UNIVERSITY OF CALIFORNIA
Santa Barbara

Abusing Hardware Race Conditions for High Throughput Energy Efficient Computation

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

by

Advait Madhavan

Committee in Charge:

Professor Dmitri Strukov, Chair

Professor Tim Sherwood

Professor Luke Theogarajan

Professor Forrest Brewer

December 2016

The Dissertation of
Advait Madhavan is approved:

Professor Tim Sherwood

Professor Luke Theogarajan

Professor Forrest Brewer

Professor Dmitri Strukov, Committee Chairperson

September 2016

Abusing Hardware Race Conditions for High Throughput Energy Efficient Computation

by

Advait Madhavan

To my Parents.

Acknowledgements

My learning experience at UCSB would not have been as enjoyable or insightful without the help of a lot of people around me. Therefore, I acknowledge them for their contributions, not only to the work presented in this dissertation but also my individual growth.

Firstly, I would like to thank my advisor, Prof. Dmitri Strukov for giving me the opportunity to be a part of the incredibly diverse research environment at UCSB and Strukov group. I would specifically like to thank him for discussing a variety of brilliant ideas with me as well as providing an environment of freedom and support. I was allowed to make mistakes, direct my research along my personal scientific interests and was provided a safety net, all for which I am very grateful.

I would also like to express my gratitude towards my committee members, with whom I was lucky to have many direct interactions resulting from classes to research. Prof Luke Theogarajan's circuit classes were very insightful and inspirational and he himself has been a constant source of scientific and psychological support for me. I am grateful to have been treated so warmly by him and his group which has allowed me to be part of more than one research environment at once. Prof. Tim Sherwood's computer architecture class at UCSB was probably one of the best classes I every took. I have learnt so much from Prof Tim, from making convincing arguments, breaking down problems effectively, to maintaining a healthy relationship between graduate school and life. Prof. Forrest Brewer's advanced VLSI and chip design classes were the first tape-outs that I was involved in. Since those classes in the early years of graduate school, Prof. Brewer has been a constant source of ideas and I am grateful to have been exposed to the tremendous breadth and depth of his knowledge.

I would also like to thank my fellow lab-mates with whom I spent a lot of time getting my hands dirty. In spite of being in different groups, Melika Payvand and I were lucky to work on multiple collaborative chip design projects together. We spent a lot of late nights in the lab doing chip tape-out work which led to the growth of a beautiful friendship and I am very grateful to her for that. I admire her for the dedication and care with which she approached surroundings and I hope for some of that to be rubbed off onto me as well. I thank members of Luke's group (Luis, Mohamed, Aaron, Jennifer, Melika, Danielle) who made the 4110 a very lively place to work. Also, I would like to thank Mirko, Farnood, Bhaswar, Xinjie and Michael with whom I have had countless discussions regarding research and who have directly helped me with my work.

I am grateful to friends of mine from high school(Karun, Ishaan, Aneesh, Apoorv, Ghodu, Aditi) for always encouraging me and keeping my spirits up

when I talk to them and visit them. My roommates (Geetak, Sagar, Nitin, Arvind (RIP), Gina, Chirag, Urmish, Manik, Ravi, Dinesh, Abhinav, Sanmay) and friends (Bugra, Mike Gama, Nikita, Billy) in SB over the years have been great to live with and have truly made these years at UCSB truly memorable. I would also like to thank my parents(Madhavan Kutty and Sreedharan Sudha) and my grandparents (P S Nayar and Vasini Nayar) for creating an environment of learning in the house as well as my sister for all the bickering and squabbling we went through. Gopal, Shanta, Siddharth and Kunal will always be a part of my family and I am eternally grateful for the boundless support that I have received from them. Last but not the least, I would like to thank Tanya Das, who has been with me through thick and thin for the last four years, has been a constant source of love and kindness and has taught me many valuable life lessons. I could not have done it without her. I would like to thank you all for your efforts.

Curriculum Vitæ

Advait Madhavan

Education

Ph.D., Electrical and Computer Engineering, University of California, Santa Barbara, 2016

M.S., Electrical and Computer Engineering, University of California, Santa Barbara, 2014

B.Tech, Instrumentation and Control Engineering, Netaji Subhas Institute of Technology, New Delhi, India, 2010.

Publications

- A. Madhavan, T. Sherwood, and D.B. Strukov, Race logic: A hardware acceleration fordynamic programming algorithms, in: Proc. ISCA14, Twin Cities, MN, June 2014, pp.517-528.
- A. Madhavan ,T. Sherwood, and D.B. Strukov, Abusing Hardware Race Conditions todo Useful Computation, Micro Top Picks, Special Edition, June 2015.
- A. Madhavan, T. Sherwood, and D.B. Strukov. Energy efficient computation with asynchronous races Proceedings of the 53rd Design Automation Conference (DAC) June 2016. Austin, TX.
- A. Madhavan, G. Adam, L. Gao, and D. B. Strukov, Analog and digital computing withmemristive devices, extended abstract in: Nature Conference on Frontiers in ElectronicMaterials: Correlation Effects and Memristive Phenomena, Aachen, Germany, June 2012
- A. Madhavan and D.B. Strukov, Mapping image and network processing tasks ontohigh-throughput CMOL FPGA circuits, in: Proceedings of VLSI-SoC12, Santa Cruz, CA,Oct. 2012.
- Melika Payvand, Advait Madhavan, Miguel Angel Lastras-Montao, Amirali Ghofrani, JustinRofeh, Kwang-Ting Cheng, Dmitri Strukov, Luke Theogarajan, A Configurable CMOS MemoryPlatform for 3D- Integrated Memristors. Proceedings of ISCAS 2015.
- Justin Rofeh, Avantika Sodhi, Melika Payvand, Miguel Angel Lastras-Monao, AmiraliGhofrani, Advait Madhavan, Sukru Yemenicioglu, Kwang-Ting Cheng and Luke Theogarajan,Vertical Integration of Memristors onto Foundry CMOS Dies using Wafer-Scale Integration, Proceedings of the ECTC 2015.

Abstract

Abusing Hardware Race Conditions for High Throughput Energy Efficient Computation

Advait Madhavan

We propose a novel computing approach, called Race Logic, which utilizes a new data representation to accelerate a broad class of optimization problems, such as those solved by dynamic programming algorithms. The core idea of Race Logic is to deliberately engineer race conditions in a circuit to perform useful computation. In Race Logic, information, instead of being represented as logic levels (as is done in conventional logic), is represented as a timing delay. Computations can then be performed by observing the relative propagation times of signals injected into a configurable circuit (i.e. the outcome of races through the circuit).

In this dissertation I will introduce race condition based computation and talk about multiple VLSI implementations. We first begin by considering a synchronous approach, which uses simple clocked delay elements. Though this synchronous implementation outperforms highly optimized conventional implementations of the well-studied, DNA sequence alignment problem, its third order energy scaling with problem size and limited dynamic range of timing delays are its major pitfalls. Next, in the search for energy efficiency, we study asynchronous designs

in order to understand the performance trade-offs and applicability of this new architecture. Finally, I will present the results of a prototype asynchronous Race Logic chip and demonstrate that Race-Based computations can align up to 10 million 50 symbol long DNA sequences per second, about 2-3 orders of magnitude faster than the state of the art general purpose computing systems.

Contents

List of Figures

Chapter 1

Introduction

The seemingly insatiable advancement in technology over the last half a century has had a significant and wide ranging impact on society and has ushered in, what is called by sociologists, the "information age". Named rightly so, our ability to store and manipulate information has followed more or less a steady exponential over the last five to six decades. To get a feeling of the enormity of this progress, a short comparison between representative examples is in order. The ENIAC (Electronic Numerical Integrator and Computer)[6] is considered as the first electronic general purpose computer and was designed in the 1940's primarily to calculate artillery firing tables for the US Army, Ballistic Research Laboratory. It could perform 60-second trajectory calculations in 30 seconds, what would usually take a skilled person with a desk calculator about 20 hours. It consisted of

$\approx$ 100 000 components, primarily resistors, vacuum tubes, capacitors and relays, and had a machine cycle of 200 microseconds, allowing it to compute 5000 additions, 357 multiplications and 38 divisions a second. It took up a total floorspace of $167m^2$, weighed 30 tons, consumed 180kW of power and cost half a million dollars. Today, the iPhone6 uses a 64bit Cortex A8 ARM chip which is composed of $\approx$ 1.6 billion transistors. It operates at a frequency of 1.4GHz and can process approximately 1.2 instructions per cycle in one of its 2 cores, which amounts to 3.36 billion instructions per second. The power consumption of this chip is around 2.3W with a footprint of $89mm^2$ and the entire 700 dollar phone weighs in at 129 grams. Trillions of dollars and countless man-hours of research, development and production have allowed information storage and processing to move beyond its initial military and aerospace applications and become ubiquitous in the world today. From massive datacenters that host the internet to household appliances, smart-phones, automotive applications, and with the upcoming surge of the Internet of Things (IoT), the estimated number of devices that are powered by silicon hardware is estimated to reach 50 billion by the year 2020.

In spite of this continued proliferation of silicon computing for a host of everyday applications, silicon itself is facing some major, potentially industry changing, challenges. For many years, consistent improvement in Complimentary Metal Oxide Semiconductor (CMOS) processing technology fuelled large increases in pro-

cessor performance and throughput. Predominantly, improvements in lithography based techniques allowed engineers to build transistors with smaller feature sizes each advancing process generation. Known colloquially as Moore's Law, what this meant was, for the same chip area, computer architects had an exponentially larger number transistors to work with, each with exponentially higher energy efficiency. The results of this can be seen in a modern microprocessor that is a highly complex piece of machinery, generally consisting of multiple cores with each core employing sophisticated mechanisms to improve performance.

However, in recent years, the law of diminishing returns seems to be catching up. As of 2016, 10nm devices are still under commercial development with industry leaders such as Intel and TSMC announcing production and commercial release to start in early 2017. Development of 7nm and 5nm nodes have also been projected by the year 2021. The 5nm node was assumed to be the end of Moore's law as quantum effects such as tunneling would cause gate leakage to go beyond permissible limits and hence significantly reduce yield. As the increase in the number of transistors that can be put on a chip seems to be slowing down, powering them efficiently is already a much more immediate concern. Leakage currents have begun to consume a significantly larger portion of the power budget than before, hence pushing CMOS scaling out of its traditional regime. As a result, after the 130-90nm technology node, though transistor areas continue to

shrink, their switching energy no longer declines. Moreover, practical concerns such as thermal management (cooling) and battery life keep the power budget fixed. Therefore, with each new technology node, the number of transistors that can be switched simultaneously is exponentially decreasing.

This energy problem that is being faced by computing, in literature, is known by multiple monikers. The growing divide between available transistors vs utilizable transistors has caused researchers to dub this the *utilization wall* [46]. Other terms such as Dark/Dim silicon refer to the underutilized transistors that have to be turned off or under-clocked to maintain fixed power budgets [12]. Dark Silicon and the Utilization wall that causes it have caused architects to shift their focus from designing the highest performance circuits, as these are almost guaranteed to exceed the power budget. Instead, as power and energy become more expensive relative to chip area, newer architectures will have to focus on spending chip area to buy energy efficiency, which opens the door to specialization and heterogeneity. Researchers have predicted that future chips are going to be populated by a large number of application specific hardware accelerators that are going to be used only part of the time. This is already being implemented in cellphones where multiple digital functions have been integrated on one chip, each having its own dedicated processing hardware.

In this dissertation we propose a novel computing approach, called Race Logic, which utilizes a new data representation to accelerate a broad class of optimization problems, such as those solved by dynamic programming algorithms. The core idea of Race Logic is to deliberately engineer race conditions in a circuit to perform useful computation. Information, instead of being represented as logic levels as is done in conventional logic, is represented as a timing delay. Computations can then be performed by observing the relative propagation times of signals injected into a configurable circuit (i.e. the outcome of races through the circuit). The main advantage of this novel approach is that the set of arithmetic and logical operations that can be most efficiently expressed changes, leading to new trade-offs and architectures. Through the manipulation of the natural delay chaining inherent to digital designs, the basic operations of MIN, MAX, and ADD-BY-CONSTANT can be implemented in a way that results in superior latency and energy efficiency for certain classes of problems.

We will begin this dissertation by exploring some general themes and approaches to energy efficiency at both architecture and circuit levels as obtained from literature. Starting with a more detailed study of the scaling laws that give rise to the Utilization Wall, we will discuss performance vs energy trade-offs, digital vs analog design trade-offs, as well as the role of approximations in improving energy efficiency of computing systems. These general themes will be detailed in

Chapter 2 and employed in the understanding of Race Logic by comparing it to other architectures.

Chapter 3 introduces Race Logic and using a very simple example, elaborates on various implementation methods. It also introduces an important application, the DNA sequence alignment problem that is used as a toy problem in the rest of this thesis.

Chapter 4 performs a case study utilizing the DNA sequence alignment algorithm. It first introduces a traditional systolic array method of performing such a computation and then presents a synchronous Race Formulation with the interest of comparison between them. A more general and energy efficient implementation of Race Logic is also presented.

Chapter 5 addresses some of the drawbacks of synchronous Race Logic, especially in terms of energy efficiency. It pushes for improving energy efficiency by removing the clock which translates to using asynchronous delay elements that are less power hungry but also more prone to variations. In this chapter we also perform a variation analysis to look at how process variations at the delay element level affect the functional correctness of the architecture and what the demands of the architecture are.

Chapter 6 presents the architectural, circuit level and floor-planning decisions that went into designing a Race Logic chip. We also present layout level details with die photographs and chip results.

Chapter 7 presents a summary and discussion of this work, and future directions that can be taken.

Chapter 2

Background and General themes

2.1 Half a century long Exponential

Among the many innovations that have enabled the technology revolution, lithography of silicon has been a significant one. One can say that at the centre of the steady increase of performance per watt per unit area, has been the battle against physical laws and variations to constantly improve fabrication techniques and enable scaling of silicon. Figure 2.1 plots CPU data over the last forty years and shows the impact of scaling on important performance parameters [7]. At first glance one can see that the number of transistors integrated on a single chip has been the only metric that has seen a steady exponential rise over the last four decades. Though yield and chip area were a concern in the early years, our

ability to power these transistors effectively and extract as much performance as possible continues to be a challenge that will shape the course of the semiconductor industry today.

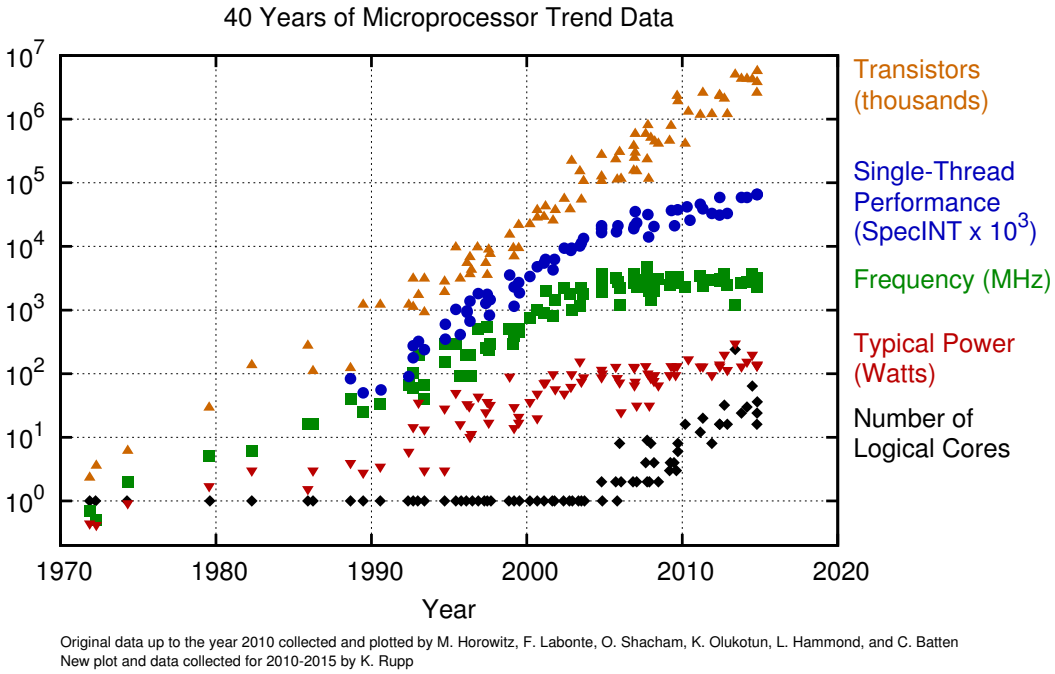


Figure 2.1: Microprocessor trend data over the last 40 years

The predominant VLSI technology in the early years from 1971-81 was based on PMOS or NMOS technology with depletion mode loads. The earliest chips of this era include Intel's and history's first fully integrated microprocessor, the Intel 4004, which ran at a clock speed of 740kHz and could perform addition of two 8bit numbers in about 850μs. Successors of 4004 include the 8080, which

extended the 4bit word size to 8bits and upgraded the clock speed to 2MHz, and the famous 8086, a 16bit microprocessor which pushed clock speeds to 10MHz. To be compatible with the then prevalent TTL technology, the supply voltage was kept at a constant $5V$ while the minimum feature size scaled down from 10μm to about 1μm. As a result, the saturation current didn't scale effectively, causing power dissipation to grow enormously, affected by commensurate increases in chip area, complexity and operating frequency. This first power crisis caused a shift from NMOS to CMOS designs which cut down large leakage transients, hence reducing power dissipation by an order of magnitude. The power savings thus gained from switching to CMOS was large enough to keep the supply voltage (V_{dd}) constant for almost another decade.

In 1974, Robert Dennard explored different methods of scaling CMOS devices and suggested a "constant field" scaling method in which the voltage levels were also scaled with lithographic dimensions [8]. He pointed out that this would allow power density to remain a constant, as the reduction in energy per switching event, due to lower supply voltage, would exactly match the increase in the number of transistors, as both obeyed square law behaviour. By the mid-nineties, as transistors began to scale into the sub-micron regime, still with a 5 volt V_{dd}, internal device electric fields grew to large values and pushed Dennard scaling into effect. $V'_{dd}s$ have systematically scaled down by roughly $10V/\mu m$ with each suc-

cessive generation from 0.5μm down to 130nm. In spite of this improved Dennard scaling, power dissipation increased by almost 50X from the early nineties to the mid two thousands, as shown in figure 2.1. This was predominantly as a result of various architecture and circuit design optimizations such as deeper pipelines, better sizing of gates, out of order execution, performance driven architectures and larger chips. During this time frame, processor speed also saw a significant boost with clock frequency almost doubling every few generations, scaling much faster than gate speed, a testament to circuit and architectural innovations of this era. Representative chips of this era include the Intel's Pentium 4, PentiumD, AMD's AthlonXP, Opteron and IBM's PowerPC970, Xenon.

As clock frequency continued to increase, engineers had to face the power wall again. As hobbyists and computer enthusiasts were liquid cooling their overclocked 8 GHz CPUs, the law of diminishing returns showed that boosting clock frequency was not the most effective way of extracting performance per watt. Though it continued to improve single thread performance for many applications, it was seen that energy used to open up data and instruction level dependencies would provide much higher overall performance gains. This slowing down of clock frequency scaling is clearly visible in figure 2.1 and coincides with a single thread performance slowdown and an increase in number of logical cores. This signifies the start of the multicore era where power constraints guided the search for improved performance

Parameter	Description	Equation	Classical Scaling	Dennard Scaling	Leakage limited
W,L	width, length		1/S	1/S	1/S
t_{ox}	oxide thickness		1/S	1/S	1/S
V_{dd}	supply voltage		1	1/S	1
C_{gate}	gate capacitance	WL/t_{ox}	1/S	1/S	1/S
$I_{d,sat}$	saturation current	$WV_{dd}^2/Lt_{ox}, WV_{dd}/t_{ox}$	S	1/S	1
F	frequency	$I_{d,sat}/C_{gate}V_{dd}$	S^2	S	S
P	dynamic power	$NC_{gate}V_{dd}^2F$	S^3	1	S^2

Table 2.1: Different types of scaling ranked in historical order and their effect on chip parameters. S denotes scaling factor, > 1

by exploiting parallelism. Though out-of-order execution and deep pipelines in single core machines already allowed for effective sharing of resources, multiple simpler cores provide a better energy and price point compared to complex high frequency uni-processors. Representative chips of this era include, Intel Core series (i3,i5,i7), Xeon E7 and AMDs FX Bulldozer.

Unfortunately no exponential can last forever and now engineers have hit a fundamental physical limit which prevents the threshold voltage, V_{th} from scaling.

Since the thermal voltage ($V_T = kT/q$) doesn't scale, reducing the threshold voltage causes an exponential increase in leakage current. Leakage current in today's designs is already large enough (approx 40% [47]) that it needs to be accounted for in the power budget and hence, V_{th} is set as a result of power optimization and not by technology scaling. During the Dennard scaling regime, V_{dd} scaling was complemented by commensurate lowering of the threshold voltage, but this cannot continue if the scaling is leakage limited as shown in Table 2.1. The dynamic power in this leakage limited regime grows exponentially and hence, due to a fixed power budget, leads to exponential reduction in active switching circuitry each process generation. In a fixed amount of time, there is only a certain amount of energy dissipation that can be tolerated by the cooling system, and with leakage accounting for a size-able portion of that energy, *edges have become expensive.*

Race Logic includes this general purpose computing research wisdom by encoding information in timing delay which in a logic circuit manifests as a single rising edge propagating through a set of spatially laid out parallel paths. Smaller and larger magnitudes can be encoded on the same wire with only a single edge, with the smaller delay encoding smaller magnitudes while a larger magnitude would mean longer delays (or vice versa, based on the kind of encoding). In both cases a single edge on a wire encodes the signal with only the arrival times determin-

ing the magnitude which results in less wires (less area and capacitance) and less switching activity (less toggling) compared to traditional approaches. This encoding scheme results in superior energy efficiency, not only as a result of this single edge based temporal coding, but also as a result of the simplicity in performing some operations that this form of coding provides.

2.2 Approaches to Energy Efficiency

As the performance per watt metric begins to take a front seat in terms of design constraints in modern systems, a wide variety of options is available. Exploring the design space of circuits or architectures, we see a certain trade-off between energy and performance of designs that informs our choices regarding the desired implementation. This trade-off is shown in Figure 2.2. Interestingly, the qualitative shape of this curve always remains the same for different architectures and circuits and it follows the law of diminishing returns[16].

A first glance analysis of the curve reveals that at low energy points (bottom left region), small changes in energy can lead to large changes in performance, while at high performance regions (top right region), a marginal increase in performance costs a lot of energy. The choice of what implementation is picked depends upon the application and in most cases lies along the pareto frontier. In other words, the energy of the architecture is reduced at the cost of increasing

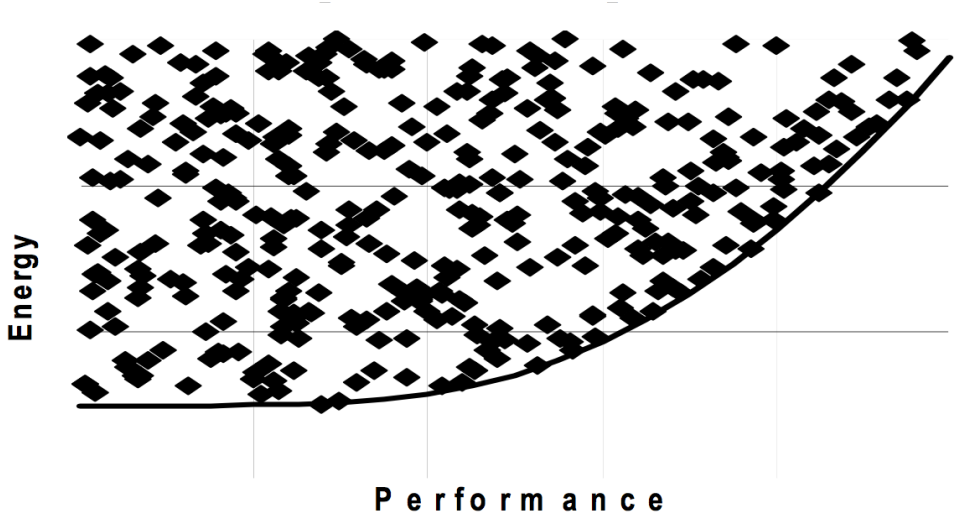


Figure 2.2: Energy vs Performance design space for a given function. The pareto curve shows the boundary of possible solutions in this space. The most favourable point in this space is the extreme bottom right as that would provide high performance at very low energy cost.[16]

the latency. Similar examples to these are architectures that are partitioned such that specific regions of the architecture are connected to different power supplies and body biases. Timing critical paths are adjusted for performance while other regions are optimized for energy efficiency.[16]

Another subset of techniques have no energy cost for keeping performance fixed. This is generally implemented by reducing the energy that is being wasted by the system. Clock and power gating strategies are good examples of such techniques as they turn off power and clocks to regions of the architecture/system that are not producing useful outputs. The dual of this could thought of as systems that improves performance at no extra energy cost. Exploiting parallelism with

multiple functional units would allow each functional unit to be running at a slower rate, instead of just one functional unit running at the maximum rate. For example, due to the shape of the curve shown in figure 2.2, the marginal energy saved by running a functional unit at a half the speed could be used by other another functional unit and would overall achieve the same performance while still keeping the energy within acceptable bounds. This argument is in essence, the logic behind the surge of multi-core architectures.

The best techniques though, are those that improve both energy and performance, hence having negative energy cost for performance improvement[16]. These techniques generally involve problem reformulation, a new information representation, or algorithmic changes that reduce the overall computation that needs to be performed. Most of these methods are highly application specific and require changes at both architectural and implementation levels. Such techniques are generally accompanied by custom hardware that allow orders of magnitudes of energy savings and performance gains. As a simple example, performing multiplication and division operations can be complex and require special hardware for binary encoding schemes, but using logarithmic [45] number systems makes performing such operations as straightforward as addition and subtraction. CORDIC [49] algorithms extend this idea to perform vector rotations in a bit-serial manner by using clever encodings that allow computation for arbitrary angles by using simple

iterative shift and add procedures. Many algorithms designed along these lines were conceived at a time when chip area was extremely expensive and performing parallel computation operations such as square root would be prohibitively expensive in area, but it is perhaps worth revisiting these ideas in the new context of power efficiency.

During the course of development of the Race Logic formulation, we have utilized most of the above techniques to shrink the energy-delay product of Race based computations compared to traditional approaches. Firstly (as we will see in chapter 3), as a result of temporal information representation, some arithmetic operations such as MIN, MAX and ADD BY CONSTANT become trivial to implement. The simplicity thus afforded allows for an ease of implementation which results in significant performance gains as compared to standard implementations. Also, as will be presented in more detail in chapter 4, we try to minimize energy consumption in our synchronous Race Logic implementation by employing clock gating strategies.

2.3 The Role of Approximations

Another means of reducing power dissipation in computation systems that has come to the fore during recent years, is to trade off functional correctness for energy efficiency. Some methods leverage the error tolerance of the application

such as those in domains of image processing, machine learning and computer vision. Though this error tolerance can originate from imperfect human perception, lack of a universal best result, or redundancy in the input, it allows the outputs of these algorithms to be numerically approximate rather than accurate. This relaxation on numerical exactness provides some freedom to carry out imprecise or approximate computation[48]. The freedom for approximation can allow for increased simplicity of designs, which can leave room for architectural and circuit level innovations to boost performance or reduce energy consumption. Some implementations identify functionally non-critical regions of the algorithms and use VOS (voltage overscaling) to save power[29], while other provide VOS cores with a low precision error compensation core[41]. Other methods explicitly target designing imprecise functional units such as adders or multipliers that are optimized for energy efficiency, speed, or wiring delay. Stochastic methods analyse the probabilistic nature of circuit types and design out low probability worst cases which reduce performance[17]. An interesting approach dubbed Razor utilizes this principle in a general purpose computing system by implementing real time error checking using shadow registers and literally runs at the edge of functional correctness[11]. The system then utilizes DVS (Dynamic Voltage Scaling) to reduce the energy dissipated by the system until the system is running on the *"razor's edge"* of failure.

As we will see in Chapter 5, Race Logic embraces the approximate computing paradigm as we forgo synchronous implementations due to high power dissipation that results from continuous clocking. Since information is encoded in timing delays, moving away from a synchronous domain could result in a significant reduction in noise immunity. We move to an asynchronous approach with analog components which is by nature more prone to process variations, power supply switching variations, and other environmental factors. We perform an error tolerance study on our application space to understand how variations in the timing delay from these aforementioned components affect the functional correctness of the application and show that the performance of the architecture is minimally affected.

The general themes discussed in this chapter lie at the heart of computing and outline different methods, generally used to extract performance and energy efficiency in modern computing systems. Race Logic is no exception. Starting from the information representation that helps preserve edges in an attempt to reduce dynamic power(discussed in chapter 3), to the application variation study (performed in chapter 5), the general themes presented here will be called upon repeatedly to push performance per watt of our designs.

Chapter 3

Race Logic and Application Study

Race conditions are notoriously knows for causing errors in software and hardware systems. They occur as a result of the output of a certain block being dependant on the sequence, or the timing of other events in the system. When the designers of such a system are unaware of these timing intricacies, these race conditions manifests as bugs. Hardware race conditions result in hazards or glitches that cause the output of a system to flip once or multiple times but systems such as Karnaugh Maps can be used to identify them before they happen and design them out of the system. Software races on the other hand are trickier to reproduce and debug since they are as a result of relative timing difference between

interfering threads. Problems occurring in production systems can therefore disappear when running in debug mode, when additional logging is added, or when attaching a debugger, often referred to as a "Heisenbug".

In this work we propose a method that makes positive use of race conditions. How we plan to do this is by setting up a set of race conditions in a system that are input dependant. A signal navigates these set of race conditions and the path that is taken by this signal (unknown beforehand), solves the problem. The main idea of Race Logic is to encode information in a timing delay. We do this by defining a unit delay element that acts as a fundamental unit of computation as shown in figure 3.1(a). How it functions is by delaying an input transition to its output by 1 unit. Aside from the physical implementation of such a delay element, a few properties remain constant.

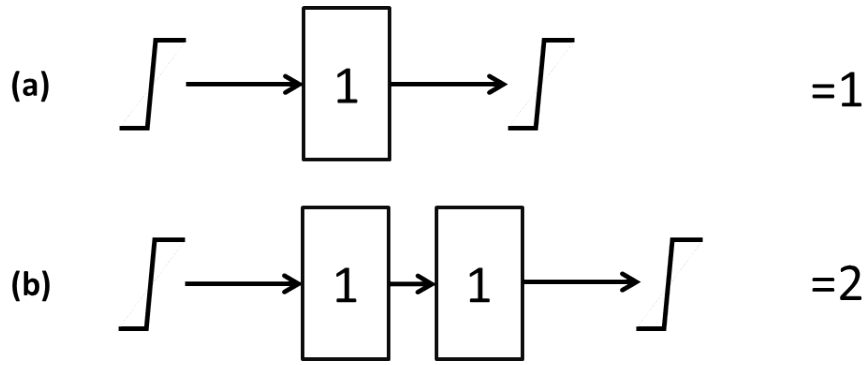


Figure 3.1: Delay domain representation. (a) Unit delay element that delays input rising edge by 1 "unit" time step, (b) stacking two delay elements one after another performs addition operation.

One such property is addition; as shown in figure 3.1(b), the simple act of chaining two unit delay elements result in an input transition to be delayed by a value of two unit delays. In this representation a slower delay corresponds to a larger value and a faster delay to a smaller one. MIN and MAX operations can also be implemented trivially with such a delay encoding.

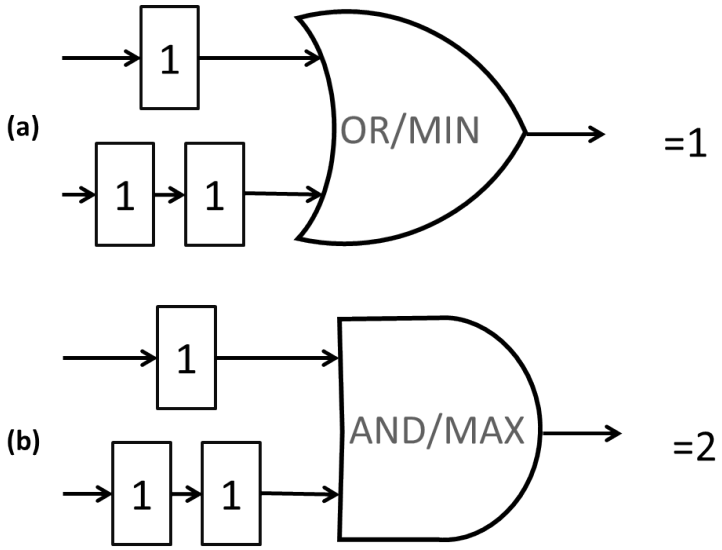


Figure 3.2: Delay domain computation. (a) Using an OR gate to implement MIN function by selecting a first arriving of multiple edges , (b) Using an AND gate to implement MAX function by selecting a last arriving of multiple edges.

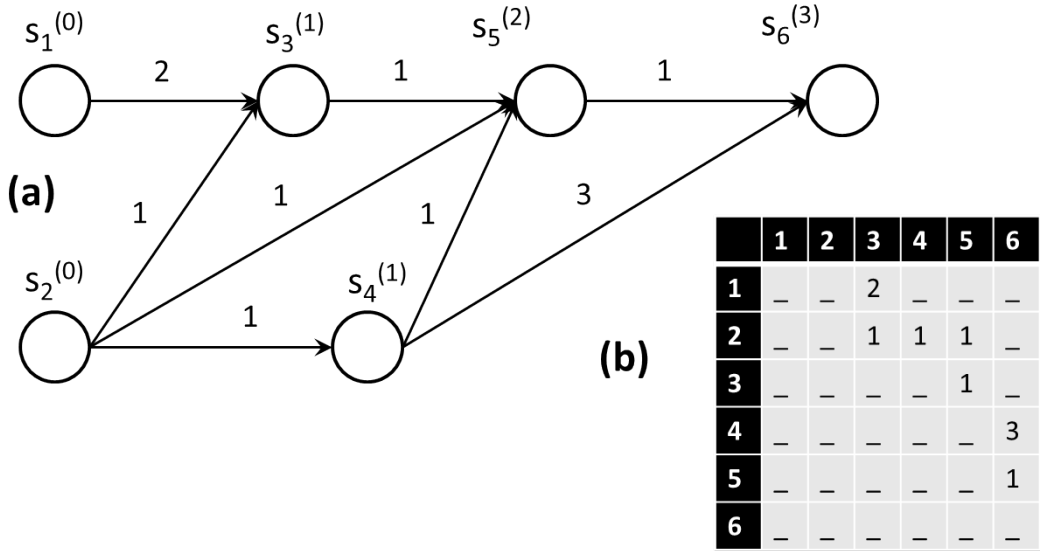
In figure 3.2 we see two parallel delay paths each with a different delay. If we trigger rising edges at the inputs of these two parallel paths simultaneously, we see that the first arriving edge is going to be the one with MIN delay while the one arriving later will the MAX of the two. Selecting the first arriving signal is equivalent to logical OR operation as the output rises if any of the inputs are

triggered. The dual of this would imply using the AND operation for the MAX function.

3.1 Hardware Implementations

To see how this could ever useful, lets consider a directed acyclic graph (DAG) as shown in figure 3.3(a) which is helpful for solving many practical problems in bio-informatics, data mining, logistics, control, and information processing. Typically, a certain weight W_{ij}(figure 3.3(b)) is associated with each edge of such graph and is generally representative of a cost of satisfying certain criteria. Optimization problems then, focus on finding a path in the graph that optimizes (maximizes or minimizes) this cost function, or equivalently, navigates the shortest or longest path on such a graph.

Because DAGs can be linearized, i.e. arranged on line so that all edges go from left to right, dynamic or linear programming techniques are the most common way of solving these problems. Conceptually, dynamic programming relies on solving progressively larger sub-problems starting with the set of trivial ones, until all of them are solved, using the results of previous calculations for each new step. The topological ordering of nodes in groups (by the longest distance from root) is convenient because the score for nodes with similar distances can be calculated concurrently, since their values depend only on the scores of the lower distance



	1	2	3	4	5	6
1	_	_	2	_	_	_
2	_	_	1	1	1	_
3	_	_	_	_	1	_
4	_	_	_	_	_	3
5	_	_	_	_	_	1
6	_	_	_	_	_	_

Figure 3.3: Toy DAG problem: (a) Directed acyclic graph with numbered nodes and edges. The superscript denotes the linearization index of each node while the subscript denotes the node number. (b) Equivalent scoring function that shows the edge weights (W_{ij}, i being the first column and j being the first row) connecting the nodes in tabular form.

nodes. The score for a given node calculated as a result of such recursive procedure represents the maximum or minimum distance from the root on a graph defined by weight matrix W. The actual path can be easily extracted by backtracking the graph, i.e. traversing the graph in the reverse direction and always choosing a particular edge which contributed to the score of the current node. The same procedure can be used to find optimal path for any pair of nodes, provided that such path exists.

To explain how score functions (figure 3.3(b)) are implemented with Race Logic, consider the job of one node in the graph shown in figure 3.3(a). It needs to

choose the MIN of multiple different inputs, where each of those inputs is penalized by a constant value. If values are represented by a delay from a reference point t (the start of the computation), we can add a constant c to a value by simply delaying it by c time steps. More concretely a score of n, is represented by a Boolean signal 1 appearing at the output of the node n unit delays after t. As seen before, when a signal is encoded in time, the min operation on a node in the graph receiving multiple inputs is equivalent to passing along the first arriving 1, which can be implemented with a simple OR gate.

Why this works for each and every node in our DAG is by virtue of the principle of optimality which is at the heart of dynamic programming. It states that the largest problem to be solved is fundamentally identical to all its sub-problems and hence the shortest path to a given node is going to be constructed out of shortest paths of its topological predecessors. Similarly, the AND gate passes the last arriving 1, since the optimal longest path is going to consist of longest path at each previous node, causing the AND gate to perform the MAX operation. The shortest/longest path DAG problem is therefore solved by measuring the time to propagate the signal from the root node(s) to the output node(s) for a graph, in which all nodes are replaced with OR/AND gates while edges are replaced with corresponding delays. Figure 3.4 (a,b) shows an example of a particular DAG

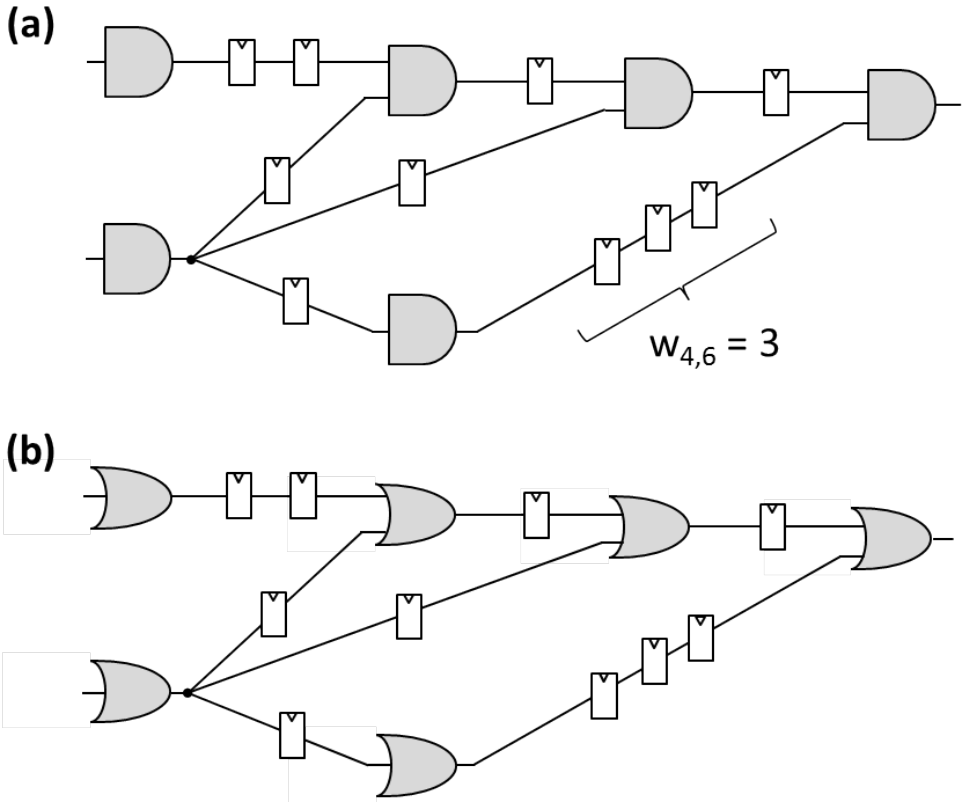


Figure 3.4: Race formulation: (a) Longest path AND race formulation achieved by replacing edges with flip flops and nodes with AND gates. (b) Dual shortest path OR race formulation.

with two input nodes and one output node converted to AND-type and OR-type Race Logic circuits.

For synchronous Race Logic, the unit delay is assumed to be equal to one clock cycle so that D Flip Flops (DFF) gates implements delay elements. In particular, DFFs can be shift chained for the cases where the edge weight is a small number or, alternatively, an encoded configuration can be used to implement larger weights. Note that for practical reasons very large weights (or more the specifically max

weight ratio) should not be too large, unless the weight is truly infinite (which can be implemented as a missing edge). The edit graph can be now thought of as a very deep pipeline, with competing paths to the final node from the root node, with all the flip flops initialized to 0. To initiate a race computation, both for the OR and AND types Race Logic, the input nodes are given a steady value of 1. With every new clock cycle, the 1 signal propagates down the edges of the graph until it reaches another node, where it gets delayed until the other inputs of the node are also 1 in the case of AND-type Race Logic, or until it just propagates through to the next edge in the case of OR-type Race Logic. For the specific DAG shown in Figure 3a, it takes two cycles for the 1 signal to propagate to the output node and it can be easily verified that this corresponds to the shortest path. Note that the shortest/longest path value can be converted back to the traditional representation with a simple counter which simply counts the number of clock cycles taken to reach the output.

Figures 3.3 and 3.4 describe the main idea for digital race logic. In these figures the weights are assumed to be fixed, so that a particular implementation finds the shortest or longest path in a fixed-weight graph. Implementing a fixed weight graph in a hardware, such as field programmable logic arrays (FPGA), just to find a shortest path may not practical. This is because implementation time overhead (e.g. configuration time in case of FPGAs) is rather substantial and

will certainly overwhelm the running time of finding a shortest path between one pair of nodes. Instead, a more practical situation is to have a fixed weight graph in which some edges are controlled by some external condition, e.g. matching condition between two letters in a string (Figure 3.5). This simple modification allows for efficient reuse of the same race logic hardware, because the external conditions modify the structure of the DAG and result in different shortest path between the same nodes.

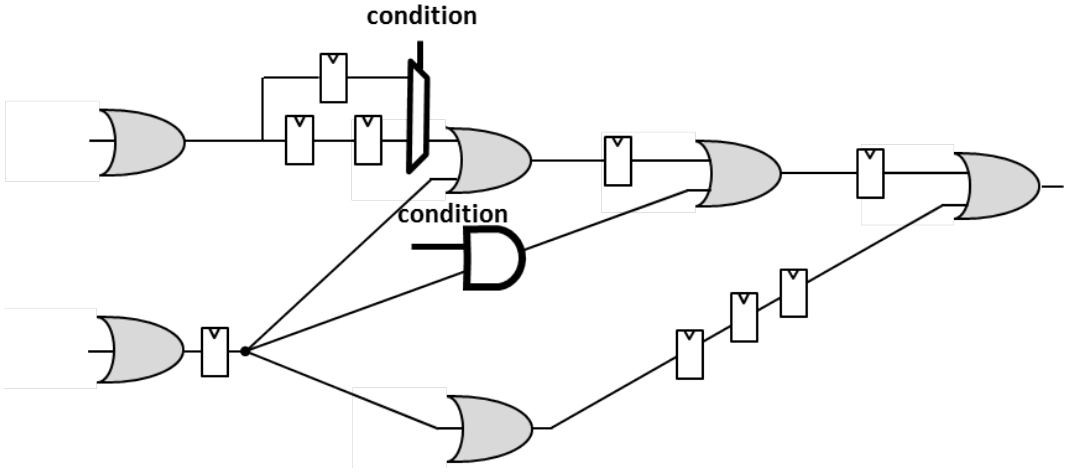


Figure 3.5: Race formulation: (a) Longest path AND race formulation achieved by replacing edges with flip flops and nodes with AND gates. (b) Dual shortest path OR race formulation.

In general, the simplest operation with race logic is to check whether the score is larger (or greater) than a certain value, i.e. sleaf $\geq$ S. This is trivially accomplished, e.g. by checking logic after S + 1 clocks (e.g. with the help of down-counter loaded with S) if flip-flop at the output of the leaf node in question

has value of 1 propagated from the root node(s) in digital logic (Fig. 4a). By feeding every input of graph node to a circuit shown on Figure 4c the actual path corresponding to the shortest or longest path can be extracted. The idea, which works for strictly nonzero weights, is to provide an extra flip-flop per node input to trace which input(s) arrived first (or last) for OR (AND) type race logic. Initially, these flip-flops are cleared. Once the signal has propagated to one of the inputs, the particular flip-flop is toggled which disables (via EN signal) toggling of other flip-flops associated with inputs of a given node. The toggled input flip-flops can then be used to extract the actual path. Lastly, note that the number of flip-flops can be greatly reduced by sharing it among different nodes output edges. For example, Figure 3.5 shows that only one flip-flop is required to implement connections from gate 2 to gates 3, 4, and 5 as compared to three in the original implementation on figure 3.4.

3.2 Sequence Alignment

While the point of our work is to explore the computational potential of races, we use the well-studied problem domain of sequence alignment to test the potential of this new logic. In this section we will touch upon the extensive body of prior work on sequence alignment, talk about an important systolic array realizations and discuss other implementations ranging from supercomputing platforms

to FPGA related applications. A common problem in bio-informatics is to estimate the similarity between DNA or protein sequences and there are 2 major types of motivations for DNA based sequence alignment. Currently, most whole genome projects use a shotgun sequencing strategy for genome sequencing . In a first step, genomic DNA is sheared into small random fragments. Depending on the technology, these are sequenced independently to a given length. Powerful software algorithms are then utilized to piece the resulting sequence reads back together into longer continuous stretches of sequence (contigs), a process known as de novo assembly. This is generally done when the reference sequence is unknown or there is no attempt being made to match to a reference sequence. In such applications sequence alignment forms the base alignment algorithm that is used by contig and scaffold generating software to map out the entire sequence. On the other hand, detection of issues such as aberrant methylation, in DNA sequences can that lead to cancer detection as well as a whole host of other genetic diseases, that allow a patients DNA to be mapped against a known disease to check for vulnerabilities. Alignment of 300M short reads to the Human genome takes roughly 5 hours when running on a system with dual 12-core Intel Xeon processors and 100GB of RAM. Accelerating alignment would shorten the diagnosis time, thereby allowing faster responses and increasing the number of patient samples that can be analysed per day. This would facilitate bridging the gap between research and

practice, enabling the diagnosis techniques developed to become part of routine clinical procedures.

Moving to an engineering perspective, these input patterns can have different alphabet sizes varying from 4 in the case of DNA (A, G, C, T representing the neucleo-bases) to 20 for a closely related protein comparison problem, in which strings consists of letters representing a particular amino acids [10]. A typical string similarity metric originating in information theory is the Levenshtein distance, also known as edit distance. It can be intuitively understood as the number of edit operations namely: Insertions, deletions, and substitutions, which are required to convert one string to another. To understand these edit operations let us consider string P = ACTGAGA of length N = 7 and string Q = GATTCGA of length M = 7.

Figure 3.6 shows two methods of converting P to Q. Strings that have a higher degree of similarity between each other will require less number of edit operations to convert from one to another. An alternate representation of such an operation that is very insightful is shown in figure 3.7

Figures 3.6(a), (b) show two methods of converting string P to Q. Here, columns with top row spaces represent insertions while bottom row spaces are deletions, and when lumped are known as indels. Columns with the same characters in both rows are known as matches and different ones are known as mis-

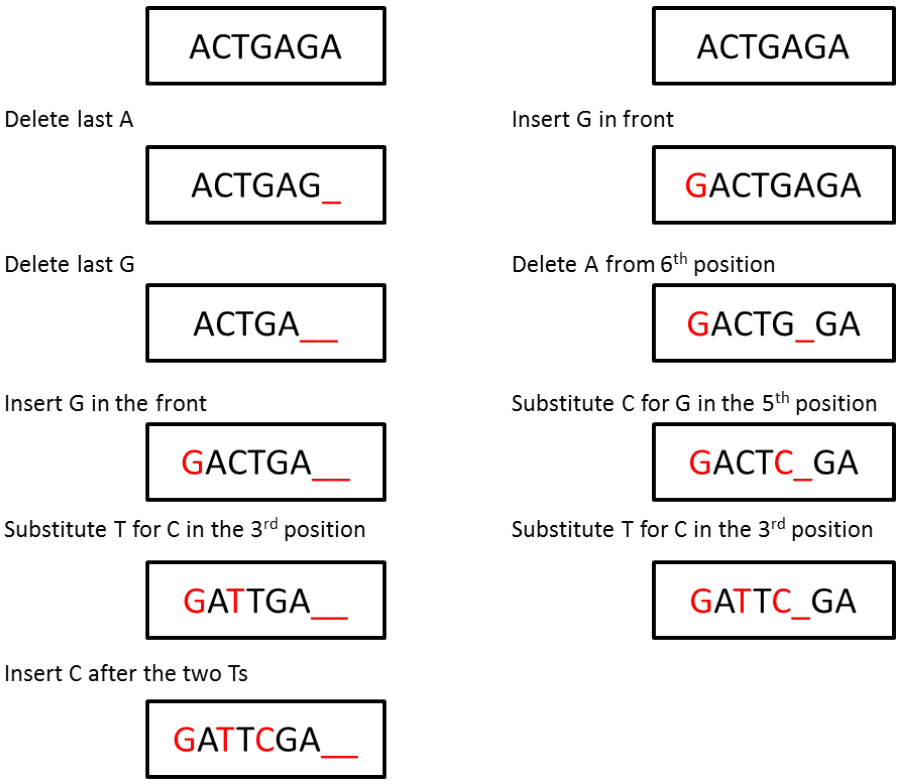


Figure 3.6: Edit operations: Multiple different ways in which simple edit operations such as insert delete and substitute can be used to convert toy DNA sequence P = ACTGAGA" to Q = GATTCGA.

matches. In particular, the first method (Fig. 1a) involves deleting letters C, G and A and inserting G, T and C, while the second method (Fig. 1c) deletes string P completely and inserts string Q. An important point to note is that even though the alignment shown in Figure 1c has no matches and is the worst case, it is still an allowed alignment. Also, the number of matches plus the number of mismatches plus the number of indels can be equal to the sum of the length of the two strings, i.e. N+M in our case, as is shown in Figure 1c, but can never exceed

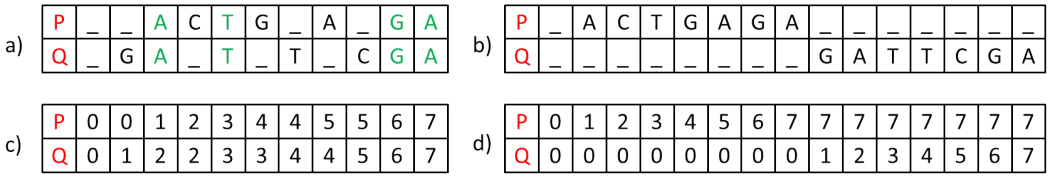


Figure 3.7: Alternate alignment representations: panels (a) and (b) show two different methods of aligning the sequences P and Q from figure 3.6. Panels (c) and (d) are the respective alignment matrices.

it. Figures 3.6(c), (d) are alternate representations for two considered alignments, where the number in any position denotes the number of symbols present in figures 3.6(a), (b) up to that particular position. Note that there is only an increase in numerical value at a particular position when it houses a symbol and not a space. This representation is known as the alignment matrix as each column can be thought of as a coordinate in a two dimensional N M grid which composes the edit graph(3.8).

The edit graph is a directed acyclic graph (DAG) that is a two-dimensional representation of all the possible alignments between the two strings. Any particular alignment is just a path in this graph where every edge corresponds to an edit operation. The arrows show all the possible alignments; the vertical arrows representing insertions, horizontal arrows representing deletions and diagonal arrows representing matches/mismatches. For example, dark blue and dark red arrows on Figure 3.8 correspond to the two specific alignments shown on Figure 3.6(a), (b), respectively. Analyzing the merit of any particular path in the graph is equiv-

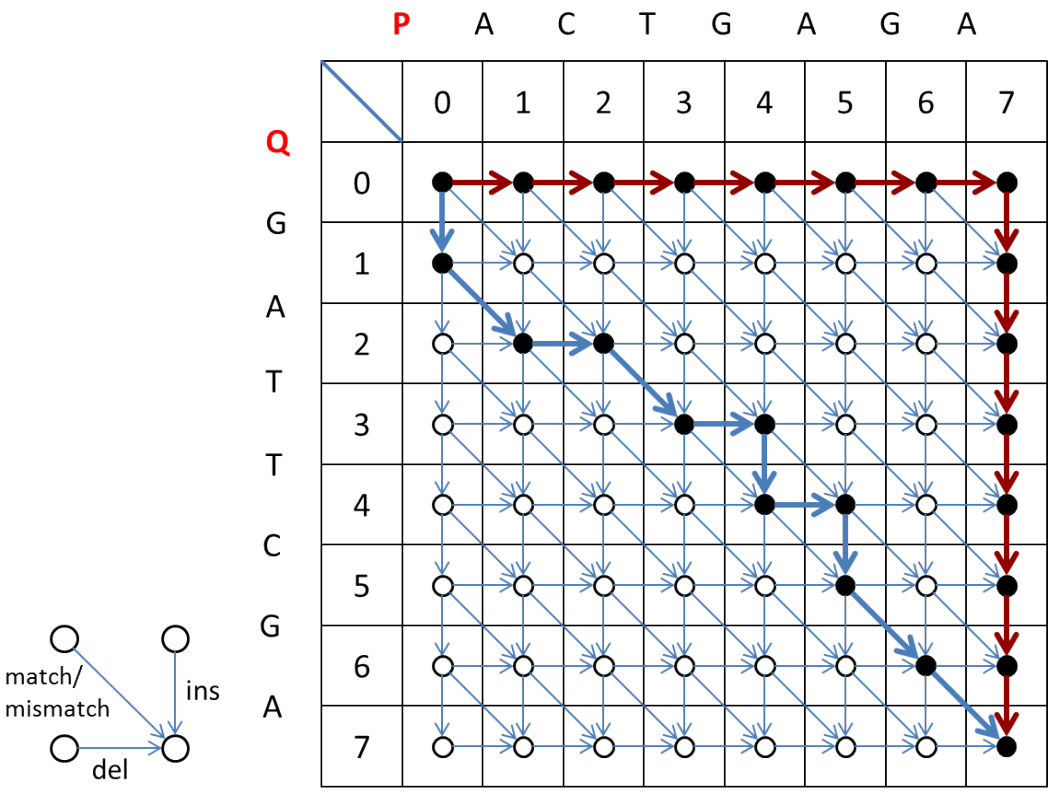


Figure 3.8: Edit Graph representation of sequence alignment. Dark red and dark blue paths are representative of the alignments shown in 3.7 (a) and (b) respectively.

alent to analyzing the merit its corresponding alignment. Given any two strings there are a large number of different paths and alignment matrices, each with its own arrangement of matches and indels. To determine the relative merit of one particular alignment over another the concept of a score matrix is introduced, which effectively defines the weight for each edge in the edit graph. Determining the goodness of the alignment is therefore finding either the longest path in the graph in the case when matches are assigned the highest values in score matrix ,

or, alternatively, the shortest path in the opposite case. Note that in general the penalty for the mismatch may also depend on a particular pair of letters.

(a)

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(-, P_j) \\ s_{i,j-1} + \delta(Q_i, -) \\ s_{i-1,j-1} + \delta(Q_i, P_j) \end{cases}$$

(c)

	A	C	T	G	_
A	1	0	0	0	0
C	0	1	0	0	0
T	0	0	1	0	0
G	0	0	0	1	0
_	0	0	0	0	0

(b)

$$s_{i,j} = \min \begin{cases} s_{i-1,j} + \delta(-, P_j) \\ s_{i,j-1} + \delta(Q_i, -) \\ s_{i-1,j-1} + \delta(Q_i, P_j) \end{cases}$$

(d)

	A	C	T	G	_
A	1	2	2	2	1
C	2	1	2	2	1
T	2	2	1	2	1
G	2	2	2	1	1
_	1	1	1	1	1

Figure 3.9: Score matrices: Panels (a) and (c) show score matrices that convert the edit graph problem to a longest path problem by rewarding matches with higher scores and mismatches with lower ones. The dual of this shown in panel (b) and (d).

Example max and min score functions are shown in figure 3.9(a),(b) respectively, where i and j are row and column indices as shown in 3.8. Applying equation in figure 3.9(a) and score matrix from figure 3.9(c) converts the alignment problem to a longest path problem by rewarding matches with an increase in the score by 1, while using equation in figure 3.9(b) and score matrix from figure 3.9(d) penalizes indels by 1 and mismatches by 2 and is equivalent to a shortest path problem. It is also worth mentioning that finding longest and shortest path with score ma-

trices in figure 3.9(c),(d) are equivalent problems. The shortest path formulation is more suitable for the considered implementation is it provides a solution in a shorter time and hence is more conducive to circuit implementations.

Not only is the edit graph representation a handy tool for visualizing paths and their corresponding alignments, it is also closely tied to the concept of dynamic programming. Each node on the edit graph calculates the score corresponding to the optimal solution of the sub-problem i.e. either shortest or longest path (depending upon the score matrix) from the root node to itself. Adjacent nodes utilize these optimal solutions to calculate their own score as the computation wave proceeds along the diagonal. The edit graph itself consists of all possible alignments represented as paths from the root node to the end node and hence the above method guarantees searching of the entire space for the most optimal alignment between the given strings.

The above concepts were first brought to light in seminal works by Needleman and Wunsch [34] and Smith and Waterman [43] which in-turn spurred a considerable research interest in software and hardware methods for string comparison. Algorithms for string comparison vary from brute force dynamic programming methods to heuristic solutions that do not search the entire space but provide quick solutions [2]. The major obstacle for hardware implementations of DP sequence alignment had to do with the area complexity as the similarity metric for

comparison i.e the score is cumulative and increases with array size. Any ASIC implementation would then need processing elements(PEs) that can store this worst case cumulative score and would hence lead to large, string length dependent sizes.

Lipton and Lopresti proposed a systolic array solution, that using maximum score dependent modular arithmetic, limits the number of bits of data that needs to be stored as well as shared between processing elements [24]. Hence, they were able to make sure the area scaling issues are mitigated, at the cost of extra circuitry outside of the systolic structure to recalculate the original score. To reduce the interconnect overhead, the systolic architecture utilized a tight encoding scheme that interleaves the alphabet and scores. Lipton and Lopresti not only addressed the area issue but theirs was the first paper to talk about anti-diagonal independence of elements in the edit graph and utilized this property for fine grain parallelism. The resultant hardware was a linear systolic array whose processing elements could differentiate between the alphabet and scores as perform comparison and addition operations. Newer architectures [14, 15] have built upon this Lipton and Lopresti work by adding markers in processing elements to trace back optimal similarity paths. Other platforms on which sequence alignment has been performed, range from networked DEC Alpha workstations [18], to GPUs [25, 19], to Supercomputers. SIMD supercomputing platforms, such as MasParMP2 [18], that have multiple low complexity processing elements

that perform fine grain computation on multiple data streams parallely, achieving high performance at reasonable cost. On the other hand, general purpose MIMD supercomputer, Paragon [18] seemed to have performance on par with the networked DEC Alpha workstations [18]. Other implementations include FPGA based reconfigurable platforms such as Jbits [13] that utilize re-configurability to create a custom architecture using the entire string as a parameter or by defining custom instructions on FPGAs that can handle multiple input and flags at the same time to speedup computation [22]. Custom ASIC implementations such as BioSCAN utilize heuristics and a very high density implementation that results in high performance [42]. More recently, multiple freely available software tools for alignment have been developed , including Soap [23], BWA [21], and Bowtie [20]. These tools utilise the latest pattern matching algorithms and hardware technologies to perform the alignment process quickly. However, there is reliance on extensive computing resources to deliver this performance. For example, the 1000 genome project uses a 1192-processor cluster to align reads, while the BGI Bio-cloud computing platform has a current total of 14774 processors delivering 157T flops of performance though a shortcoming of such an approach is that simply scaling across more machines cannot keep up with the projected growth of sequenced data which far exceeds Moores Law.

3.3 Sequence Alignment as a Kernel

Though DNA sequence alignment is one specific application, the major concept of string, sequence, time series alignment can be found in multiple places in literature. Though the nature of the edit graph is the same in all these variations, the meaning held behind the similarity measures is different.

Dynamic Time Warping (DTW) is one such method that was first introduced in the data mining community in the context of mining time series [4]. Since it is a flexible measure for time series similarity it is used extensively for ECGs (Electrocardiograms) [5], speech processing [36], and robotics [40]. The concept of a warp is slightly different from the levenshtien edit distance as it it measures the amount of squishing or elongation that needs to be done to covert one signal to another. This is generally done in speech/spoken word recognition. A figure similar to the edit graph is shown in figure 3.10(a) which shows how the warping function maps the squished data points from b_j to b_J to elongated data-points in a_i to a_I. This different interpretation is by virtue of the problem, the objective of which is to eliminate timing differences between the two speech patterns, A and B. Since speech recognition and DNA sequence alignment are different problems, the constraints on the problems are also different. DNA sequences, as has been discussed before, has some similarities between the strings by virtue of randomness and hence scores above a certain value can be discarded as non-alignments.

Similarly in speech recognition applications, there are some constraints that each warping path must follow. Outside of the simpler constraints such as monotonicity and continuity, the work by Sakoe and Chiba3.10(b), develops a global constraint band, called the Sakoe Chiba band, which limits the scope of warping. Important work done in this paper, analyses symmetric and asymmetric score matrices, as well as more complex connectivity patterns and concludes that symmetric scores perform better than asymmetric ones.

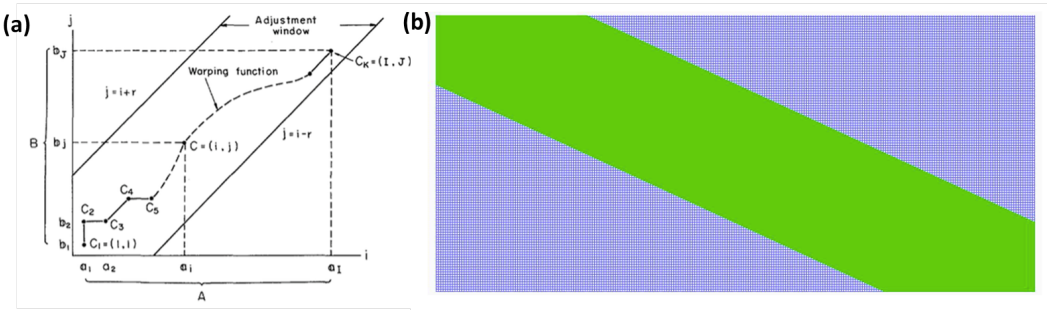


Figure 3.10: Panel (a) taken from [38], shows how the warping function locally stretches and squishes a to convert it to b. (b) taken from [32] shows the Sakoe-Chiba band that limits the warping score.

These constraints allow an efficient mapping to Race Logic. Firstly, due to the constraint band, the scaling of the problem can be reduced from 2D scaling to 1.X D scaling based on the size of the band. Also, higher similarity already map to lower scores in such a representation, and the weights are defined to be non-negative.

Another interesting application that uses dynamic programming based graph solutions is the stereo correspondence algorithm which is the process of obtaining depth information from a pair of binocular images. In more detail, assuming 2 cameras looking at the same 3D imagery, slightly displaced from each other, we need to find out which points in the image from camera 1 correspond to which points in camera 2. It is clear from inset in figure 3.11 that some points in each image will have no corresponding points in the other image as cameras will have slightly different fields of view and as well as occlusion from objects. Though different approaches exist such as auto-correlation, cooperative algorithms etc, we concern ourselves with the dynamic programming based approaches that simplify the image to image correspondence problem with a scan-line to scan-line matching problems. The scan-line matching problem, takes horizontal scan-lines from each image and matches them against each other to understand regions of similarity between them. Regions that are dissimilar are treated as occlusions (similar to the in-dels in DNA sequencing) and are used to extract depth information, and aggregating this data across all scan-lines in the image allows full reconstruction on the 3D scenery.

Many of these algorithms including beat retrieval in music processing and image seam carving utilize such regular 2D graph structures structures to convert matching to shortest path optimisation problems. In a lot of these applications,

the weights that are used are normalized to some maximum value and are hence scalable. DNA Global sequence alignment is known to be score matrix scaling invariant, while DTW also allows for weights to be scaled to within an order of magnitude of each other. More application specific details tend to bring out custom details such as the global constraint band in speech processing which lend themselves very nicely to Race Formulations. We can see that Race Logic is not just limited to the DNA sequence alignment problems but to a much wider set of problems that are pervasive in computation today.

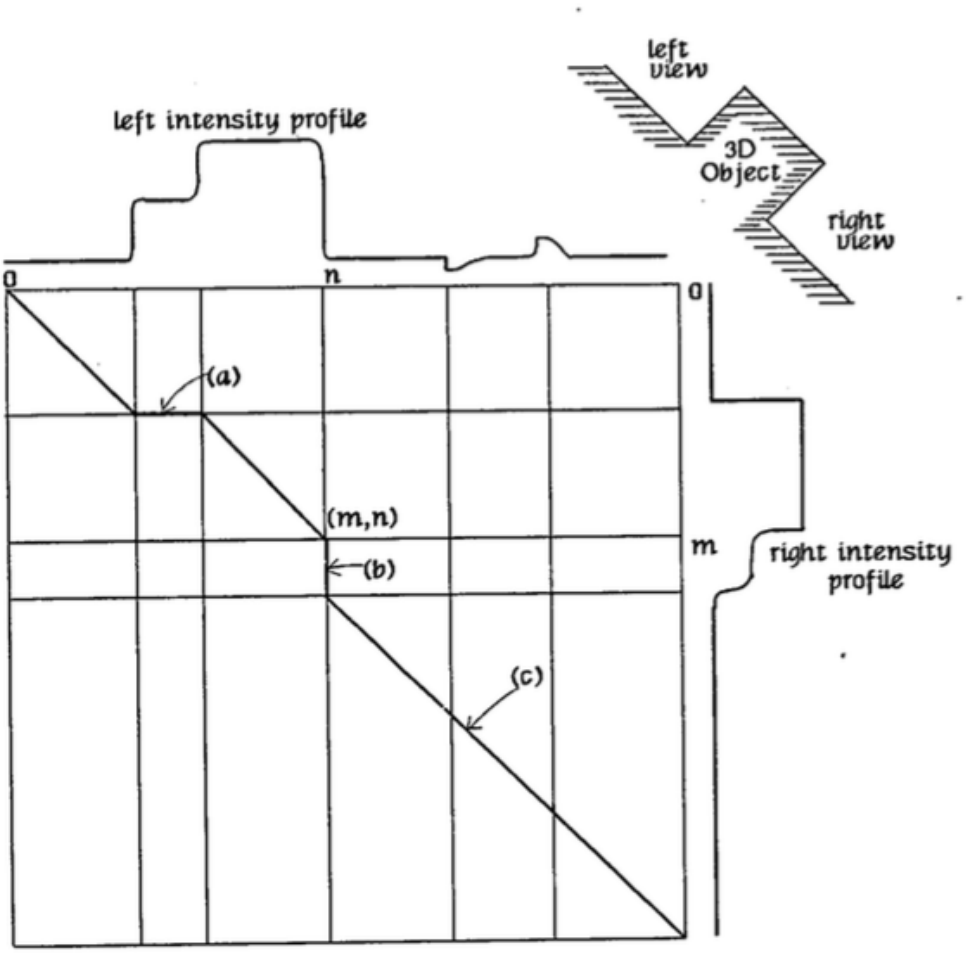


Figure 3.11: Shows the process of determining occlusions and corresponding regions in a stereo image. The left and right intensity profiles are as shown for 2 views of the same object. Edit graph like structure used to determine regions of correspondence and that of occlusion. Taken from [35]

Chapter 4

Race Logic and Application Study

Since the fundamental principle of Race Logic is encoding information in delays, the choice of delay element has a central role in the performance of the Race Logic architecture. In a synchronous implementation, the delays are constructed out of D-flip-flops. This allows for a global clock network to be responsible for quantization of timing intervals, whose time period is only determined by the number of logic level between said flip-flops. In this chapter, we will use the sequence alignment problem as our benchmark and use it to compare previous ASIC implementations against a synchronous race formulation with flip flops as delay elements.

To summarize our contributions,

- We show that our synchronous Race Logic implementation is both feasible and practical through a synthesizable ASIC implementation for an important instance of dynamic programming optimization.
- We demonstrate that our specific implementation is more efficient (as measured in latency, throughput/area, and energy) than best known traditional designs by factors of 4, 3, and 200, respectively.
- We describe the design space of Race Logic for this class of applications more broadly, and propose and model several important optimizations that make this new class of designs even more useful.

Now we have seen that the string similarity problem is equivalent to searching for a shortest/longest path in the edit graph, it is pretty clear how we can convert this to its equivalent race formulation. The implementation that we are comparing against is the Lipton and Lopresti systolic architecture,[24] which was first demonstrated in the year 1985. Though this problem is not representative of DNA sequence alignment today, as the field has evolved exponentially in the last couple of decades, we still use it as a comparison point as it boasts a high degree of optimization at multiple different levels:

- It is a linear (1D) architecture, to solve a 2D problem. Newer, more advanced architectures are 2D in nature, and have poorer area scaling behaviour.

- The score function used is very simple. This allows for elimination of a certain set of dependencies, as score values at each node depend only on their diagonal predecessors. This anti diagonal independence was first observed by these authors and exploited to improve fine grained parallelism of this architecture.

- More area optimizations preformed by interleaving input data and the score for each node. With each clock tick, up-to N parallel operations performed based on internal score state as well as newly shifted in data.

- Processing Element (PE) sizes are kept to a minimum by utilizing modular arithmetic techniques, thus ensuring only the least significant bits are transferred between processing elements. This is also an advantage of the simple score matrix as adjacent blocks cant have scores differing by a large value. Since, modular arithmetic is being used, sizes of adders and comparators are also relatively small and allow fast operation speed.

The lack of flexibility (limited to only one score matrix) and size of the score matrix makes this architecture a very specific one which allows for a lot specifically tuned optimizations. This results in a low area footprint, high throughput

architecture. Though it has been implemented in very old technology, these optimizations still hold their value today.

Figure 4.1 details the functioning of the Lipton and Lopresti architecture. The array consists of a series of processing elements arranged linearly with data moving in two anti-parallel directions. This allows each character in the sequence to be compared with every other. Each row in the figure shows a new time step as the two input sequences to be compared are interleaved with the initial scores that have been calculated before hand. When two non-null characters enter a processor from opposite directions, a comparison is performed, indicated by the dark circles in figure. The result from each comparison calculates a score of an equivalent node (indicated by the circled numbers) which , which is shifted out at the next clock cycle and is used to calculate scores from further nodes that are dependent on it. Finally the output scores are calculated after all dependencies are satisfied and shifted out with the strings. The red dashed lines show the edit graph within the temporal functioning of the array and provides a pictorial insight into how information flows through the systolic array.

On the other hand, figure 4.2 shows two equivalent race logic implementations of the sequence alignment problem with the score matrices from figures 3.9(a), (c). The grey dashed lines show the unit cell that, in both cases, consists of a flip flop, that is shared between multiple parallel paths to conserve area and power,

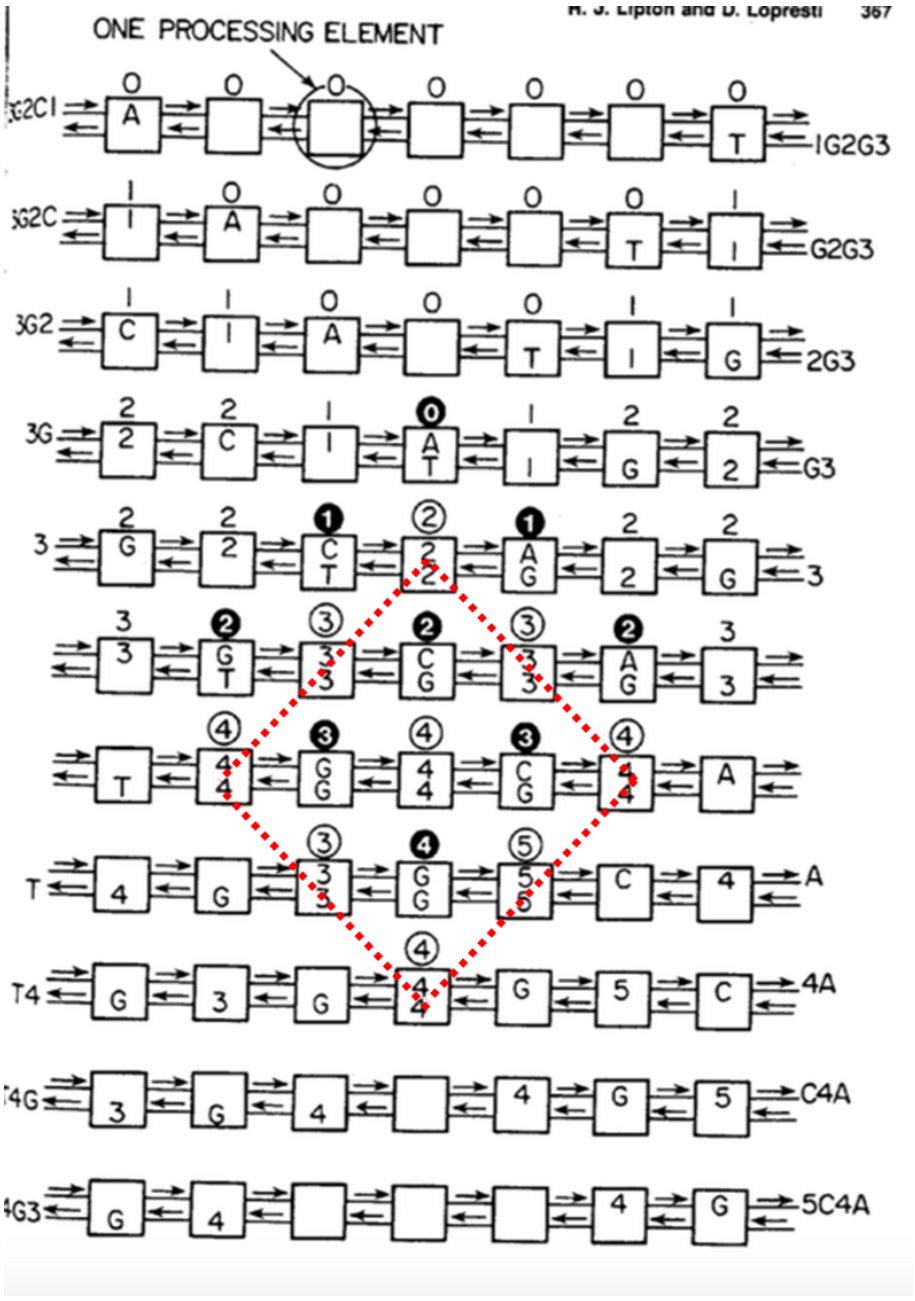


Figure 4.1: Original figure from the Lipton and Lopresti systolic engine.[24]

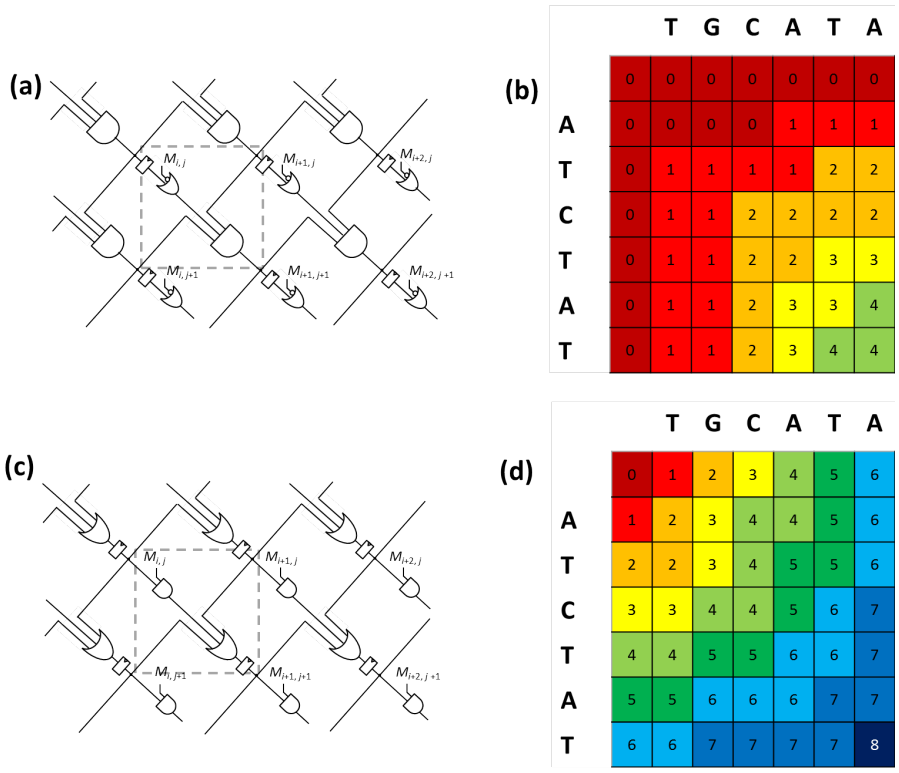


Figure 4.2: Race logic implementation of sequence alignment. Panels (a) and (c) show the circuit level diagrams for Race implementation (AND and OR) of score matrices from 3.9(a,c). Panels (b),(d) show the propagation of the race wavefront at each clock cycle until it reaches the output.

a control signal or a condition which is determined by matching of the relevant nucleotides, and an OR/AND gate to select first or last arriving races. Shift registers are used to input and store the nucleotide sequences, which go through XOR based comparison(not shown in figure) to generate the $M_{i,j}$ signals. Once the data has been completely shifted into place, the array is ready to compute. In some sense, the information is already encoded in the spatial layout of the DFFs that depend on the input arrays and a rising edge is fired at the input and kept

high, until an output an edge is received at the output. A counter is used to count the number of cycles that it takes and hence computes the final score. Note that the OR gate based implementation is used instead of the AND gate based one as it rewards matches with lower latency and hence allows for faster operation and higher throughput. The DFFs can then be reset, with new input sequences to be barrel shifted in, which refreshes the array with new input data for the next race to begin.

To make sure that the comparison is a fair one we implement the Lipton and Lopresti systolic architecture using a recent standard cell technology and include all of the area optimizations as well as encoding schemes that were implemented in the original architecture. The process used is an AMIS $0.5\mu m$ process and the studies are done using both OSU standard cells as well as AMIS standard cells to study the tradeoffs involved in using a different standard cell set. For the implementation of both architectures, a parametrized and scalable Verilog code is synthesized using Synopsys Design Vision tool to get estimates of area. Power and timing information is obtained using Synopsys Primetime tool using a representative set of input vectors for multiple problem sizes. Since the power consumed by the architectures is highly dependent on the input vectors, random input vectors cannot be used. A specific set of input vectors that follow the correct encoding is generated using a test-bench. These simulations are performed

using the Modelsim tool, which generates toggle information of each net on the synthesized netlist. This toggle information is then used by Primetime tool with 100% coverage (confidence metric) to estimate power values.

4.1 Analytical Estimates and Results

Among the simulated performance metrics, area and latency scaling with string length are the easiest to understand. The area of the Race Logic scales quadratically with N (problem size) while the systolic array scales linearly, though the systolic array requires 2N+1 processing elements due to the interleaving of data and scores. Figure 4.3 shows this area scaling behaviour. The crossover point is representative of the relative complexity of the systolic processing element(PE) with respect to the Race Logic cell. The systolic PE has adders, flip flops, comparators and encoders(detecting data vs score) which cause the cell to be complex compared to the race logic cell which contains a flip flop and 2 gates. At about N = 18 (324 Race Logic cells) does the area scaling of Race Logic catch up with the area of the systolic array.

For both architectures, the latency scales linearly with N. For the systolic array, it takes 2(2N+1) cycles to complete the computation independent of the data, while in the worst case scenario(4.4(a)), for Race Logic, i.e. when the strings are completely mismatched, it takes 2N-1 cycles and only N-1 cycles in best case

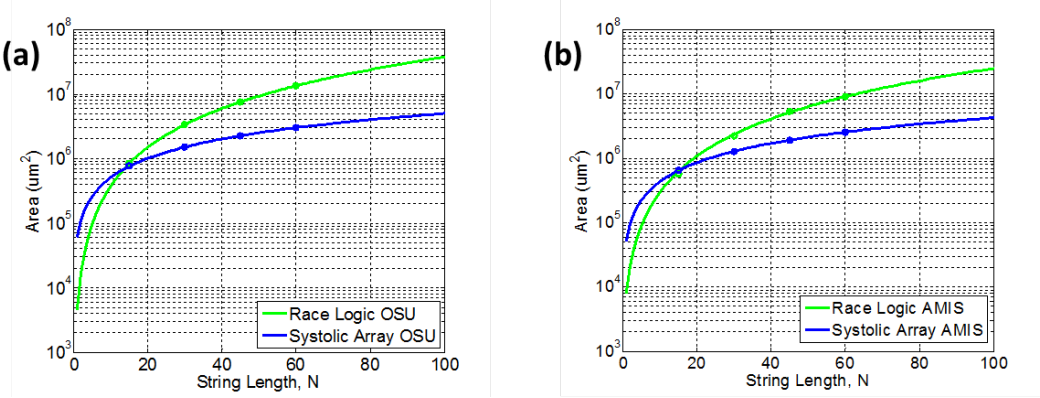


Figure 4.3: Area scaling with respect to problem size for multiple standard cell sets.

scenario(4.4(b)), i.e. when the strings are completely aligned. Also since the Race Logic cell design has just 2 logic levels between flip flops, it can be clocked faster than the systolic cell that has a more complex design. This allows the worst case performance of Race Logic to outdo the systolic array, as shown in Figure 4.5

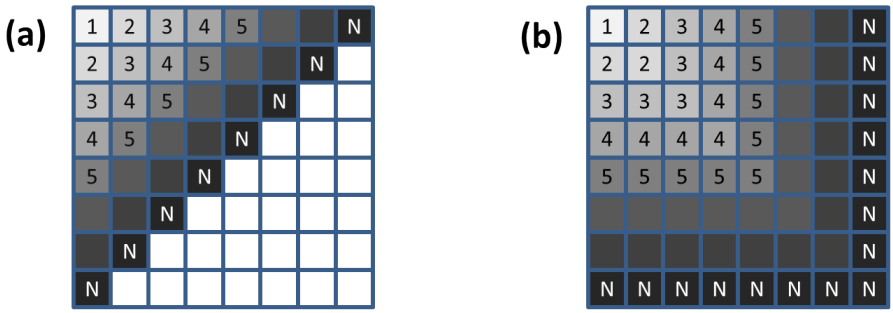


Figure 4.4: Flow of the wavefront in the worst case(panel (a)) and best case (panel(b)) alignments. Though these cases will never occur in testing real data, they are good test cases to get bounds on performance of the architecture.

The overall throughput, shown in figure 4.6, reflects both the area and latency tradeoff in one figure. For smaller values of N, the latency advantage of Race

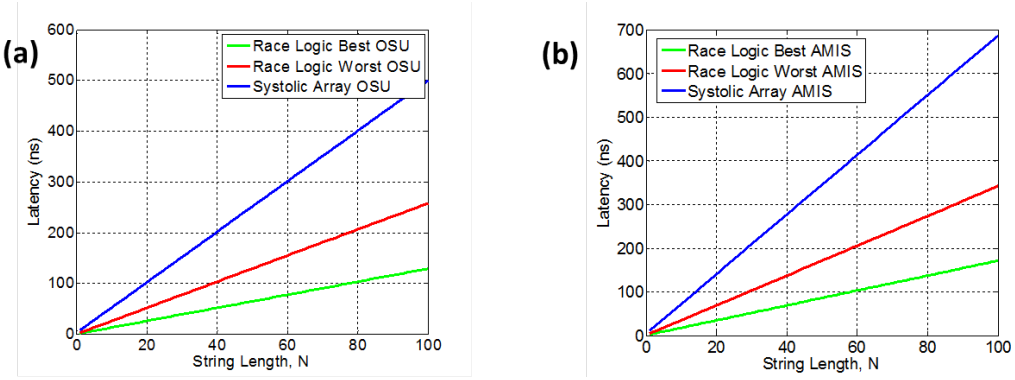


Figure 4.5: Latency with respect to problem size for multiple standard cell sets

Logic allows higher throughput, but as N grows, it gets overwhelmed by increases in area. Crossover points for higher throughput happens around N=60, 70 for OSU vs AMIS standard cells respectively (figure 4.6).

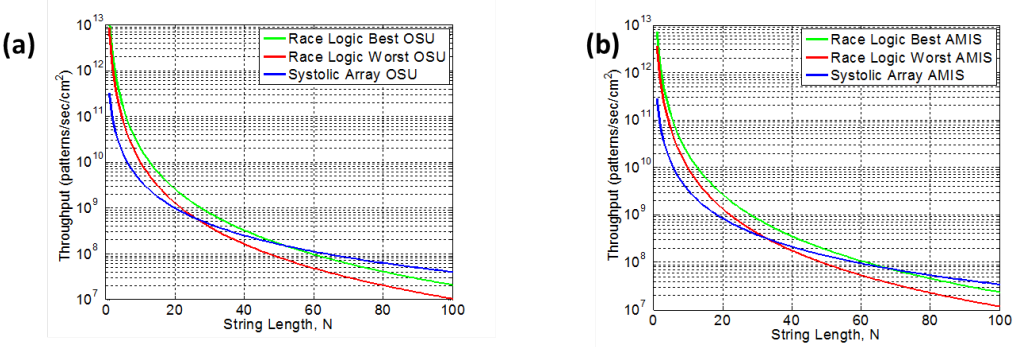


Figure 4.6: Throughput with respect to problem size for multiple standard cell sets

Derivation of energy and power requires a little more detailed analysis. Lets assume that C_{clk} corresponds to the capacitances of DFFs that are clocked every cycle, and hence having an activity factor of 1, while the $C_{non-clk}$ corresponds to

all other capacitances that have data dependent activity factors. For both the best and the worst case scenarios all the non-clocked capacitances in the entire architecture are charged once per comparison. This can be seen very easily in the worst case, by following the horizontal and vertical edges on the edit graph, but is similar in the best case as the propagating 1 uses the diagonal delay elements to propagate to the extreme topological east and south blocks of the architecture. Hence the dynamic power dissipated can be written as

$$P = C_{clk}V_{dd}^2N^2f + C_{non-clk}V_{dd}^2N^2\alpha f \tag{4.1}$$

where α is the activity factor that is data dependent, V_{dd} is voltage supply, and f is a frequency of operation. Energy consumed per comparison can be calculated by multiplying power by the time taken per operation. Therefore energy dissipated per comparison for the best case and worst cases are

$$E_{best} = C_{clk}V_{dd}^2N^3 + (C_{non-clk} - C_{clk})V_{dd}^2N^2 \tag{4.2}$$

$$E_{worst} = 2C_{clk}V_{dd}^2N^3 + (C_{non-clk} - 2C_{clk})V_{dd}^2N^2 \tag{4.3}$$

correspondingly. The Equations 4.2 and 4.3 define the scaling law of energy and power with respect to N. Since C_{clk} and $C_{non-clk}$ are not known parameters we estimate them from fitting. The resulting equations from fitting for both the AMIS and OSU standard cell libraries are

$$E_{AMIS,best} = 2.65N^3 + 6.41N^2 \tag{4.4}$$

$$E_{AMIS,worst} = 5.30N^3 + 3.76N^2 \tag{4.5}$$

$$E_{OSU,best} = 1.05N^3 + 5.91N^2 \tag{4.6}$$

$$E_{OSU,worst} = 2.10N^3 + 4.86N^2 \tag{4.7}$$

where the units of energy are in pJ.

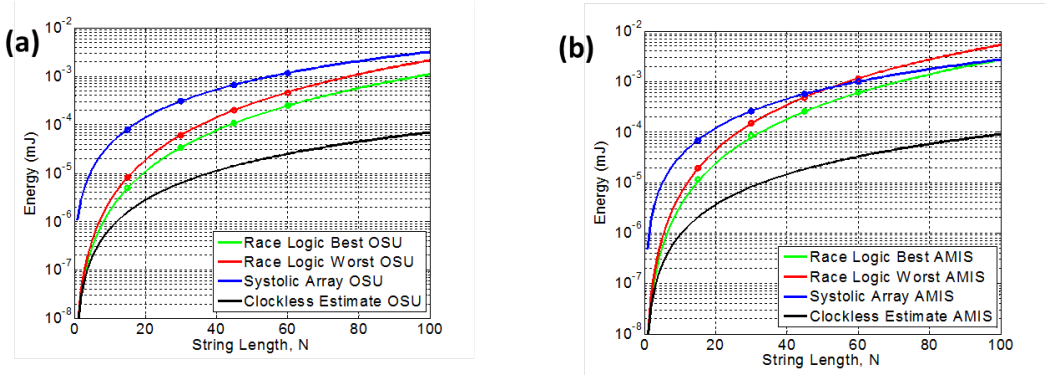


Figure 4.7: Energy with respect to problem size for multiple standard cell sets

The cubic energy scaling of Race Logic can be seen clearly in figure 4.7 as the red and green lines have a steeper slope that the blue of the systolic array that has a shallower quadratic slope, as 2N processing elements are running for 2N cycles. For the OSU standard cell set the crossover point is after 100 but for the AMIS cell set it lies around to 50 to 60 bases mark. As a reference a clockless estimate is also shown which shows the power spent only in the edge propagating

down the array and not its clocking. Note both systolic and clockless estimates have quadratic energy scaling behaviour.

4.2 Energy Optimized Architecture

One of the drawbacks of Race Logic is its third order energy scaling with string length N. By observing Equations 4.2 and 4.3 we can see that the capacitance associated with the clocked region of the fabric constitutes the cubic behavior. This is due to the fact that the area scales quadratically and the time taken per computation scales linearly, but most importantly this area is clocked every cycle. Fortunately, using a strategy known as a clock gating, this term can be greatly reduced. By exemplifying the worst case; i.e. the case of maximum energy per computation, from our given score matrix, we can see that there is a time dependent wavefront of the propagating Boolean 1 as is shown in Figure 4.4. This wavefront represents the cells where the flip flops are changing state from Boolean 0 to 1. The key observation is that cells that are away from the wavefront, i.e. the ones which have already changed state to 1 as well as the ones that are still 0 at this particular clock cycle are going to retain their state for the next clock cycle and hence do not need to be clocked.

By employing a data dependent clock gating strategy we can turn off regions of the chip that are not being utilized to save power. Due to the regular structure

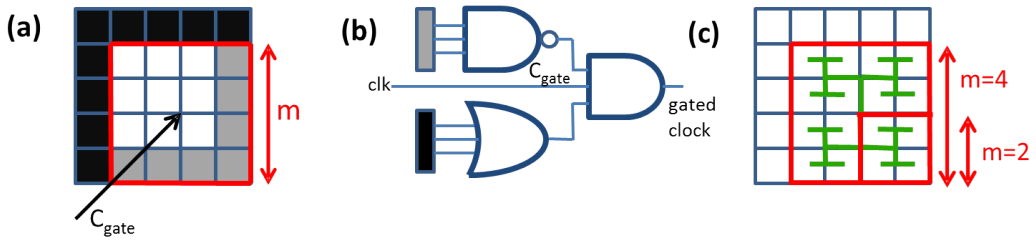


Figure 4.8: Clock gating: (a) granularity of the clock gating cell, pictorial representation of parameter m (b) clk gating circuitry (c) htree fabric and multiple clock gating granularities

of the Race Logic fabric the clock network can be designed as an H-tree. One of the major parameters that would determine the power savings would be the granularity of the H-tree, in other words, the number of cells that would be gated at once. Let us look at a 4x4 group of cells (multi-cell region) as shown enclosed in Red in Figure 7a. This multi-cell region can be thought of as the smallest group of cells that can be gated at once. During the operation of the circuit, if the cells that are grey in color have the Boolean value 1 then it means that the wavefront has crossed this multi- cell region and their values are not going to change in this operation. Also if the cells that are in black have the Boolean value 0, it means that the wavefront has not yet approached this multi-cell region. For both the above cases, the multi-cell region shown in Figure 7a doesnt need to be clocked. By activating the clock of the multi-cell region on the arrival of the Boolean 1 on the black cells and deactivating it when all grey cells are 1 we can ensure that this multi-cell region is clocked only for a limited period of time, hence reducing

energy consumption. Very fine granularity of this multi-cell region would increase energy dissipation due to a large number of multi-cell regions that require every cycle clocking, while very coarse granularity would mean clocking one multi-cell region for very long, also increasing energy dissipation.

To calculate the optimal granularity, we introduce a parameter m, which is the side length of one multi-cell region as shown in figure 7a. Now the worst case energy dissipation for the clocked part of the architecture is as follows,

$$E_w = C_{clk}V_{dd}^2N^2(2m-2) + C_{gate}V_{dd}^2N^2/m^2(2N-2) \tag{4.8}$$

where the first term represents the entire clocked capacitance being activated only for 2m-2 cycles (i.e., the worst case number of clock cycles one multi-cell region remains active) and the second term represents the gating capacitance that the clock distribution network still has to clock, with Cgate is the actual capacitance, (N/m)2 is the number of multi-cells regions and 2N - 2 factor is the total number of cycles. Solving for minimum energy, we get

$$m = \sqrt[3]{\frac{2C_{gate}(N-1)}{C_{clk}}} \tag{4.9}$$

The energy advantage due to clock gating is significant as it reduces the order of the equation from third order to fractional order, pushing it closer to the clockless estimate of quadratic behaviour. This causes the crossover point to be shifted by

more than 2 orders of magnitude, from 100 to 50,000. Figure 4.9 shows both pre and post gating energy dissipation.

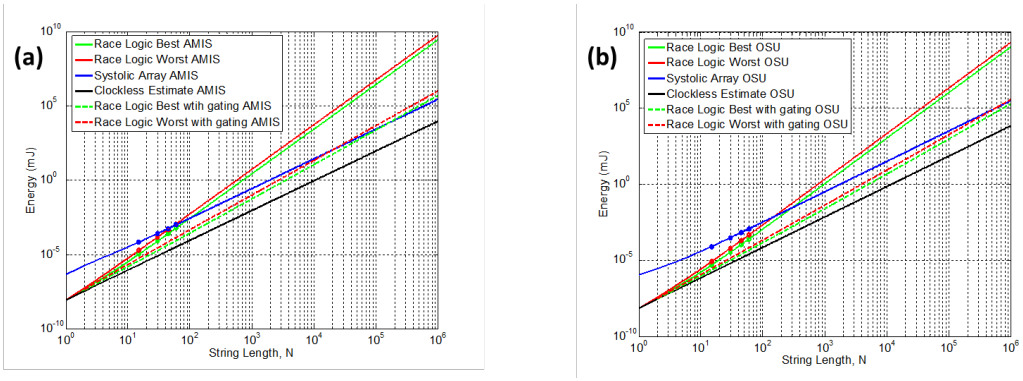


Figure 4.9: Clock gating energy savings: energy dissipation before and after clock gating for multiple standard cell sets.

Th power density of the system is also affected by the clock gating strategies as shown in figure 4.10. Race Logic shows a much darker operation as compared to the systolic array. The continuous clocking of the systolic array with every cycle is very power hungry, on the other hand, the time taken for the race to propagate through all the flip flops is once every N cycles at best which causes a much slower duty cycle. Essentially there is less computation to be done and it is being spread across a larger area of silicon, causing it run more power efficiently.

Similar trends can also be seen in the pareto plot shown in figure 4.11. This figure shows the energy vs latency for a certain problem size, N= 30. Since we would prefer lower energy and high speed, the most desirable place to be in would be the bottom left of the plot. The black lines show constant energy delay surfaces.

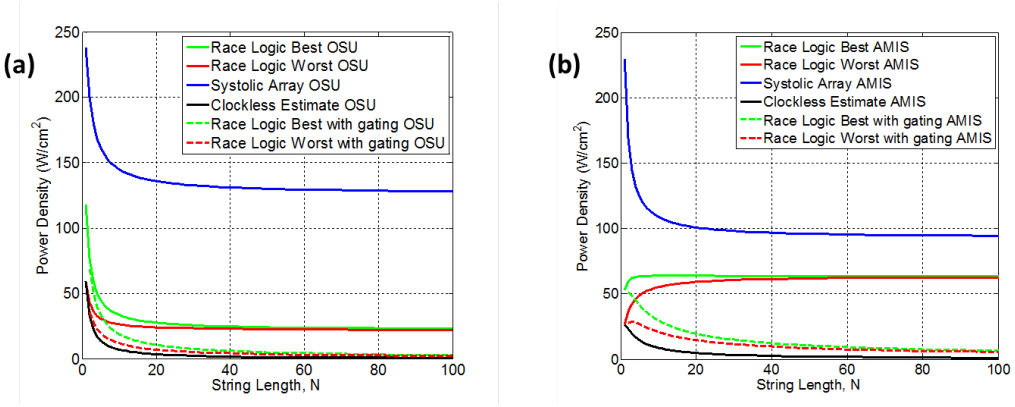


Figure 4.10: Clock gating power density: power savings before and after clock gating for multiple standard cell sets.

We can see that Race Logic outperforms systolic array implementations and pushes the pareto points closer to the origin. Not that the clockless estimate is the lower bound on energy as it assumes quadratic scaling and no power spent in delivering timing information.

4.3 Generalized Synchronous Race Architecture

The score matrix which was considered in a previous example is simple and easy to implement. However, score matrices for sequence alignment have evolved considerably from the time Lipton and Lopresti systolic architecture was published. Nowadays, important properties of the score matrix such as symbol size (N_{SS}) and dynamic range (N_{DR}) change from application to application and even the same application can have different dynamic ranges. One such example, the

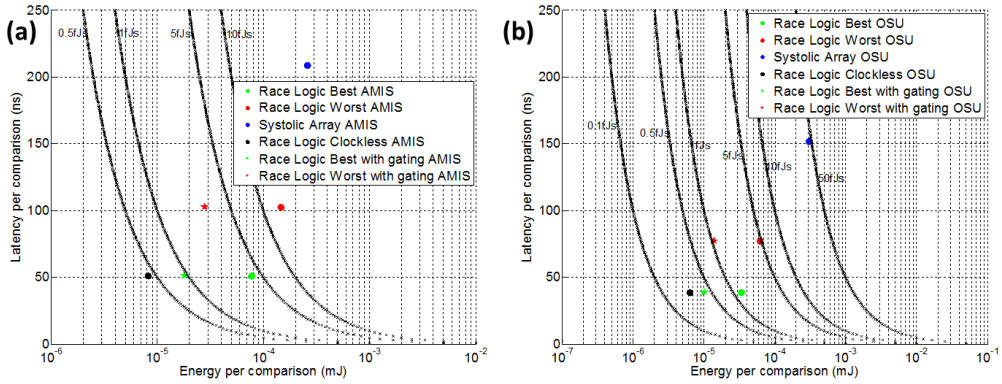


Figure 4.11: Pareto plots comparing Race Logic with systolic implementations for array size N= 30, across different standard cell sets.

modern amino acid score matrices, which go by industrialized acronyms such as BLOSUM62(block Substitution Matrix) and PAM250(Point Accepted Mutation), are large, complex matrices with above 400 elements and are highly tuned as a result of statistical behavior of amino acid sequences [9]. Therefore, it is worth investigating generalized Race Logic which can deal with different types of score matrices. The first step is to convert any given matrix to the form which can be used with generalized Race Logic architecture.

Using BLOSUM62 as an example (Fig. 4.12), we prepare the score matrices for its Race Logic realization, by keeping a few things need to be kept in mind. Firstly, since our preferred type of Race Logic is the OR race, we must ensure that the highest similarity corresponds to the smallest score and hence the lowest latency. The BLOSUM62 score matrix rewards perfect matches with positive

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 4.12: Block Substitution matrix or BLOSUM62.

scores, substitutions with negative scores and indels are generally of the lowest score. It is possible to invert the score matrix from longest path to shortest path one by understanding the value in score matrix are obtained. In particular, the origin of these score matrices are based on log-odds score calculations as shown below,

$$S(a,b) = \frac{1}{\lambda} log \frac{P_{ab}}{f_a f_b} \tag{4.10}$$

where S(a,b) is the score for symbols a, b, P_{ab} is the joint probability of alignment of a and b, f_a and f_b are the probabilities of alphabet a and b by themselves and λ is a scaling factor to get integer values of scores. By inverting the above equation, and changing the scaling factor we can convert all diagonal elements

from positive to negative and non-diagonal from negative to positive. The next step is to obtain the equivalent score matrix with all positive weights since negative or zero weights cannot be implemented in a straightforward way in Race Logic. The solution to that problem is to add a fixed bias to values of score matrix corresponding to the indels and double of that fixed bias to the remaining ones, as the latter are one rank ahead in the edit graph (figure 3.8) [1].

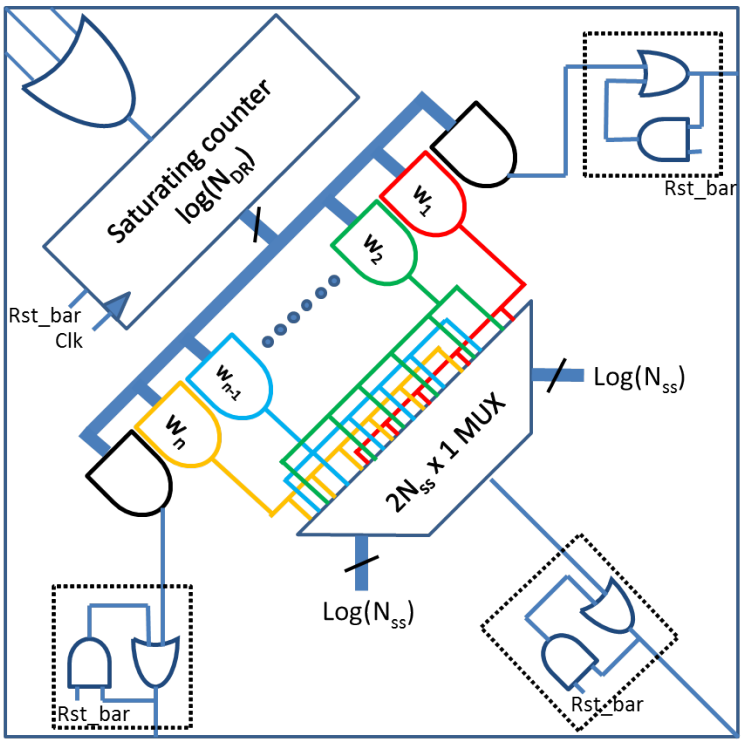


Figure 4.13: Generalized race logic architecture which allows a larger symbol space and high dynamic range compared to the implementation previously discussed

The score matrix now consists of elements ranging from 1 to NDR, with the scores along the diagonal being the smallest and indels being the largest. As can

be seen in Figure 2c, modern score matrices contain a lot of repeating scores. When using one hot encoded DFFs for realization of delay, the area of a single Race Logic cell scales linearly with dynamic range and hence may have serious area repercussions for large values of dynamic range. Binary encoding with a saturating up-counter allows us to save on area by reducing the number of DFFs for the same NDR as well as making sure that the counter doesnt overflow and restart the count. The generalized structure of complex Race Logic cell is shown in Figure 4.13, the Boolean 1 signal can come in either from the left, diagonal or top of the cell, which then passes through the OR gate and enables the saturating counter that begins to count 0 to NDR clock cycles. The output of the each colored gate, represents a specific weight, which will trigger as soon as the desired weight is reached and the weight that is desired can be selected from the MUX whose inputs are the encoded forms of the alphabet. To ensure that the output signals that are generated are not pulses but fixed Boolean 1s, the set on arrival circuit is placed which is reset at the end of each computation.

4.4 Discussion and Conclusions

As it is mentioned earlier, area scaling of the Lipton and Lopresti architecture is linear in N, while Race Logic has quadratic area behavior. In spite of such unfavorable area scaling laws, the constants associated with Race Logic are smaller

than that of the systolic architecture due to the simplicity of the fundamental cells (Figs. 4.2(a, c)), which arises due to the choice of data representation. Note that Lipton and Lopresti architecture requires a linear systolic array of 2N + 1 element for a string of length N.

One important observation is that worst case scenario for comparison, i.e. complete mismatch of strings, is not representative of the typical needs of applications. More specifically, the typical requirement is to determine if string similarity is above a certain threshold. For example, due to the large volumes of DNA data availability, there is a need to know if sequences are genuinely aligned (share a common ancestral sequence or function) or are aligned by chance [9, 30]. Statistically, it is known that the probability of small similarity regions in strings is fairly high and goes down exponentially as the length of the similarity goes up. Therefore, in such applications a similarity threshold is defined below which strings are assumed to be similar by chance and not due to genuine alignment.

This means that in our OR-type race implementation, a smaller score can be attributed to a higher level of similarity and a threshold score can be decided, beyond which the architecture will not look for similarity i.e. if the count exceeds the threshold value, the architecture will treat it as if the required match was not found and move on to the next pattern. This feature is very useful as the maximum possible score is known at each instant in time, and not only at the end

of the computation. This also means that with increasing dynamic range, the best case scenario becomes more representative of a typical situation and the latency does not necessarily scale with dynamic range N_{DR}. For Lipton and Lopresti architecture, however, the entire computation has to complete, before which the maximum score can be ascertained.

Energy consumption is where the idea of Race Logic really shines through. Due to the fact that computation occurs only along the wavefront, the race logic structure can be gated effectively to save on energy. The systolic array on the other hand is linear and hence needs to be clocked every cycle. Hence, for small value of the string comparisons, Race Logic outperforms the systolic array for both the best and worst case scenarios (Figs. 4.9). In general, architectures that focus on energy savings do so by reducing the latency (in other words trading off energy for time). Such architectures are not of much use when it comes to high performance architectures [16]. In our case it is the novel data representation that allows this architecture to maintain a fast operating speed as well as be energy efficient. A related result is that energy-delay product (Fig. 4.11) and power density (Fig.4.10) are much smaller for Race Logic. The latter is also far away from maximum value of 200 W/cm2 as defined by International Technology Roadmap for Semiconductors [50].

Even despite unfavorable area scaling law, the throughput per area of best case scenario of Race Logic is considerably better than that of the systolic array for $N < 70$ as can be seen in Figure 4.6. It is also worth mentioning that the Race Logic architecture was first implemented on an FPGA due to ease of use and re-configurability but the results were completely unexpected. The latency and energy numbers were very large, and upon further inspection, we realized that due to the large capacitance of the global wiring, the interconnect delay was more than the gate delay. Similarly, the energy numbers were larger than expected due to unnecessary charging the discharging of global capacitances. We concluded that the Race Logic architecture was not suitable for mapping on a general purpose FPGA and this is why we proceeded to perform a standard cell based custom design flow. We believe that a full custom design flow with tweaks to clocked capacitance values as well as appropriate sizing of the devices would yield further improvements in results.

Finally, the most optimal implementation of Race Logic is asynchronous and in the analog domain. In each of the figures 4.9, 4.10, 4.11, black curve shows the clock-less estimate that light the goal of 2D energy scaling. It theory, the clock-less estimate is representative of the case in which each race logic cell is responsible for generating its own timing, and there is no need for a timing distribution centre. We can envision such an outcome with the two following solutions. In an attempt

to eliminate the clock, we could use asynchronous delay elements such as current starved inverters and distribute bias voltages to keep power dissipation low. Another solution involving resistive switching devices [44] uses them to implement configurable edge weights, which would provide increased advantages in area and energy efficiency.

Chapter 5

Asynchronous Race Logic

The answer to the question, "Can races be used to perform computation?" is a resounding "Yes". In the previous chapter we saw how clocked, synchronous race conditions were set up in a fabric based on the 2 input patterns. The input pattern matching problem was mapped to a shortest path in a graph problem, by spatially laying out delays based on match/mismatch conditions and a rising edge sets up a race that navigated this delay maze. Moreover, the race implementation outperformed a highly optimized systolic array approach in latency and energy, after clock gating strategies were used to reduce energy consumption. Though certain advantages of race based computation were enumerated, the application that was used to demonstrate these advantages, was a very specific one. With a

more general application will come a larger score matrix and symbol set which will have a direct impact on area, and energy scaling.

To ensure Race Logic has a wider applicability, we need to first address certain issues that are directly related to the size of the problem.

- Area scaling: As Race Logic is a direct representation of the graph, it follows graph scaling laws. In most dynamic programming based optimization problems, the whole search space needs to be searched causing the graph size to become very large. Moreover, as score matrices become more complex with increases in dynamic range, This requires careful cell design to reduce area of the cell itself to ensure scaling doesn't make the architecture prohibitively expensive.

- Energy scaling: The energy scaling in synchronous implementation is essentially third order. This is fundamentally as a result of the clock inputs of the 2D array. Since these clock inputs trigger every cycle, they contribute to N^2 capacitive charges and discharges every cycle for $\approx N$ cycles, leading to a cubic energy scaling factor. Though the problem itself is quadratic, it is evident that the square law energy scaling cant be overcome, but as shown before, clock gating strategies can be used to reduce this third order energy scaling and push it closer to the second order limit. Moreover, as

dynamic range begins to increase, the number of flip flops in a cell would go up causing energy to become a bigger issue.

In Race Logic, we can see that solution to the problem lies within the mesh of delays and a rising edge accesses that particular delay path which is of interest to us. In the synchronous implementation, the timing information (delay value) is not intrinsic to the delay element, though it depends on it (through set up and hold time limits for combinational logic). It is generated by the clock signal and distributed to different flip flops via clock distribution network. It is the dynamic power dissipated by this network that contributes a significant portion to the overall power dissipation of the synchronous Race Logic structure. In this chapter we propose an asynchronous version of the previously described Race implementation, which replaces D flip-flop based delay elements with current starved delay elements as a solution to the aforementioned issues. This will serve multiple purposes. Firstly, currents starved inverters have a much smaller footprint than D flip flop based delay elements. Moreover, timing distribution methods can be developed that essentially consume very little static/dynamic power. Also, we utilize a more modern process compared to the synchronous case and perform the case study again for a more general range of score matrices.

One advantage of a clock network is that it provides quantization of timing intervals. Also due to the regular structure of Race Logic, skew and jitter can

be minimized by spending more power on the clock network. Even in the face of process variations, we can be sure of the functional correctness of the system. This is not as trivial in the asynchronous case. Since the delay elements are bias dependent, process variations and other environmental noise sources will have a larger impact on the delay value and hence may lead to inaccurate results. In this chapter, will also perform a system level functional accuracy study to determine the level of accuracy necessitated by the system and the energy efficiency tradeoff that results from it.

To summarize our contributions,

- We design, for the first time, a full custom, synthesized, asynchronous Race Logic array in 0.18 m technology that can report similarity between 5 to 50-symbol long sequences and simulate its performance against the well-studied DNA sequence alignment problem in the interest of understanding its scaling behaviour.

- We quantify the eect of process variations both from a device and system level standpoint and show that at least for the considered application process variations have minimal impact on the performance and functionality of the asynchronous Race Logic.

- We show that the asynchronous Race Logic architecture is up to 10X more energy efficient, and 4X denser compared to previously reported synchronous and state of the art systolic implementations, while also being faster.

5.1 Delay element and Current source

One of the goals of this work is to design Race Logic in which timing of delay elements is not determined by a clock but by its own intrinsic delay. Ideally, such a delay element should be controllable to account for an order of magnitude of dynamic range, required for more general purpose graph traversal applications, as well as be tolerant to supply, process and mismatch variations. Different schools of programmable delay elements such as [51, 3, 27, 28, 37] were investigated. Capacitive control options like the ones discussed in [51, 3] do not seem promising due to square law scaling with respect to dynamic range of the delay, which leads to prohibitively large area costs. [51] proposes the use of the neuron MOS mechanism, by using capacitive coupling based charge sharing to control the gate charge of an inverter and hence current starving it. [3] proposes multiple sized shunt mos-capacitors to be charged by an inverter. These methods all include large cell sizes as the values of the largest capacitor increase exponentially with each new bit added to the dynamic range. Moreover, the digitally controlled network shown in [3] has monotonicity issues that have been resolved with further

area expenditure. Problems of charge sharing in complex pull-down networks as shown in [37] are addressed in [28] but suffers from a lot of charge injection noise. We found that a cascode current source, controlled by a variable resistor is best suited for our application [31].

To elaborate more on the choice of delay element let us consider figure 5.1 The standard way of constructing a delay element is by charging a capacitor with a controllable current and using a thresholding element to detect a voltage crossing. This is generally implemented using an inverter and controlling the current that is charging its output capacitance and is known as a current starved inverter. Transistors M_N and M_P behave like digital switches while the transistor M_{CS} is the current control transistor that discharges the output capacitance at a fixed rate, hence giving the required constant delay. The design decision at this point is the placement of the current control transistor. In figure 5.1a the current control transistor is placed in the discharge path with its drain connected to the source of the NMOS switch, while in figure 5.1b the current control transistor splits the output node of the inverter. Though the two topologies seem similar, transient simulations reveal that charge sharing due to switching of M_N causes considerable differences in both timing and variability. In the former case, the rising edge switches transistor MN into the linear region which causes charge sharing between C_{out} and C_{int} causing the output voltage to drop instantaneously

(not controllably) to $\frac{V_{DD}C_{out}}{C_{int}+C_{out}}$. This voltage is then discharged by the current control transistor M_{CS}.

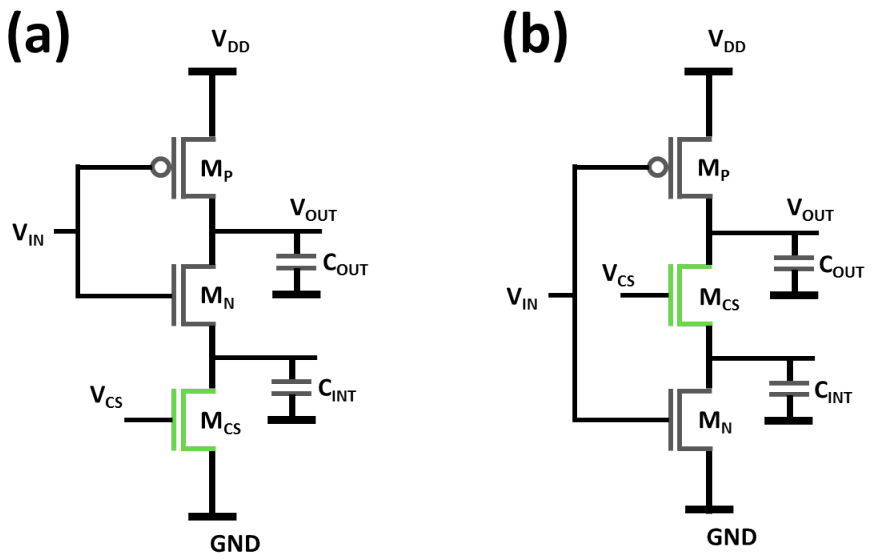


Figure 5.1: Delay element choices are based on the location of the current control transistor. (a) shows the current control at the tail of the switching transistor M_N whereas in panel (b) the current control splits the output node

In the latter case, the rising edge causes C_{int} to be discharged through M_N quickly, while the output voltage stays at V_{DD}. This is then followed by a controlled discharge of the output node through M_{CS}. Not only does the latter method produce longer and more controllable timing delays for the same bias, but is less susceptible to mismatch variations as well. This is due to the fact that output voltage starts discharging from V_{DD} in the latter case vs $\frac{V_{DD}C_{out}}{C_{int}+C_{out}}$ in the former. The capacitances C_{int} and C_{out} are subject to mismatch variations

and hence add uncertainty to the delay of the element. It is worth noting that figure 5.1 shows only a single bias node for ease of explanation. In reality, the current control is performed by both bias and cascode nodes with optimal biasing to ensure minimal current variation with V_{out} during the discharge phase. The digitally switching transistors, MN and MP are minimum sized, while the bias and cascode current control transistors are sized to match the current mirrors in the current source.

The precision of the delay element, barring its own process and mismatch variation, depends upon the precision of the current that is discharging the output capacitance. Hence the current source has a very important role to play. It should be tolerant of process variations and should provide a constant current independent of any power supply variations that may occur due to injection from digital switching activity. Moreover, the current source should be a variable one, preferably controlled by an external source, such that variety of timing delays can be implemented. For the specific case of comparison against work in [26], the dynamic range of the delay does not need to be very large, but for implementation with real data (e.g. DNA nucleotide sequences from NCBI [33]) the dynamic range needs to vary by about an order of magnitude, hence requiring the current source to produce low variability currents over such ranges. In this regard an op-amp based current source was designed that pins a fixed voltage across an off-chip variable

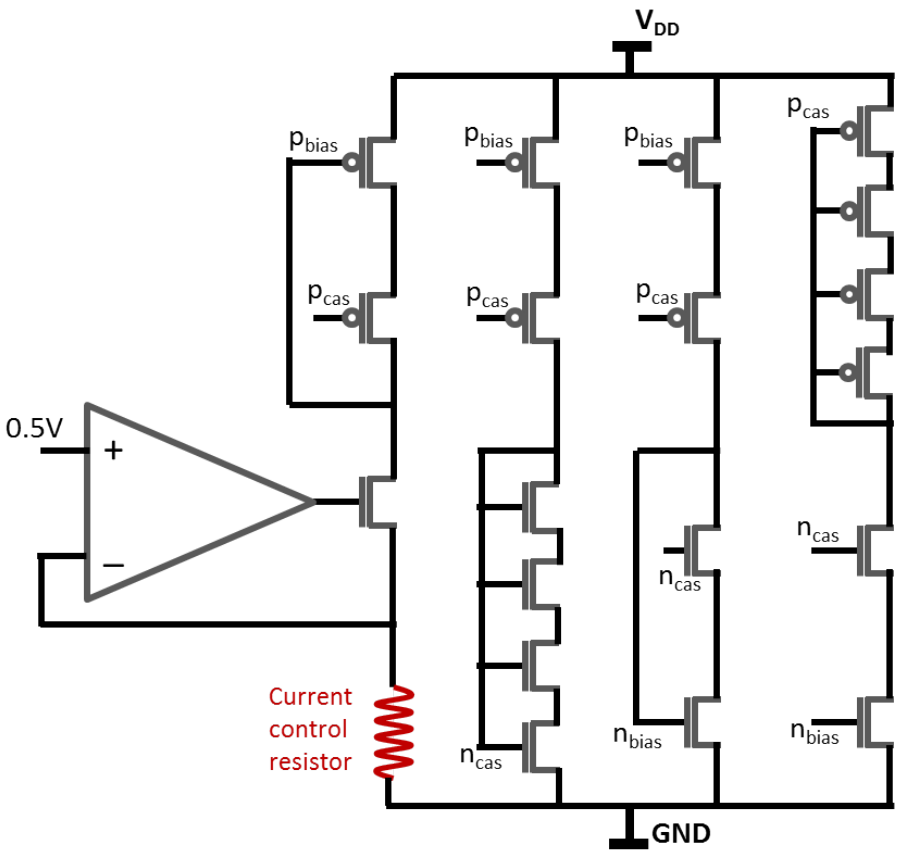


Figure 5.2: Current source design: The current source works by pinning 500mV across a resistor that is variable and externally controllable, using a high gain op-amp. The current that is generated is largely independent of process or supply variations. This current is then mirrored to generate bias and cascode voltages to be distributed to the rest of the array.

resistor, hence utilizing the resistor to act as the current control element. This makes the design largely independent of process and supply variations. The bias voltage thus generated is then redistributed across all the required delay elements. Depending upon the number of independent delays needed (which is three for the considered case), replicas of the current source are made with different resistances,

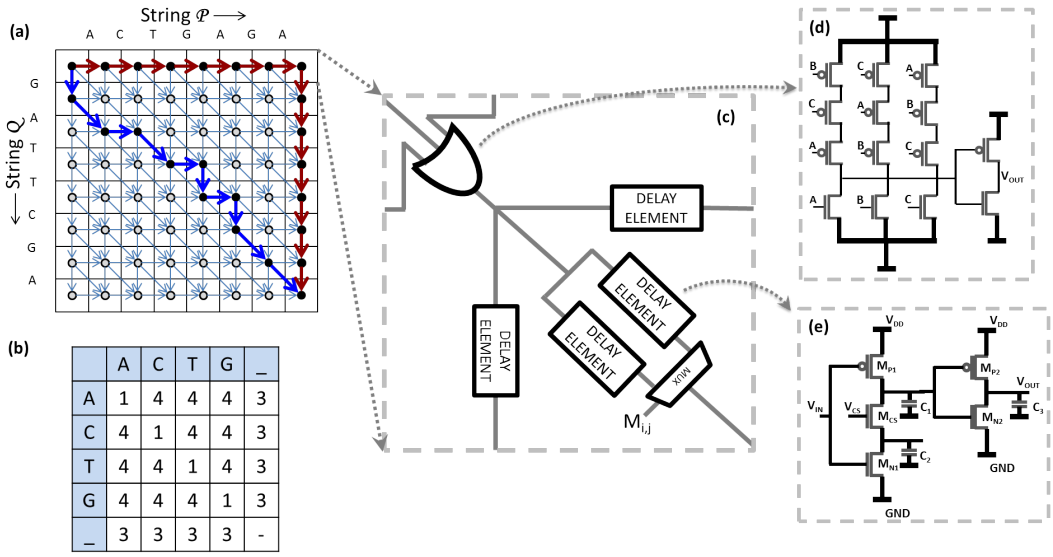
each of them variable, to tune to the required current. This design is allows area savings compared to delay element designs in [28] where each cell consists of a controllable elements that take up considerable area. Another important aspect of the current source design was to size the bias node transistors relatively large, such that they would behave as low impedance nodes and be more tolerant to charge injection from switching activity from nearby digital nodes.

5.2 Top level Architecture

Similar to the synchronous design, in our attempt to design asynchronous Race Logic circuit for DNA sequence alignment problem we make the most of its repetitive nature and partition its implementation into unit cells. The unit cell for an OR-type Race Logic is shown in figure 5.3c. There are three inputs - top, left and diagonal, which receive a rising edge from the preceding and adjacent cells, and the first arriving input is detected by an OR gate. The symmetric gate design shown in Fig. 3b was used to aim for equal delay on all input to output paths. This is as a result of the asymmetry in the design of a classical OR gate which favours the input closest to the output. Since the asynchronous design cant afford the luxury of a clock to swallow such errors, there is possibility for different paths to select different OR gate inputs and cause errors to add up with increasing problem size. The first arriving signal then passes through three delay

elements, the top and bottom being always a constant delay for the considered application (representative of a insertion/deletion) and the diagonal one, selected through the multiplexer, being dependent on the match mismatch criterion. The multiplexer is a pass-gate based analog multiplexer and such pass gates are also added in series with the delay elements on the off diagonal to ensure similar delays on all paths. The values of delay elements are set according to the particular score matrix. Though having fixed delays might be acceptable in some cases , in this paper we consider a more general case when the delay value can be programmed, thus allowing asynchronous Race Logic to solving a broader variety of alignment problems.

The issue of charge injection into the bias nodes is a serious one and can cause large systematic errors in the delay of the circuit if not resolved. The V_{cs} node of the delay elements (Fig. 5.3e) are shared by the entire array and are hence subject to charge injection from the switching activity in the bottom NMOS transistor. Though the entire array does not switch repeatedly and simultaneously (which would cause a large simultaneous charge injection event), there is switching activity which is staggered by the delay elements themselves. This is fundamentally the same problem as in the synchronous case. The source of timing information instead of being a clock is now the a current. This information (the value of the current) has to be distributed to all delay elements throughout the array. In the



	A	C	T	G	_
A	1	4	4	4	3
C	4	1	4	4	3
T	4	4	1	4	3
G	4	4	4	1	3
_	3	3	3	3	-

Figure 5.3: Top Level Architecture: Panel (a) shows the Edit Graph used in the previous chapter and (b) shows a more commonly used score matrix in DNA sequence alignment. Panel (c) shows the unit cell of the asynchronous Race Logic architecture. It is similar to the synchronous approach in many ways, but differs in the design of the delay elements and the OR gate as shown in panels (e) and (d) respectively.

lowest power scheme, the bias voltages generated from the current source shown in figure 5.2 can just be routed to the V_{CS} nodes in figure 5.3(e). The drawback of this is that the global bias nodes thus created are going to be susceptible to various kinds of noise, such as switching noise, substrate coupling etc. One solution to overcome this problem that minimally power hungry is to use the area above the entire array by high-quality MIM caps ($4fF/m^2$) and bypass the bias nodes. The large value of the capacitance significantly reduces variation in voltages from the injected charge. Another, albeit more power hungry (local) solution is to partition the array into blocks and regenerate the bias voltages locally for

each of these blocks while capping them with the aforementioned MIM caps. This significantly reduces the load on the primary bias network and decouples large sections of switching activity from each other and hence provides a more precise delay values. It can be seen upon inspection that this is very similar to the clock gating idea presented in chapter 4. The system is partitioned into blocks, but instead of removing useless energy from the system, energy is being supplied to those partitions to reduce variability and provide a better noise tolerance. The clock in the synchronous architecture has a similar function, i.e. is improving the noise tolerance of the system by quantizing timing intervals at the expense of energy. As the clock contributed to cubic scaling of energy and after gating, shifted to between quadratic and cubic, in this case, we start at quadratic with just a single current source and all the energy being spent in switching and in our effort to improve performance we add local biasing circuits that push the energy scaling by increasing the quadratic constants.

5.3 Results and Discussion

We have designed asynchronous Race Logic implementation for solving DNA sequence alignment problem for variable string lengths from N = 5 to N = 50. Data points were fitted to analytical equations for all desired metrics shown in Fig. 5. All the simulations were done in Cadence 6.1.0 using the Silterra 0.18 m

process. Though the implemented delay elements are programmable with roughly 10X dynamic range, in the simulations we used specific score matrix in Fig. 2b.

5.3.1 Simulation Results

For the purpose of comparison, similar to synchronous Race Logic studies [26], we focus on two possible alignments resulting in the best and the worst case scores, which represent a perfect match and a complete mismatch, respectively. (The complete mismatch case results in 2N indels for a string length of N and rather unlikely in practical applications - see discussion in the next subsection.) Moreover, Dennard constant-field scaling laws were applied to make sure that the work in [26], which was performed in 0.5 micron, was scaled down to effectively be in the same process. In addition, previously reported performance results for the synchronous Race Logic were adjusted to account for larger dynamic ranges of the delays in the asynchronous Race Logic. The Dynamic range for each delay element is show in figure 5.4 as a variation of the delay vs the value of the resistor.

In particular, the following changes were made to the synchronous architecture. For the dynamic range to be increased from a factor of two in the synchronous design, to an order of magnitude, the number of D flip-flops has to be increased from 1 to 4 (assuming logarithmic encoding), which in-turn adds extra gates for score selection and latching. The latency is also affected as a result of addition of

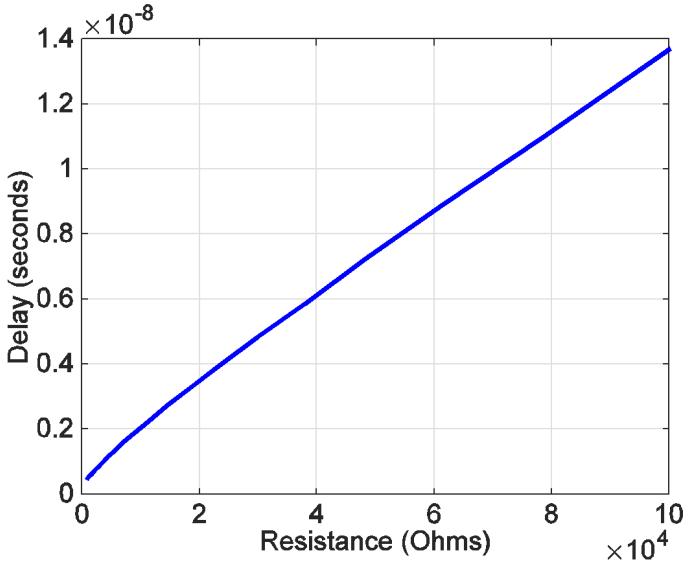


Figure 5.4: Variation of Delay with Resistance.

extra logic levels in the critical path between two D flip-flops, which manifests in the slowing down of the clock.

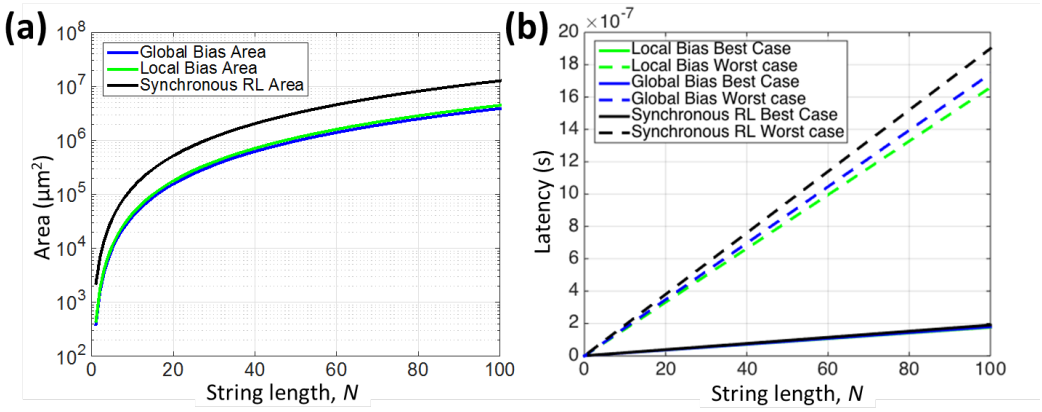


Figure 5.5: Simulation Results: Panel (a) shows the area scaling of the Synchronous Race Logic implementation and the Local Bias and Global Bias cases. (b) shows the latency of the best and worst case delay for the same.

Another repercussion of each unit cell housing 4 DFFs is in the power and energy, because the clocked capacitance dominates the energy scaling of the synchronous architecture. The energy numbers that are reported for the synchronous design are after clock gating strategies, that attempt to reduce the third order energy scaling that exists as a result of continuous clocking of the entire Race Logic fabric. Nonetheless, the cubic scaling can be reduced but not completely eliminated. Figures 5.5,5.6,5.7 show various performance metrics for asynchronous Race Logic and compare it with previously reported adjusted results for synchronous Race Logic. In general, even with pessimistic assumptions regarding the increase in area and energy of the synchronous design, asynchronous Race Logic performs better in almost all metrics (and as a result significantly outperforms highly-optimized conventional implementation) except for power density 5.6, which is due to larger area of the synchronous architecture. The global biasing implementation is much more energy efficient and area efficient compared to local bias one. This is because in the global bias case, the distribution of timing information happens in terms of bias voltages which are gate-connected and hence consume negligible power, while the MIM caps supply any instantaneous charge required.

In the local bias case, once the global bias voltages are distributed, local biases are regenerated for 5X5 array regions, which contribute to the cubic scaling of

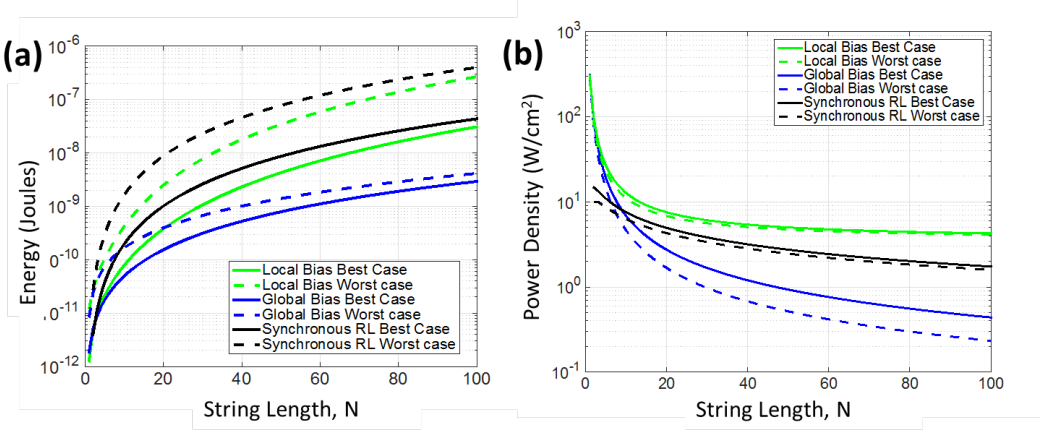


Figure 5.6: Simulation Results: Panel (a) shows the energy scaling of the Synchronous Race Logic implementation and the Local Bias and Global Bias asynchronous cases. (b) shows the power density for the same.

energy. This behavior is visible in the figure 5.6 as the global bias case has square law energy scaling and is considerably lower than the scaled synchronous as well as the local bias case which have cubic behavior.

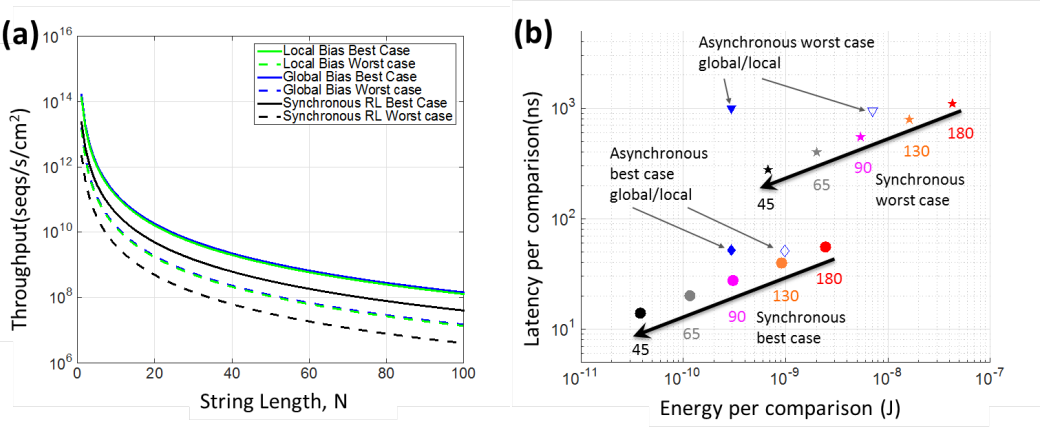


Figure 5.7: Simulation Results: Panel (a) shows the throughput of the Synchronous Race Logic implementation and the Local Bias and Global Bias asynchronous cases. (b) shows the power density for the same.

One possible concern is that performance and energy efficiency advantages may be lost when implementing circuits with more aggressive and more practical CMOS process nodes, in particular because asynchronous Race Logic might be more sensitive to process variations due to its inherently analog design, compared to a purely digital one. To address this concern we first note that the relative area of the circuitry, that is sensitive to mismatch variations in asynchronous Race Logic, is approx. 60%, and therefore such circuitry can be aggressively scaled without increasing variations. To further compare scaling behaviour, figure 5.7b shows the estimated performance for scaling of the synchronous design relative to the 180 nm asynchronous design in this work. The latency estimates assume Dennard scaling, while the energy estimates also account for leakage power at low technology nodes. Though latencies of scaled synchronous versions always outperform the asynchronous version, the energy performance of the 180 nm asynchronous version is comparable to the energy performance between 90 nm and 130 nm for the best case and between 45 nm and 90 nm for the worst case. Moreover, in the next subsection we show that for the considered application variations in delay can be effectively tolerated at the functional level without much penalty in performance or energy efficiency.

5.3.2 Variation Analysis

To understand how the Race Logic implementation of the edit graph is affected by the delay variations in each of its elements, and how it affects the functional correctness of the overall architecture, a MatLAB model of the circuit architecture was designed into which variations were purposefully introduced. To make sure that our assumptions were realistic, we took real data from the chromosome 1 of the Human Genome and simulated the process of shotgun sequencing followed by realignment 5.8, which is a typical procedure in the de-novo sequencing [10]. In shotgun DNA sequencing, a section of the DNA sequence is split in into thousands of strands of equal length at random locations. Since there are so many strands that are taken, there is a chance of some of them overlapping which is what we are looking for. Strands with the maximum overlap (above a certain statistical significance) can be thought of as belonging to a contig (contiguous region of DNA) and can hence be used in reconstruction of DNA sequence. Figure 5.8c shows an example of score histogram when comparing a particular DNA string with others in a given section with an overall number of nucleotides being 20 times the number of nucleotides in the section (coverage of 20). We then use Monte Carlo technique to simulate signal propagation timing (i.e. score) in the Race Logic fabric for two pairs of DNA strings (Fig. 5.8b). In each run of Monte Carlo simulation, delay elements were initialized by adding normally distributed random variable with

specific standard deviation to the exact delay value determined from the score matrix.

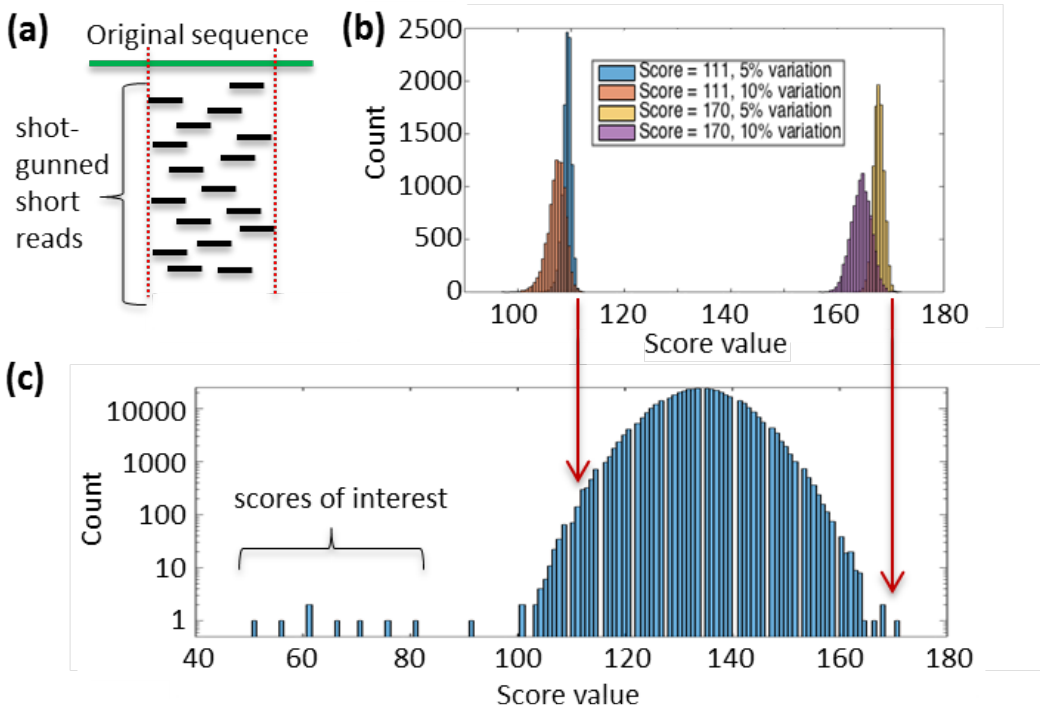


Figure 5.8: Preliminary variation study: (a) A cartoon of shotgun sequencing process, (b) Monte Carlo simulations (10,000 runs) of the alignment scores for two particular pairs of DNA strings for the OR-type Race Logic with variation-prone delay elements, and (c) representative score statistics in human genome shotgun sequencing. The reference DNA strings is the same in both panels (b) and (c). The query strings used in panel (b) were chosen such that their alignment scores with the reference DNA string correspond to the start and very end of the hump in the score distribution, which is highlighted with red arrows. (The simulation results show that the distributions on both panels are rather insensitive to the particular choice of DNA strings.

The chosen values of standard deviations are crudely representative of pessimistic and optimistic scenarios for process and mismatch variations in the simulated 180 nm process. An interesting detail is that, though variations for each

delay element are symmetric around mean value, the total score is almost always lower compared to the exact value. This is an artifact of OR-type Race Logic which favors fast signals over the slow ones. The detailed analysis of the results of Fig. 6 suggests practical and efficient solution for tolerating variations and noise in asynchronous Race Logic. Indeed, it is known that the probability of small similarity regions in DNA strings is fairly high and goes down exponentially as the length of the similarity goes up [30]. Therefore, the similarity threshold can be defined below which the strings would be assumed similar by chance and not due to genuine alignment. Practically, that means that for OR-type Race Logic we could define a certain threshold beyond which the architecture will not look for similarity and move on to the next string. As Fig. 6c shows there are only few strings, e.g. with scores below 100, which should be considered for alignment, while the vast majority of strings would be discarded. Interestingly, this means that with an increasing dynamic range, the best-case (rather than worst case) paths become more representative of a typical situation. When looking for alignments with small scores in such score distribution, variations and noise can be resolved efficiently by slightly increasing the threshold. For example, for the 10%-variation case all strings of interest with a score value below 90 can be detected by setting the threshold to ≤ 92 and the penalty of such adjustment would be identifying few false-negative strings whose score is in between 90 and 100.

From Fig. 6, it is clear that the increase in runtime due to threshold adjustment and additional work of finding false negative would present just a minor overhead considering that most of the time will be spend screening the strings with high (≤ 100) scores.

5.4 Conclusion

Our objective was to design an asynchronous Race Logic array and in particular address some issues pertinent to synchronous implementation such as third order energy scaling and difficulty with handling of a more complex score matrix. To accomplish this goal we propose an asynchronous architecture in which the delay is controlled by current starved inverter based delay element. This design allows for a larger dynamic range at lower area and energy costs. As a specific example, we implemented the well-studied shotgun DNA sequence alignment problem and compared synchronous versus asynchronous design styles. The simulation results for the synthesized design in 0.18 m technology show that, asynchronous implementation is at best 10X more energy efficient, 4X denser and has slightly smaller delays as compared to synchronous one, which by itself significantly outperformed conventional highly-optimized systolic array implementation for sequence alignment problem. Moreover, we study the effects of process variations at both a device and system level and show that for at least the considered application, the

process and mismatch variations have limited impact on the asynchronous Race Logic performance and functionality.

Chapter 6

Chip Design and Testing

To test the potential of Race Logic, we designed a chip that would utilize the asynchronous race conditions discussed in the previous chapter to perform DNA sequence alignment on real DNA sequences. This chapter will go into the details of the chip architecture and the circuit, layout and floor-planning level decisions that went into designing it.

6.1 Objectives

Since this design is a first prototype of Race Logic, flexibility in testing of the array was a major concern. This section details some of the constraints that govern the design process. Our major objectives are to,

- Design a functional prototype of the Race Logic Asynchronous architecture and test functional accuracy and correctness for real DNA sequence data with score matrices from real biological test data.

- Test the architecture for performance and energy numbers.

To be able to achieve the above, a few points need to be kept in mind,

- The size of the array should be representative of real world NGS sequencing data. Common sizes are between 25 and 100 base pairs(bp), with some ranging as high as 200bp.

- From the previous chapter, we see that the fastest delay is of the order of a nanosecond with the slowest delay being approx one order of magnitude larger at 15 nanoseconds. This delay is tunable with an external resistor. Our unit delay here is going to be the smallest delay that can be achieved by such a system. This constrains our problem in the following ways,

  - We need a system that can measure such delays with relative accuracy.

  - Measurement system should be on chip

  - Since the value of delay itself is variable the measurement system should also be adjustable to measure a range of delays.

- Since power numbers also need to be measured, some specific considerations have to be kept in mind.
  - We need a system that measures power of the array itself versus the power of surrounding circuitry
  - Also measuring the switching power of the Race Logic array itself separately from the bias power will also yield interesting results.
  - If smallest delay is of the order of magnitude of one nanosecond, and the array is of size 50X50, the fastest race would be of the order of magnitude of 50 nanoseconds. The change in current drawn by the array will last only for that long and needs to be measured accurately.
- Characterization of current source to compare against simulation models.

The next section introduces the details of the architecture and test plans that arise from the aforementioned constraints.

6.2 Architecture and Test Plan

This chip was designed in a 0.18 um Siletrra process with 2mm X 2mm of available area. The process consisted of 6 metal layers and provided high density MIM (Metal Insulator Metal) capacitors with a capacitance of $4fF/um^2$. Fan out

of four (FO4) delay of a minimum size inverter is $\approx$ 70ps. Due to area constraints we chose the size of the array to be 50x50.

To have a variable delay measurement system, we decided to go for a variable delay clock generator whose delay could be divided down and measured externally on an oscilloscope. This method can serve multiple functions; firstly, a variable delay clock that can be characterized externally will provide a relatively accurate measurement for internal chip events that happen at a much smaller timescale. Moreover, since the array inputs are programmable, the fastest delay path through the array, which is the diagonal one, can be isolated by choosing the same input patterns P and Q. This shortest diagonal path, also known as the best case in the previous chapters, consists of 50 delay elements in series with each other. Since the architecture requires a counter to start with the signal entering the top left corner of the array and stop with it coming out of the bottom right corner, we don't need to develop an isolated test structure to characterize the delay elements. If the bias is adjusted on the current source controlling the diagonal path and on the clock generator such that the value of the counted signal is exactly 50, we can be sure that our clock has been set to the unit delay of the Race Logic Array. We can also use other score matrix configurations to isolate other specific paths and characterize the major delay elements, diagonal match, diagonal mismatch and off-diagonal respectively.

To be able to measure power, according to our constraints, we decided to split the chip power across 3 major domains. The first domain would cover the switching energy spent in the array. This consists of the energy spent in the actual switching of the race array and not the power spent in bias networks. The second domain would would be connected to all the bias circuitry throughout the array, as well as the main bias generation for amplifiers, current sources etc. The third domain would qualify everything else, meaning the input circuitry, clock generation etc. Also multiple modes of operation of the chip were decided upon. A pattern mode to check for functional correctness, and a burst mode for power measurement. In the burst mode, a single pattern match operation would be performed on a system repeatedly at maximum throughput to get an idea of the worst case power dissipation of the system.

In the design of the system, we partition the architecture in to 3 major regions. Input/Output circuitry, Clock and Control Logic and finally Array and Bias Circuitry. Fig 6.1 shows the major partitions of the architecture and the next few sub-sections talk about these partitions in detail.

6.2.1 Input/Output Circuitry

The I/O system consists of a set of shift registers that are responsible for shifting in DNA sequence data that are to be aligned with each other. To implement

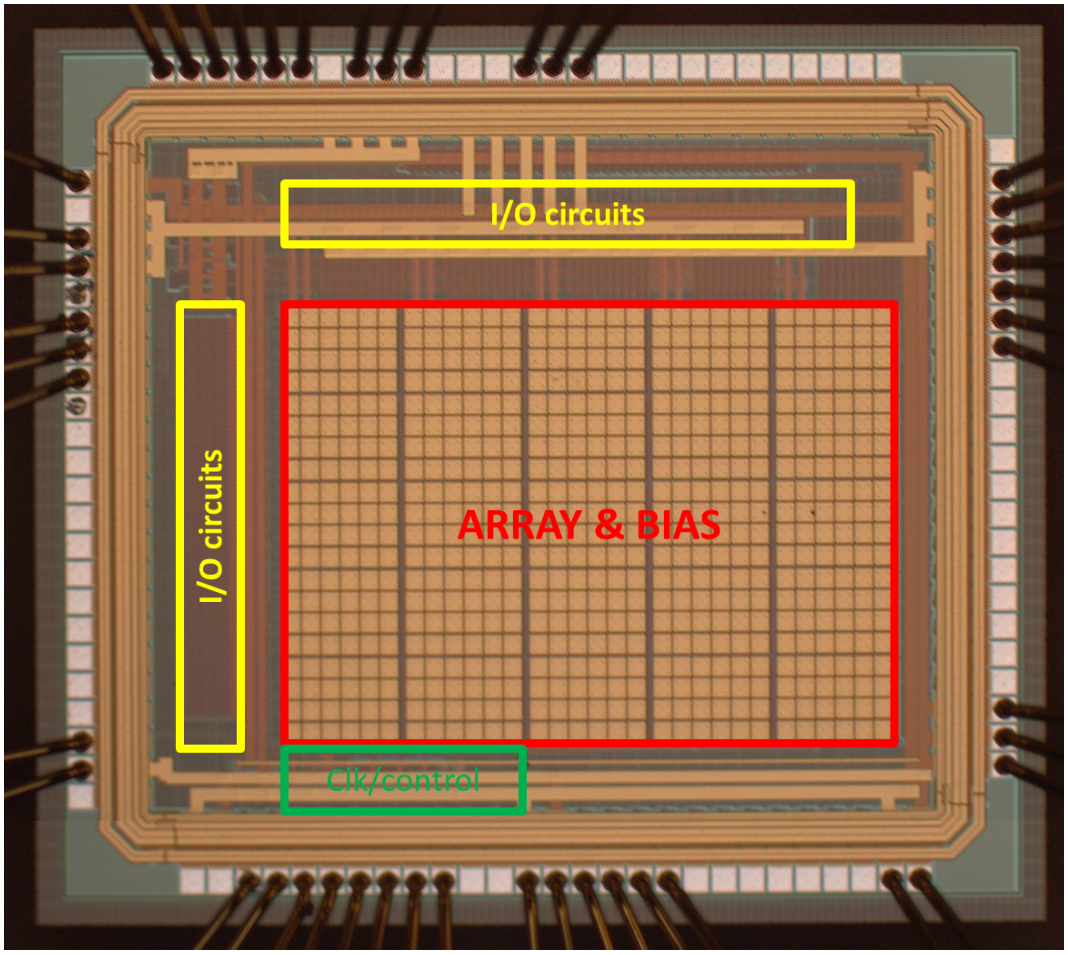


Figure 6.1: shows the micrograph of a fabricated chip with architectural block highlighted

the symbols, a simple binary encoding is used to represent the respective nucleobases with $A =$ "00", $G =$ "01", $C =$ "10" and $D =$ "11". The system can load upto 4 patterns at once and run them in either in patterned mode or burst mode. This is implemented by constructing a single 250 element long 2-bit shift register in which the first 50 elements are the pattern to be compared against, and last 200 consist of the 4 patterns that are being compared. A 4X1 MUX is used to choose

between the patterns, and in the patterned mode, the select inputs are marched up to cycle between the patterns one after another and after the final pattern has been matched the system stops. In the burst mode the cycling continues until the user initiates a stoppage. The idea here is to allow enough time for correct measurement of power through the external system. This shift register in total had 3 inputs and 2 outputs as the data can also be shifted out to ensure that the shift register behaves like a 2 bit scan chain with 250 elements.

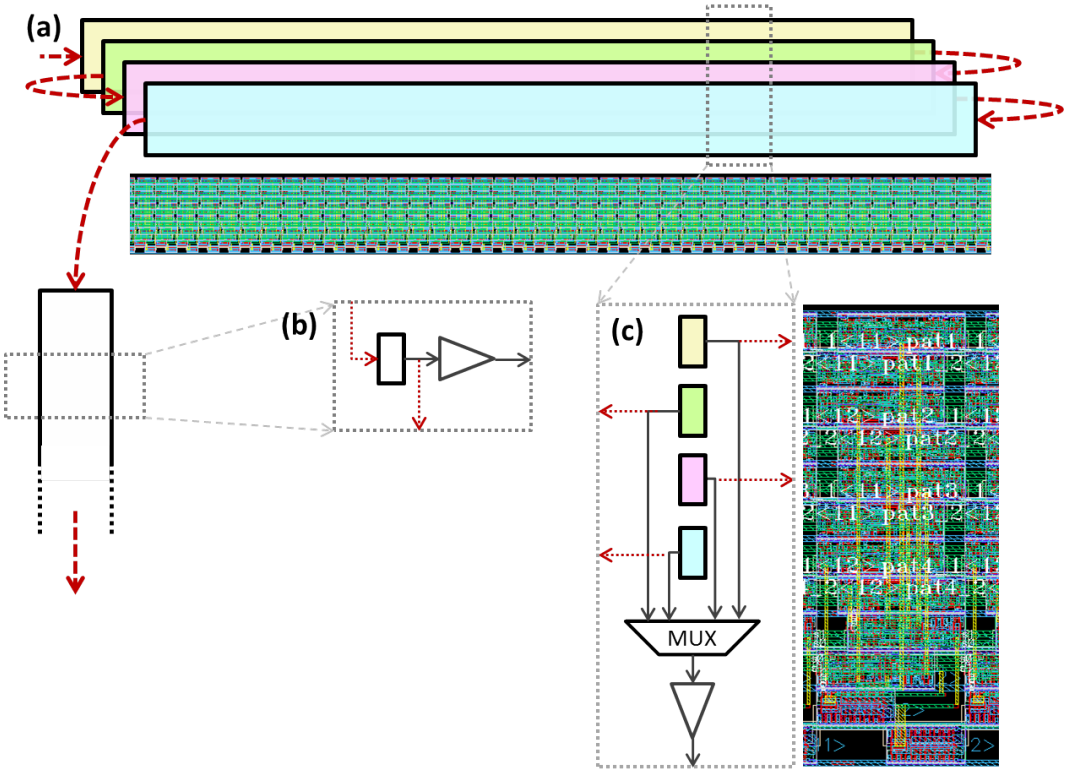


Figure 6.2: Shift in circuit and Scan chain: Panel (a) shows the overall 50X4 shift registers scan chained to store four patterns(P1 to P4) extending into the Q pattern. (b) shows the details of the single Q bit element while (c) shows the details of the P bits and how different 50 long sequences are buffered into the Race Array

The Race logic system outputs a final score, which is the similarity metric between the input strings that have been compared. In this implementation that would be four output scores, as 4 patterns can be matched in one operation. The counter that is used to count the score receives its input from the clock generator block, which has a variable delay clock. We estimate this latency to be varying from 700 ps to 10ns. Also, this counter should be able to count atleast 50 times the dynamic range of the score matrix used. We used a 11 bit counter. Since the 11 bit counter had to work at 700ps, we designed the counter using a prescalar, such that it could handle such high speeds. The idea here is that a smaller size counter is used to scale the input clock(the LSBs of the count) such that the latter MSBs counts can get a longer propagation time for the carry chain. To operate in burst mode, the counter has a output that was parallel loaded to shift registers which were then shifted out with a traditional SPI interface. 2X1 MUXes are used to change the loading from parallel load inputs to shift chaining. The counter/shifter is also a scan chain that can be used the test the structures.

6.2.2 Array and Bias Circuitry

The array consists of repetitions of a fundamental unit cell whose structure is shown in fig figure(a). The cell consists of 4 delay elements and 3 delay control inputs(one for each diagonal match and mismatch and 1 shared between the off

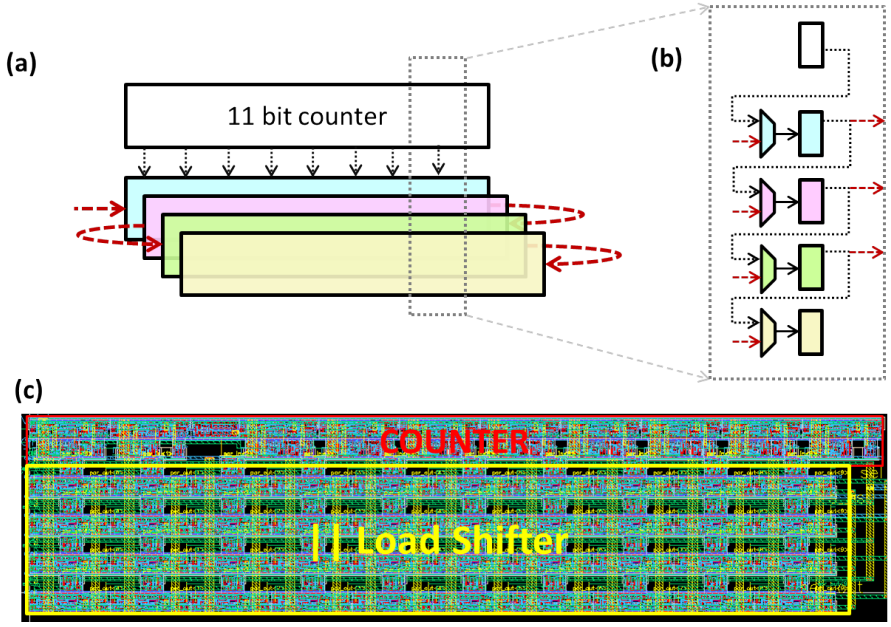


Figure 6.3: Counter circuit and shifter register output: Panel (a) shows the overall 11 bit counter parallel loading in to the first shift register while (b) shows the details of how the counter values are parallel loaded into each shift register for one MUX configuration and to a scan chain shift register for another. (c) Shows the layout of the blocks from (a) and (b)

diagonal ones) shared between them. The delay elements are the same as in the previous chapter with a current starved stage followed by a regular minimum size inverter to sharpen the edge. Since only rising edges go through the structure, the current starved inverter, only uses Nbias and Ncas control nodes. The data values that come from the P and Q arrays on the topological north and west of the array, also have 4 inputs into each cell (A1, A2, B1, B2) which go into XOR based matching circuitry to decide which diagonal element is going to be chosen. The MUX is not a digital one but a pass gate based MUX to make sure that both paths have similar delay characteristics.

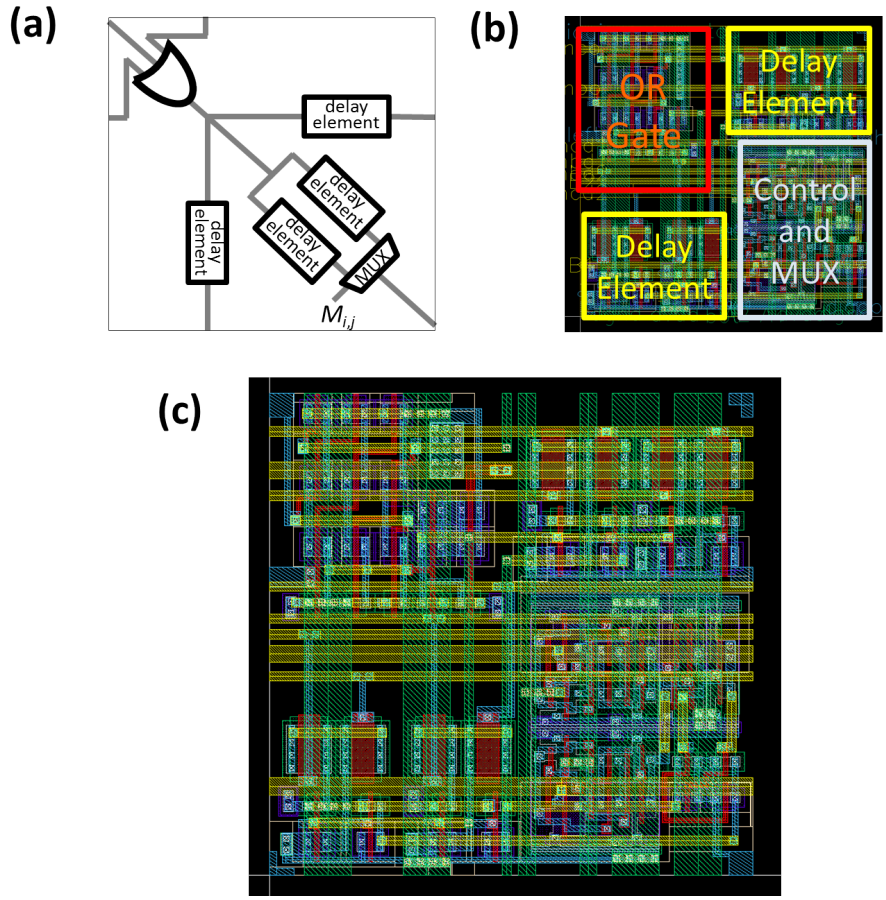


Figure 6.4: Race Logic cell: Panel (a) shows a simplified circuit of the Race Logic cell while (b) shows the layout of the panel (a). Panel (c) shows a detailed version of (b) to accentuate the tiling structure. It can be seen that the corners are designed to lock into each other and allow diagonal connections while bias nodes go across the cell while power runs from top to bottom.

Since the off diagonal paths are not switched between, there is no multiplexer but pass gates were added in series to ensure similar delay across all paths. The OR gate at the input of the cell is designed with a symmetric NOR followed by a NAND to add a Reset to the system. One point to notice is that in an

asynchronous system, there are no flip flops that can be reset, so we added a reset control to the OR gate that suppresses any input and drives a 0 into all the input nodes. The timing on the reset circuitry is adjusted externally with the 3 delay elements in parallel with an AND gate(longest path race) to ensure that all delay elements have been Reset.

The cell in the array was designed and laid out in such a way that they could be tiled and no further routing would be required, after correct placement of one cell surrounded by 8 nearest neighbours. Since this kind of tiled structure is going to be used, the entire routing of the power grid as well as all routing wires are to be performed within the design of the cell. Another concern is that since the high density Dual MIM caps take up metal layers 4, 5 and 6, there are only 3 available layers to perform the rest of the routing, power and reset distribution etc.

In this first prototype, functional correctness and accuracy was of utmost importance. To ensure a reduction in switching noise, the array was partitioned into regions of 10X10 which were then connected to local bias networks that allowed decoupling of switching activity from one part of the array to another. More intricate partitioning can be done to further reduce coupling such as alternating domains, or partitions where nearest neighbours are on different domains. The local bias networks have to be designed for all 3 delay values and sized such that the bias nodes are low impedance nodes to absorb any switching as soon as pos-

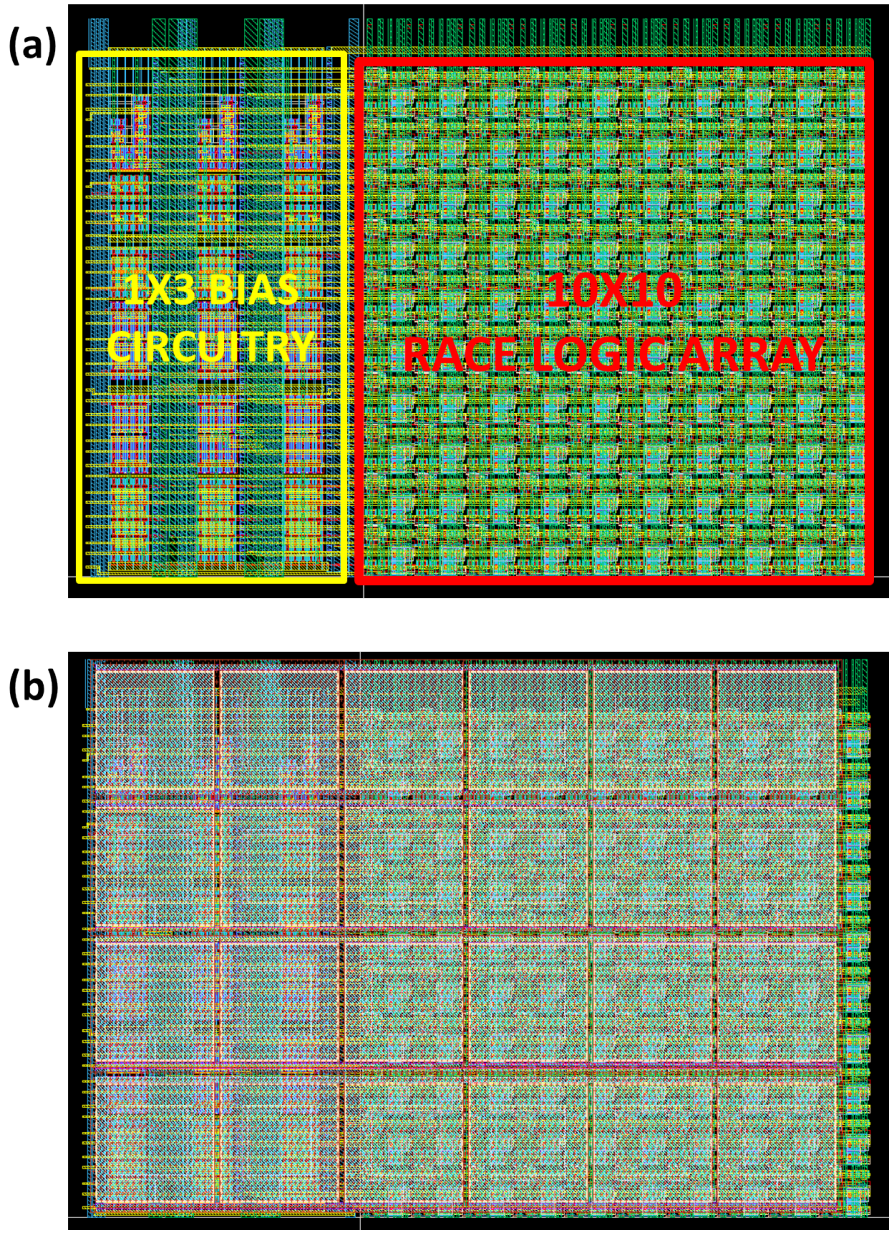


Figure 6.5: Race Logic Array: Panel (a) shows a 10X10 Race Logic array with the relevant local bias circuitry without metal layers 4, 5 and 6. On careful inspection, the tiles nature of this granularity is also visible. (b) shows the same layout with MIM Caps on top of the array in the top 3 metal layers.

sible. As a result the N_{bias} and N_{cas} branches were sized larger while making the P_{bias} and P_{cas} branches at minimum size. Dual MIM caps were used on metal layers 4 5 and 6 above the array to provide decoupling capacitance for all 6 bias nodes. At this granularity, tiling was performed again to ensure that placement of this 10X10 cell complete with power and bias supply, next to similar surrounding cells would automatically route the whole architecture. An important point to notice here is that the bias circuitry and array circuitry are on different power domains to ensure separate current measurement. The entire 50X50 array with its 25, 10X10 partitions are shown in figure 6.6. The array has 200 data inputs (100 on top and 100 on the left) which are used to set up the edit graph, one race input, one race output, and one reset.

To generate all the biases that were used in this chip, the current source explained in Chapter 5 was also designed and laid out. This current source, in order to be tolerant to process and supply variations, uses an off chip resistor to determine its current by using an Op-Amp to pin voltages across the resistor. The Op-Amp is a single stage, P input, folded cascode, single output amp which has a high gain and doesn't need to drive any large loads. The amplifier works in uA range and has its own current source to bias it. This current source uses an on chip resistor for supply independence. The tunable current source inclusive of all the branches, Op-amp and its own current source is shown in Figure 6.7

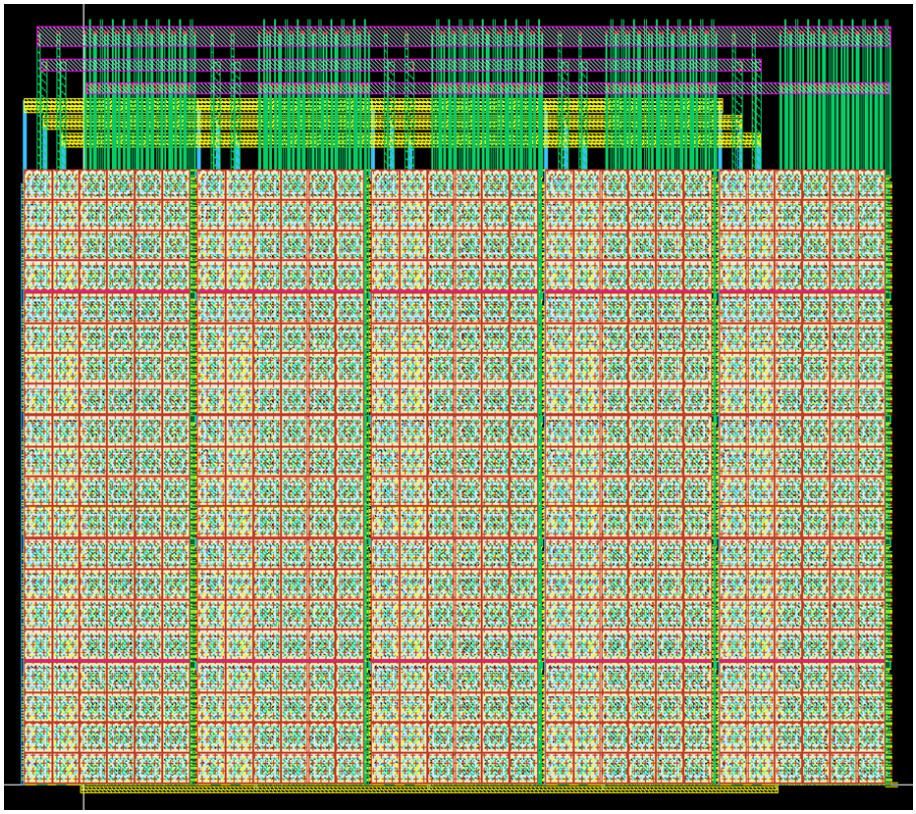


Figure 6.6: Race Logic Array: 50 X 50 RAce logic array complete with bas circuits and MIM capacitors.

6.2.3 Clock and Control Logic

The function of the clock generator block, as discussed before, is to generate an output clock that is going to be used with the counter to count the time period of the critical path in unit delay steps. This clock is also going to be externally calibrated. To design this system, instead of using current starved inverters as a delay line, we decided to directly vary the supply voltage of a 11 stage ring oscillator. Power supply variation allows larger dynamic range is

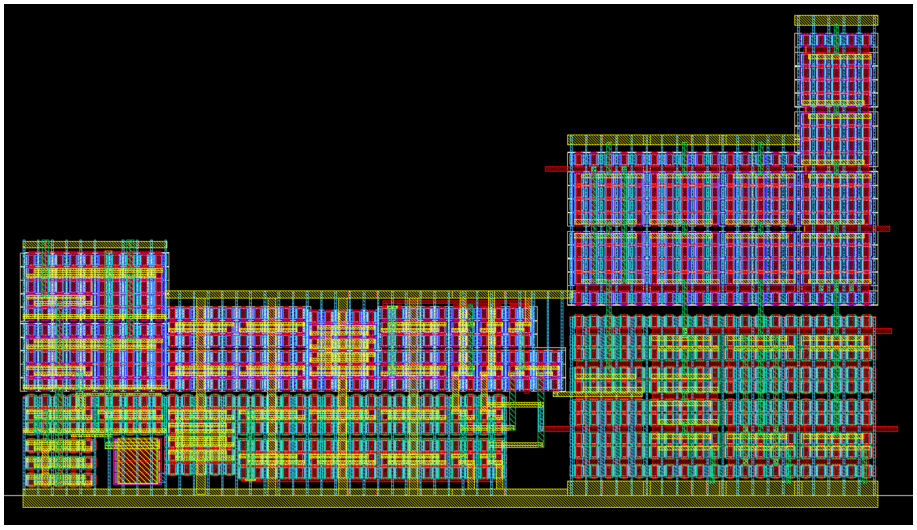


Figure 6.7: Tunable current source for current biasing using off chip resistors.

delays and can hence be used to generate 700 ps to 15 ns time period clocks. An differential amplifier based level shifter was used to scale the clock signal generated as cross-coupled structures didn't provide the speed required at the fastest operating corners. This was as a result of the fact that the time taken in the cross-coupled structure to fight the feedback loop and have the positive feedback kick in was too high. This clock output goes through buffers to the rest of the circuit as well as through a 12 bit counter that divides it down such that it can be sent off chip for measurement.

The function of the control logic block is to take all the elements discussed in this section and create a simple state machine based interface that can be used externally for delay and power measurement. The state machine of such a

control logic block is responsible for interfacing with the array, and counter blocks, starting the race computation (array and counter), detecting the end of the race resetting the array and counter etc.

6.3 Results

To test the functionality of the architecture, we ran similarity measures on real DNA sequences from chromosome 1 of the human genome by partitioning it into 50 symbol long sequences taken at random places with a coverage of 15. These output scores were compared against scores gotten from simulation values and measured for error. We were able to characterize the array and program it with regularly used score matrices such as [1, 2, 1] [1, 6, 4] [1, 4, 3] where the format is described as [match, mismatch, indel].

The results for the score matrix [1, 4, 3] is shown in figure 6.8. The errors are not repeated as the same measurement repeated multiple times does not result in a different value of score. Due to an artifact in the circuit, there is some systematic error causing the score to be 1 or 2 values higher than it is supposed to be, and given that , the average error between the simulated score and demonstrated score is about 2.9%. The best case pattern matching speed is approx 100ns approx, 10 million patterns per second to 2.5 million patterns per second, while a more representative value, would be data depedant and would lie around 5 million

patterns per second. On the other hand state of the art algorithms such as BWA and bowtie, bowtie2, bowtie2GP on a 3 GHz 32 GB server can map 1 million 36bp long short reads in 2140 seconds, 490seconds, 640seconds, 500seconds, respectively, an approximate 3 order of magnitude increase in performance without even considering power numbers.

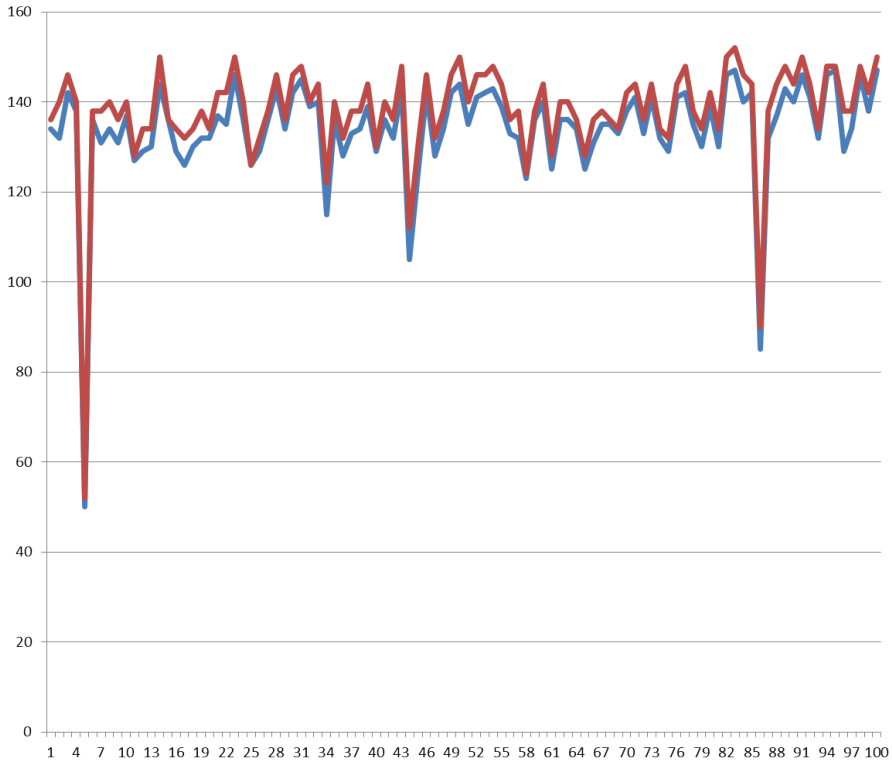


Figure 6.8: Simulation vs Real data with the blue line representing simulated value and red line representing real data. The x axis represents different sequence alignments while the y axis represents the score.

For different score matrices the power numbers were different. For the common [1, 4, 3] score matrix, the static leakage of the array was about 2.2mA while the

dynamic switching current of the array for the best case was 6.1mA while for the worst case it was 4.1mA. When the score matrices are not loaded with any values, the static leakage across the local biases is very low at approx 0.8mA but as the score matrices get large, for example for [1, 4, 3], the static power drawn is about 29mA, whereas for more dense score matrices such as [1, _ , 1] it can be as high as 37.6mA. In an attempt to be functionally accurate, the timing distribution circuit was designed to be large and power hungry and hence the static power dissipation of the local bias circuit takes up a much more significant portion of the power than the dynamic switching current.

Chapter 7

Summary and Conclusion

As we look to application specific designs for improvements in performance and energy efficiency, we often only consider the universe of design options cleanly broken apart into digital systems which encode values as bits on a wire, and analog systems which encode values as continuum of levels on a wire Sharpeshkar in [39] draws on some basic differences between digital and analog circuits by dividing them into 4 regions comprising of combinations of continuous/discrete signals and time. Discrete Signal and Discrete Time(DSDT) consist of our standard Microprocessors and DSPs while Continuous Signal and Continuous Time (CSCT) represent standard analog circuits such as Op-amps, filters etc. Other regions such as DSCT and CSDT are not of much relevance to our work. Since in Race Logic, the signal is encoded in time, discrete time intervals automatically mean discrete

signals which is what has been used in both our synchronous and asynchronous designs. We compare Race Logic implementations to standard Digital and Analog implementations that know. Since Race Logic encodes multiple bits of information on each wire, it shares a lot of properties of analog systems which give it a unique place in between two classical domains.

No.	ANALOG	DIGITAL	RACE LOGIC
1	Compute with continuous values of physical variables in some range, typically voltages between the lower and upper power-supply voltages.	Compute with discrete values of physical variables, typically the lower and upper power supply voltages.	Compute with discrete values of physical variables, typically with the delay encoded in the rising edges between the power supply voltages

2	Primitives of computation arise from the physics of the computing devices: physical relations of transistors, capacitors, resistors, floating-gate devices, Kirchoffs current and voltage laws and so forth. The use of these primitives is an art form and does not lend itself easily to automation. The amount of computation squeezed out of a single transistor is high.	Primitives of computation arise from the mathematics of boolean logic: logical relations like AND, OR, NOT, NAND, and XOR. The use of these primitives is a science and lends itself easily to automation. The transistor is used as a switch, and the amount of computation squeezed out of a single transistor is low.	Primitives of computation arise from the physics of the computing devices such as delay stacking for addition and choosing the first/last arriving edges hence implementing MIN/MAX functions using OR/AND boolean primitives. Each time the output of OR/AND gate goes high, it represents the optimal shortest path till that node. Computation squeezed out from each gate is high.
3	One wire represents many bits of information at a given time.	One wire represents 1 bit of information at a given time.	One wire represents many bits of information at a given time.

4	Computation is offset prone since it is sensitive to mismatches in the parameters of the physical devices. The degradation in performance is graceful. Noise is due to thermal fluctuations in physical devices.	Computation is not offset prone since it is insensitive to mismatches in the parameters of the physical devices. However, a single bit error can result in catastrophic failure. Noise is due to round-off error.	Computation is sensitive to process and mismatch variations and switching current injection. Circuit and architectural techniques can be used to greatly reduce their effect. Variation study shows that performance degradation is graceful
5	In a cascade of analog stages, noise starts to accumulate. Thus, complex systems with many stages are difficult to build.	Round-off error does not accumulate significantly for many computations. Thus, complex systems with many stages are easy to build.	Assuming Gaussian noise distribution on delay elements, the fractional deviation of the noise with respect to the value of the signal decreases, but gets biased based on the choice of OR/AND gates. 50 long cascade stage works within 3% error demonstrated.

We should be clear that we do not believe Race Logic is any sort of replacement for traditional design practices in general purpose logic. Rather it is a

new type of data encoding / representation with the opportunity to play a role in improving the energy efficiency or speedup of specific information processing algorithms. Though the sample problem chosen was that of DNA sequence alignment, we have seen that string matching, time warping, scan-line matching are all different monikers for this dynamic programming based formula for solving similarity based optimization problems. Just as multiplication and division operations, which traditionally require complex hardware for binary encoding schemes, can be simplified to addition and subtraction with logarithmic number systems, a delay encoding transforms other problems, such as MIN-MAX, into a far easier to compute space. Of course in both examples there will be many other relationships that are then harder to calculate.

At this point we have proof that some useful computations can be done in this new space but many open questions remain. What other sorts of other computations can be efficiently solved with races? What are the limits to the expressiveness of a Race Logic? Are other timing-based encoding schemes possible? What are the best ways to efficiently implement the expressive and programmable delay elements? While these, and many other, questions remain, we do at least know now that compositions of the MIN, MAX, and ADDBY-CONSTANT primitives provided by this work are sufficient (with the proper routing of values) to solve an important class of bio-informatics similarity problems.

As our field searches for ways to continue to turn transistors into value without having to toggle those transistors and have them consume power, Race Logic points in an interesting and little explored new direction with the potential to outperform traditional designs by a significant degree.

Bibliography

[1] Stephen F Altschul. "Amino acid substitution matrices from an information theoretic perspective". In: *Journal of molecular biology* 219.3 (1991), pp. 555–565.

[2] Stephen F Altschul et al. "Basic local alignment search tool". In: *Journal of molecular biology* 215.3 (1990), pp. 403–410.

[3] Pietro Andreani et al. "A digitally controlled shunt capacitor CMOS delay line". In: *Analog Integrated Circuits and Signal Processing* 18.1 (1999), pp. 89–96.

[4] Donald J Berndt and James Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series." In: *KDD workshop*. Vol. 10. 16. Seattle, WA. 1994, pp. 359–370.

[5] EG Caiani et al. “Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume”. In: *Computers in Cardiology 1998*. IEEE. 1998, pp. 73–76.

[6] The History of Computing Project. “ENIAC, USA 1946”. In: (2013).

[7] Andrew Danowitz et al. “CPU DB: recording microprocessor history”. In: *Communications of the ACM* 55.4 (2012), pp. 55–63.

[8] Robert H Dennard et al. “Design of ion-implanted MOSFET’s with very small physical dimensions”. In: *IEEE Journal of Solid-State Circuits* 9.5 (1974), pp. 256–268.

[9] Sean R Eddy et al. “Where did the BLOSUM62 alignment score matrix come from?” In: *Nature biotechnology* 22.8 (2004), pp. 1035–1036.

[10] Robert Ekblom and Jochen BW Wolf. “A field guide to whole-genome sequencing, assembly and annotation”. In: *Evolutionary applications* 7.9 (2014), pp. 1026–1042.

[11] Dan Ernst et al. “Razor: A low-power pipeline based on circuit-level timing speculation”. In: *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*. IEEE. 2003, pp. 7–18.

[12] Hadi Esmaeilzadeh, Emily Blem, et al. “Dark silicon and the end of multicore scaling”. In: *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE. 2011, pp. 365–376.

[13] Steven A Guccione and Eric Keller. “Gene matching using JBits”. In: *International Conference on Field Programmable Logic and Applications*. Springer. 2002, pp. 1168–1171.

[14] Dzung T Hoang. *A systolic array for the sequence alignment problem*. Department of Computer Science, Univ., 1992.

[15] Dzung T Hoang and Daniel P Lopresti. “FPGA implementation of systolic sequence alignment”. In: *International Workshop on Field Programmable Logic and Applications*. Springer. 1992, pp. 183–191.

[16] Mark Horowitz et al. “Scaling, power, and the future of CMOS”. In: *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest.* IEEE. 2005, 7–pp.

[17] Jiawei Huang, John Lach, and Gabriel Robins. “A methodology for energy-quality tradeoff using imprecise hardware”. In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. 2012, pp. 504–509.

[18] Richard Hughey. “Parallel hardware for sequence comparison and alignment”. In: *Computer applications in the biosciences: CABIOS* 12.6 (1996), pp. 473–479.

[19] Ali Khajeh-Saeed, Stephen Poole, and J Blair Perot. “Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors”. In: *Journal of Computational Physics* 229.11 (2010), pp. 4247–4258.

[20] Ben Langmead and Steven L Salzberg. “Fast gapped-read alignment with Bowtie 2”. In: *Nature methods* 9.4 (2012), pp. 357–359.

[21] Heng Li and Richard Durbin. “Fast and accurate short read alignment with Burrows–Wheeler transform”. In: *Bioinformatics* 25.14 (2009), pp. 1754–1760.

[22] Isaac TS Li, Warren Shum, and Kevin Truong. “160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)”. In: *BMC bioinformatics* 8.1 (2007), p. 1.

[23] Ruiqiang Li et al. “SOAP: short oligonucleotide alignment program”. In: *Bioinformatics* 24.5 (2008), pp. 713–714.

[24] Richard J Lipton and Daniel Lopresti. “A systolic array for rapid string comparison”. In: *Proceedings of the Chapel Hill Conference on VLSI*. 1985, pp. 363–376.

[25] Yang Liu et al. “Gpu accelerated smith-waterman”. In: *International Conference on Computational Science*. Springer. 2006, pp. 188–195.

[26] Advait Madhavan, Timothy Sherwood, and Dmitri Strukov. “Race logic: a hardware acceleration for dynamic programming algorithms”. In: *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*. IEEE. 2014, pp. 517–528.

[27] Nihar R Mahapatra, Alwin Tareen, and Sriram V Garimella. “Comparison and analysis of delay elements”. In: *Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on*. Vol. 2. IEEE. 2002, pp. II–473.

[28] Mohammad Maymandi-Nejad and Manoj Sachdev. “A digitally programmable delay element: design and analysis”. In: *IEEE transactions on very large scale integration (VLSI) systems* 11.5 (2003), pp. 871–878.

[29] Debabrata Mohapatra, Georgios Karakonstantis, and Kaushik Roy. “Significance driven computation: a voltage-scalable, variation-aware, quality-tuning motion estimator”. In: *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM. 2009, pp. 195–200.

[30] Richard Mott. “Alignment: statistical significance”. In: *eLS* (2005).

[31] Przemyslaw Mroszczyk and Piotr Dudek. "Tunable CMOS delay gate with reduced impact of fabrication mismatch on timing parameters". In: *New Circuits and Systems Conference (NEWCAS), 2013 IEEE 11th International*. IEEE. 2013, pp. 1–4.

[32] Ghazi Al-Naymat, Sanjay Chawla, and Javid Taheri. "SparseDTW: a novel approach to speed up dynamic time warping". In: *Proceedings of the Eighth Australasian Data Mining Conference-Volume 101*. Australian Computer Society, Inc. 2009, pp. 117–127.

[33] *NCBI BLAST webpage*. http://www.ncbi.nlm.nih.gov/blast/.

[34] Saul B Needleman and Christian D Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[35] Yuichi Ohta and Takeo Kanade. "Stereo by intra-and inter-scanline search using dynamic programming". In: *IEEE Transactions on pattern analysis and machine intelligence* 2 (1985), pp. 139–154.

[36] Lawrence Rabiner and Biing-Hwang Juang. "Fundamentals of speech recognition". In: (1993).

[37] Martin Saint-Laurent and Madhavan Swaminathan. "A digitally adjustable resistor for path delay characterization in high-frequency microprocessors".

In: *Mixed-Signal Design, 2001. SSMSD. 2001 Southwest Symposium on.* IEEE. 2001, pp. 61–64.

[38] Hiroaki Sakoe and Seibi Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), pp. 43–49.

[39] Rahul Sarpeshkar. "Analog versus digital: extrapolating from electronics to neurobiology". In: *Neural computation* 10.7 (1998), pp. 1601–1638.

[40] Matthew D Schmill, Tim Oates, and Paul R Cohen. "Learned models for continuous planning." In:

[41] Byonghyo Shim, Srinivasa R Sridhara, and Naresh R Shanbhag. "Reliable low-power digital signal processing via reduced precision redundancy". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12.5 (2004), pp. 497–510.

[42] Raj K. Singh et al. "BIOSCAN: a network sharable computational resource for searching biosequence databases". In: *Computer applications in the biosciences: CABIOS* 12.3 (1996), pp. 191–196.

[43] Temple F Smith and Michael S Waterman. "Identification of common molecular subsequences". In: *Journal of molecular biology* 147.1 (1981), pp. 195–197.

[44] Dmitri B Strukov et al. “The missing memristor found”. In: *nature* 453.7191 (2008), pp. 80–83.

[45] Earl E Swartzlander and Aristides G Alexopoulos. “The sign/logarithm number system”. In: *IEEE Transactions on Computers* 24.12 (1975), pp. 1238–1242.

[46] Michael Taylor. “A Landscape of the New Dark Silicon Design Regime”. In: *Micro, IEEE* (Sept. 2013).

[47] James W Tschanz et al. “Dynamic sleep transistor and body bias for active leakage power control of microprocessors”. In: *IEEE Journal of Solid-State Circuits* 38.11 (2003), pp. 1838–1845.

[48] Ajay K Verma, Philip Brisk, and Paolo Ienne. “Variable latency speculative addition: A new paradigm for arithmetic circuit design”. In: *Proceedings of the conference on Design, automation and test in Europe*. ACM. 2008, pp. 1250–1255.

[49] Jack E Volder. “The CORDIC trigonometric computing technique”. In: *Electronic Computers, IRE Transactions on* 3 (1959), pp. 330–334.

[50] Linda Wilson. “International technology roadmap for semiconductors (ITRS)”. In: *Semiconductor Industry Association* (2013).

[51] Renyuan Zhang and Mineo Kaneko. “A feasibility study on robust programmable delay element design based on neuron-mos mechanism”. In: *Proceedings of the 24th edition of the great lakes symposium on VLSI*. ACM. 2014, pp. 21–26.