INL away from the actual delay (red). Figure 6.16 shows the same measurement but with both TDLs operational, with the active-preemptive locking mechanism. The mechanism can be seen to ensure < 1 LSB DNL while switching between the coarse folded delays and the TDLs. The INL drops to < 1 LSB as well. The 1-TDL DNL and INL from Figure 6.15 are re-plotted in the background in Figure 6.16 for reference and comparison. From the control perspective, the same number of steps can cover a larger range of delays in the 2-TDL approach as opposed to the 1-TDL approach due to the the 1-TDL having to constantly cover up the mismatched delay after every folded delay switch.

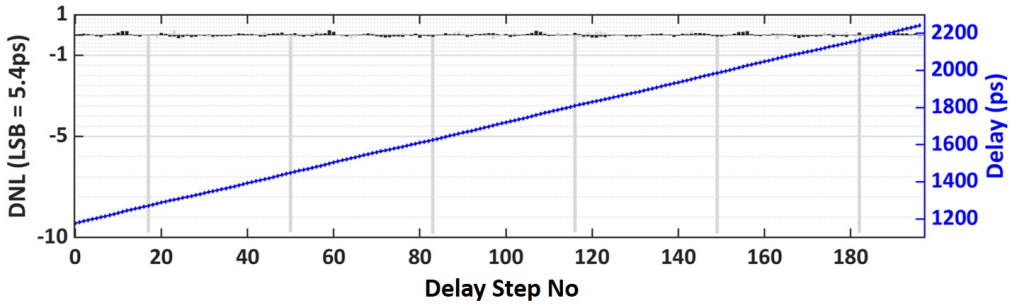


(a) DNL measurement over a contiguous 200 step interval.

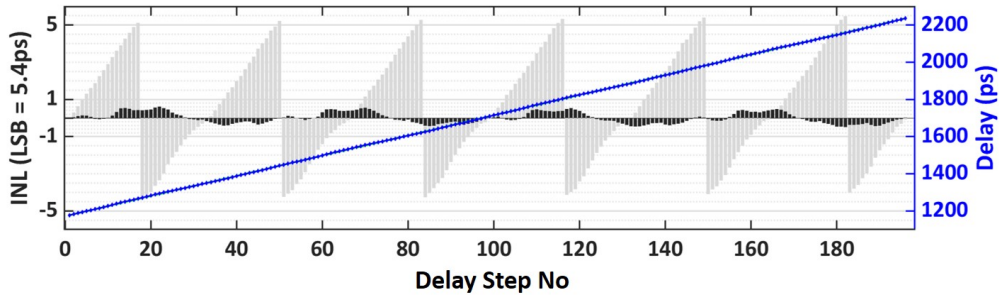


(b) INL measurement over a contiguous 200 step interval.

Figure 6.15: DNL and INL measurements for 1-TDL operation.



(a) DNL measurement over a contiguous 200 step interval.



(b) INL measurement over a contiguous 200 step interval.

Figure 6.16: DNL and INL measurements with 2-TDL operation using preemptive locking.

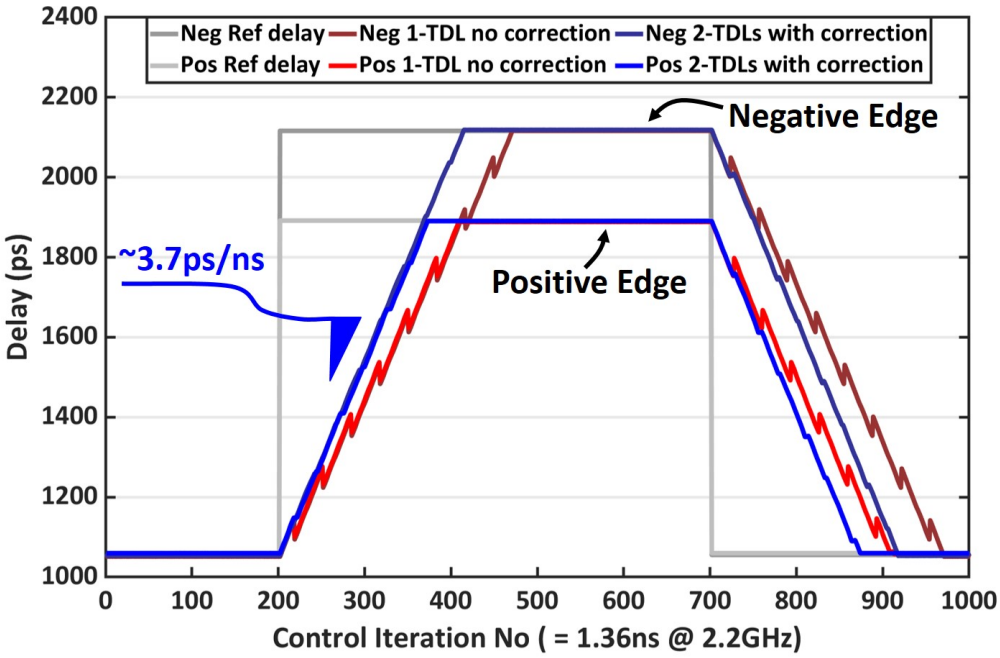


Figure 6.17: Dual-edge control comparison for step response.

6.6.3 Measured DLL Dynamics

The previous measurements show static characteristics of the control and delay line. In order to observe the dynamic characteristics of the delay line, it is connected as a DLL. A phase step is applied to a settled delay line state and the corresponding corrections are observed over time and plotted in Figure 6.17. The corrections cannot be observed in real-time since the delay line needs to be disconnected from the main control loop and connected to the measurement circuit, however, the functional progression of the control scheme does not change regardless of operational speed. Consequently the x-axis is measured in control iteration numbers as a unit of time. At the highest frequency of operation (2.2GHz) at $1/3^{rd}$ the bandwidth, each iteration takes 1.36ns. In order to further exemplify the independent edge tracking mechanic, two different phase steps are applied, one for each edge. This can be thought to model duty cycle distortion. From the figure, the DLL is able to adjust and track both delays independently with the negative edge settling higher than the positive edge.

The 1-TDL vs 2-TDL mechanisms are also compared. The red curves are for a single TDL without the preemptive locking correction mechanism. It can be seen that over a large step, the single TDL based delay line experiences large DNL movements. Similar movements can occur with run-time voltage and temperature variation in a statically referenced delay line. A large DNL, such as that, can cause missed samples while sampling. The blue curves show the dynamic response of the fully functional 2-TDL delay line. Here the DNL goes to zero, and before every coarse switch there is a small transition period where the preemptive delay line is locking to the active delay line. The average slew-rate for the 2-TDL mechanism is 3.7ps/ns. As long as the run-time delay variation is less than the slew-rate, the TDL transition period becomes indistinguishable from regular adjustments.

6.6.4 DLL Comparison Table and Discussion

Although the literature on DLL designs is diverse and rich, select state-of-art DLLs (with an emphasis on minimizing area) are summarized in Table 6.1. The performance of this work is compared to said DLLs. Other than high area and low performance FPGA implementations, dedicated synthesizable DLLs could not be found and as such are absent from the comparison. The DLL presented in this work is the first dedicated fully-synthesizable DLL. Other low area designs such as [75], [76], and [77] require manual design intervention in layout to minimize PVT based matching issues. The synthesizable implementation presented in this work is designed to be immune to DCD and placement mismatches, as synthesis is more prone to such issue. The active area for designs such as [76] and [77] seem small, but their lowest operational frequency is 1.65GHz and 1.5GHz respectively which drastically reduces the size of the delay line. [75] makes an effort to increase their $f - ratio$ while maintaining small area at the cost of limited run-time range corresponding to low DNL, and limited run-time

Table 6.1: Comparison of proposed synthesizable DLL with SoA DLLs.

Reference	[75] JSSC'15	[76] ISCAS'18	[77] TCAS-I'18	**[This work]**
Technology (nm)	65	65	130	22
Voltage (V)	1.0	1.0	1.2	0.8
Active Area (mm^2)	0.0153	0.02	0.0077	**0.0032**
Synthesizable	No	No	No	**Yes**
f_{min}, f_{max}	3MHz, 1.8GHz	1.65GHz, 7GHz	1.5GHz, 3.3GHz	471KHz, 2.2GHz
f-ratio	600	4.2	2.2	**4670**
Lock time (Cycles)	5	6	32	32
Power (mW)	9.5	7.1	7	**2.1**
Resolution (ps)	5.2	2.5	5.3	5.4
Full range DNL	> 1 LSB	-	> 1 LSB	**< 1 LSB**

range of duty cycle correction. This work achieves a much smaller area than all three designs with a much higher $f-ratio$ and large duty cycle correction range due to independent edge tracking.

The power consumed by the proposed delay line is lowest at 2.1mW running at 2.2GHz. The power is further reduced by ~9% in steady state due to the dead-zone in the phase detector. The lock-times for [75] and [76] are lower and therefore better than this work. This is because general DLL applications such as SerDes, benefit from quick locks as such power hungry IPs consumes idle power while the DLL achieves lock during wake-up (after a sleep cycle). For the application that this work is targeting, namely SNR-10, the channels are expected to run continuously till a system power cycle. Hence quick locking is not of importance to the intended application. Additionally, [76] can reach much higher frequencies of operation with much finer resolution compared to this work. This is at the cost of extra area due to large shift registers, as well as a very small run-time tuning range, for low DNL. With regards to resolution of the proposed delay line, the current resolution design-point is around 1.2% cycle error, and since the frequency is lower than [76], the resolution delay

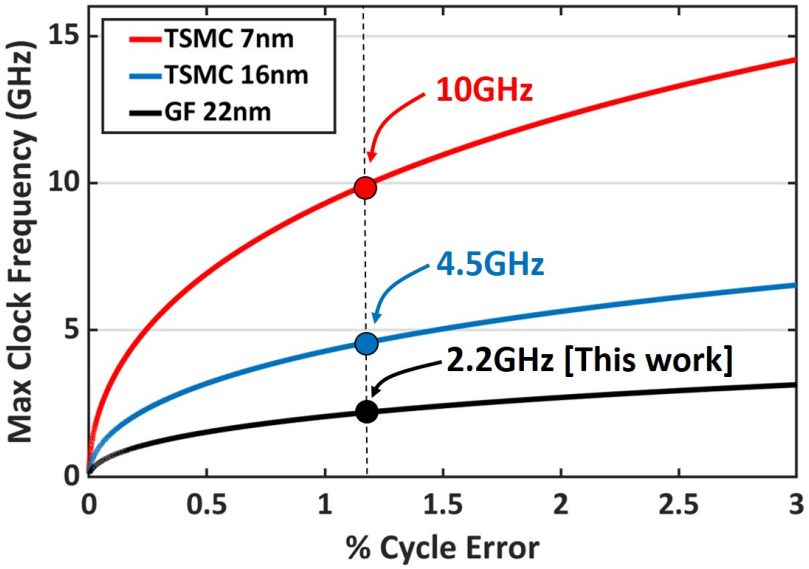


Figure 6.18: Synthesizability and frequency scaling in advanced technology nodes.

is higher (in terms of pico-seconds). This means that w.r.t. percentage cycle error, the proposed delay line has lower error. With regards to frequency, the sythesizable nature of the delay line and the DDE demands a lower maximum frequency than what would be otherwise achievable through conventional analog design and layout techniques. However, the target application of 'many parallel I/Os' rather than 'a few serial I/Os' doesn't require high frequency communication per pin. In fact, since the delay line is synthesizable, its speed scales proportionally with technology (all error thresholds equal). Figure 6.18 exposes this dependence. At the resolution error of 1.2%, an identical delay line as the one proposed will achieve 4.5GHz DDR speed in 16nm and 10GHz DDR speed at 7nm. This translates to a per-pin data rate of 9Gbps for 16nm and 20Gbps for 7nm. These rate far exceed (by over 2×) any internal data signals that those technologies might sport. Therefore it is well within reason to assume that the proposed delay line architecture, if implemented in any of the advanced technologies, will always be able to support the required maximum frequencies due to the parallel I/O nature of the application. Last, the proposed delay line ensures < 1 DNL across its entire delay range, which for such a wide delay range is a first.

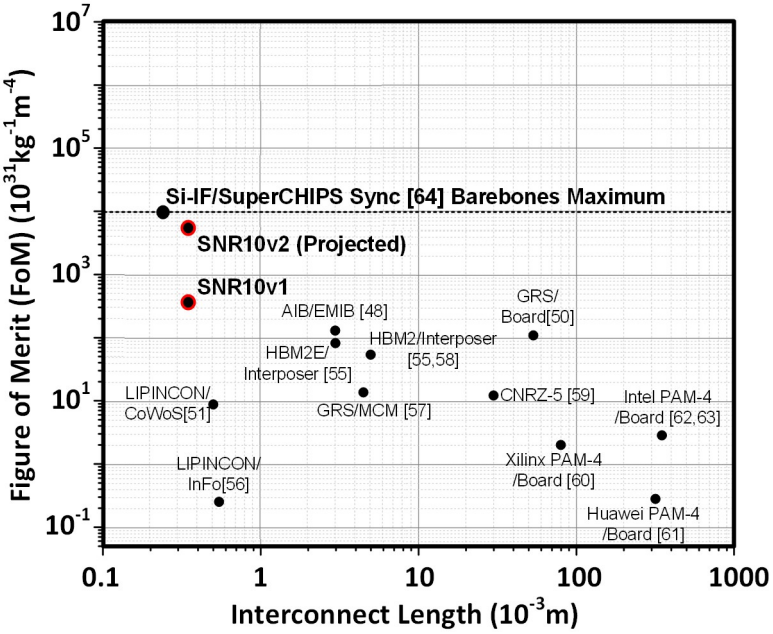


Figure 6.19: Projected maximum UCLA FoM for SNR-10v2 channel.

Intended for use in a future SNR-10 version 2 (SNR-10v2) channel, the DLL enables the use of DDR as well as higher speed clocks. The expected UCLA FoM [49] of the channel described in Figure 6.1 is shown in Figure 6.19. With more amortization of resources, a faster clock rate at DDR and a lower voltage to match, the DLL enables the SNR-10v2 channel to approach (with in 4×) the theoretical maximum performance of a bare-bones synchronous channel.

CHAPTER 7

Conclusion

The work presented in this dissertation addresses the balance between flexibility, scalability, and efficiency. This balance is achieved by leveraging domain-specificity in the design of the core and the interconnect. A multi-layer domain-specific interconnect design methodology is used to make a scalable and densely connected compute fabric. For lowering the interconnect area and compile-time, this work advances the understanding of multi-layer information flow in switchboxes, introduces methods of representation for these switchboxes, and presents compiler-oriented hardware minimization approach to designing these switchboxes. In addition, this work advances inter-dielet communication for the next generation of MCMs by demonstrating the first 10-µm pitch integration of a processor on the Si-IF using an energy-efficient, lowest area-per-I/O, die-to-die communication channel. Finally, to scale up the bandwidth of the channel, this work advances portable and synthesizable timing correction circuits for integration in MCM based dielet I/Os. Together these advancements allow for increasing the energy- and area-efficiency of flexible compute fabrics to near-ASIC performance ($\sim$5$\times$). Three chip tape-outs across two technologies and one integration technology are used to verify the results. Key techniques learnt during the development of this work consist of the statistics based interconnect optimization, the HVCC based switchbox optimization, the amortization based channel-area optimization, and the folded-delay based DLL's DNL and area optimization. These techniques allow for the development of efficient,

scalable, and reconfigurable architectures and serve to combat the rising cost of re-design in advanced technology nodes due to constantly evolving standards. This dissertation describes, in detail, the challenges encountered in the development of each of these techniques, and provides solutions to overcome these challenges. Below is a summary of the specific contributions of this work.

7.1 Summary of Research Contributions

- A coarse-grain domain-specific core design is presented for DSP applications that balances core granularity and core utilization to maximize efficiency. Connections are made 1-hop to minimize compile effort.

- A domain-specific, multi-layered interconnect network design methodology is presented using the connection statistics of a domain. Each interconnect layer caters to a different distance allowing $> 85\%$ coverage of the DSP domain with 3 layers. The network is delay-less, avoiding retiming at compile time, it works at 1.1GHz independent of the mapped algorithm, and has the *right* amount of flexibility for the domain. For longer distance connections, a registered layer 4 is proposed.

- A formal method to design a sparse, multi-layered switchbox is presented, based on a hyper-matrix representation which preserves path information. A new $\textcircled{\mathbf{R}}$ tensor product is proposed to concatenate simpler switchboxes to build the hyper-matrix. A novel HVCC based measure of routability is presented for a hyper-matrix switchbox. Using these tools, an optimal switchbox-search-space traversal method is presented that does as well as the base line brute-force search in terms of MCBF, and exceeds it in area-efficiency, resulting in a hardware-compiler co-optimized sparse switchbox design.

- A static compiler is presented which achieves < 1s compile times on the UDSP. The compiler leverages design provisions in the UDSP compute fabric, such as 1-hop core connections, 2-distance delay-less interconnect, and a disjoint-path switchbox with high I/O mapping probabilities, to achieve fast compile-times for moderate sized kernels. The static compiler is extended to a dynamic compiler and a further extension is proposed as RTRA to maximize active utilization of the array and provide accelerator as a service.

- Unidirectional I/Os, a light weight communication channel handshake, and a simple low-area redundancy-repair scheme are presented which allow for the I/Os to be more tightly integrated, in addition to increasing the yield of the assembly process.

- The first and only (as of the time of writing) 10-µm pitch protocol is demonstrated to work on a fine-pitch interposer, the Si-IF, with the lowest area per I/O of 137µm^2 (as of the time of writing). The SNR-10 channel is also demonstrated to be portable to GF 22nm.

- A fully synthesizable, 0.38pJ/bit, 3 clock cycle latency, 297Gbps/mm SNR-10 channel implementation is presented in TSMC 16nm. Using the channel, a 2×2 UDSP MCM's cross-sectional dielet bandwidth is measured to be 493Gbps using only two layers on an interposer giving it the highest UCLA FoM of any protocol (as of the time of writing).

- A 2×2 UDSP MCM assembly is presented which is within 4.2× energy-efficient and 6.4× area-efficient compared to an equivalent ASIC. The architecture sports a kernel-independent, high-throughput compute fabric which runs at 1.1GHz. This is the 1st functional processor on the Si-IF with 10-µm pitch I/O bumps.

- A low-area, sythesizable, folded delay-line architecture is presented for the coarse delay stage along with a synthesizable DDE that makes up a hierarchically synthesizable

TDL for the fine delay stage. This allows the delay line to be portable across advanced technologies, incurring low development costs and enabling quicker time-to-market.

- A method to eliminate DNL across the entire delay range is presented. The technique decouples the matching of the coarse and the fine stages and simultaneously ensures zero DNL. It enhances the robustness of the design to digital P&R based variation. In addition, a simple method to eliminate DCD is also presented that can individually track the positive and negative edges. A modified high-speed counter is presented which decouples the counter's operational speed and the counter's size, resulting in a large f_{max}/f_{min} ratio.

- A fully synthesizable, low-area - $0.0032mm^2$, and low power - 2.1mW, $<$ 1 LSB full range DNL, adequately high frequency - 2.2GHz, delay line is implemented in GF 22nm with DCPD and a large frequency tuning range.

7.2 Future Work and Directions

This dissertation opens avenues into broader applicability of the general ideas and techniques listed above. Suggested future directions from this work are discussed below.

- Possible automation in the design of the core can be investigated. Such a core design tool would need objective ways to measure utilization and efficiency. Using these as cost functions, the tool may even be able to give an algorithm-difference score, which will allow the designer to decide on either a heterogeneous design or a homogeneous design.

- The network design can be further optimized by compressing the 3 layers into a single layer multi-layer switchbox. Layer compression will eliminate the vertical connections between layers making it more sparse and area efficient. In addition, the compressed layered, can then be further optimized to eliminate loop back connections that are not going to be used (e.g. the current network allows a connection to go to layer-1 of a vertical stack from the left side and come back out to the left side, this can be avoided by removing the possibilities of such connections).

- Optimizations in the switchbox are possible, by using the multi-layered design methodology, to co-design the sending and receiving networks together rather than individually. This can further compress the area occupied by the switchboxes leading to better energy and area efficiencies without loss of meaningful flexibility.

- The switchbox design methodology is fundamentally maximizing disjoint paths for a particular connection cost. Applications of this method can go beyond switchboxes to other uses such as memory scatter-gather and pruning of network layers in a neural

network to lower computation complexity. For such applications, adjusting the connections in feedback or designing for the worst case can be replaced by judicious choice of connections in a feed forward way, saving design time while getting optimal results.

- Integer Linear Programming (ILP) techniques can be leveraged to produce more compact compiled programs, leading to greater active utilization. The compiler can be extended to include C++ based code with an interpreter and DFG generator.

- Infrastructure for accelerator as a service (ACaS) has been discussed. Building its components in a hardware dielet form and integrating it onto an SiP can be explored. There is room for improvement in runtime scheduler heuristics with possibilities of bringing down reconfiguration time to $<$ 100ns. The UDSP programming bandwidth can be increased as well.

- The possibility for increasing the SNR-10 channel size to achieve better I/O area density has been discussed. With much of the challenges addressed and the ground work for clock-correction laid out along with a possible high-density channel footprint, the SNR-10v2 channel can be designed and implemented in an advanced CMOS process with increased bandwidth.

- The UDSP can be coupled with a hardware memory controller and hierarchical memory and caching units using the SNR-10 interface. This would allow for high-throughput data processing on large data sets such as images. The MCM can be expanded to a wafer-scale graph processor with heterogeneous memory and compute dies to make a peta-scale compute fabric.

- With the presented architecture of a synthesizable delay line, possibilities of tighter oscillator timing can be explored with advanced technology nodes. There could be a possibility of a smaller coarse delay with a much smaller TDL (of 10-16 steps), doubling

the maximum frequency of operation. The size of the delay line can be further reduced by limiting independent edge action (and implementing a DCC along side it) and freeing up extra area.

- A faster delay line can then be implemented as a multiplying DLL (MDLL) to recover 50% duty-cycle clock, forming a duty-cycle correction engine. This can enable DDR functionality in SNR-10v2.

- Implementation of SNR channels across heterogeneous dielets on a wafer can also be investigated where cross technology dielet communication can be posed at the highlight of the work. Alignment standards for such dielets would need to be ironed out. In addition, inter-channel distance behaviour of different Tx driver sizes can also be explored (from a process perspective) to gain more insight into the number of layers that can be theoretically supported as well as how tightly can the chips be integrated.

References

[1] L. Eeckhout, "Is Moore's Law Slowing Down? What's Next?," in IEEE Micro, vol. 37, no. 4, pp. 4-5, 2017.

[2] T. Simonite, "Intel Puts the Brakes on Moore's Law," MIT Technology Review, March 2016. [online] Available: http://www.technologyreview.com/s/601102/intel-puts-the-brakes-on-moores-law.

[3] D. Takahashi, "Intel: Moore's Law Isn't Slowing Down," VentureBeat, March 2017. [online] Available: http://venturebeat.com/2017103/28/intel-moores-law-isn't-slowing-down.

[4] F. Adi, W. David, "The Accelerator Wall: Limits of Chip Specialization," in IEEE International Symposium on High Performance Computer Architecture (HPCA), 2019.

[5] Y. S. Shao and D. Brooks, "Research infrastructures for hardware accelerators," Synthesis Lectures on Computer Architecture, vol. 10, no. 4, pp. 1–99, 2015.

[6] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, et al., "Applied machine learning at facebook: A datacenter infrastructure perspective," 2018.

[7] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, et al., "In-datacenter performance analysis of a tensor processing unit," in International Symposium on Computer Architecture (ISCA), pp. 1–12, ACM, 2017.

[8] S. Ram, "Dark Silicon Overview, Analysis and Future Work," 2015.

[9] A, Kanduri, A. M. Rahmani, P. Liljeberg, A. Hemani, A. Jantsch, H. Tenhunen, "A Perspective on Dark Silicon," The Dark Side of Silicon: Energy Efficient Computing in the Dark Silicon Era, 2017.

[10] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, et al., "Conservation cores: Reducing the energy of mature computations," in International Conference on Architecture Support for Programming Languages & Operating Systems (ASPLOS), pp. 205–218, ACM, 2010.

[11] T. Nowatzki, V. Gangadhan, K. Sankaralingam, and G. Wright, "Pushing the limits of accelerator efficiency while retaining programmability," in Symposium on High-Performance Computer Architecture (HPCA), pp. 27–39, 2016.

[12] M. Khazraee, L. Zhang, L. Vega, and M. B. Taylor, "Moonwalk: NRE Optimization in ASIC Clouds," in International Conference on Architecture Support for Programming Languages & Operating Systems (ASPLOS), pp. 511–526, ACM, 2017.

[13] "ZYNQ RFSoC DFE Backgrounder," AMD Xilinx, 2020. [online] Available: https://www.xilinx.com/products/silicon-devices/soc/rfsoc.html.

[14] 3GPP Releases. [online] Available: https://www.3gpp.org/specifications/67-releases.

[15] International Business Strategies (IBS) report, 2018.

[16] A. Saran, Investor Day, Marvell, pp. 42, 2020.

[17] A. Podobas, K. Sano and S. Matsuoka, "A Survey on Coarse-Grained Reconfigurable Architectures From a Performance Perspective," in IEEE Access, vol. 8, pp. 146719-146743, 2020.

[18] B. Mei, A. Lambrechts, J. Mignolet, D. Verkest and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," in IEEE Design & Test of Computers, vol. 22, no. 2, pp. 90-101, March-April 2005.

[19] D. Burger et al., "Scaling to the end of silicon with EDGE architectures," in Computer, vol. 37, no. 7, pp. 44-55, July 2004.

[20] J. Lopes, J. Sousa. “Versat, a Minimal Coarse-Grain Reconfigurable Array," in VECPAR, June, 2016.

[21] R. Prabhakar et al., "Plasticine: A reconfigurable architecture for parallel patterns," 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 389-402, 2017.

[22] F. Yuan, "Energy-Efficient VLSI Architectures for Next-Generation Software-Defined and Cognitive Radios," UCLA, 2014.

[23] F. -L. Yuan, C. C. Wang, T. -H. Yu and D. Marković, "A Multi-Granularity FPGA With Hierarchical Interconnects for Efficient and Flexible Mobile Computing," in IEEE Journal of Solid-State Circuits, vol. 50, no. 1, pp. 137-149, January 2015.

[24] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications," in ACM Computing Surveys, vol. 52, is. 6, Article 118, November 2020.

[25] C. H. Stapper and R. J. Rosner, "Integrated circuit yield management and yield analysis: development and implementation," in IEEE Transactions on Semiconductor Manufacturing, vol. 8, no. 2, pp. 95-102, May 1995.

[26] Lin, X.-W., et al., "Heterogeneous Integration Enabled by State-of-the-Art 3DIC and CMOS Technologies: Design, Cost, and Modeling," in IEDM, paper 3.4, 2021.

[27] T. Burd et al., "“Zeppelin”: An SoC for Multichip Architectures," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 133-143, January 2019.

[28] S. Naffziger, K. Lepak, M. Paraschou and M. Subramony, "2.2 AMD Chiplet Architecture for High-Performance Server and Desktop Products," 2020 IEEE International Solid- State Circuits Conference - (ISSCC), pp. 44-45, 2020.

[29] "Nvidia Tesla V100 GPU Architecture," White paper, August 2017. [online] Available: https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[30] M. Deo, "Enabling Next-Generation Platforms Using Intel's 3D System-in-Package Technology," Intel white paper. [online] Available: https://www.intel.com/content/www/us/en/silicon-innovations/6-pillars/emib.html.

[31] CoWoS-R, TSMC. [online] Available: https://3dfabric.tsmc.com/english/dedicatedFoundry/technology/cowos.html.

[32] Broadcom and TSMC, "TSMC and Broadcom Enhance the CoWoS® Platform with World's First 2× Reticle Size Interposer," Hsinchu, Taiwan, March 2020.

[33] P. Gupta and S. S. Iyer, "Goodbye, motherboard. Bare chiplets bonded to silicon will make computers smaller and more powerful: Hello, silicon-interconnect fabric," in IEEE Spectrum, vol. 56, no. 10, pp. 28-33, October 2019.

[34] Xilinx Product Specification (Version 1.6), XC4000E and XC4000X Series Field Programmable Gate Arrays, May 1999.

[35] Bolsens I., "Programming Modern FPGAs," Xilinx, MPSOC, August 2006.

[36] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 203-215, Feb. 2007.

[37] I. Anagnostopoulos, A. Bartzas, I. Vourkas and D. Soudris, "Node resource management for DSP applications on 3D Network-on-Chip architecture," 2009 16th International Conference on Digital Signal Processing, pp. 1-6, 2009.

[38] P. Sudeep, and N. Dutt. "On-Chip Communication Architectures System on Chip Interconnect," Elsevier / Morgan Kaufmann Publishers, 2008.

[39] C. C. Wang, F. -L. Yuan, T. -H. Yu and D. Markovic, "27.5 A multi-granularity FPGA with hierarchical interconnects for efficient and flexible mobile computing," 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers - (ISSCC), pp. 460-461, 2014.

[40] A. DeHon and R. Rubin, "Design of FPGA interconnect for multilevel metallization," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 10, pp. 1038-1050, October 2004.

[41] U. Y. Rathore, "Next Generation Dynamically Reconfigurable DSP in 16nm Technology," UCLA, 2018.

[42] J. Rose and S. Brown, "Flexibility of interconnection structures for field-programmable gate arrays," in IEEE Journal of Solid-State Circuits, vol. 26, no. 3, pp. 277-282, March 1991.

[43] S. A. Cook, "The complexity of theorem-proving procedures," Proceedings of the third annual ACM symposium on Theory of computing, pp. 151–158, 1971.

[44] R. M. Karp, "Reducibility Among Combinatorial Problems," In R. E. Miller; J. W. Thatcher; J.D. Bohlinger (eds.). Complexity of Computer Computations. New York: Plenum. pp. 85–103, 1972.

[45] I. Koren, C. M. Krishna, "Defect Tolerance in VLSI Circuits, Fault-Tolerant Systems (Second Edition)," pp. 341-371, 2021.

[46] B. Clark and Y. Hill, "IBM Multichip Multilayer Ceramic Modules for LSI Chips-Design for Performance and Density," in IEEE Transactions on Components, Hybrids, and Manufacturing Technology, vol. 3, no. 1, pp. 89-93, March 1980.

[47] G. Corongiu and J. H. Detrich, "Large-scale scientific application programs in chemistry and physics on an experimental parallel computer system," in IBM Journal of Research and Development, vol. 29, no. 4, pp. 422-432, July 1985.

[48] D. C. Kehlet, "Accelerating innovation through a standard chiplet interface: The advanced interface bus (AIB)," Intel, 2019. [online] Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/accelerating-innovation-through-aib-whitepaper.pdf.

[49] S. Jangam, "Heterogeneous Integration on Silicon-Interconnect Fabric using fine-pitch interconnects ($\leq$ 10 µm)," UCLA, 2020.

[50] J. W. Poulton et al., "A 1.17-pJ/b, 25-Gb/s/pin Ground-Referenced Single-Ended Serial Link for Off- and On-Package Communication Using a Process- and Temperature-Adaptive Voltage Regulator," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 43-54, January 2019.

[51] M. Lin, T. Huang, C. Tsai, K. Tam, K. C. Hsieh, C. Chen, W.Huang, C. Hu, Y. Chen, S. K. Goel, C. Fu, S. Rusu, C. Li, S. Yang, M. Wong, S.Yang, and F. Lee, "A 7-nm

4-GHz Arm[1]-Core-Based CoWoS[1] Chiplet Design for High-Performance Computing," in IEEE Journal of Solid-State Circuits, vol. 55, no. 4, pp. 956-966, April 2020.

[52] T. H. Cormen and J. C. Fan, "Dense Gray codes, or easy ways to generate cyclic and non-cyclic Gray codes for the first n whole numbers," 2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2016.

[53] U. Rathore, S. Nagi, S. Iyer, D. Markovic, "A 16nm 785GMACs/J 784-Core Digital Signal Processor Array with a Multilayer Switch Box Interconnect, Assembled as a 2×2 Dielet with 10µm-pitch Inter-Dielet I/O, for Runtime Multi-Program Reconfiguration," IEEE International Solid- State Circuits Conference - (ISSCC), pp. 52-53, 2022.

[54] C. Liu, J. Botimer and Z. Zhang, "A 256Gb/s/mm-shoreline AIB-Compatible 16nm FinFET CMOS Chiplet for 2.5D Integration with Stratix 10 FPGA on EMIB and Tiling on Silicon Interposer," 2021 IEEE Custom Integrated Circuits Conference (CICC), pp. 1-2, 2021.

[55] JEDEC Standard, Jesd235c: High bandwidth memory (HBM) DRAM, January 2020. [online] Available: https://www.jedec.org/document_search?search_api_views_fulltext=jesd235.

[56] Mu-Shan Lin et al., "A 16nm 256-bit wide 89.6GByte/s total bandwidth in-package interconnect with 0.3V swing and 0.062pJ/bit power in InFO package," 2016 IEEE Hot Chips 28 Symposium (HCS), pp. 1-32, 2016.

[57] J. W. Poulton, W. J. Dally, X. Chen, J. G. Eyles, T. H. Greer, S. G. Tell, J. M. Wilson, and C. T. Gray, "A 0.54 pj/b 20 gb/s ground-referenced single-ended short-reach serial link in 28nm CMOS for advanced packaging applications," in IEEE Journal of Solid-State Circuits, vol. 48, no. 12, pp. 3206-3218, December 2013.

[58] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained DRAM: Energy-efficient dram for extreme bandwidth systems," in 2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 41-54, 2017.

[59] A. Tajalli, M. Bastani Parizi, D. A. Carnelli, C. Cao, K. Gharibdoust, D. Gorret, A. Gupta, C. Hall, A. Hassanin, K. L. Hofstra, B. Holden, A. Hormati, J. Keay, Y. Mogentale, V. Perrin, J. Phillips, S. Raparthy, A. Shokrollahi, D. Stauffer, R. Simpson, A. Stewart, G. Surace, O. Talebi Amiri, E. Truffa, A. Tschank, R. Ulrich, C. Walter, and A. Singh, "A 1.02-pj/b 20.83-gb/s/wire USR transceiver using CNRZ-5 in 16-nm FinFET," in IEEE Journal of Solid-State Circuits, vol. 55, no. 4, pp. 1108-1123, April 2020.

[60] M. Erett, D. Carey, J. Hudner, R. Casey, K. Geary, P. Neto, M. Raj, S. McLeod, H. Zhang, A. Roldan, H. Zhao, P. Chiang, H. Zhao, K. Tan, Y. Frans, and K. Chang, "A 126mw 56gb/s NRZ wireline transceiver for synchronous short-reach applications in 16nm FinFET," in 2018 IEEE International Solid - State Circuits Conference - (ISSCC), pp. 274-276, 2018.

[61] M. LaCroix, H. Wong, Y. H. Liu, H. Ho, S. Lebedev, P. Krotnev, D. A. Nicolescu, D. Petrov, C. Carvalho, S. Alie, E. Chong, F. A. Musa, and D. Tonietto, "6.2 A 60Gb/s PAM-4 ADC-DSP Transceiver in 7nm CMOS with SNR-Based Adaptive Power Scaling Achieving 6.9pJ/b at 32dB Loss," 2019 IEEE International Solid- State Circuits Conference - (ISSCC), pp. 114-116, 2019.

[62] Y. Krupnik, Y. Perelman, I. Levin, Y. Sanhedrai, R. Eitan, A. Khairi, Y. Shifman, Y. Landau, U. Virobnik, N. Dolev, A. Meisler, and A. Cohen, "112-gb/s PAM4 ADC-

based SerDes receiver with resonant AFE for long-reach channels," in IEEE Journal of Solid-State Circuits, vol. 55, no. 4, pp. 1077-1085, April 2020.

[63] J. Kim, A. Balankutty, R. K. Dokania, A. Elshazly, H. S. Kim, S. Kundu, D. Shi, S. Weaver, K. Yu, and F. O'Mahony, "A 112 Gb/s PAM-4 56 Gb/s NRZ Reconfigurable Transmitter With Three-Tap FFE in 10-nm FinFET," in IEEE Journal of Solid-State Circuits, vol. 54, no. 1, pp. 29-42, January 2019.

[64] S. Jangam, U. Rathore, S. Nagi, D. Markovic and S. S. Iyer, "Demonstration of a Low Latency (< 20ps) Fine-pitch (≤10µm) Assembly on the Silicon Interconnect Fabric," 2020 IEEE 70^{th} Electronic Components and Technology Conference (ECTC), 2020.

[65] MATLAB Simulink, 2022. [online] Available: https://www.mathworks.com/products/simulink.html.

[66] R. Nanda, C. Yang, D. Markovic, "DSP architecture optimization in Matlab/Simulink environment," UCLA, pp. 14-16, 2008.

[67] S. Nagi, "Efficient, Scalable and High Throughput Run-Time Reconfigurable Architectures (RTRA) for ACcelerator As a Service (ACAS)," UCLA. 2022.

[68] M. Y. Lanzerotti, G. Fiorenza and R. A. Rand, "Microminiature packaging and integrated circuitry: The work of E. F. Rent, with an application to on-chip interconnection requirements," in IBM Journal of Research and Development, vol. 49, no. 4.5, pp. 777-803, July 2005.

[69] S. Jangam, U. Rathore, S. Nagi, D. Markovic and S. S. Iyer, "Demonstration of a Low Latency (<20 ps) Fine-pitch (≤10 μm) Assembly on the Silicon Interconnect Fabric," 2020 IEEE 70th Electronic Components and Technology Conference (ECTC), pp. 1801-1805, 2020.

[70] K. Sahoo, U. Rathore, S. Jangam, D. Markovic, and S. Iyer, "Functional demonstration of < 0.4pJ/bit, 9.8µm fine-pitch dielet-to-dielet links for advanced packaging using Silicon Interconnect Fabric," *"in press"*, 2022 IEEE 70th Electronic Components and Technology Conference (ECTC), 2022.

[71] A. Stillmaker, B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm," Integration, Volume 58, pp. 74-81, 2017.

[72] S. Lip-Kai, M. Sulaiman and Z. Yusoff, "Fast-Lock Dual Charge Pump Analog DLL using Improved Phase Frequency Detector," 2007 International Symposium on VLSI Design, Automation and Test (VLSI-DAT), pp. 1-5, 2007.

[73] A. Ghaffari and A. Abrishamifar, "A Wide-Range Delay-Locked Loop with a New Lock-Detect Circuit," 2006 13th IEEE International Conference on Electronics, Circuits and Systems, pp. 1168-1171, 2006.

[74] Y. -L. Lo, P. -Y. Chou, H. -H. Cheng, S. -F. Tsai and W. -B. Yang, "An all-digital DLL with dual-loop control for multiphase clock generator," 2011 International Symposium on Integrated Circuits, pp. 388-391, 2011.

[75] J. -S. Wang, C. -Y. Cheng, P. -Y. Chou and T. -Y. Yang, "A Wide-Range, Low-Power, All-Digital Delay-Locked Loop With Cyclic Half-Delay-Line Architecture," in IEEE Journal of Solid-State Circuits, vol. 50, no. 11, pp. 2635-2644, November 2015.

[76] D. Park and J. Kim, "A 7-GHz Fast-Lock 2-Step TDC-based All-Digital DLL for Post-DDR4 SDRAMs," 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018.

[77] E. Bayram, A. F. Aref, M. Saeed and R. Negra, "1.5–3.3 GHz, 0.0077 mm^2, 7 mW All-Digital Delay-Locked Loop With Dead-Zone Free Phase Detector in 0.13 µm CMOS,"

in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 1, pp. 39-50, January 2018.