CONCEPTS IN RELIABLE AND OPTIMAL SYSTEMS DESIGNS

A Project Report

submitted by

MOHAMMED SHOAIB

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

MAY 2008

THESIS CERTIFICATE

This is to certify that the thesis titled **CONCEPTS IN RELIABLE AND OPTIMAL SYSTEMS DESIGN**, submitted by **Mohammed Shoaib**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Kamakoti. V
Research Guide
Assoc. Professor, Dept. of CSE
IIT Madras, Chennai 600 036

Srinivasan. S
Research Guide
Professor, Dept. of EE
IIT Madras, Chennai 600 036

Place: Chennai

Date: 01st May 2008

ACKNOWLEDGEMENTS

"One can pay back the loan of gold, but one dies forever in debt to those who are kind."-
A Malay Proverb

I am thankful and beholden to God for his infinite mercy and blessings showered on me. I am superlatively saddled by my parents support, encouragement and prayers in making this all happen. My brother, Adnan, has been my single most important mentor, buddy and card-carrier throughout my five year stint at IIT Madras. I would like to acknowledge his help and thank him for all the wonderful discussions I had with him - every one of them was so important in reaching this milestone.

I am delighted to express my deepest sense of gratitude to Dr. Kamakoti, my advisor and guide at the Computer Science and Engineering (CSE) Department at IIT Madras. He kept me focused with his tremendous help, energy and motivation all through the wonderful year I spent at The Reconfigurable and Intelligent Systems Engineering (RISE) lab. The resources at the lab have been impeccable and I would like to thank him for helping me grow in such a world class facility. This overture would be left void if I equivocate the kind concern and guidance of Dr. Srinivasan, my co-advisor from the Electrical Engineering (EE) Department at IIT Madras. His forbearance has been phenomenal in carrying me along the various crests and troughs in my graduate career. In fact, the entire faculty at the EE department has been responsible for casting my foundation in VLSI and the CSE faculty has collectively been responsible for helping me build upon this strong foundation. Dr. Shankar, Dr. Debdeep and Dr. Ravindran have been instrumental in helping me look beyond the horizons of current technology. Their qualitative excellence and commitment to research has helped me transpire constructively and continue working on topics of significant importance.

This rubric would be incomplete without acknowledging the friendly support and enormous motivation from Noor, my closest pal at the RISE lab. His beneficial cona-

tus has been pivotal in this work - technical and otherwise. I would like to thank him for introducing me to evolutionary computing and helping me with various simulations, paper writing, discussions and outages. Vimal, a third year undergraduate student in CSE at IIT Madras has also been very helpful through various simulations and discussions. I gratefully acknowledge his contribution to this work and thank him for all the useful inputs. I would like to thank my other lab mates - Venkat for his terrene and seraphic discussions at the lab, Rajesh for his terrene discussions outside the lab and Chester for his seraphic discussions within and outside the lab. I would also like to express my appreciation and thanks to Kumar Dhirendra Pratap Singh Yadav from the EE department who has been a wonderful companion and a good friend of mine at IIT Madras.

I cannot put in words, my expression of gratitude to the members of RISE lab at IIT Madras for the numerous inputs, ideas and discussions, all of which were so very useful in shaping up this work. Finally, I would like to mention significant motivation and fun-times stocked up by my dorm mates - Saurabh, Nitin, Gaurav, Shashank, Lokesh and Dhirendra. Their repose at restaurant lounges, movie theaters, gymnasiums and sports grounds was something I'll have to thank for - a kinesis which was subservient to my survival in research.

"What we have done for ourselves alone dies with us; what we have done for others and the world remains and is immortal "- Albert Pike

SHOAIB, MOHAMMED

Indian Instt. of Tech. Madras

ABSTRACT

KEYWORDS: Reliable Systems; Mathematical Optimization; Genetic Algorithms; Reconfigurable Hardware.

Designing reliable and optimal systems is one of the most challenging goals in contemporary electronics. As technologies scale and information processing gets critical, reliability in high performance systems plays a vital role in maintaining prolonged uptime and dependable outputs. In this work, we provide insights into designing reliable systems with optimal configurations using a popular stochastic technique called the Genetic Algorithm (GA). GAs are highly tunable algorithms which work with large amounts of data and simple operators to yield optimal solutions to difficult problems. The thesis uses abstractions at the transistor, gate and systems level to demonstrate the heuristics and challenges in the same. In a transistor optimization problem the thesis propounds a novel technique for evolving transistor net lists directly from truth table descriptions of arbitrary digital circuits. A salient feature of the proposed technique is the bypassing of gate level representation and optimization in the VLSI design flow. This leads to generation of custom and semi-custom library cells on the fly. The second problem involves finding input vector pairs that cause maximum power dissipation in digital circuits. A modified genetic search and a vector partitioning approach are used to obtain good lower bounds for the same. GA heuristics for FIR Filters are presented in a third problem. The thesis demonstrates that the filters thus designed are self-adaptive; respond to arbitrary frequency response landscapes; have built-in coefficient error tolerance capabilities; and have a minimal adaptation latency. As a byproduct of this, it also proposes a novel flow for the complete hardware design of what is termed as an Evolutionary System on Chip (ESoC). Finally a fourth optimization problem outlines the design methodology of a SEU tolerant Distributed RAM using Configurable Logic Blocks (CLBs) on FPGAs incorporating unused CLB BlockRAMs for high speed On-Chip memories and SEU resistant Tri-State Buffers (BUFTs) for the ECC.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ANN	Artificial Neural Network
ATG	Automatic Test Generation
ASIC	Application Specific Integrated Circuit
BiET	Built-in Error Tolerance
BUFT	Tristate Buffer
CHE	Complete Hardware Evolution
CLB	Configurable Logic Block
CRF	Coeffficient Register File
CROOT	Cone Root
DRAM	Distributed Random Access Memory
DSP	Digital Signal Processing
DWC	Doubling With Comparison
ECC	Error Correction Control
ECC	Error Correcting Codes
ER	Elitism Rate
ESoC	Evolutionary System-on-Chip
FD	Frequency Domain
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
ftDRAM	fault tolerant Distributed Random Access Memory
FWL	Finite Word Length
GA	Genetic Algorithm
GO	Genetic Operator
HD	Hamming Distance
IC	Integrated Circuit
IR	Incircuit Reconfiguration

IIR	Infinite Impulse Response
IP	Initial Population
JTAG	Joint Test Action Group
LL	Label List
LUT	Look-Up-Table
MBU	Multiple Bit Upset
MEU	Multiple Event Upset
MOS	Metal Oxide Semiconductor
MR	Mutation Rate
MTTR	Mean Time To Repair
NDS	Non-Dominated Set
NoC	Network on Chip
PDPE	Peak Dynamic Power Estimation
PI	Primary Inputs
PO	Primary Outputs
PODG	Primitive Operator Directed Graph
PoS	Product of Sum
RAM	Random Access Memory
SA	Simulated Annealing
SD	Spatial Domain
SEB	Single Event Burnout
SEE	Single Event Effect
SEGR	Single Event Gate Rupture
SET	Single Event Transient
SEU	Single Event Upset
SHE	Single Hard Error
SoP	Sum of Product
SRAM	Static Random Access Memory
TMR	Triple Modular Redundancy
TTM	Time-to-Market
WE	Write Enable

NOTATION

S_j	Input signal and gate node of the j^{th} transistor
n_j^1	Source node of the j^{th} transistor
n_j^2	Drain node of the j^{th} transistor
T_{ij}	Transistor type (CMOS/NMOS) between nodes i and j
$L(W)_{ij}$	Transistor channel length (width) in μm
R_{ij}	Uniform Random Selector
δ_{ij}	Select Scalar for the GA
ξ	Genetic copy operator
ε	Genetic fitness evaluator
l	Length of truth table input
f_i	i^{th} chromosome fitness in genotype matrix
$\oplus$	Logical XOR operator
Λ	Fitness sorting operator
C_g	Output capacitance of gate g
V_{dd}	Rail supply voltage
P_R	Energy per clock cycle (peak power)
O	Complexity order of an algorithm
$\bigcup$	Set union operator
(v_i, v_j)	Random input vector pair to circuit
$\asymp$	Asymptotically equal to
h_i	i^{th} filter coefficient (tap weight)
δ_p	Filter passband ripple
δ_s	Filter stopband ripple
ΔF	Filter transition bandwidth
$G_{id(ev)}$	Filter ideal (evaluated) response
f_n	Filter normalized frequency - $\frac{f}{f_s}$
f_s	Filter sampling frequency
f_c/f_{cutoff}	Filter cutoff frequencies
f_{Nyq}	Filter nyquist frequency
Δ^{mean}	Mean inverse coefficient space value
δ^{step}	Inverse coefficient space step for interpolation
$N(\mu, \sigma)$	Std normal distribution - mean μ & SD σ
ξ_{curr}	Current evaluated fitness operator
:	Matrix concatenation operator
$\bullet$	Vector dot product
ϕ	Circuit clock signal
$T_{WCLK(H)}$	Clock write (hold) period

CHAPTER 1

MOTIVATION AND BACKGROUND

Typical engineering systems are described by a very large number of variables, and it is the designer's task to specify appropriate values for these variables. Skilled designers utilize their knowledge, experience, and judgment to specify these and design effective engineering systems. Because of the size and complexity of the typical design task, however, even the most skilled designers are unable to take into account all of the variables simultaneously. Design optimization is the application of numerical algorithms and techniques to engineering systems to assist in improving the system's performance, reliability, and cost [Adoptech (2008)]. Optimization methodologies can be applied during the development stage to ensure that the finished designs will have high performance, reliability and low cost. Alternatively, optimization methods can be applied to existing designs to identify potential improvements. This thesis is dedicated to providing the tools and knowledge required to generate practical designs that are useful in the real world. It demonstrates the use of a probabilistic algorithm called the Genetic Algorithm (GA) for the same. The thesis also presents insights into reliability concepts on a reconfigurable platform.

1.1 Design Optimization Paradigms

Optimization refers to the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. An optimization problem can be represented as follows

$$Given: \ A\ function\ f\ : A \rightarrow R\ from\ some\ set\ A\ to\ the\ real\ numbers \tag{1.1}$$

$$Find: \ An\ element\ x_0\ in\ A\ such\ that \tag{1.2}$$

$$f(x_0) \leq f(x)\ \forall\ x\ \in\ A\ (\text{"}minimization\text{"}) \tag{1.3}$$

$$f(x_0) \geq f(x)\ \forall\ x\ \in\ A\ (\text{"}maximization\text{"}) \tag{1.4}$$

In Eq.(1.2) through Eq.(1.4), typically, A is some subset of the Euclidean space R_n, often specified by a set of constraints, equalities or inequalities that the members of A have to satisfy. The domain A of f is called the search space, while the elements of A are called candidate solutions or feasible solutions. The function f is called an objective function, or cost function. A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an optimal solution.

Many real-world and theoretical problems may be modeled in this general framework. Sometimes this technique is referred to as energy minimization wherein the concept is the association of the function f with the energy of a system. Generally, when the feasible region or the objective function of the problem does not present convexity, there may be several local minima and maxima, where a local minimum x^* is defined as a point for which there exists some $\delta > 0$ so that for all x such that $||x - x^*|| \leq \delta$, the expression $f(x^*) \leq f(x)$ holds; that is to say, on some region around x^* all of the function values are greater than or equal to the value at that point. Local maxima are defined similarly.

A large number of algorithms proposed for solving non-convex problems including the majority of commercially available solvers are not capable of making a distinction between local optimal solutions and rigorous optimal solutions, and will treat the former as actual solutions to the original problem [Optimization (2008)]. The branch of applied mathematics and numerical analysis that is concerned with the development of deterministic algorithms that are capable of guaranteeing convergence in finite time to the actual optimal solution of a non-convex problem is called global optimization. The local and global optimization concepts are critical in problem validation. The evolutionary heuristics using GAs proposed in this thesis provide insights into both of these using three different problems.

1.2 Evolutionary Optimization

Why use evolution as an inspiration for solving computational problems?

The mechanisms of evolution seem well suited for some of the most pressing computational problems in many fields. Many computational problems require searching through

a huge number of possibilities for solutions. On an alternative note, search problems can often benefit from an effective use of parallelism, in which many different possibilities are explored simultaneously in an efficient way. But what is needed is both computational parallelism (i.e., many processors evaluating sequences at the same time) and an intelligent strategy for choosing the next set of sequences to evaluate.

Many computational problems require a computer program to be adaptive - to continue to perform well in a changing environment. Others require computer programs to be innovative - to construct something truly new and original. Finally, many computational problems require complex solutions that are difficult to program by hand. In the current context, the best route to intelligence is through a bottom-up paradigm in which humans write only very simple rules, and complex behaviors emerge from the massively parallel application and interaction of these simple rules.

In evolutionary computation the rules are based on natural selection with variation due to crossover and/or mutation; the hoped-for emergent behavior is the design of high-quality solutions to difficult problems and the ability to adapt these solutions in the face of a changing environment. Biological evolution is an appealing source of inspiration for addressing these problems. Evolution is, in effect, a method of searching among an enormous number of possibilities for solutions. In biology the enormous set of possibilities is the set of possible genetic sequences, and the desired solutions are highly fit organisms - organisms well able to survive and reproduce in their environments. Evolution can also be seen as a method for designing innovative solutions to complex problems. Seen in this light, the mechanisms of evolution can inspire computational search methods. Of course the fitness of a biological organism depends on many factors - for example, how well it can weather the physical characteristics of its environment and how well it can compete with or cooperate with the other organisms around it. The fitness criteria continually change as creatures evolve as typified by the FIR filter problem in Chap.4. So evolution is searching a constantly changing set of possibilities. Searching for solutions in the face of changing conditions is precisely what is required for adaptive computer programs. Furthermore, evolution is a massively parallel search method: rather than work on one species at a time, evolution tests and changes millions of species in parallel. Finally, viewed from a high level the rules of evolution are remarkably simple: species

evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce thus propagating their genetic material to future generations. Yet, these simple rules are thought to be responsible, in large part, for the extraordinary variety and complexity as we see in the biosphere [Mitchell (1998)]. The following sections briefly outline common methods used in the artifical evolution process. More details on these can be found in the excellent book by Greenwood and Tyrrel [Greenwood and Tyrrel (2007)].

1.2.1 Genetic Algorithms: Principles of Natural Selection

Evolution in hardware is very similar to that in nature. However, the one major difference between what happens in nature and what happens in electronics is that the hardware evolution is completely artificial with tunable operation models and heuristics. Hardware evolution is categorized into two major categories based on the evolution style.

- **Extrinsic** or software evolution.
- **Intrinsic** or hardware evolution.

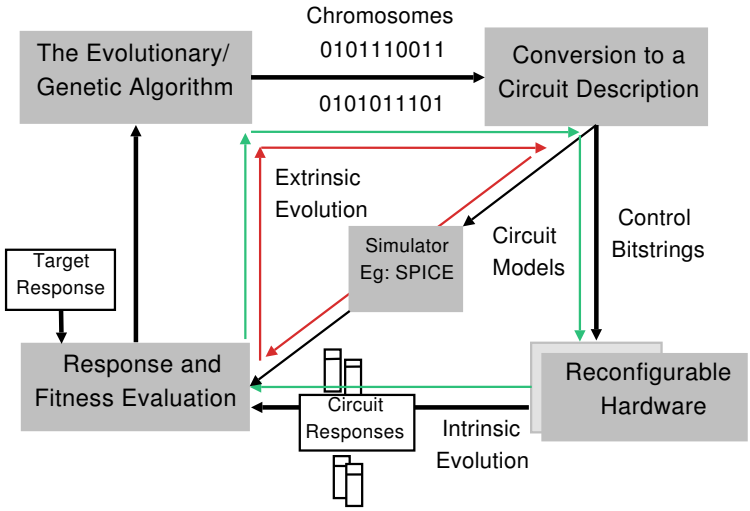


Fig. 1.1: Evolutionary hardware synthesis - Intrinsic and Extrinsic

The two evolutionary synthesis mechanisms are shown in Fig.1.1. The figure shows how a Genetic Algorithm (GA) is used for artificial evolution. After a population generation, the population is varied using mutation/crossover followed by a simulation and evaluation

either in software (extrinsic) or on a reconfigurable platform (extrinsic). The evaluated value of the current solution determines the termination or cycling criterion for the GA.

1.2.2 A Basic GA Cycle

Genetic Algorithms are stochastic algorithms with simple operations like Random number generation, string copy and exchange. They are used for hardware evolution and mimic the process of natural selection to solve difficult problems. GAs fare well with problems having a large search space and their solutions tend to "mature with time". A basic Genetic Algorithm (GA) cycle is demonstrated as a flow algorithm in Fig.1.2. The gene

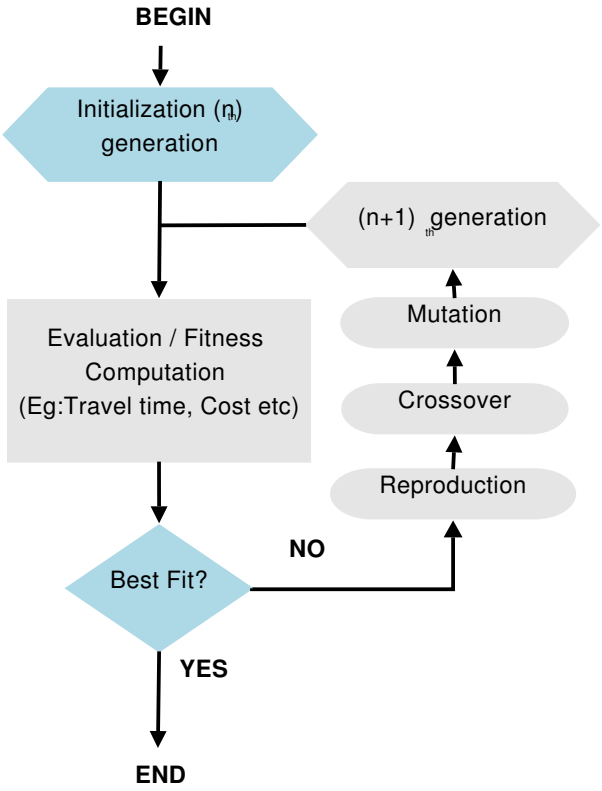


Fig. 1.2: A basic GA cycle

structure in a GA models the hardware parameters, the on field performance is evaluated as fitness, the structure is varied by crossover and mutation and the fitter chromosomes are chosen for future iterations much like an intelligent biological system. A GA cycle can be summarized using Eq.(1.5).

$$\varphi(n+1) = \zeta(\varepsilon(\upsilon(\varphi(n)))) \tag{1.5}$$

where the variables stand for the initial population generation, population variation, evaluation and selection. The following paragraphs discuss details about each of these operators in a GA cycle.

1. Initial Population (IP) Generation (φ)
2. Population Variation - Crossover and Mutation (υ)
3. Population Evaluation - Fitness (ε)
4. Selective Progression (ζ)

1.2.3 Representation (φ)

The first major hurdle in modeling a physical problem to be compatible with a genetic algorithm is its representation using a data structure that encodes all the problem parameters. A cue from the biological hierarchy of genetics can be obtained. The genetic hierarchy has a *gene* as its most basic block which has a locus defined in the system. A collection of genes form a *phenotype* which form a part of a *genotype*. The *genome* is a selection of genotypes and constitutes the highest level of hierarchy in the genetic abstraction. There are five common encodings which could be used for population representation.

Integers Consider an RC circuit with the values of R and C to be encoded. The values could be represented using integers - R$[470, 220]$ for $[470\Omega, 220\Omega]$ and C$[10, 47]$ for $[10\mu F, 47\mu F]$

Binary Strings The values of the capacitance and resistance above could also be represented using unique binary strings - R$[000, 100]$ for $[470\Omega, 220\Omega]$ and C$[00, 01]$ for $[10\mu F, 47\mu F]$

Real Numbers Given a l bit binary string, the real number representation can form a compact representation for the same. To translate from a binary representation, in this scheme, a real number x is assigned to a current population model such that $x \in [x_{min}, x_{max}]$ The encoding scheme is shown in Eq.(1.6). The decoding can be done by substituting $\widetilde{x}$ into Eq.(1.7). This kind of a modeling is often used in assigning tap weights on digital filters

and the popular IEEE 754 standard.

$$\widetilde{x} = \sum_{i=0}^{l-1} b_i.2^i \quad ; b_i \in \{0, 1\} \tag{1.6}$$

$$x = x_{min} + \widetilde{x}.\left[\frac{x_{max} - x_{min}}{2^l - 1}\right] \tag{1.7}$$

Graphs Graphs are another way of IP representation. This scheme is particularly useful in modeling circuit topologies and neural networks. The graphs could be directed or undirected as shown in Fig.1.3 **Hybrids** For the IP representation, hybrid representation

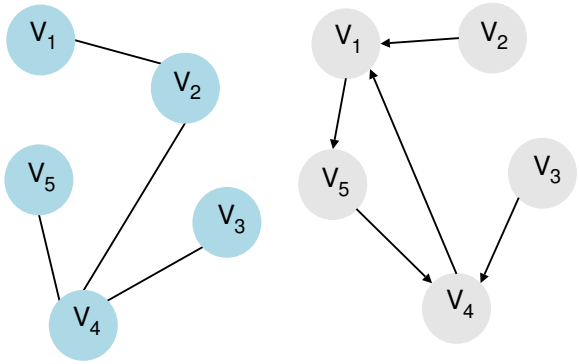


Fig. 1.3: IP representation using directed and undirected graphs

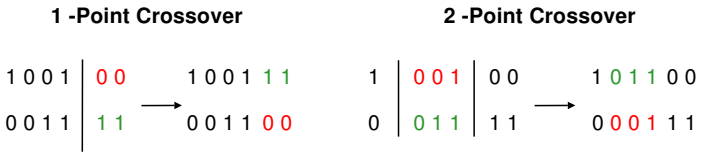
schemes using graphs, real numbers, integers and so on could be used. An example when this is particularly useful is a problem which involves both circuit topologies and component values. Typical design challenges are to identify a proper encoding and IP representation scheme for the problem at hand. A choice of IP modeling can be aptly suited for a specific problem type.

1.2.4 Variation (υ)

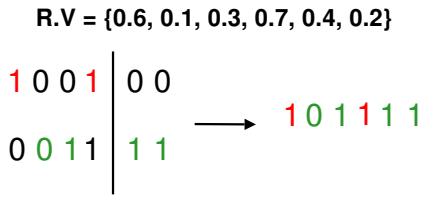
Variation is the variety inducing process in a given population. This is often a random process which changes the encoded parameters of an IP. The result of a variation operation is the creation of newer offspring which are hopefully better ones. There are two major types of variation operations in the genetic cycle *viz.* mutation and recombination.

Recombination: Exemplified below are two types of recombination operations. There could be variations of these or newer ones.

n-Point Crossover The n-point crossover mechanism involves the selection of one or more random points (may be heuristic based) in a chromosome representation followed by an interchanging of genetic material among two or more chromosomes Fig.1.2.4.



Uniform Crossover This is another popular type of the crossover operation. This the generation of a random selection probability with its cardinality equal to the number of genes in evolving chromosome. A recombination strategy in this involve choosing a gene from the first IP vector if the random selection vector has a probability value ≥ 0.5 else from the second IP vector. A new recombination vector is hence produced using the two IP vectors after variation using the selection probability vector as shown exemplified in Fig.1.2.4



1.2.5 Evaluation (ε)

Evaluation is the third major step in the genetic process. In the evaluation step, an objective function (Φ) is modeled such that, $\Phi : X \rightarrow \Re$. Where X is the chromosome under iteration for the given problem. This mapping needs to be a very intelligent one as the final outcome of the GA is strongly dependant on the fitness function. This usually involves the inclusion of all the optimization parameters in the problem. Oftentimes, they are conflicting but some other times they are independent. The objective function (Φ) is either maximized or minimized. Depending on this, a given chromosome is selected if $\Phi(X(n+1)) \geq \Phi(X(n))$ or if $\Phi(X(n+1)) \leq \Phi(X(n))$. An example objective function

is shown in Eq.(1.8)

$$\Phi(s) = \frac{1}{\sum_{i=1}^{M}[G'(i) - G(i)]^2 + \epsilon} \ , \ \epsilon << 1 \tag{1.8}$$

where ϵ is used to keep the fraction finite and the denominator of the fitness is the square of the difference of objectives between the current and the past iteration.

1.2.6 Selection (ζ)

Selection of a given chromosome from an IP pool to move on to future generations involves problem heuristics and results in maturing solutions and competitive species. Five typical selection schemes are outlined in below.

Uniform Selection is a scheme where the chromosomes from a population pool are selected randomly with an equal probability. Of course the convergence rates of this scheme are slow but this provides the highest variety of populace to move on to future generations.

Fitness Proportional Selection involves the selection of a chromosome from a pool with a probability proportional to its fitness value as shown in Eq.(1.9)

$$prob(i) = \frac{\Phi_i}{\sum_{j=1}^{N} \Phi_j} \tag{1.9}$$

Fitness Ranking Selection is similar to fitness proportional selection but the difference being that the selection probability is this case is proportional to the exponential transform of the ranks assigned to chromosomes based on their fitness values, Eq.(1.10)

$$prob(i) = \frac{1 - exp(-r)}{C} \qquad r : rank, \ C : norm \ const \tag{1.10}$$

q-Tournament Selection is a biased selection scheme which involves the selection of a chromosome from a random set of q individuals chosen from the pool of chromosomes under the current iteration Eq.(1.11).

$$\Phi_{max} \ from \ Random \ q \ individuals. \tag{1.11}$$

Truncation Selection is one of the most popular selection schemes which involves the choice of promoting μ offspring with high fitness values among a set of $(\lambda+\mu)$ individuals as shown in Eq.(1.12)

$$\{\Phi^1_{high}, \Phi^2_{high} \ldots \Phi^{\mu}_{high}\}\ among\ (\lambda+\mu)\ offspring. \tag{1.12}$$

The methods of IP Generation, Variation, Evaluation and Selection proposed in this section are just an example of the methods currently popular in literature. There are numerous other variations of these. The most important aspect of a GA is the adjustment of these methods based on problem heuristics. In Chap.3 and Chap.4 we exemplify the importance of these heuristic adaptations using two important problems in circuit and systems design. After a genetic approach to optimization, the thesis proposes a reliable system design on a reconfigurable platform in Chap.5. Hence, in the following sections, the thesis presents some basics about reconfigurable hardware, reliability and soft errors.

1.3 Reconfigurable Hardware and Reliability

Field Programmable Gate Arrays (FPGAs) are programmable digital logic chips which can be programmed to do almost any digital function.

1.3.1 Logic Cell

FPGAs are built from one basic *logic-cell*, duplicated hundreds or thousands of times. A logic-cell is basically a small lookup table (LUT), a D-flipflop and a 2-to-1 MUX (to bypass the flipflop if desired). Fig.1.4 outlines a typical logic cell showing the multiplexer, the flip flop and the LUT. The LUT is like a small RAM and has typically 4 inputs and can implement any logic gate with up to 4-inputs. For example, an AND gate with 3 inputs, whose result is then ORed with another input would fit in one 4-input LUT.

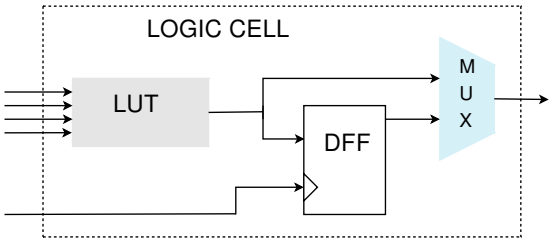


Fig. 1.4: A typical FPGA Logic Cell

1.3.2 Interconnect and Routing

Each logic-cell can be connected to other logic-cells through interconnect resources (wires/muxes placed around the logic-cells). Each cell can do little, but with lots of them connected together, complex logic functions can be created. FPGAs also have fast dedicated lines in between neighboring logic cells. The most common lines are called carry chains. Carry chains allow the creation of arithmetic functions like counters and adders efficiently. Also, within the FPGA fabric, there are interconnect buffers which are tri-stated AND-OR logic elements and can be used to realize hardwired logic functions in special scenarios.

1.3.3 Internal RAM

In addition to logic, all new FPGAs have dedicated blocks of static RAM distributed among and controlled by the logic elements. There are many parameters affecting the RAM operation. The main parameter is the number of agents that can access the RAM simultaneously. Based on the access patterns, they can be classified as

- Single-port RAMs: only one agent can read/write the RAM.
- Dual-port or quad-port RAMs: 2 or 4 agents can read/write.

Dual port RAMs are efficient in getting data across clock domains where each agent can use a different clock.

Blockram vs. Distributed RAM

There are two types of internal RAMs in an FPGA: blockrams and distributed RAMs. The size of the RAM needed in an application usually determines which type is used.

1. The big RAM blocks are blockrams, which are dedicated areas in the FPGA. Each FPGA has a limited number of these and if unused, they are lost (they cannot be used for anything but RAM).
2. The small RAM blocks are either in smaller blockrams or in distributed RAM. Distributed RAM allows using the FPGA logic-cells as tiny RAMs.

Distributed RAM brings a lot of flexibility in the RAM distribution in an FPGA, but isn't efficient in term of area as a logic-cell can actually hold very little RAM. There are a large number of block RAMs within the Xilinx family of FPGAs [fpga4fun (2008)]

1.3.4 Errors in High Performance Memories

In the field of high-performance communication memory devices, it is critical for designs to be immune to soft errors or single-event upsets. As device technology scales, the area efficiency of memory devices decreases, and a device's natural resistances against SEUs (single-event upsets) decreases. A reconfigurable platform like an FPGA makes the hardware even more susceptible as device configurations in the FPGA are stored in SRAM cells.

SEUs are random and rarely catastrophic, and they do not normally destroy a device. Many systems can tolerate some level of soft errors. An occasional bad bit may be unnoticeable and unimportant in many applications. However the use of memory elements in mission-critical applications to control system functions makes soft errors more impactive and lead to not only corrupt data, but also a loss of function and system-critical failures. Getting worse, not better. Poor system design is a common source of SEUs. High performance memory devices normally comprise SRAM cells, combinational logic, and latches. In high-performance memories, the area efficiency is usually low. Past research [Lima *et al.* (2003)], [C Carmichael and Caffrey (1999)] shows that combinational logic is less susceptible to soft errors than memory cells because of natural resistances set up by masking. However, these natural resistances could diminish as devices scale and technologies advance[C Carmichael and Caffrey (1999)].

Check information for memories could be used to serve as a correction mechanism for SEUs. They serves two purposes; First, when a check word is read from memory, the

check information can help determine whether any of the data bits have changed. In ECC detection, the check information can help determine whether a single bit or more than one bit has changed. Second, if only a single bit has changed, ECC correction helps determine which bit changed and facilitate correcting the data by flipping the bit back to its complementary value. An ECC-detection circuit detecting a change in one or more bits in a word of data is broadly categorized as an ECC error. These errors can further be categorized as functions of the number of bits in the error. ECC circuits described in literature can correct single bits and report multi-bit errors [Mastipuram and Wee (2004)]. These can be implemented for correction in hardware or software. However, the ECC logic designed in hardware is itself SEU prone as the logic for the ECC in a reconfigurable platform would again have to reside in LUTs. The concluding parts of this thesis present insights into the design of a reliable high performance on chip memory using redundant logic elements in FPGAs. This provides a robust and reliable memory for high performance applications with a built in hardwired ECC.

1.4 Context and Objective

The general framework of this thesis is the design of optimal and reliable systems. For this we use heuristic genetic algorithms and reconfigurable hardware. We look at three problems at three abstraction levels in the first three chapters as shown in Fig.1.5. The problems in the figure are summarized with short code names - cmosGAsim for the second chapter, pVirus for the third, firGA for the fourth and ftDRAM for the fifth. The first problem (cmosGAsim) addresses the issue of obtaining optimal transistor netlists with no redundancy for arbitrary truth table specifications. This leads to what is known as the gateless custom VLSI design flow which may help in designing custom library cells on the fly with digital CMOS logic styles. The second problem (pVirus) is the power virus problem which is an early peak dynamic power estimation problem. This is a critical problem from the perspective of system reliability and design guard-banding. We use a heuristic GA and vector partitioning methods for the PDPE problem. The third optimization problem (firGA) works at the system level describing the design of optimal filters automatically which are reliable and robust using a novel, heuristic GA. The final prob-

lem addressed by the thesis is that of the reliability and efficient design of high speed memories (ftDRAM). We use a reconfigurable platform (FPGAs) for demonstrating the same. Hence, in the context of reliability and optimality of system designs, this thesis contributes significantly by providing new insights into important issues. The major ob-

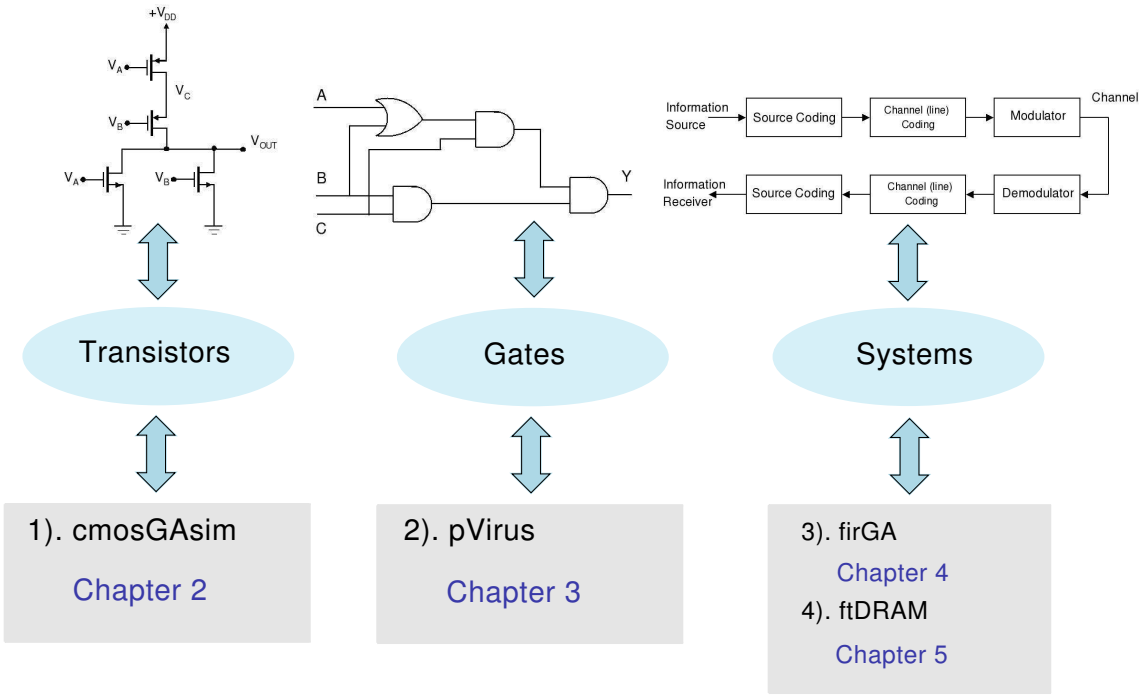


Fig. 1.5: The thesis framework - optimization abstractions

jectives of the thesis are to show new methods for optimization and design of reliable systems. This is effected using a reconfigurable platform as well as other grounds. Besides this the thesis also intends to point out the importance of the various steps in genetic optimization for a given problem. The thesis also aims to bring out the effective use of design abstraction, layout tools, simulation software and the genetic methods in heuristic scenarios. It aims to show that the performance of systems can be enhanced by intelligent design choices and systems can be made reliable using redundant logic elements in a hardware platform.

1.5 Organization of Thesis

To address the optimization problems, the thesis follows an *abstraction* organization, *i.e* it work on optimization problems at the transistor level and move upto the system level. Reliable and adaptive systems being at the core of all this. The thesis is organized as follows

$Ch.2$ **Genetic transistor level optimization:** A genetic approach to gateless custom VLSI design flow - Problem showing the usage of GAs to obtain optimal transistor netlists directly from truth table descriptions.

$Ch.3$ **Genetic gate level optimization:** Vector parts and genetic search methods for PDPE in digital circuits - Problem of reliable early power estimates for effective guard-banding and optimal system synthesis.

$Ch.4$ **Genetic system level optimization:** Hardware based genetic evolution of FIR filters - Problem of designing Self-adaptive and reliably responsive filters for arbitrary frequency responses.

$Ch.5$ **System level Reconfigurable Design:** A SEU tolerant distributed CLB RAM for in-circuit reconfiguration - Problem of designing reliable memories on reconfigurable platforms.

App. **CMOS Concepts, PDPE Results and ECC:** The three appendices at the end of the thesis describe basics of CMOS logic design, Hamming ECC and the PDPE results on ISCAS'85 benchmark circuits using the proposed techniques.

CHAPTER 2

A GENETIC APPROACH TO GATELESS CUSTOM VLSI DESIGN FLOW

2.1 Introduction and Background

Automation in modern microelectronics has resulted in multiple methods for the design and optimization of digital circuits. These automated methods incorporate gate level optimizations followed by the use of standard library cells to map the designs to hardware. In several custom designed Integrated Circuits (ICs), the use of standard libraries translates to hardware redundancy. This is because, standard library cells, due to their limitations in size and number, often result in having several unused transistors for the custom logic. Consequently, the use of tailored cells for specific applications become necessary. Application specific generation of such cells utilizes on-chip hardware effectively besides providing fast and flexible designs operating with lesser redundancy and better performance.

Realization of custom circuits - cells or blocks, involve three implementation phases as described in [Lefebvre and Marple (1997)]:

- Creation of transistor circuit topologies which provide a specific digital function.
- Sizing and ordering of the transistors in the circuit topology.
- Placing, routing and compacting the transistors in layout.

Each of the above stages involve trade-offs which must be optimized across all stages. This chapter proposes to address the first phase of transistor circuit topology creation, automatically.

In literature, much attention has been given to the sizing [Rogenmoser *et al.* (1996)], [Heusler and Fichtner (1991)] and placing of transistors [Lefebvre and Marple (1997)],

[Ho *et al.* (1997)] and [Bahuman *et al.* (2002)] in custom and semi-custom circuits. Using Genetic Algorithms (GAs) in [Ho *et al.* (1997)], Murphy, et. al, describe the placement optimization of a cell followed by the *extraction of* the netlist. They employ sigmoidal transistor characteristics for an Artifical Neural Network (ANN) model unlike the switch characteristics in our case. Their major aim is the use of primitive components and reduction of the parasitic capacitance rather than a topological optimization. In [Bahuman *et al.* (2002)] is described a GADO model for a custom cell. But their starting point is the placement optimization unlike configuration optimization as in our case.

This chapter focuses on the direct transistor netlist generation using Genetic Algorithms for optimization. Genetic methods have been applied in the past for gate level synthesis [Hounsell and Arslan (2000)] besides specific optimization methods for Pass Transistor Logic (PTL) [Cho and Lee], Complementary PTL (CPL) [Yano *et al.* (1990)], Differential PTL (DPTL) [Pasternak (1993)], Double PTL (DPL) [Suzuki *et al.* (1993)] and other non-complementary MOS logic styles. Mazumder and Rudnick, in [Mazumder and Rudnick (1999)] provide a good insight into the problems in VLSI design and synthesis techniques.

The methods we use in this chapter involve the use of Genetic Operators to evolve transistor netlists for a certain functional requirement described by an input truth table. The netlist generation, because of the characteristics of the genetic operators, inherits direct optimization at the transistor level. This gives an optimized transistor netlist which would otherwise be derived after converting the truth table to its min-terms and then applying the Boolean simplification operations. In the latter case, the netlist obtained would still not be optimized to exploit internal topological optimizations of the transistors. The main contribution of this work is to propose the genetic methodology of direct evolution for the transistor netlist from functional descriptions of circuits. The chapter also provides a methodology for incorporating a gateless optimization algorithm in the custom circuit design flow to provide the creation of custom library cells *in-situ* as shown in Fig. 2.1. This favorably enhances the performance of custom circuit syntheses and provides better utilization of transistor resources besides automation and simplification of the design flow, by eliminating the boolean optimization step.

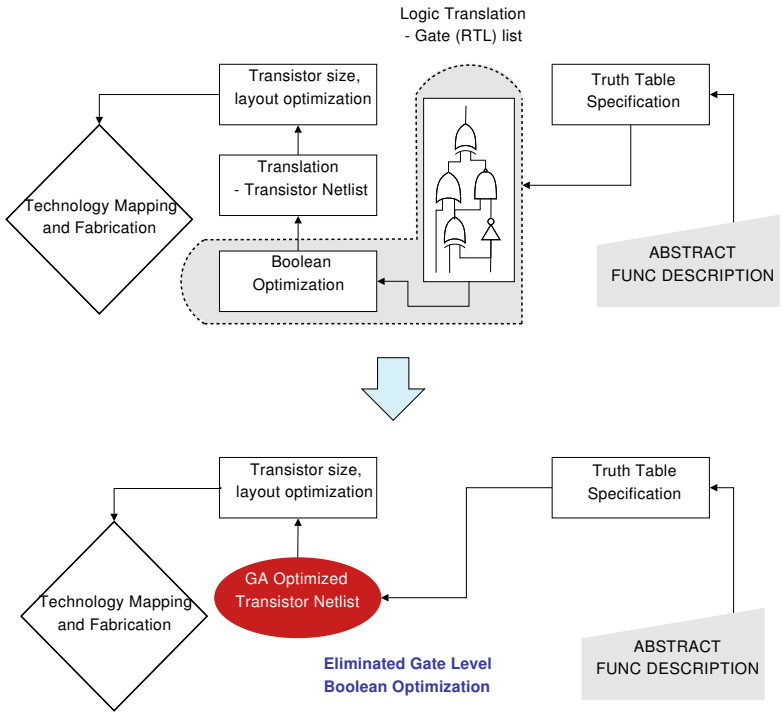


Fig. 2.1: Custom design flow simplification using the proposed genetic method : The genetic flow eliminates the intermediate Boolean simplification step from the normal gate optimized flow by providing a direct transistor optimized netlist, ready for size/layout tuning in custom circuits.

Although described in an earlier chapter, we reiterate the basic concepts of Genetic Algorithms [Godlberg (1989)] necessary to understand the rest of the chapter.

Genetic algorithms work on a set of *chromosomes/genotypes* called the population. Each chromosome represents a solution to the problem which is associated with a *fitness* value that reflects how good it is compared to the other solutions in the population. The variation process comprises of *crossover* and *mutation*, which concoct material by partial exchange among *genotypes* and by random alterations of data strings. The frequency of these operations is controlled by certain pre-set probabilities which require heuristics appropriate for the particular problem at hand. The *representation, variation, evaluation* and *selection* operations constitute the basic GA cycle or *generation*.

Table 2.1: Gene structure in a chromosome using .sim encoding [a]

T_{ij}	S_{ij}	n_{ij}^1	n_{ij}^2	W_{ij}	L_{ij}	f_i	δ_{ij}
p	CLK	V_{dd}	1	4	2		
n	$R_{ij}(A_1..A_N)$	1	$R_{ij}(2..\overline{N+1})$	4	2		
n	$R_{ij+1}(A_1..A_N)$	$R_{ij+1}(S_{ik}..S_{ij-1})$	$R_{ij}(2..\overline{N+1})$	4	2		
.	.	.	.	.	.		
.	.	.	.	.	.		
n	CLK	$max[n_{ij}^{1,2}\ldots n_{ij+N}^{1,2}]$	gnd	4	2	f_i	δ_{ij}

[a] T_{ij}: Transistor type, S_{ij}: Transistor gate node, W_{ij}: Transistor width in μm, L_{ij}: Length in μm, N: Number of Inputs, R_{ij}: Uniform Random Selector

2.2 Genetic Topological Synthesis

2.2.1 Representation and initial population

Gene representation in the Allele: Representation of the genes for evolution is a critical choice to keep the circuit topology valid and provide faster convergence. Eq. (2.1) shows the chosen representation. The representation follows in line with the genetic ideology and the IRSIM simulator codes as described in Appendix.A. A thorough understanding of the material Appendix.A is highly recommended before proceeding. Each transistor is represented as a triplet $\langle S_j, n_j^1, n_j^2 \rangle$, where S_j stands for the node to which the input signal and the gate of the j^{th} transistor are connected. n_j^1 and n_j^2 are the nodes to which the source and drain of the j^{th} transistor are connected Section.A.2 in Appendix.A. A *chromosome* is a sequence of such triplets which are equal in number to the input signals determined from the truth table. These form netlist inputs to the *IRSIM* simulator for fitness evaluation and selection. A subsequence of these signal triplets can be used for mapping the inputs (or transistor gates) to one of the variables $A_1, A_2 \ldots A_N$. In other words, the S_j values of the triplets in the subsequence shall be a one-one mapping from among A_i's, $0 \leq i \leq N$.

Gene Structure: The internal gene structure of a chromosome is shown in Table. 2.1. The representation using a *Perl parser* is designed to conform to the spice netlist. This can be directly used for evaluation, using the *IRSIM* switch level simulator.

$$\left[S_j\ n_j^1\ n_j^2 \middle| S_{j+1}\ n_{j+1}^1\ n_{j+1}^2 \middle| \ldots\ldots \middle| S_{j+N-1}\ n_{j+N-1}^1\ n_{j+N-1}^2\right] \tag{2.1}$$

The methodology for the generation of the **initial population** is shown in Fig. 2.2 and

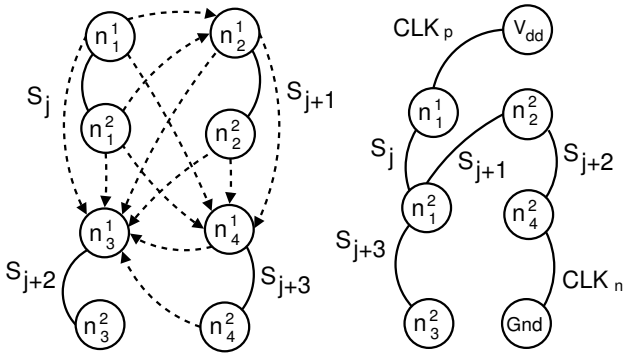


Fig. 2.2: Initial population generation with the corresponding transistor topology and SFG for the netlist. The dotted lines contribute to the random node choice for the next chromosome. The final obtained netlist is functionally valid.

Table 2.2: A model .sim file for the circuit example in Fig.2.3

line	*text*					
1	*units: 100 tech: cmos*					
2	**type**	**gate**	**source**	**drain**	**length**	**width**
3	p	ϕ	5	1	L_{51}	W_{51}
4	n	ϕ	4	0	L_{40}	W_{40}
5						
6	n	V_A	1	2	L_{12}	W_{12}
7	n	V_B	2	3	L_{23}	W_{23}
8	n	V_C	3	4	L_{34}	W_{34}
9	n	V_D	1	4	L_{14}	W_{14}

in Table. 2.1. The first node, n_0^1, for the i^{th} chromosome in the population is chosen to be 1 to which the *pMOS*, *CLK* signal is connected. The second node is randomly chosen between $(N{+}1)$ and 2. These form the source and the drain for the first *nMOS* transistor. For the second and subsequent j^{th} device, the first node is chosen randomly from among the previously chosen nodes, $\langle n_{ik}^1, n_{ik}^2 \rangle$, where $0 \leq k \leq j$ and the second node is chosen randomly between $(N{+}1)$ and 2. This ensures circuit connectivity and appropriate intermixing of the nodes to provide a broad outreach in the search space. A matrix of P chromosomes is chosen this way to form the valid initial population. The circuit interconnection and topological configuration needs to remain in tact. We summarize the rules followed for such a limit by the Example circuit shown in Fig.2.3. The figure shows an example of a dynamic CMOS circuit under simulation. The circuit stands represents the

function $ABC + D$. The genetic algorithm described in this chapter evolves a random initial interconnection towards this topology. Consider the IRSIM model file for the circuit shown in Table.2.2. The gate, source and drain of each transistor is written as explained in Appendix.A. The length and the widths of the transistors are decided based on follow up performance characterization. The interconnectivity is what is modeled in the .sim file. The drain and the source are interchangeable terminals in the irsim switch level simulator.

From Table.2.2, we model the initial population assuming each transistor as an element in the chromosome. The modeling of the circuit into a genotype is shown in Fig.2.4. The circuit connectivity is captured in this compact genetic representation. Simple obser-

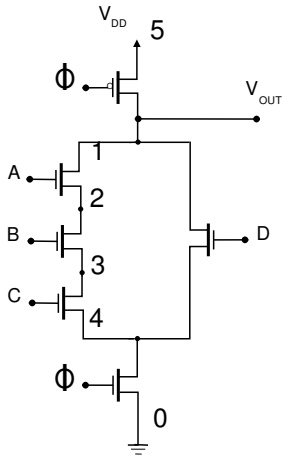


Fig. 2.3: Dynamic CMOS simulation circuit example.

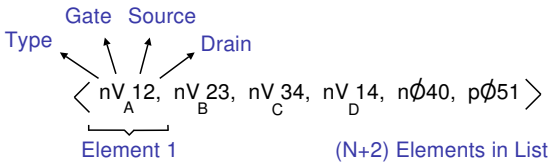


Fig. 2.4: Circuit to chromosome translation.

vations from the chromosome model yield that for the given circuit with $N = 4$ inputs, the total circuit nodes T_n is 12, the total intermediate circuit nodes I_n is 4. Another premise of generalization would mean that for a circuit with N inputs, T_n can assume a maximum value of $(2N + 5)$ and I_n can go upto $(N + 1)$. After modeling the chromosome as described above, we proceed with the variation operation.

2.2.2 Variation: Crossover and Mutation

Algorithm 1 CROSSOVER(n)

Require: An integer $0 \leq n_{ij}^{1,2} \leq N+1$
Ensure: Network connectivity and $n_{0j}^{1} = 0$.

for all i such that $0 \leq i < P$ **do**
  for all j such that $0 \leq j < \delta_j$ **do**
    $S_{ij}^{'}$ = $R(S_{ij} \ldots S_{ij+N-1})$
    $n_{ij}^{1,2} = \xi(n_{ij}^{1,2})$
    if $j = 0$ **then**
      return $n_{ij}^{1} = 1$
    end if
  end for
  for all k such that $\delta_j \leq k < N$ **do**
    $S_{ij}^{'}$ = $R(S_{ij+1} \ldots S_{ij+1+N-1})$
    $n_{ij}^{1,2} = R(n_{ij}^{1,2} \ldots n_{ij+\delta_j}^{1,2})$
    n_{ij}^{2} = $\xi(n_{ij+1}^{2})$
  end for
end for

The initial population generated as described in Sec. 2.2.1 is used to proceed with the evolutionary variation. **The crossover** operator algorithm is shown in Algorithm. 1. For the j^{th} chromosome in a population size of P, δ_j transistor selections from the j^{th} chromosome and $(N\text{-}\delta_j)$ selections from the $(j{+}1)^{th}$ chromosome are used. The copy operator, ξ is used to copy the corresponding nodes for the δ_j devices. The first node is then set to 1 as in Sec. 2.2.1 to ensure circuit connectivity in the *phenotype*. For the rest of the N-δ_j transistors, the first node is chosen randomly from among the previous $2\delta_j$ nodes. The second node for all the genes in the chromosome other than the first one is copied (ξ) from the $(j{+}1)^{th}$ chromosome. The select-scalar, δ_{ij} is calculated as shown in Eq. (2.2).

$$\delta_{ij} = log_2 l - \left\lceil \frac{f_i}{log_2 l} \right\rceil \tag{2.2}$$

where, l is the length of the truth table input by the user and f_i is the fitness value for the i_{th} *chromosome* set in the *genotype* matrix. **The mutation** operator works similar to the initial population generation described in Sec. 2.2.1. These randomly mutated and the varied chromosomes $(P_{co/m})$ total to P-1. These are appended at the end of the chromosome matrix with P elements to obtain a total of $2P$-1 chromosomes in the next generation. The operations described in this section follow certain intrinsic rules which

can be summarized as follows:

- Output consistency
  - The first I_n for the first element is always a 1
  - The T_n list contains at least two nodes numbered 1
- Network interconnection
  - I_n's for an element i contain at least one of the I_n's from elements 1 to (i-1)
  - The highest I_n is limited to $(N+1)$ and T_n to $(N+2)$
- Stimulus clock
  - Element $(N+1)$ is connected between nodes I_n^{max} and 0
  - Element $(N+2)$ is connected between nodes $(I_n^{max}+1)$ and 1

2.2.3 Selection and termination:

Algorithm 2 SELECTION(n)

Require: An integer $0 \leq n_{ij}^{1,2} \leq N+1$.
Ensure: P: Population; G: Generation

1: **for all** g such that $1 \leq g < G$ **do**
2: $P_g = P_i + P_{co/m}$
3: $f_g = \varepsilon(P_g)$ {irsim $*.proc\ g_j.sim\ g_j.cmd$}
4: $P_{ng} = \Lambda[P_g(1 \ldots P)]$ {sort and select P}
5: **if** $g = 0.1G$ **then**
6: $P_g^{'} = P_i + P_m$ {variation mutation}
7: $P_g^{'} = \Lambda[P_g(1 \ldots P)]$ {sort select}
8: **end if**
9: **end for**

With the variation generated $P_{co/m}$=P-1 chromosomes appended at the end of the initial population, P_i=P, a new *genotype* matrix is created. For all of these, the **fitness** (ε) is evaluated. This is an $\oplus$ (*XOR*) operator with the input truth table supplied by the user. This compares the deviation of the current chromosome from the required truth table. The ε is evaluated using the *IRSIM* simulator. From the resulting $2P$-1 chromosomes generated, the sorting operator Λ, sorts the chromosomes according to their fitness values and the top P of them are selected to move on to the next generation. This way, only the best characteristics of a generation are passed on to the next generation. After about 10% of the total generation size G, mutation is introduced to increase the generation gap and introduce diversity in the current population. When the best fit chromosome is found from

the sorted matrix, the algorithm is **terminated**. The selection operation is algorithmically described in Algorithm. 2. An elitist model is also used in the design.

2.3 Experimental Results

Test case simulations for the proposed design flow were run using an embedded switch level *IRSIM* simulator [Appendix.A]. Fig. 2.5 shows the evolved fitness values over the iterations numbers for a test input truth table whose boolean functions are shown in Fig. 2.6 and Fig. 2.7. The netlist obtained from the genetic evolution using the operations described in Sec. 2.2 provides a quick way of custom generating library cells. The fitness value is evaluated by summing up the exclusive-OR vector derived using the input (required) truth table from the user and the evolved truth table response for the stimulus vector obtained from *IRSIM*. *When the fitness value reaches* 0*, the two responses match and the netlist obtained from the genetic method is valid.* Fig. 2.6 and Fig. 2.7 show

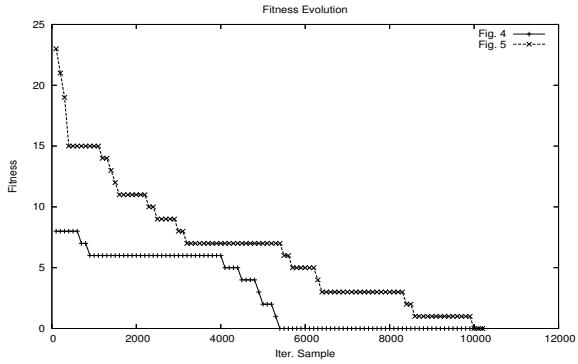


Fig. 2.5: The fitness evolution samples over intermediate generations, plotted for cases in Fig.2.6 and Fig.2.7 with four and five inputs respectively.

evolved netlist translations to dynamic CMOS circuit schematics. Our genetic methodology guarantees a convergence to a point which gives the simplified netlist incorporating all the boolean simplification rules. This means that the final netlist obtained after evolution has the minimum transistor count. The netlist could be used directly as a library cell instead of simplifying the logic at the gate level and then translating it to the transistor netlist which may still miss out a few optimizations in the transistor topologies. This transistor level simplification can be carried ahead into layout level optimization. Better transistor sizing can also be obtained by using the stochastic methods described in [Ro-

genmoser *et al.* (1996)]. This would be the next step in optimization to obtain the best layout and size for*characterizing a cell.* Table. 2.3 shows the convergence rates and pa-

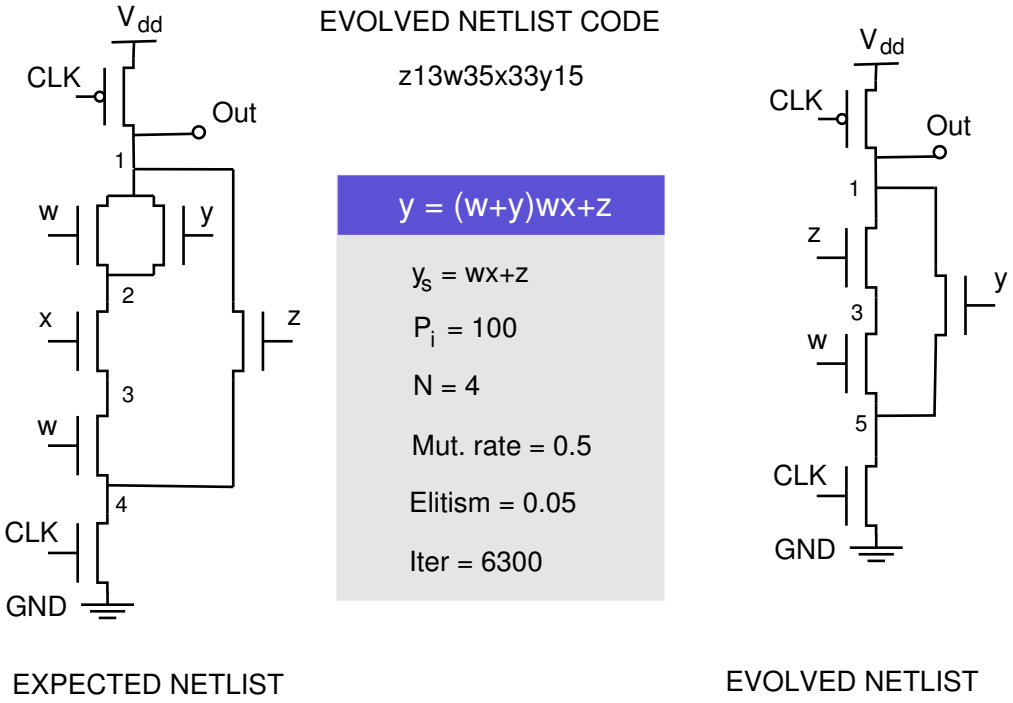


Fig. 2.6: Evolved SPICE netlist for the test case with N=4, P=100 and G=23

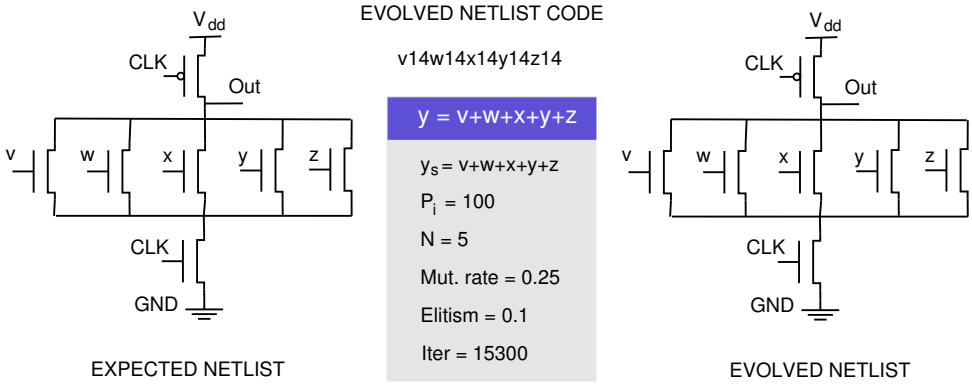


Fig. 2.7: Evolved SPICE netlist for the test case with N=5, P=100 and G=153

rameters used for the experiments conducted on a few test cases for the proposed genetic methodology. It is evident that the convergence is practical, albeit stochastic. Our genetic methodology could easily be incorporated into any digital circuit design flow to enable the creation of *dynamic library cells* on the fly. Fig. 2.1 shows one such methodology where the genetic netlist creation forms the fundamental step in the process of custom and semi-custom circuit design.

A neat example of the technique described in this chapter is shown in Fig.2.8, Fig.2.9, Fig.2.10 and the final netlist is rewritten in Fig.2.11 The boolean function represented by

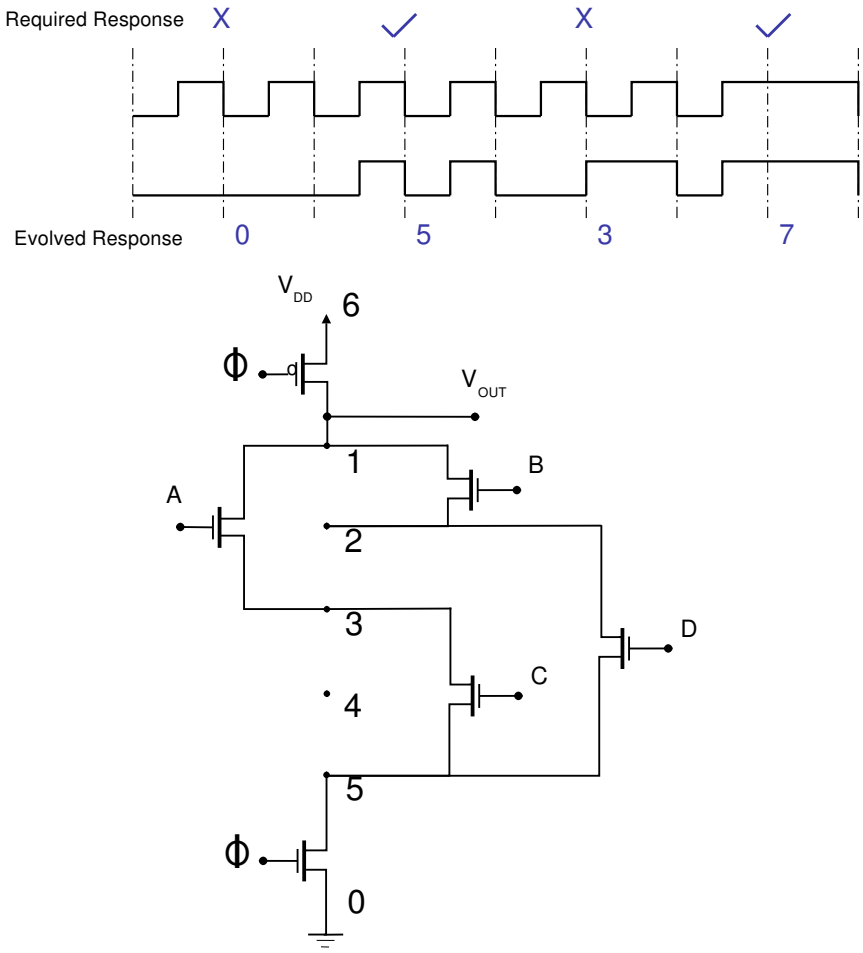


Fig. 2.8: Evolving transistor netlist: function AC+BD

the initial random vector is $AC + BD$ This random vector is generated at random and is represented as $\langle nV_A13, nV_B12, nV_C35, nV_D25, n\phi50, n\phi61\rangle$. A crossover is effected between nodes 3 and 2 of signals V_C and V_D to keep the circuit topologies intact pertaining to the rules explained in Sec.2.2.2. The hexadecimal truth table equivalent of the function represented by Fig.2.8 is $h'0357$. After the heuristic crossover, we obtain a new vector $\langle nV_A13, nV_B12, nV_C25, nV_D35, n\phi50, n\phi61\rangle$. The function represented by this topology is $h'0357$. Both of these functions have a fitness value of 12. These are chosen among a set of initial random populations. A crossover among these would result in the final netlist shown in Fig.2.10. This is the final netlist which has a fitness value of 16 - meaning that the exact truth table requirement has been achieved. The final netlist has the hex function

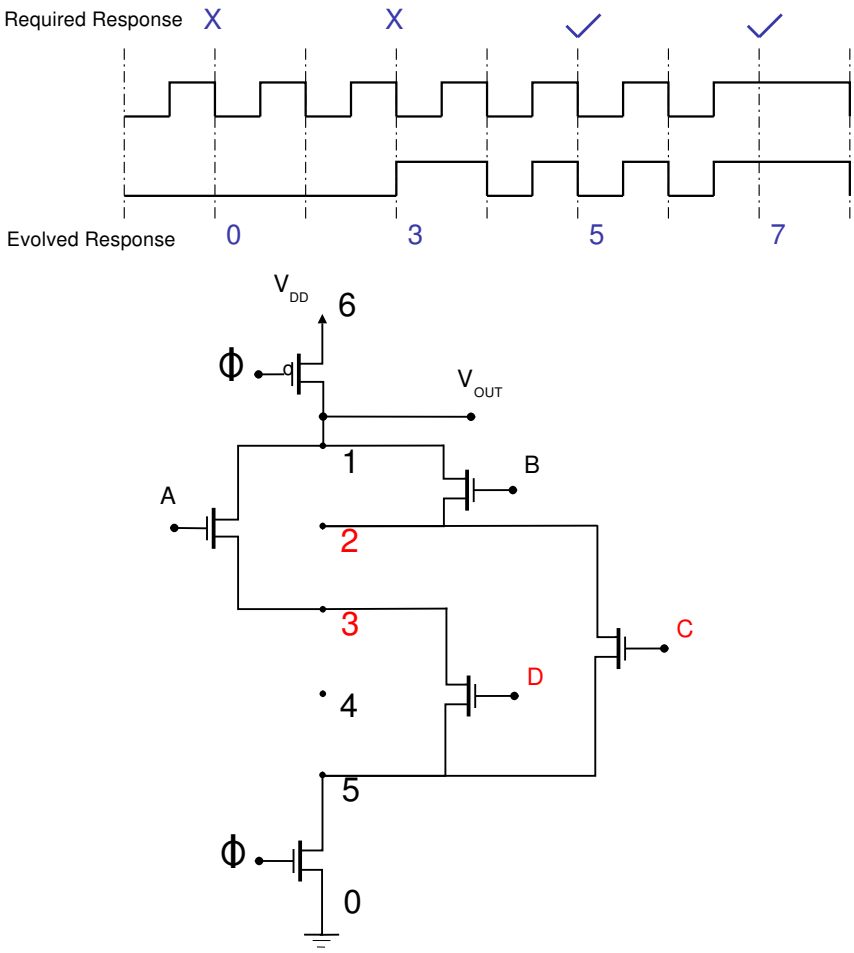


Fig. 2.9: Evolving transistor netlist: function AD+BC

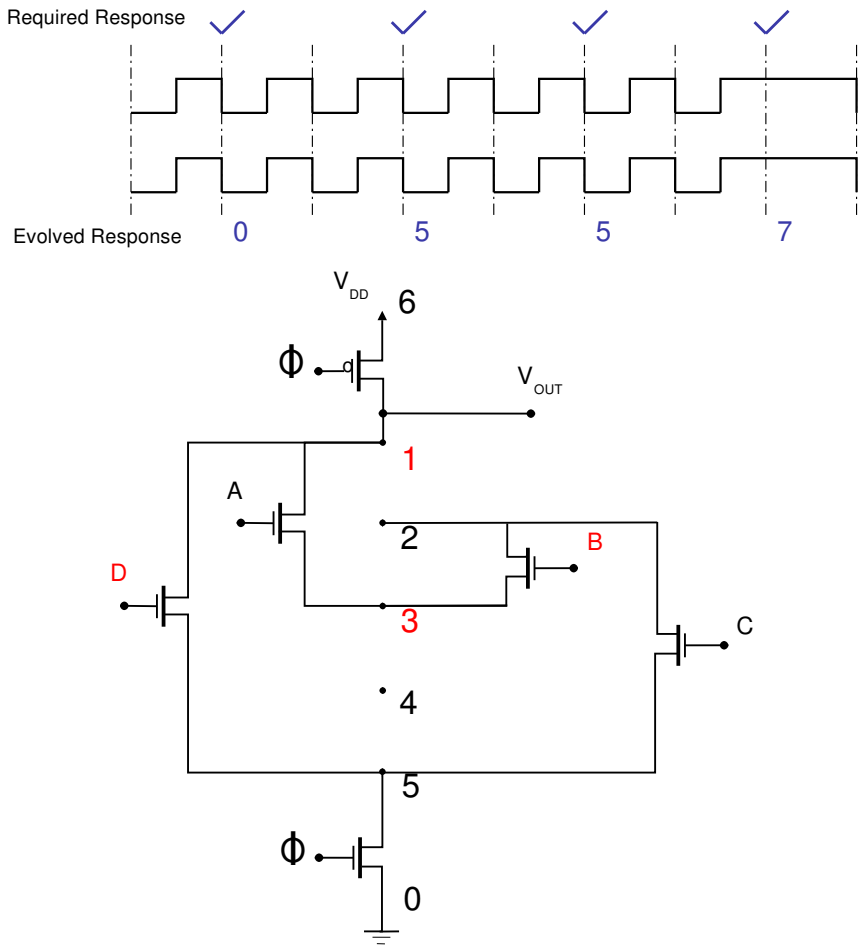


Fig. 2.10: Evolving transistor netlist for example function ABC+D

$h'0557$ and the boolean equation $ABC + D$. The netlist in Fig.2.10 is written simplified in Fig.2.11 which aptly reflects the required netlist. This is modeled as the chromosome - $\langle nV_A13, nV_B32, nV_C25, nV_D15, n\phi50, n\phi61\rangle$.

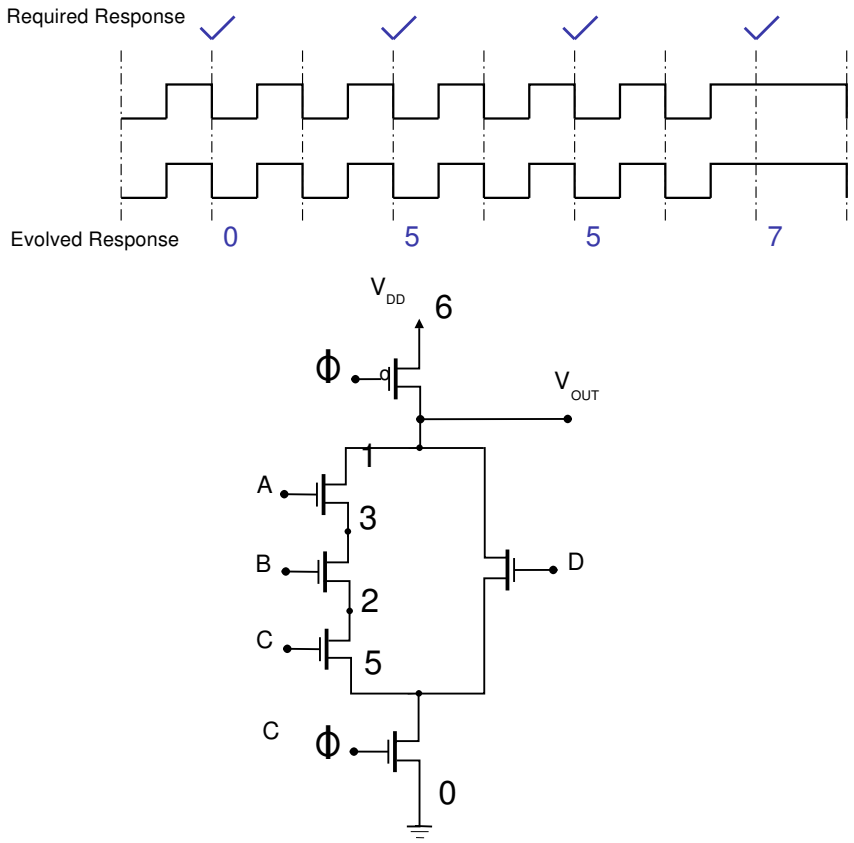


Fig. 2.11: Final evolved transistor netlist - ABC+D

2.4 Future Work

From the design methodology and the results described in Sec. 2.2 and 2.3, it is evident that our methods are scalable to incorporate four and five input truth tables with extremely practical speeds of convergence. It is important to note that the search space complexity in the problem is of the order 2^{2^n}, which is enormous even for a four or five input truth table. For higher order functions, convergence rates become a serious issue. However, the *shannon's decomposition*, laid out in [Woods and Casinovi (1996)] and the equation

below, could be effectively used to exploit some parallelism.

$$f(x_n, x_{n-1}, \ldots, x_0) = f(x_n, x_{n-1}, \ldots, 1)x_0 + f(x_n, x_{n-1}, \ldots, 0)\overline{x_0}$$

The decomposed functional topology with fewer inputs can easily be evolved in parallel using our genetic methods and the whole truth table can be realized by combining the decomposed parts using the dynamic CMOS logic. Scalable functionality can hence be incorporated into the proposed method at various levels. Experiments with the scalable models could be the future direction for the current design methodology. This would make it feasible to work with arbitrary truth table inputs. This automatic transistor level optimization of the topology starting from truth tables, completely avoiding the Boolean simplification approach is the first known methodology to the best of our knowledge. The transistor sizing [Rogenmoser *et al.* (1996)] and placement optimization [Lefebvre and Marple (1997)], [Ho *et al.* (1997)] and [Bahuman *et al.* (2002)] can easily be incorporated in the current model for the complete genetic custom design flow. The sizing of the netlist obtained using our methods can be done as described in any of these works in the literature.

2.5 Summary

The results and designs set out in this chapter describe the implementation techniques for the genetic evolution of optimized transistor netlists starting from truth table descriptions. This low level optimization methodology can be effectively used to generate custom library cells on the fly. Although demonstrated for the dynamic CMOS and the domino logic case, the design methodology can generically be extended to include static CMOS and other logic styles using the principle of duality. Appropriate heuristics in the evolution of the transistor netlist starting from truth table descriptions are shown to perform practically. These include Boolean simplifications and other optimizations using genetic operators of crossover and mutation. The methodology bypasses the normal way of Sum of Product (SoP) or Product of Sum (PoS) formulation of a function description followed

by gate level optimization and transistor netlist generation. The netlist obtained directly from the functional specification can be used for layout preceded by optimizations for sizing and placement.

Table 2.3: Exemplary Circuit results for test truth table inputs with the mutation and elitism rate over multiple generations - ER: Elitism Rate, MR: Mutation Rate.

BOOLEAN EXPR	ER	MR	N	ITER	FINAL NET
$Y = (x+y)(yz+x)+xy$ $Y_s = xz$	0.1	0.1	3	1300	$x12, y22, z234$
$Y = x\overline{yz} + \overline{x}y\overline{z} + xy\overline{z}$ $+\overline{xy}z + x\overline{y}z + \overline{x}yz$ $Y_s = x+y+z$	0.1	0.1	3	2300	$x14, y14, z14$
$Y = wx\overline{yz} + wx\overline{y}z + wxyz$ $+wxy\overline{z} + \overline{wx}yz$ $+\overline{w}xyz + w\overline{x}yz$ $Y_s = wx+yz$	0.1	0.1	4	6600	$w14, x14, y12, z25$
$Y = (w+y)wx+z$ $Y_s = wx+z$	0.05	0.5	4	6300	$d13, a35, b33, c15$
$Y = v+w+x+y+z$ $Y_s = v+w+x+y+z$	0.1	0.25	5	15300	$v14, w14, x14, y14, z14$

Researchers in the literature have looked at using genetic algorithms for placement driven/ routing driven transistor netlist generation. These are however slower methods for convergence. We provide an intermediate layer of configurational evolution which is rapid and highly scalable. in this chapter, we also provide insights into a new methodology for a typical VLSI CAD flow which could be altered to include our genetic methods and we name it the gateless VLSI custom design flow.

CHAPTER 3

VECTOR PARTITIONING AND GENETIC SEARCH METHODS FOR PDPE IN DIGITAL CIRCUITS

3.1 Prelude

Performance efficiency and operative reliability of hardware designs relies heavily on its power profiling and guard-banding for heat dissipation. An early power estimate for the circuit would mean a reduction in costly back annotations during a typical VLSI CAD design cycle. The power estimation problem has alternative apellations - Peak Dynamic Power Estimation (PDPE) and the power virus. PDPE techniques at the various stages of the design flow can be broadly classified into simulation based techniques and non-simulation based techniques. Another way to look at things is based on the methodology of power estimation - the PDPE can be a static or dynamic power estimate. Static estimation techniques exploit probabilistic analyses and circuit properties using graph models whereas simulations are run in the dynamic estimation methodology. Fig.3.1 [Najeeb *et al.* (2007)].

3.1.1 Early estimation techniques

Non-simulation based estimation techniques are often quite rapid in producing PDPE estimates. However the error margin in these estimations are high as internal signal correlations cannot be accurately modeled using a theoretical modeling tool. Simulation based techniques are accurate estimation methods as they involve the entire input signal states. Signal value assignments to circuit nodes can be evaluated using external simulators. These can handle glitches, signal correlations and delays more accurately. The downside to this however is that the estimation latencies are high which is often quite high. But given the accurate estimations we get, this might be considered a worthy one

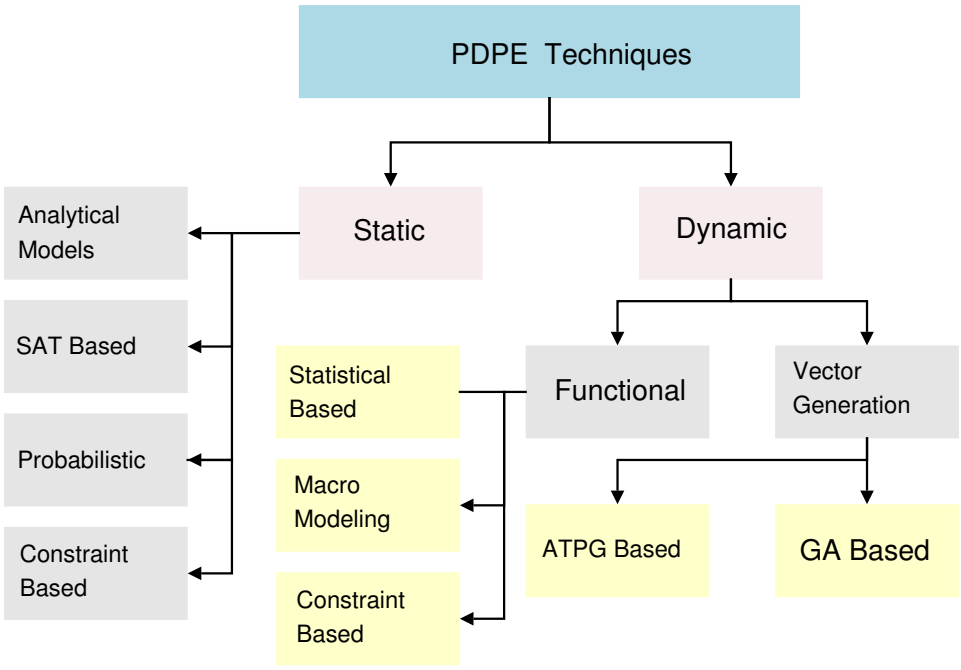


Fig. 3.1: PDPE Techniques in digital VLSI design

time investment. The power estimates strongly depend on various other circuit parameters and technology based factors too. Many of these are however not readily available nor are precisely characterized at the time of power estimation. Some of these factors are the actual input sequences that would be applied to the circuit on field - delay of the circuit elements, logic function, logic style, spatio-temporal correlations and circuit structure [Pedram (1996)].

Delay models for the circuit under test is another vital criterion in PDPE techniques. There are three popular models - the unit delay model, the zero delay model and the variable delay model. As the names signify, the delay models assign delays to individual gates in the circuit and run the simulations. The sensitivity of PDPE to gate delays has been extensively studied and it has been shown that the delay models have a strong bearing on the final PDPE [Hsiao *et al.* (1997*a*)]. Variable delay models are the most accurate ones in estimation as they are based on a realistic characterization of gates in the circuit from fabrication data. These can account for circuit glitches and other variations. In this work we focus on two dynamic estimation methodologies using a variable delay model. The techniques exploit an external simulator to yield accurate results for PDPE and are based on a genetic search and partitioning methodology. These fall under the category of *vector generation dynamic techniques* for PDPE.

3.2 Handling the PDPE problem

With the advent of portable and high-density microelectronic devices excessive power dissipation is a problem of extreme propensity. The continuing decrease in feature size, increase in chip density and clock frequency in recent years have invigorated concerns about excessive power dissipation in modern VLSI chips. High power dissipation may lead to drops in performance or in extreme cases cause burnout and damage to circuits. Peak power dissipation of a circuit determines the thermal and electrical limits of components and system packaging requirements [Pedram (1996)]. Faster Times-To-Market (TTM) and expensive redesign cycles necessitate accurate and efficient power estimation at an early design phase. The peak power consumption corresponds to the highest switching activity generated in the circuit during one clock cycle. This was also explained in Chap.1 and presented as the Peak Dynamic Power Estimation (PDPE) problem in digital circuits. The energy per clock cycle (peak power) in the combinational portion of a circuit can be computed as:

$$P_R = \frac{V_{dd}^2}{2 \times frequency} \times \sum_{\forall\, g} [toggles(g) \times C(g)] \tag{3.1}$$

where the summation is performed over all gates g. $toggles(g)$ is the number of times gate g has switched from 0 to 1 or vice versa within a given clock cycle, C_g is the output capacitance of gate g and V_{dd} is the supply voltage. This work assumes that the output capacitance for each gate is equal to the number of fanouts. Therefore, the total switching activity (toggles) is the parameter that needs to be maximized for maximum P_R among all possible input vector pairs. Given that the circuit has n primary inputs, there are 4^n possible input vector pairs to be considered for an exhaustive search. Thus the search space for the vector pairs is huge even for reasonably large values of n.

As described in Sec.3.1, searching in this huge multi modal search space poses problems in avoiding local maxima and in getting out of *frozen extrema*, where fixed search algorithms tend to get indefinitely stuck [Wenzel and Hamacher (1999)]. Stochastic search models such as those based on Genetic Algorithm (GA)s [Hsiao *et al.* (1997*a*)], [Hsiao *et al.* (1997*b*)], [Hsiao *et al.* (2000)] provide very tight lower bounds on peak power. This

clearly outperforms random search methods. In [Hsiao (1999)], Hsiao presents four genetic search methods based on node, path, cone and distance heuristics. These methods use GA in an *underterministic* way. The initial population uses *random vectors* and no heuristics are used to generate quick approximate solutions to act as preludes to future evolution. Uniform crossover (which is equivalent to large scale mutation) and tournament selection employed in [Hsiao (1999)] adds more randomness to the search process. Also the *Genetic Spot Optimization* used in [Hsiao (1999)] has an abstract definition of *expansion* with no construed relationship in circuit models.

3.3 Previous Work

Besides the genetic methods described in Sec.3.2 several non-stochastic approaches have also been proposed in the literature to estimate the maximum power consumption in CMOS circuits [Hsiao *et al.* (2000)], [Devadas *et al.* (1992)], [Wang and Roy (2000)] and [Chou *et al.* (1994)]. In [Devadas *et al.* (1992)], the problem of worst-case power computation was transformed to a weighted max-satisfiability problem. Its limitations were constrained scalability and no provision for delay incorporation. In [Kriplani (1994)] and [Kriplani *et al.* (1993)], switching time windows were used with partial input enumerations for correlation resolution. Symbolic transition counts was introduced in [Manne *et al.* (1995)]. Automatic Test Generation (ATG) based techniques have also been proposed in the literature [Wang *et al.* (1996)]. Inherently, they are limited by their lack of adaptation to handle delay parameters. In [Devadas *et al.* (1992)] and [Wang and Roy (2000)] a test generation strategy was devised for finding test patterns that would produce the maximum power. [Devadas *et al.* (1992)], takes exponential times with respect to the number of levels in the circuit and hence lacks scalability. Static timing analysis was used in [Wang and Roy (2000)] to find the time instants at which the gates can switch and this information was used to maximize energy dissipation in a clock cycle. However, this approach requires complete and specific information about the circuit and has complexity proportional to the number of gates and fan-in. Hence, it would take a large computation time to create or valid sequence.

3.3.1 Contribution

This chapter proposes two methodologies for power virus generation for combinational circuits which could be easily extended for sequential circuits. The first method is named the *vector partitioning method* and is based on dividing the power virus search space into subsets enabling a fast parallelized search. The second method incorporates the use of favorable pattern matching among sample power virus vector pairs leading to newer *chromosomes* for future iterations in a GA with controlled $crossover$. Both of these methods incorporate a variable delay model which is a distinguishing feature from previous works.

The Vector Partitioning Method This is based on the divide and conquer algorithm [T H Cormen and Stein (2005)]. It solves a closely related subproblem of finding the Power Virus sub-vector ($PV_{sv}^{i},\ i = 0 \ldots m$) by partitioning the input vector into sub-vectors. Combining the PV_{sv}^{i}'s after a quicker search leads us to the required Power Virus Vector (PV). A commonly argued disadvantage of this divide-and-conquer method is its slow recursion. However, a demonstrated tight lower bound on power estimates on ISCAS'85 benchmark circuits offsets this latency disadvantage to make this method considerably accurate. The convergence rate of this method can be enhanced by a smart choice of the partitions. This involves the exploitation of the cone-based partitioning method proposed by [Saucier *et al.* (1993)]. Independent/ fanout free regions can be explored and the inputs to those can be optimized using the vector partitioning method. The resulting vectors can be ported forward to the inputs to ultimately lead to the Power Virus vector.

The Genetic Search Method This works with an Initial Population vector quadruple (IP_vq). A pair each among the IP_vq is considered as a PV for the given circuit. The PV pairs in the IP_vq are chosen so as to maximize the Hamming distance among the two vector pairs. This forms the initial choice of the genetic population. The Hamming distance is the number of differing bit values between the two vectors. This provides a deterministic IP base. From the two PV pairs, correlating bit patterns are chosen to move over to future generations providing a *heuristic variation* mechanism. The PV and hence the peak power estimation using this method is demonstrated to perform much better than

the best reported techniques in the literature.

Delay Models Switching activity of a given node in a circuit is not only dependent on the output capacitance of the node, but also heavily on the gate delays in the circuit, since multiple switching events can result due to uneven circuit delay paths. Glitches and hazards are not taken into account in a zero-delay framework [Najeeb *et al.* (2007)] and the power dissipation measures are off greatly from the actual powers [Hsiao *et al.* (1997*a*)]. The unit delay model offsets these inaccuracies to an extent by uniform delay modeling of the gate levels. This provides a better inclusion of power fluctuations [Hsiao (1999)]. A variable delay model is incorporable in our simulator design to account for more accurate estimation of the peak power.

3.4 Algorithms for Power Virus Generation

Problem Formulation: As described in Sec.3.2, given a circuit, consider two input vectors $V_1 :< x_1, x_2, \ldots x_n >$ and $V_2 :< y_1, y_2, \ldots y_n >$ such that there is a maximum power dissipation when the circuit undergoes a transition from the initial state which is got by applying V_1 as the input to the circuit, to the final state, which is got by applying V_2 in sequence. The two vector pairs V_1 and V_2 form a Power Virus Pair. The problem of finding this pair is called the power virus problem. This is also known as *Peak Power Estimation in VLSI circuits*. In this work we propose to find V_1 and V_2 which give maximum power dissipation in the circuit.

Each gate in a digital circuit is modeled as a node. The type of the gate determines the node characteristic and the i/o connections of the gate are translated to edges. Hence, wires contribute to edge weights and gates to node weights. The number of toggles on every edge determines the switching activity in the circuit and the number of fanins and fanouts weight P_R according to Eq.(3.1).

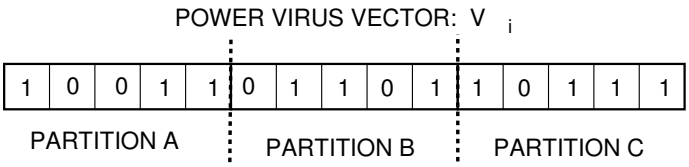


Fig. 3.2: Vector partitioning the PV

3.4.1 Vector Partitioning

The first method we demonstrate for approaching the power virus problem is that of vector partitioning. Fig.3.2 shows an exemplary partitioned input vector. The primary input power virus vector pV is partitioned into $m = 3$ sub-vectors named as partitions A, B and C respectively. The optimization algorithm is then run on the individual sub-vectors and finally combined to give the power virus vector PV. Algorithm. 3 describes a pseudo-code for vector partitioning. The power virus vector pair consists of a pair of input sequences to be applied to a given circuit successively. We model, each of the the input sequence as a bit vector. Consider such a sequence of word length n corresponding to the n primary inputs of the circuit. As explained earlier, this input vector is partitioned into m subsequences, each of length $\frac{n}{m}$. The choice of m is a separate issue which would be discussed in the following paragraphs. Recursive search through the entire initial search space of order n would have a search complexity of order $O(2^n)$. Due to the partitioning, inherent parallelism in modern computation machines can be exploited aptly to search recursively in a reduced search space of order $O(m \times 2^{n/m})$. This leads to a faster convergence to the PV and is also demonstrated in Table.3.2 in Sec.3.5 to be better in peak power determination compared to random search methods. This method has a quicker convergence than the complete random search.

The choice of m, the number of partitions of the virus vector is an important problem. We propose to use the cone based partitioning method demonstrated in [Saucier *et al.* (1993)]. The method is outlined in Algorithm. 3. The cone based partitioning algorithm is described in Appendix C. As shown in Algorithm. 3, the cone based partitioning method is used to determine the clusters (Cluster$_i$). Each of the clusters determined form the sub-sequence (PV$_{sv}$) for which the optimization algorithm needs to be run locally.

Two random input vector pairs of length equal to that of Cluster$_i$ are chosen ((v_1,v_2)

Algorithm 3 VECTORPARTITIONING

Require: CROOT $\leftarrow$ I/P $\forall$ nodes$_{pri-o/p}$
// # Nodes $\Leftrightarrow$ # Cones

repeat
  scan: N_d = CROOT[i] $\longmapsto$ P(N_d)
  LL(N_d) = $\bigcup_{i=0}^{\#P(N_d)}$LL(P(N_d)$_i$) //Label List
  $i++$
until CROOT $\neq \phi$
j = m //Number of Clusters (m)
————————————LOCAL SEARCH————————————
while (j > 0) **do**
  $\rightarrow$ choose PV_{sv}^{i} = Cluster$_i$
  INIT :: choose rand_vec_pair :: (v_1,v_2),(v_3,v_4)
  $\rightarrow$ simulate: inp : (v_1,v_2): Toggl$_1$
  $\rightarrow$ simulate: inp : (v_3,v_4)/PV_{sv}^{j}: Toggl$_2$
  HD$_1$ = HD$_2$ = 0 //Vector Hamming Distance = 0
  for all (k, $1 \leq k < size(v_1)$) **do**
    if ($v_1[k] \neq v_2[k]$) **then**
      HD$_1$ ++
    end if
    if ($v_3[l] \neq v_4[l]$) **then**
      HD$_2$ ++
    end if
  end for
  if (HD$_1$ $\sim$ HD$_2$ > HD$_{Thresh}$) **AND** (Toggl$_1$) > (Toggl$_2$) **then**
    PV_{sv}^{j} = (v_3,v_4)
  else
    PV_{sv}^{j} = (v_1,v_2)
  end if
  j–
end while
PV $\asymp \bigcup_{i=0}^{m}$ PV$_{sv}$

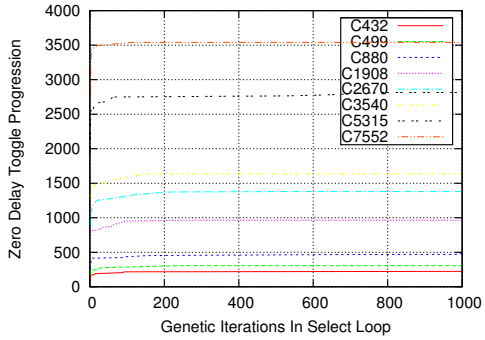


Fig. 3.3: Modified Genetic Method zero delay toggle progression

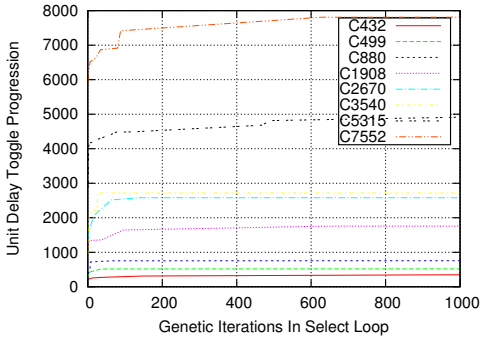


Fig. 3.4: Modified Genetic Method unit delay toggle progression

and (v_3,v_4)) in the first iteration. The circuit is simulated using these signals. The values of other signals are held constant at 0 during the local optimization process. This may not have an issue of dependance on the other inputs as the clusters determined using the cone based method are rather independant. The resulting toggles based on the application of the above two vector pairs are recorded as Toggl_1 and Toggl_2 respectively. An hamming distance between the two vector pairs is determined. We intend to choose a pair with larger hamming distance among the vectors. This is expected to result in maximum toggles at the primary inputs. This is chosen in combination with the maximum toggles in the circuit determined earlier as Toggl_1 and Toggl_2. A threshold crossing would mean that the current vector pair is the one causing the maximum local toggles in the cone cluster and is chosen as the PV_{sv} for the i^{th} cluster. This estimation is iterated over a few times and the vector pair causing the maximum local toggles is chosen. In iteration numbers greater than or equal to two, only one random vector pair is generated and the PV_{sv} is used as the first vector pair. Finally, the sub-vector components for the clusters (PV^i_{sv})'s are clubbed to give the required power virus vector (PV) for the entire circuit.

3.4.2 Genetic Pattern Matching

Furthering the exploration of the Peak Dynamic Power Estimation (PDPE) space, an intelligent search mechanism was found necessary. Genetic Algorithms were explored and the details of a modified GA are presented below. The heuristics are problem optimized. A good Initial Population (IP) generation process is the first step in a GA. We use a simple hamming distance based selection of $genotypes$. $Genes$ pairs, which are simply bit

strings corresponding to the probable PV vector, are chosen based on larger hamming distances among them. For this purpose, a random initial vector V_1 is chosen and its bits are subjected to an inversion operation. Then with an adjustable probability of about 0.1, its bits are randomly flipped to obtain V_2, the second PV vector, so that the hamming distance, $H = V_1 \oplus V_2$, is still large. Here $\oplus$ is the XOR operator. We do not use a simple inversion of V_1 to obtain V_2 because this heuristic may lead to biased IP isolation and may take large times times to converge. This is because the PV vectors do not always have the maximum hamming distance [Hsiao (1999)]. Two other $< V_1^{'}, V_2^{'} >$ pairs are then chosen in a similar way to produce a PV quadruple. These four vectors form the IP for the GA.

The genetic search methodology proposed in this chapter uses a pattern matching heuristic for $selection$. Following a deterministic IP generation, the circuit is simulated for successive applications of $< V_1, V_2 >$ and $< V_1^{'}, V_2^{'} >$. If the peak power estimate from these vector pairs improves the best recorded one from the past iterations of the GA, the best PV is updated with the current one. At this point, there is a $crossover$ among the PV pairs. A pattern matching is made among the two vector pairs $< V_1, V_2 >$ and $< V_1^{'}, V_2^{'} >$. Common bit patterns among the two vector pairs are heuristically understood to contribute to the peak power dissipation. Hence, these bit patterns are carried over to the next generation $genes$. The rest of the $gene$ bits in the next generation are generated at random. This kind of $variation$ is heuristically justified and can be supposed to produce a better PV vector pair rather quickly. The $genes$ in future generations are again subject to the same $variation$ operation leading to better and better chromosomes in which the best characteristics of previous iterations are carried forward. Algorithm. 4 shows a pseudo-code for the genetic matching algorithm proposed above. In the crossover operation, a slice of the previous best input vector (the one resulting in the maximum) toggles is passed on to continue the GA. This is based on the heuristic that the better characteristics (which result in a large toggle count in the circuit) of the vector are present in parts of the vector, the passing on of which results in finding the ultimate power virus (PV) vector. Fig. 3.3 shows the progression of zero delay toggle count over the generations using our genetic search method. The algorithm was implemented on the ISCAS'85 benchmark suite. Fig.3.4 shows the results for a unit delay model. Appendix.B shows detailed results for both the methods above and also provides the obtained power virus

Algorithm 4 GENETICMATCHING

Require: best_vec = (rand_vec_1,rand_vec_2) $\Leftrightarrow$ (rv_1,rv_2)

Ensure: [CORRESP]$_{besttoggle}$ $\Leftarrow$ simulate (rv_1,rv_2)

loop
    $\rightarrow$ choose rand_vec_pair: (v_1,v_2)(v_3,v_4)
    $\rightarrow$ simulate ckt: input $\Leftarrow$ (v_1,v_2)
    $\rightarrow$ simulate ckt: input $\Leftarrow$ (v_3,v_4)
    diff1[size(v_1)] = diff2[size(v_3)] = 0
    for all i such that $1 \leq i < size(v_1)$ **do**
        if $v_1[i] \neq v_2[i]$ **then**
            diff1[i] = 1
        end if
    end for
    for all j such that $1 \leq j < size(v_3)$ **do**
        if $v_3[i] \neq v_4[i]$ **then**
            diff2[j] = 1
        end if
    end for
    ——————————CROSSOVER——————————
    v = rand(v_1,v_2) // Choose Either
    for all k such that $1 \leq k < size(v)$ **do**
        if diff1[k] $\neq$ diff2[k] **then**
            Flip v[k]
        end if
    end for
    new_toggles $\Leftarrow$ simulate (new_v_1,new_v_2)
    DISCARD: v_3,v_4
    if new_toggles $>$ best_toggles **then**
        SWAP: (new_v_1,new_v_2) $\Leftrightarrow$ (best_v_1,best_v_2)
    end if
end loop

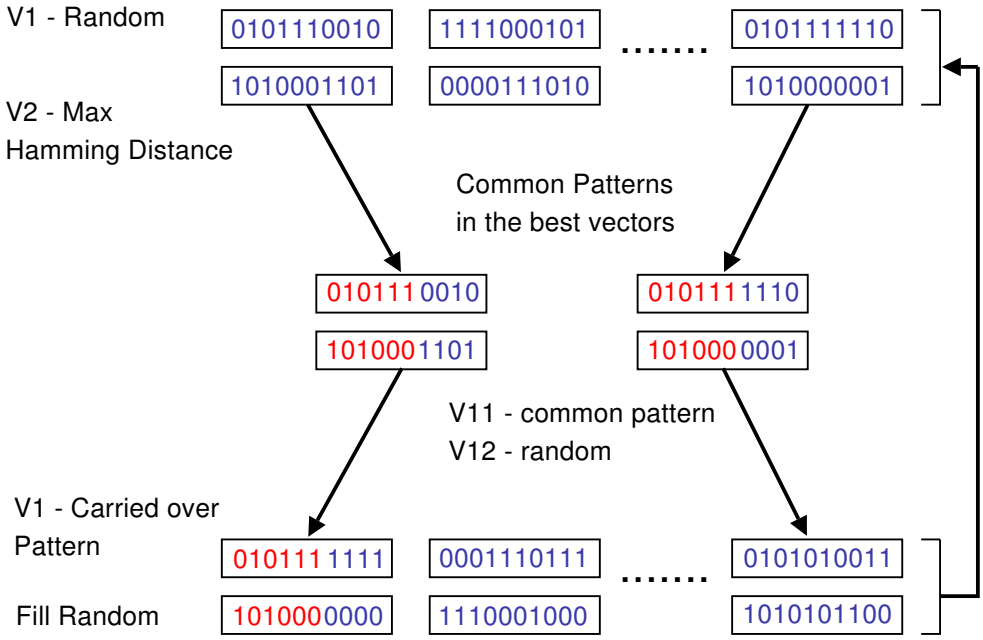


Fig. 3.5: Genetic Search methodology for PDPE in digital circuits.

vectors. Fig.3.5 shows a flow diagram for the genetic search methodology. The initial vector V_1 is generated at random. The second vector V_2 is generated based on a maximum hamming distance w.r.t the first vector V_1. Following this a circuit simulation is run and the vectors with the best fitness are selected based on a truncation selection scheme. Bit patterns among the fitter chromosomes are identified for commonality. These patters are chosen based on a n-point heuristic crossover variation scheme. The variation scheme is a little different that the conventional n-point crossover mechanism wherein parts of other vectors are joined to give a new vector. Here, we choose common bit patterns among the vectors with high fitness values followed by a random filling of the rest of the bits in the virus vector. These form elite phenotypes for succeeding generations. The genetic search algorithm is continued on this new pattern. The algorithm summary is also captured in Fig.3.6.

3.5 Experimental Results

The proposed algorithms in Sec.3.4 were implemented in C++ and tested on the ISCAS'85 benchmark suite. The algorithms were run on a Linux workstation with a 3.0 GHz Pentium IV processor and 2GB of physical memory. The convergence times of the

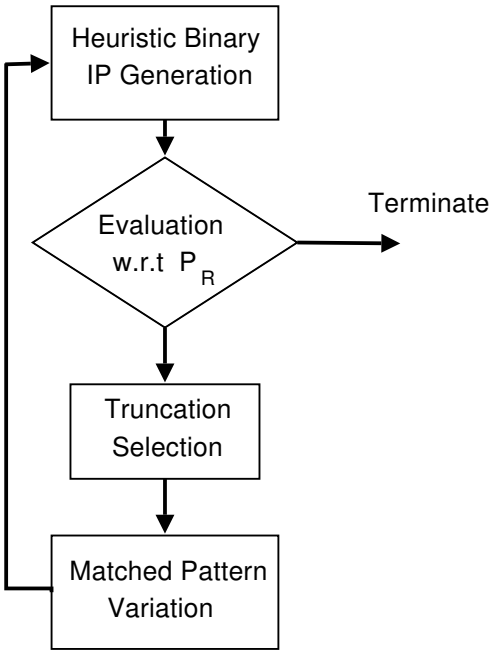


Fig. 3.6: Genetic Search algorithm for the PDPE problem.

algorithms (esp. the vector partitioning method) were practical - of the order of tens of minutes. This time frame is reasonable for a one time pre-design circuit characterization of PDPEs. Note that we use unit and zero delay models in our simulator. Table.3.1 shows the specifications for the ISCAS'85 combinational benchmark suite and the results are presented in Table. 3.2. The results shown are for weighted toggles which take care of the fanout of the switching gates as explained in Sec.3.4. The results are presented in comparison with random and a few other PDPE estimation methods found in the literature. xb

The results presented in Table.3.2 shows comparable PDPE estimates of the proposed method to the best known in literature. In the controllability based method [Najeeb *et al.* (2007)], Najeeb. et.al have used a modified D-Algortihm with a pre-defined backtrack threshold. The initialization of their algorithm is based on a random search for a vector at the primary inputs. The wires in the circuit are then optimized starting from the primary outputs and moving towards the primary inputs. An assignment conflict is resolved using a backtracking mechanism (much like in the D-Algorithm). The convergence time and rate of this algorithm is highly dependant on the preset backtrack threshold and on the initial random simulation to determine the primary inputs resulting in the maximum primary output toggle.

Table 3.1: ISCAS'85 Benchmark suite specifications

Circuit	#PIs	#POs	#Gates	#FFR Clusters	# Group2 Nets	#Levels
C432	36	7	204	76	343	13
C499	41	32	276	58	440	5
C880	60	26	470	105	755	10
C1355	41	32	620	258	1096	15
C1908	33	25	939	377	1523	20
C2670	233	140	1567	539	2216	21
C3540	50	22	1742	619	2961	24
C5315	178	123	2609	806	4509	18
C6288e	32	32	2481	1456	4832	93
C7552	207	108	3228	1446	6252	18

Table 3.2: Comparison of PDPE for ISCAS'85 Combinational Benchmark Circuits - Zero Delay Model

Circuit	Number of Weighted Toggles						
	I_{max}/ Fanout	I_{max}/ Gain	MAX_P	Random	Control Driven	Vector Partition	Genetic Match
C432	214	223	183	201	270	242	224
C499	228	210	196	272	303	249	306
C880	524	517	388	437	582	498	479
C1355	650	688	368	530	610	509	596
C1908	872	882	898	858	973	886	964
C2670	1292	1366	1161	1332	1516	1266	1330
C3540	1527	1545	1347	1531	1727	1395	1633
C5315	2644	2668	2556	2570	3007	2673	2815
C6288	2207	1799	2991	2558	2684	2266	2515
C7552	3406	3524	3556	3591	3670	3431	3541

This dependancy doesn't exist in the proposed genetic method as well as in the vector partitioning method. The PDPE estimate is fairly independant of the starting vectors (primary input assignments) and the convergence rate and time of the algorithms are of the order of a few tens of minutes running on modern machines. The results show a significant improvement in the toggle estimates compared to I_{max}/Fanout, I_{max}/Gain, MAX_P and The Random methods proposed in the literature. The PDPE estimates from the two methods are also comparable to the Controllability based estimate in [Najeeb *et al.* (2007)]. The toggle estimates for one case - $C499$ (306) better the controllability based estimate (303). Appendix.B shows the toggle progression in the genetic/vector partition methods and also gives the power virus vector pairs which when applied in sequence to the bechmark circuits give the estimated toggles.

3.6 Summary

In this chapter, we have presented two novel methods for a dynamic PDPE estimate of digital VLSI circuits. The first method uses a vector partitioning approach wherein the primary inputs are clustered using the cone based partitioning method presented in [Saucier *et al.* (1993)]. The clustered inputs lead to sub-vectors which are optimized to yeild the maximum number of toggles in the circuit. This local optimization leads to a search complexity reduction to the order of $O(m \times 2^{\frac{n}{m}}$ from $O(2^n)$. The second method uses a modified genetic algorithm which passes on better characteristics of input vectors (parts of the vector which causes the maximum toggles) to future generations of the GA. The algorithm is run with a tournament selection scheme over multiple generations. Both these algorithms were tested on the ISCAS'85 combinational benchmark suite and the results obtained show a quicker convergence to comparable PDPE estimates to the ones reported in the literature.

CHAPTER 4

HARDWARE BASED GENETIC EVOLUTION OF FIR FILTERS

4.1 Introduction

In this chapter we look at the problem of a system design and try to approach it using a heuristic based genetic methodology. The problem we tackle is the design of FIR filters with an arbitrary response. Filters are electronic circuits which perform processing functions, specifically intended to remove or enhance signal components. They are used in many cutting edge electronic applications within which they form critical elements. Filters could be analog or digital and may posses an Infinite Impulse Response (IIR type) or a Finite Impulse Response (FIR type). Finite Impulse Response (FIR) digital filters have many applications in a wide range of Digital Signal Processing (DSP) algorithms. The lack of feedback in the design makes them more reliable and robust than their IIR counterparts. In practice, *Digital Finite Impulse Response (FIR) Filters* implement Eq.(4.1) where, p is the order of the filter, h_i are the filter coefficients, $x[n]$ and $y[n]$ are the n^{th} input and output signal samples respectively.

$$y[n] = \sum_{i=1}^{p} h_i x[n-i] \tag{4.1}$$

4.1.1 Adaptive Filters

Adaptive filters are those which self-adjust their filter coefficients according to an optimizing algorithm. This could be due to changing requirements on field as well as due to application specific demands. By way of contrast, *non-adaptive filters* have static filter coefficients. Because of the complexity of the optimizing algorithms most adaptive filters are digital. They are routinely used in a wide range of DSP applications. Equalization of data transmission channels in high-speed MODEMS, noise cancellation in speaker

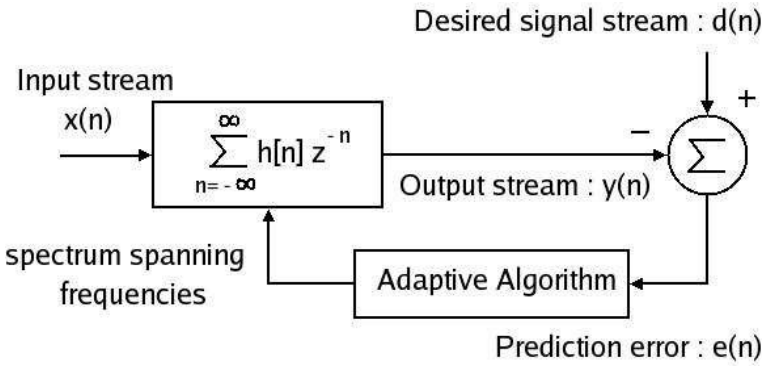


Fig. 4.1: A general adaptive FIR filter topology using the output error formulation as a feed. The prediction error ($e(n)$) could be different from a simple difference between the output and the desired signal stream.

phones, interface removal in medical imaging, canceling narrow-band interference in direct sequence spread spectrum systems and beam reforming in radio astronomy are some practical applications of adaptive systems. Conventional first generation systems typically embody basic *FIR Linear Filter* structures along with simple gradient descent or least squares algorithms for coefficient adaptation. As such, they have certain performance limitations as explained in [Sundaralingam and Sharman (1997)]. A more accurate adaptive structure is needed to realize a robust filter that adapts to the changing requirements on filed, quickly and correctly. Fig.4.1 shows a general *Adaptive Digital FIR Filter* topology. It shows an input sample stream ($x[n]$) filtered into an output stream ($y[n]$) which is compared with the desired sample stream ($d(n)$). The resulting deviation in the output is used as the prediction error ($e(n)$) to guide the adaptive algorithm which reconfigures the filter coefficients ($h[n]$) dynamically.

4.1.2 Some Approaches to FIR Filter Design

Signal processing architectures find *adaptive filter* fabrics indispensable and often need them to be capable of responding to various malfunctions caused by endogenous and exogenous factors. Consequently, *error tolerance* in adaptive systems demands a need for resilient designs. We can realize this using hardware fault tolerance which can be achieved by means of redundancy and other supplementary hardware modulation techniques [Miron Abromovici (2001)]. However the most critical component of the design - the filter coefficient values, need to be corrected in alternate, quicker ways for reduced

circuit off-times. Also, for error tolerance to be practical and effective, reduced overall on-chip *adaptation latency* is also essential. In reconfigurable systems, minimization of *adaptive complexity* is another critical objective.

In [Adams and Willson (1984)], Adams and Wilson sought *adaptation efficiency* by dividing the filter design into two stages - a computationally efficient pre-filter followed by an amplitude equalizer. For the same objective, Nevo, et.al in [Saramaki *et al.* (1988)] described FIR cascaded structures with an interpolated filter; In [Wade *et al.* (1990)], Wade, Van-Eetvelt and Darwen have described a non-interactive filter design method; And Suckley [Suckley (1991)] has furthered on it to propose band-fit filter designs using cascaded filter primitives. However, these *complex filters* exhibit little or *no error resilience* besides having a *limited adaptation range*. Their hardware implementation is plainly impractical. Realistic hardware platforms impose restrictions based on fixed register widths leading to a loss of precision in the storage of filter coefficients. The resulting filters known as Finite Word Length (FWL) filters would however accept this trade-off to benefit from faster computations. Several *simple configuration* methods have been proposed for effective FWL design of FIR filters or for filters with Canonical Signed Digit (CSD) coefficients [Cho and Lee][Kodel (1980)][Lim *et al.* (1982)]. Rounding-off the coefficients to the nearest integers has been another common practice. Coefficients were also sought to be represented as the sum of a few Signed Power-of-Two (SPT) terms. Further improvements were proposed using Primitive Operator Directed Graph (PODG) representations in [Bull and Horrocks (1991)]. However, these methods have issues of *poor response fidelity and high latency*.

Conventional filter designs hence involve multiple, often conflicting design criteria and finding an optimal solution is therefore not a simple task. Analytic or simple iterative methods usually lead to sub-optimal designs. This necessitates optimization based methods for filter design [Antoniou (2005)][Parks and Burrus (1987)][Lu and Antoniou (1992)]. However, the such methods formulated for digital filters are often complex, highly non-linear and multi-modal in nature. However, in recent years, a variety of optimization algorithms have been proposed based on the mechanics of natural selection and genetics, which have come to be known collectively as genetic algorithms (GAs) [Godlberg (1989)]. GAs are stochastic search methods that can be used to search for

an optimal solution to the evolution function of an optimization problem [Pittman and Murthy (2000)]. Holland proposed GAs in the early seventies [Holland (1975)]. De Jong extended GAs to functional optimization [Jong (1975)] based on the detailed mathematical model for the GA presented by Goldberg in [Godlberg (1989)]. In the past, GAs have been applied to a number of optimization designs including digital and analog filters [Horrocks and Khalifa (1994)]. Genetic filter designs have also been of considerable interest in recent years. Genetic evolution (application of several instances of a given GA) of FIR Filters has been of considerable interest in the recent past [Sundaralingam and Sharman (1997)][Suckley (1991)][Sabbir and Antoniou (2006)][Tufte and Haddow (2000)] and [Redmill and Bull (1998)]. However, the focus in each of these has been discrete and diverse. In [Tufte and Haddow (2000)], Tufte and Haddow demonstrate a model for GA operations on FIR filters. But their focus is not on design practicality (hardware implementation). They consider non-integer (floating point) coefficients which makes their design process all the more complex. In addition, the fitness function of the GA they use does not scan the entire frequency spectrum but considers only specific frequencies for evaluation. This limits the accuracy coefficient estimates. The trade-off not considered here is that between integer coefficients (avoiding floating point operations) and excess spectrum samples (accuracy).

Genetic algorithm operations were described in Sec.2.1. Here we present another quick synopsis to keep things in perspective. Genetic Algorithms use simple operations over a large number of iterations to optimize system goals making their implementation easy and effective. Errors in filter coefficients lead to an egress in the prediction error which could be used to correct the faulty coefficients on-field. GAs also have the flexibility to determine coefficient values for quite an arbitrary set of frequency response characteristics. Such nasty responses are a common premise of many real life applications. However the major challenge in the genetic design for FIR filters is the adaptation latency which can up shoot quite significantly owing to a poor search methodology. A GA involves a large number of heuristics based on the problem dynamics which when exploited appropriately alleviates the latency issue to a significant extent.

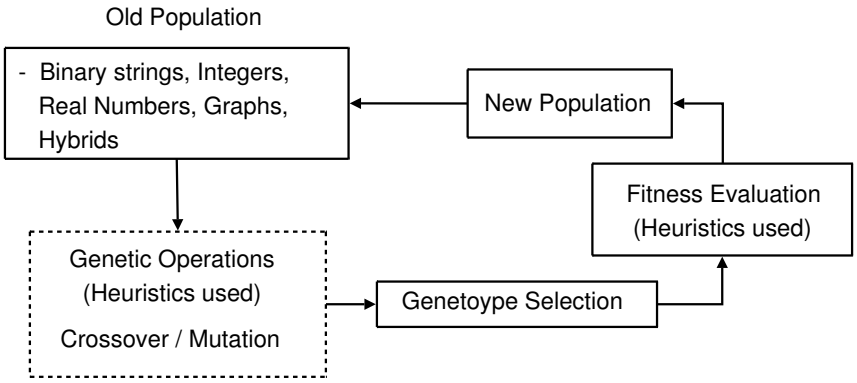


Fig. 4.2: A Generic GA Cycle - Involves four major steps Initial Population Generation, Variation, Evaluation and Selection

4.1.3 Genetic Operators from a new perspective

Genetic Algorithms (GAs) manipulate a population of individuals in each generation (iteration), where each individual, termed as a chromosome or genotype, represents one candidate solution to the problem. Within the population, individuals with better fitness survive to reproduce. Their genetic material is varied to produce new individuals as offsprings which form the seeds for the following generations. The genetic material is modeled using some data structure with finite attributes. As in nature, selection provides the necessary driving mechanism for better solutions to survive. Each solution is associated with a fitness value that reflects how good it is compared to the other solutions in the population. The variation process comprises of crossover and mutation, which concoct material by partial exchange among genotypes and by random alterations of data strings respectively. The frequency of these operations is controlled by certain pre-set probabilities which require heuristics appropriate for the particular problem at hand. The representation, variation, evaluation and selection operations constitute the basic GA cycle or generation, Fig.4.2, which is repeated until some pre-determined criteria are satisfied which also require a comprehension of the problem heuristics. With increased computing power and hardware, simulation of evolutionary systems is becoming more and more tractable and GAs are being applied to many real world problems including the design of digital filters. The crucial need in such designs is the use of appropriate heuristics for probabilistic variation and selection. This chapter presents analytical insights into the selection of the Initial Population (IP) and the Genetic Operators (GO) used by the GA. This exploits the problem heuristics to the maximum and unlike several instances

of GA reported in the literature, the IP for the GA proposed in this work is not derived at random but is derived using a deterministic procedure based on the properties of the impulse response of FIR filters. This makes the entire adaptation process fast and robust. As a byproduct, the chapter also proposes a comprehensive methodology for the design of what is termed as the Evolutionary System on Chip (ESoC).

The rest of this chapter is organized as follows. In Sec.4.2, we explore existing filter architectures and their advantages. Specifically, we dwell on the comparison between the spatial design and the frequency domain design of FIR Filters. In Sec. 4.3, we present the functions used by the evolutionary algorithm with analytical justifications for choosing them. In Sec.4.4, we propose and describe the complete Evolutionary System-on-Chip (ESoC) hardware design and its parallelism, scalability and Built in Error Tolerance (BiET), . Sec.4.5 provides experimental results for the proposed design and hardware architecture where we present simulation cases for exemplary linear phase FWL test cases. Finally we conclude in Sec.4.6.

4.2 FIR Architectures and Design

4.2.1 FIR filter architectures

Digital FIR filter outputs are related to their inputs by the following difference equation (Expanded from Eq.(4.1))

$$y[n] = h_0 x[n] + h_1 x[n-1] + \ldots + h_p x[n-p] \tag{4.2}$$

where p is the filter order, $x[n]$ is the input signal, $y[n]$ is the output signal and h_i are the filter coefficients. Hardware implementation of the filters in the time domain can have the canonical form translating equation (4.2) directly or the *broadcast(transposed direct) architecture* obtained using the inversion property of the resulting Signal Flow Graph (SFG) - Fig.4.3 and Fig.4.4. We use the broadcast architecture in our hardware design. FWL constraints require truncation of coefficient lengths besides their magnitudes. The minimum length of linear phase low-pass FIR filters to meet the frequency domain

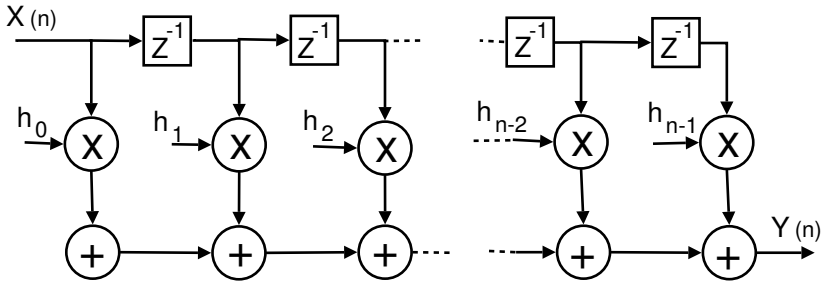


Fig. 4.3: Direct Form Implementation of Digital FIR Filters

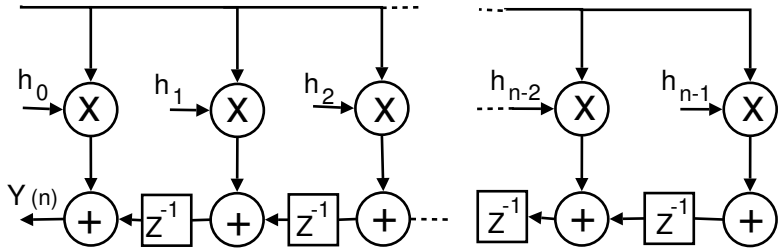


Fig. 4.4: Transposed Direct Form Implementation, $h_i's$ are the filter tap weights and Z^{-1} are the delay elements.

specifications is approximately

$$N = \frac{-20log_{10}\sqrt{\delta_p\delta_s} - 13}{14.6\Delta F} + 1 \tag{4.3}$$

where δ_p and δ_s are the passband and stop band ripples and ΔF is the transition bandwidth [Saramaki *et al.* (1988)]. Eq.(4.3) forms the initial coefficient length constraint on our filter specification which is done offline.

4.2.2 Spatial and frequency domain design

The filter coefficients used in the FIR architecture form a $sinc$ function, the Fourier transform of which is a window, Fig. 4.5. Frequency selective filters could be designed in the *Frequency Domain (FD)* for which a multitude of complex algorithms exist. They focus on the process of cumulatively windowing the uniform frequency span to approximately reach the desired frequency response. This is then inverse transformed to translate to a $sinc$ in the time domain. Some popular FD domain design algorithms are outlined in [A. V Oppenheim and Buck]. *Spatial Design (SD)* is an alternative to the Frequency

Domain Design (FDD) of FIR filters. It is advantageous over the FDD as it eliminates the need for the complex operations of transformation and inverse transformation of the response to determine its fitness in an iterative design. This has tremendous implications in hardware implementability simplifying the design flow to basic operations and hence speeds up the computational steps substantially. However a large number of samples may still be needed for an accurate determination of the fitness function for the GA. This trade-off is a design choice and we stick to simpler operations in the spatial domain for the filter operation. In the next few sections, we propose the genetic evolution of filter co-

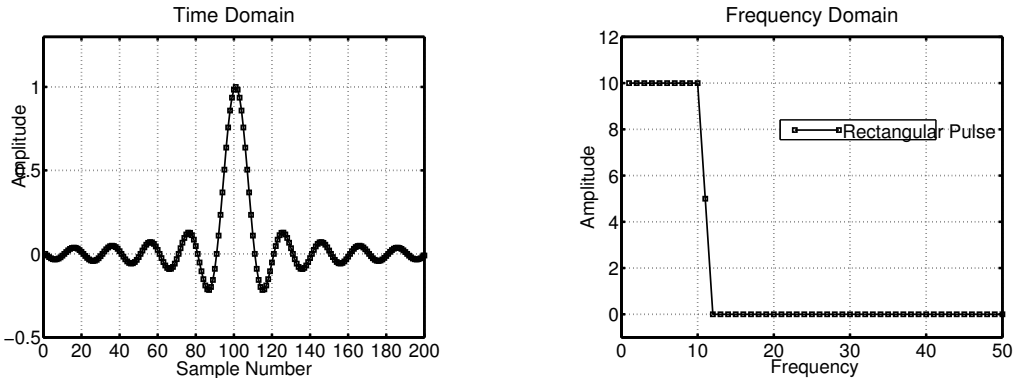


Fig. 4.5: A Sinc Function expressed as f(x)=sin(x)/x and its transform, a window function. The FIR filter coefficients form a sinc function.

efficients in the time domain including appropriate heuristics with analytical justifications and describe its complete implementation in hardware.

4.3 Evolutionary System Design

The objective of spatial design of FIR filters is to accurately estimate the coefficients h_i, $1 \leq i \leq p$, in Eq.(4.1) at any point of time. This section explains how a GA can be employed for the same. The proposed algorithm has the following steps.

Step 1 *Limiting the search space for the coefficients*

(A) Determination of the largest coefficient h^{max}

Given the order of a filter p, the first step in the FIR design process is to estimate the largest value, h^{max}, that could be assigned to the coefficients h_i, $1 \leq i \leq p$ in Eq.(4.2).

This is done as shown in Eq.(4.4) [A. V Oppenheim and Buck].

$$h^{max} = \int_0^1 (G_{id}) \mathrm{d}f_n \tag{4.4}$$

where G_{id} is the expected frequency response normalized w.r.t f_n, $f_n = \frac{f}{f_s}$, f_s is the sampling frequency and f is the range of frequencies considered in the spectrum. The summation shown in Eq.(4.5) is a trapezoidal approximation to the integral in Eq.(4.4). This is easily implementable in hardware as an alternative evaluation of h^{max}.

$$h^{max} = \sum_{j=0}^{N_j} \sum_{i=1}^{9} \frac{10^{j+1}}{2f_{Nyq}} \times G_{id}(10j + i) \tag{4.5}$$

The limits in Eq.(4.5) for i and j are chosen so as to span the frequency spectrum containing the frequency components of the input signal. f_{Nyq} in the equation is the Nyquist Frequency obtained as an input parameter and N_j is the highest decade order of the expected response. This is a tunable factor which dictates the accuracy of initial evaluation.

(B) Determination of coefficient search range h_i^{max}

The odd number (p) of filter coefficients, h_i in Eq.(4.1), are symmetric about the mean value h_q, where $q = \frac{(p+1)}{2}$. The FIR filter coefficients lie on a $sinc$ function. In general, a $sinc$ function is given by $sinc(x) = \frac{sin(x)}{x}$ as explained in Sec.4.2.2. From Eq.(4.5) the mean value h_q of the filter coefficients is determined which is the largest value among the coefficient set of cardinality p. To estimate an upper bound (h_i^{lim}) for coefficients other than h_q, we use a $\frac{1}{x}$ functional model. The upper bound is used to limit the search space for the forthcoming genetic algorithm. This model starts with h_q as the highest value of the function with a gradient modulated by the lowest cut-off frequency, f_{cutoff}, of the filter which is obtained as an input parameter. This is shown in the computation of the interpolating step in Eq.(4.7). The modulation is justified by the fact that the gradient fall for the $\frac{1}{x}$ model close to the mean coefficient for steeper cut-off filters is lesser than that for filters with wider transition bands. Fig.4.6 and Fig.4.7 show comparative effects of incorporating f_{cutoff} in the range estimation. In Fig.4.7, the normalized cut-off frequency is lower and hence the gradient fall from the mean value is slower compared to that in Fig.4.6. This model captures the fact that a large number of coefficients are close to the mean value for filters with lower cut-offs. Hence, based on the maximum coeffi-

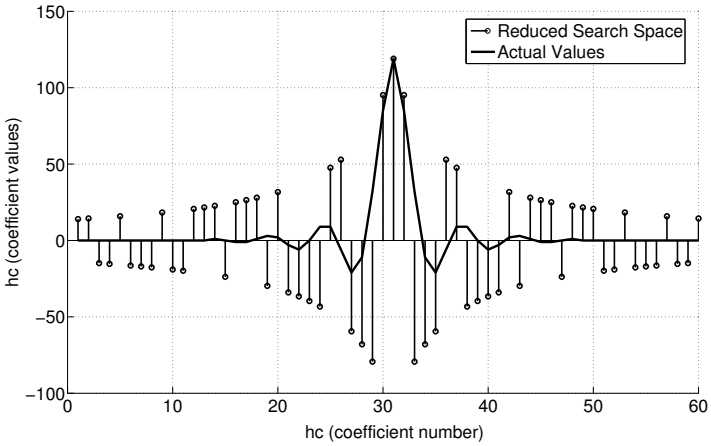


Fig. 4.6: Reduced GA search space with the Gaussian Switch. f_c=0.25 and p = 60

cient value, h^{max}, determined in Step 1(A) we can compute the maximum range for each of h_i in Eq.(4.1) by modeling a $\frac{1}{x}$ behavior. This can be done by linear interpolation in the inverse coefficient space, Eq.(4.8). As a starting point for this, we compute Δ^{mean} and δ_{step} as shown in Eq.(4.6) and Eq.(4.7). h_i^{lim}, the coefficient search range for each of the i^{th} coefficient is hence computed using δ_{step} linear increments from the mean value Δ^{mean}.

$$\Delta^{mean} = \frac{1}{h^{max}} \tag{4.6}$$

$$\delta_{step} = \frac{f_{cutoff}}{h^{max}} \tag{4.7}$$

$$\Delta_i^{lim} = \Delta^{mean} + i.\delta_{step} \tag{4.8}$$

$$h_i^{lim} = \left\lceil \left(\frac{1}{\Delta_{n-i}^{lim}} \right) \right\rceil \tag{4.9}$$

Based on the description of functional model in this section, we can determine the coefficient limit envelope h_i^{lim}, Eq.(4.9), which determines the range within which each of the filter coefficients, h_i, $1 \leq h_i \leq q$, in Eq.(4.1) lie. This methodology helps narrow down the search domain from infinity to h_i^{lim} within which the possible coefficients may lie. To translate the Δ_i^{lim} values into the coefficient limits, we take the upper integer ceil of the reciprocals of Eq.(4.8) and flip their order to get the coefficient limits.

Step 2 *User Configuration*

The filter hardware is designed to accommodate a maximum of IP_{max} Initial Populations

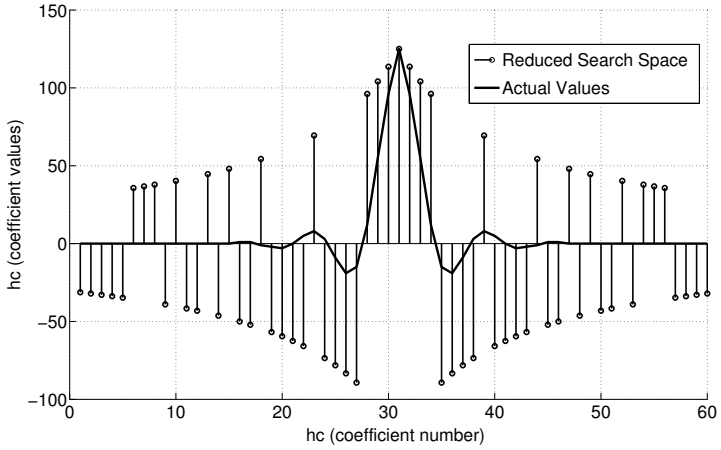


Fig. 4.7: Sample case with f_c=0.1 and p = 60. A lower f_c would mean a slower $\frac{1}{x}$ fall

(IP) in the parallel process such that a given population set can hold upto p_{max} coefficients. The user can configure the number of coefficients and the size of the IP such that they lie within the above limits. The user also inputs the ideal filter response G_{id}, the filter cut-off frequencies and the sampling rate f_s. The evolutionary hardware estimates h_{max} and h_i^{lim}, $1 \leq i \leq p$, as described in Step 1.

Step 3 *IP Generation*

As mentioned earlier, the GA generates a user defined number of chromosomes. Each chromosome is a vector $(h_1, h_2 \ldots h_p)$ of p integers such that $h_i = \pm h_i^{lim}$ (based on the binary switch) in the first iteration and are restrained within $-h_i^{lim} \leq h_i \leq h_i^{lim}$ in subsequent iterations. The h_i's start with h_i^{lim} as initial estimates and slowly evolve towards the required coefficients. It is easy to see that, each chromosome describes a filter albeit with a different frequency response. The following steps describe the GA used in the design.

Step 4 *Genetic Evolution*

There are two evolution stages in the design. At the parallel macro-evolution stage, a *random normal binary switch* toggles the sign of the coefficient limit h_i^{lim} randomly for each of the i^{th} coefficients in the GA search. This introduces more variation in the population. The GA search range is $[0, \pm h_i^{lim}]$ at this stage. At the second micro-evolution stage, we follow a similar methodology but the GA search space in this case is $[-h_i^{lim}, h_i^{lim}]$. A matrix of switched h_i^{max} coefficient sets is formed again. The number of elements of the matrix are user defined to be IP. Each of the randomly switched initial population vectors

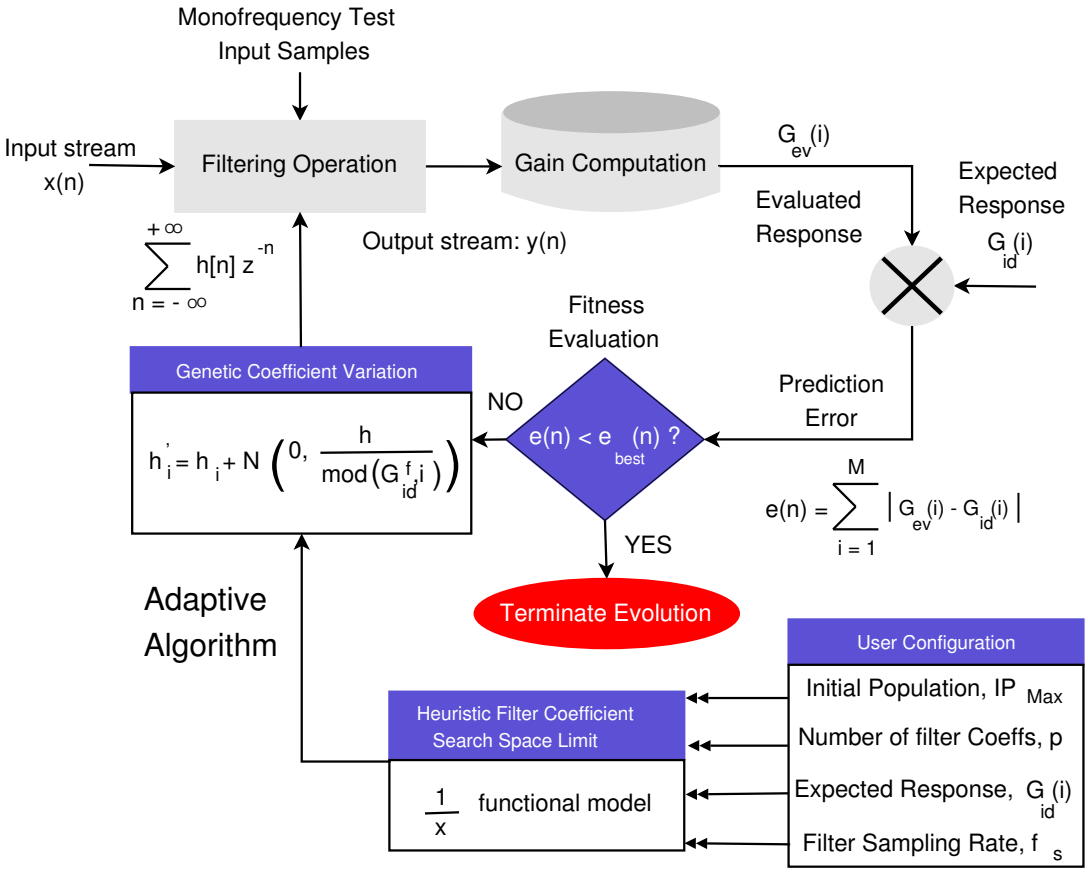


Fig. 4.8: The evolutionary adaptive filter design including the UI

are used to independently and parallely evolve towards the best fit solution. The following section explains the evolution strategy for each independent vector in the matrix.

(A) Mutation

The variation strategy of the filter involves *random normal perturbation* of the object parameter, h_i, between the preset limits decided by h_i^{lim}. The mutation incorporates progressive reduction and occasional bursts in standard deviations of the perturbation, Eq.(4.10). This can be seen as analogous to the temperature surges in simulated annealing (SA) and has shown empirically better results than other variation methodologies.

$$h_i^{'} = h_i + N\left(0, \frac{h_i}{mod(G_{id}^f, i)}\right) \tag{4.10}$$

where G_{id}^f is the G_{id} gain at frequency f, $h_i^{'}$ is the mutated coefficient value of h_i and $N(\mu, \sigma)$ represents a normal distribution function with mean μ and variance σ as shown in Eq.(4.11).

$$N\left(\mu, \sigma\right) = \frac{1}{\sigma\sqrt{2\pi}} exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4.11}$$

Fig.4.9 shows the standard deviations for the mutation of the coefficients for a symmetric test filter. The test case is of length $p = 21$. Note that the values decrease with the coefficient number. The heuristic for this being that the upper bound of h_{lim}^j forms a closer estimate of the ideal filter coefficients for near mean coefficient numbers than the farther ones the deviation between which can be modeled close to (4.11).

(B) Evaluation

The evaluation of the coefficient set is based on the fitness value computed by summing up the absolute deviations of the evolved gain G_{ev} from the expected gain, G_{id}^f over the entire frequency spectrum using the isolated filter hardware.

$$\xi_{curr}(s) = \sum_{i=1}^{M} \left[G_{ev}(i) - G_{id}(i)\right] \tag{4.12}$$

This evaluation shown in Eq.(4.12) is unlike in [Tufte and Haddow (2000)] where rather than evaluating the entire frequency spectrum, specific sinusoids are used for fitness evaluation.

(C) Selection and Termination

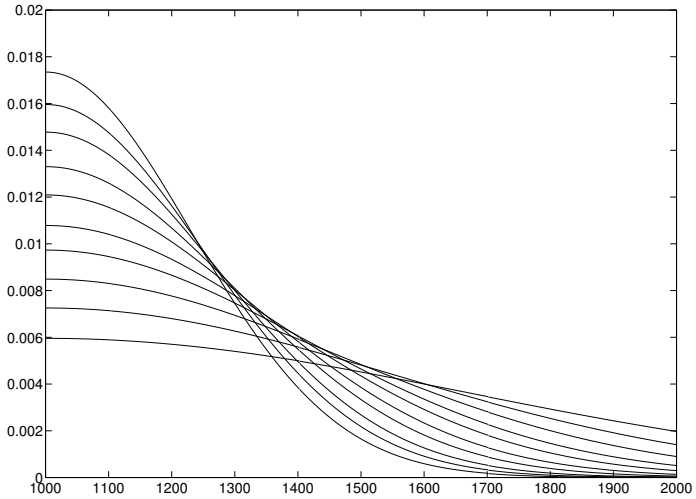


Fig. 4.9: Decreasing σ with increasing coefficient numbers j in the normal random mutation. The tallest peak stands for the mean coefficient whose deviation is the least and the deviation increases with reducing coefficient index

The selection of the chromosome set for every, parallel macro-evolution stage is based on the overall fitness criterion which is terminated after a pre-determined best fitness has been reached or when the fitness improvements over the previous generations are not substantial. Eq.(4.13) is used to estimate δ_{step} for the GA termination.

$$\delta_{step} = \begin{cases} 0 & \text{if } \xi_{curr} < \xi_{best} \\ 0 & \text{if } (\xi_{curr} - \xi_{past}) < \delta_{min} \\ \delta_{step}^{norm} & \text{Otherwise} \end{cases} \tag{4.13}$$

If the difference between the current fitness estimate, ξ_{curr}, and the best estimate over a pre-determined set of iterations, ξ_{past}, does not exceed δ_{min}, a minimal distinction criterion, the genetic algorithm is terminated. Hence obtaining the required filter coefficients.

The filter architecture is presented as a flow diagram in Fig.4.8. The user inputs the parameters as described in Step 2. The initial coefficient estimation occurs as explained in Step 1. This limits the search space. Step 3 is the IP generation which is controlled by the UI as shown. The genetic evolution step is shown as a separate block. Mono frequency sinusoidal samples spanning the spectrum are input to the filter with the initial estimate of the coeffients (tap weights). The FIR filter operation yields an output sample stream

$y[n]$ which is used to estimate the gain as shown in Eq.4.10. The gain is evaluated based on the maximum average output of the stream $y[n]$ when the input stream, $x[n]$ has unit amplitude, mono-frequency, spectrum spanning sinusoids. This evaluated gain is used to estimate the fitness and finally decide the termination as shown in Eq.(4.13) above.

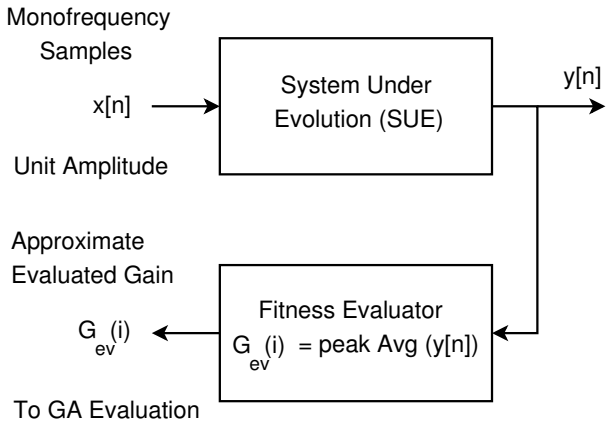


Fig. 4.10: The fitness evaluator for the system under evolution.

4.4 Intrinsic Design

Intrinsic (or hardware) design of evolvable systems has been an issue of great interest in recent times. However the implementation complexity of traditional GAs make it *very difficult to translate the algorithmic design into hardware*. Insights into approximations and simplified heuristics are essential for effective hardware translation. In [Redmill and Bull (1998)], Redmill and Bull use the Primitive Operator Directed Graph (PODG) representation to simplify the design. The GA is used to provide a Non Dominated Set (NDS) of solutions. However the entire FIR structure is evolved from scratch. In [Miller (1999)], Miller follows a similar approach using gate arrays. He points out the difficulties in hardware implementation and falls back to extrinsic evolution. *T*ufte and Haddow [Tufte and Haddow (2000)], propose the Complete Hardware Evolution (CHE) implementation of the GA pipeline. But their implementation description is only a traditional GA. The Evolutionary System on Chip (ESoC) serves to address this lacking balance between convergence speeds and implementation practicality. In the following sections, we highlight the implementation mechanism at module level hardware logic for the previ-

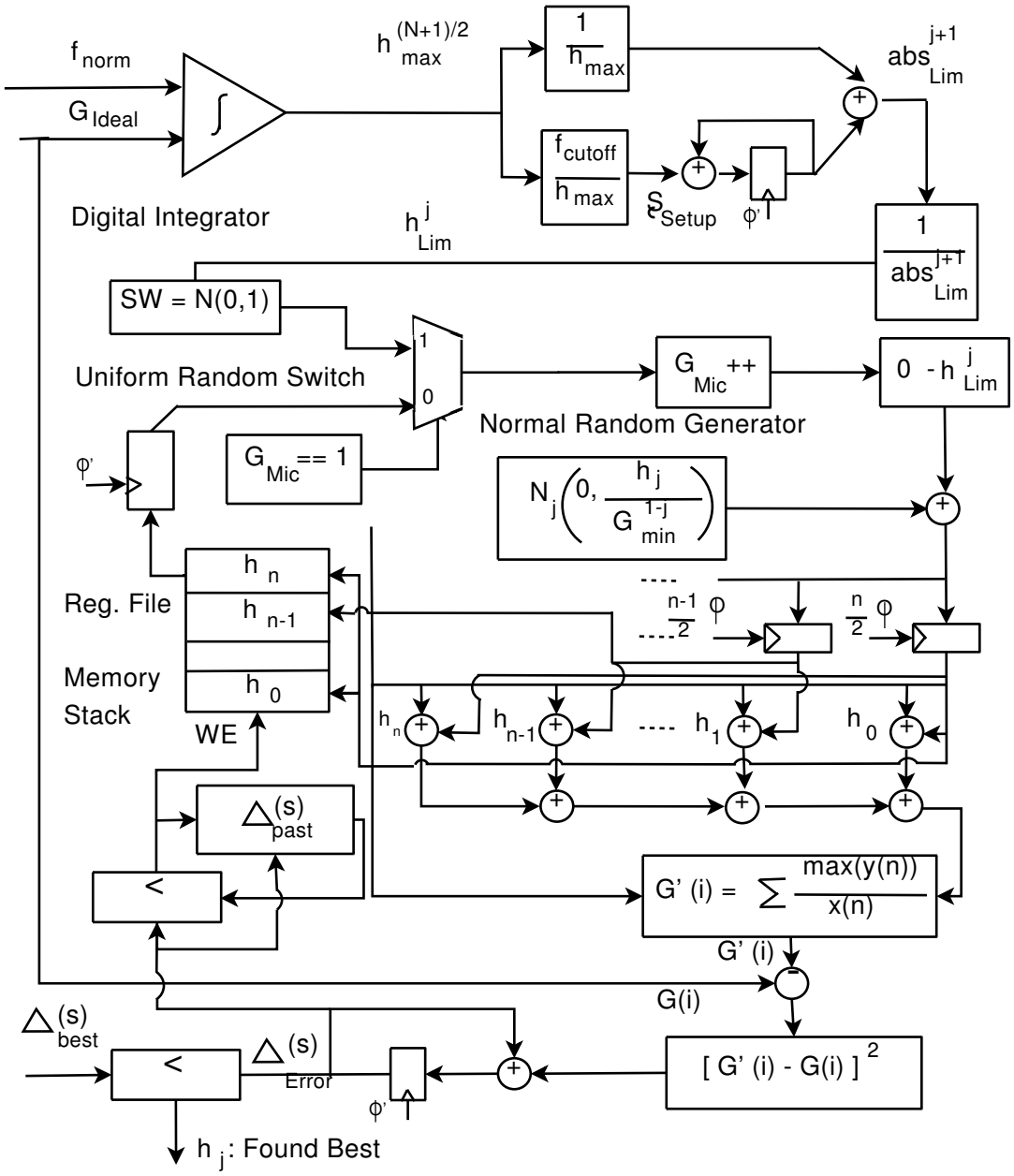


Fig. 4.11: Macro-Evolutionary System on Chip

ously described two stage heuristic evolution of the FIR filter. We demonstrate that the filter is self sustaining and independently evolving. The reconfiguration and adaptation times for the filter are quite practical and the hardware architecture described can form a backbone for evolution of FIR filters in many real life applications. The degree of scalability and the level of parallelism in the design are customizable and provide a flexibility for fast convergence of results keeping the evolutionary genetics practical.

4.4.1 The Evolutionary System on Chip (ESoC):

In the following sections, the chapter proceeds to present details of the hardware implementation of the two stage evolution mechanism described in the previous section. In a practical Evolutionary System on Chip (ESoC), we have a macro-evolution stage for a faster but coarser convergence and a micro-evolution stage for more accurate coefficient estimates with slower convergence rates.

Macro-Evolution Stage Fig.4.11 shows the Evolutionary System on Chip (ESoC) architecture that implements the GA. In other words, all the heuristics proposed in the previous sections are translated into hardware and shown implementable on a structured FPGA/ASIC platform.

The population generation block The population generator integrates the ideal gain contour specified by the user $w.r.t$ the normalized frequency to determine the maximum coefficient value. This is then randomly switched using the Normal Random, $N(0,1)$, switch. This is multiplexed through to the filter in the first macro-generation.

The evolution and filtering block The isolated filter, which comes offline temporarily, filters out a stream of spectrum spanning sinusoidal inputs to determine the corresponding outputs. The filter coefficients are the coefficient limits during the first iteration and the best fit coefficients, h_i^{best}, progressively. The mutation of the coefficients occur as in Eq.(4.10). The normal random generator is again modulated with an intrinsically self-adaptive strategy parameter. The deviation space is $[0, |h_i^{lim}|]$ in every macro-iteration. The filtering block used in the design is the broadcast architecture which provides a pipelined and fast filter implementation to evaluate the fitness span over the entire spec-

trum of frequencies in realistic time frames.

Evaluation and Selection block The fitness evaluation for multiple frequencies is done using the multiplier and accumulator. This evaluated fitness $\xi_{curr}(s)$ is compared against the best available fitness value $\xi_{best}(s)$ or with the previous fitness value $\xi_{past}(s)$ to either terminate the macro-evolution or to update the coefficient sets h_i^{past} with the current (more) fit coefficients, h_i^{curr}, respectively, Eq.(4.13). This comparison records the best coefficient set by write enabling (WE) the Coefficient Register File (CRF) from which the coefficients are read out to be used in successive generations.

Micro-Evolution Stage The Micro-evolution stage of the adaptive filter is shown in

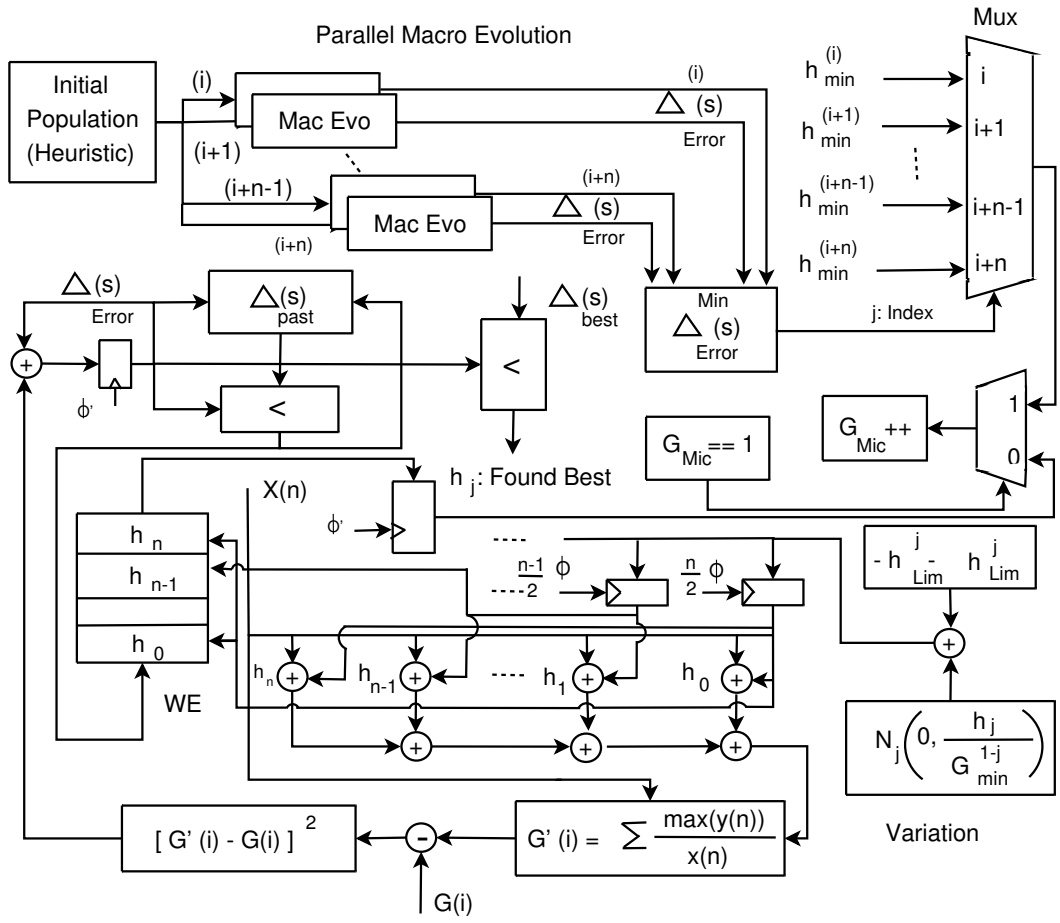


Fig. 4.12: Parallel and Scalable Micro-Evolution Stage

Fig.4.12. The parallel ESoC design is a set of Macro-Evolution stages running in parallel over a customized number of parallel processes. These arrive at their own respective fitness values. The minimum fitness, ξ_{min}^{j}, among them is chosen to be the best fitness and the micro-stage runs another GA very similar to the one in the macro-stage using the

$h_i^{best,mac}$ as the initial coefficient estimate. The sole difference between the macro and the micro stages is that the search space for the micro stage is $[-h_i^{lim}, h_i^{lim}]$. This makes this stage a little diverse in search albeit very precise. The convergence of the micro stage of evolution is a little slow compared to the macro stage. However this is faster when run in unison with the macro stage than when run alone.

Parallelism and Scalability in the design The proposed design has a high degree of customizable parallelism and scalability. This is reflected in the number of macro-stages which could be used for the GA. The trade-offs in this choice are speed and accuracy. This customizable similarity among the macro and micro stages is advantageous to the effect that the same filter and hence the GA can be run over multiple cycles providing good hardware re-usability.

Built in Error Tolerance (BiET) Isolated self repair is a built in feature of the design. The advantage of the design is that any error occurring in the system coefficients after it has evolved and is running on field can be easily corrected by re-evolving the filter . This provides a robust architecture immune to externalities. This BiET has important applications in noise cancellation and adaptive channel equalization.

4.5 Experimental results

The results presented here are from the design experiments done using the architecture in Sec.4.4. The entire ESoC architecture shown in Fig.4.11 and Fig.4.12 was emulated on MathWorks Inc. Matlab Version 7.0.0.19901(R14) and was run on a RHEL Linux workstation with a 3.0 GHz Pentium IV processor. Fig.4.13 and Fig.4.14 show the macro and micro fitness progress of some sample filter cases. An initial choice for a test filter was with cut-off frequencies f_{cutoff} = $0.22, 0.33, 0.44, 0.66$ and band gains $A_v = 10.0193, 106.667, 203.314, 300$ respectively, length $p = 21$ and sampling frequency $f_s = 18 \times 10^3 Hz$. Fig.4.15 [1] and Table.4.1 and Table.4.2 show the response charac-

[1] scaled/f_s normalized samples. The fitness is scaled *w.r.t* the best fit value based on the selection criterion in Eq.(4.13)

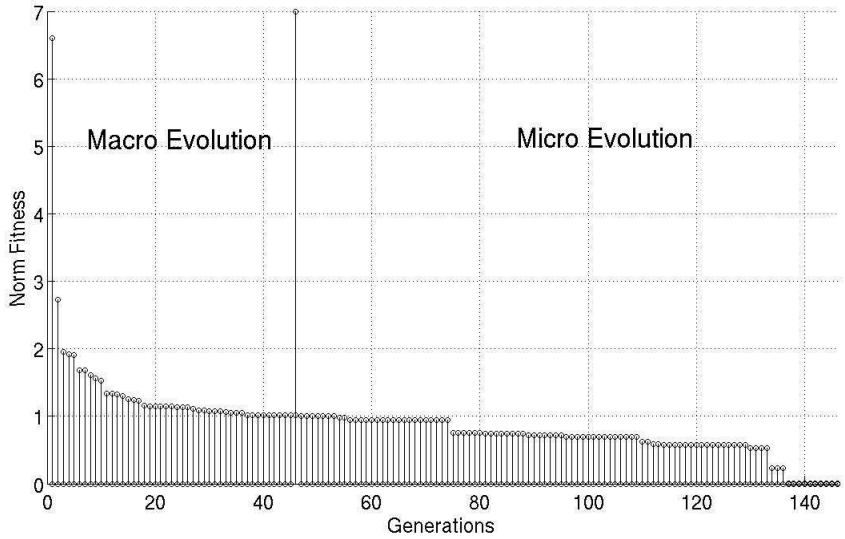


Fig. 4.13: Macro and Micro fitness evolution progress. Test case with p : 21, $\delta_{s/p}$: $[15, 250, 15]$ and $f_{s/p}$: $[0.11, 0.2]$

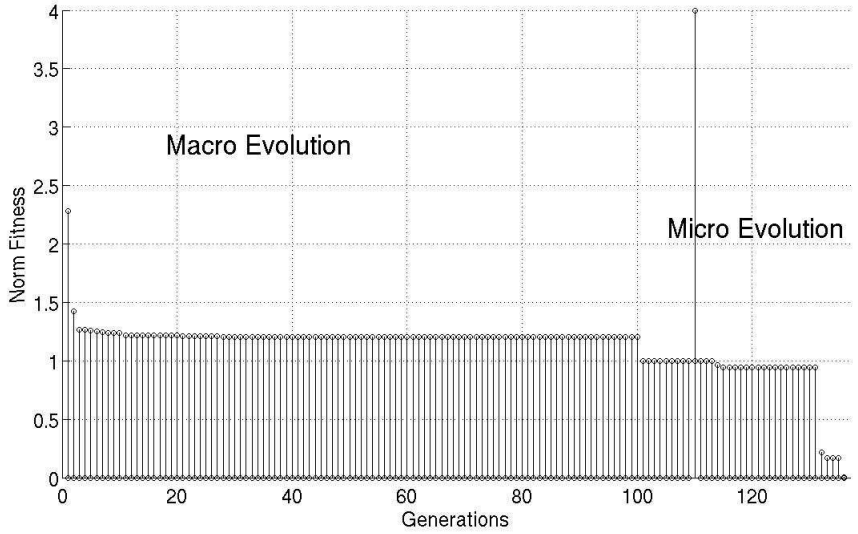


Fig. 4.14: coarser case with p : 21, $\delta_{s/p}$: $[320, 90, 100, 320]$ and $f_{s/p}$: $[0.1, 0.25, 0.43, 0.5]$

teristics. The algorithm converged within less than 100 generations (compare with 1171 generations in [Tufte and Haddow (2000)]). Fig.4.15 shows the expected frequency response of the actual filter, the response obtained from the Parks-McClellan algorithm (remez exchange in Matlab) and the evolved filter response using our genetic methodology. The response characteristics are within reasonable practical specifications and with an acceptable adaptation latency. Fig.4.16 shows the macro-fitness evolution where the fitness values evaluated using the algorithm progress *w.r.t* time.

Fig.4.17 shows the frequency response characteristics and Fig.4.18 shows the macro-fitness evolution of a band-pass filter case with cut-off frequencies and band gains as f_{cutoff} = $0.1, 0.11, 0.22, 0.33$, and $A_v = 10, 300, 300, 10$ respectively, length $p = 21$ and sampling frequency $f_s = 18 \times 10^3 Hz$. Fig.4.21 shows the micro-fitness evolution of the filter. The micro-evolution stage fine tunes the convergence of the

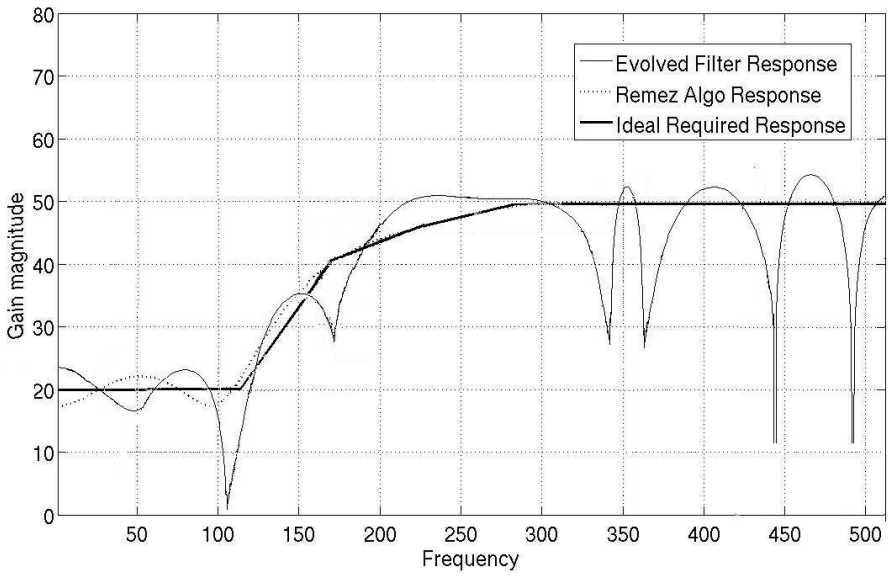


Fig. 4.15: Frequency response evolution progress. Test case with coarse spec p : 21, $\delta_{s/p}$:$[10.0193, 106.667, 203.314, 300]$ and $f_{s/p}$: $[0.22, 0.33, 0.44, 0.66]$

macro stage. The refinement is systematic and typically stops in fewer iterations than the macro-evolution stage. Fig.4.19 shows a low pass filter characteristic, Fig.4.18 shows the Macro fitness evolution and Fig.4.22 shows the Micro fitness evolution progress. The filter was designed using cut-off frequencies $f_{cutoff} = 0.22, 0.33, 0.44, 0.67$, band gains $A_v = 199.9873, 136.6667, 73.3460, 10$, length $p = 21$ and sampling frequency $f_s = 18 \times 10^3 Hz$. Implementation of the adaptable genetic filter on a dedicated ASIC

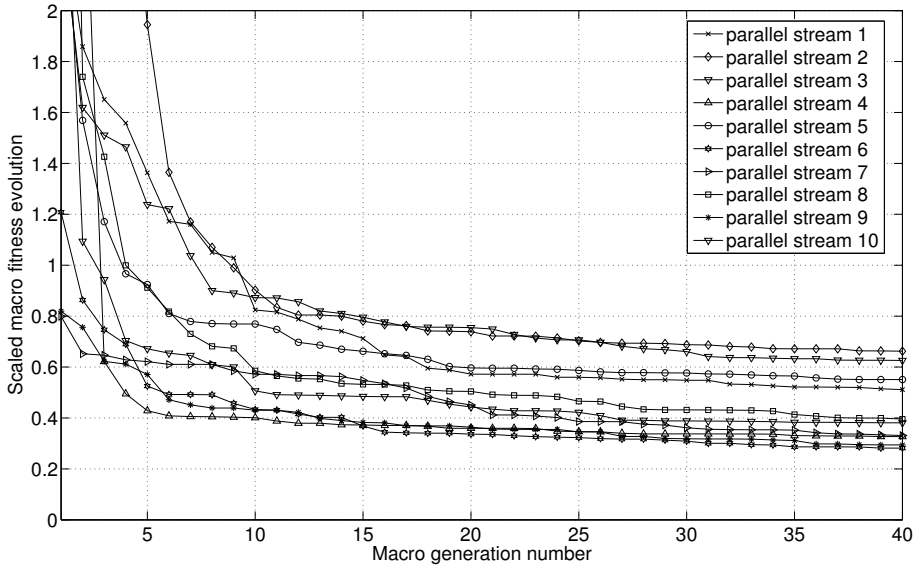


Fig. 4.16: Macro fitness evolution progress. Test case with coarse spec p : 21, $\delta_{s/p}$:$[10.0193, 106.667, 203.314, 300]$ and $f_{s/p}$: $[0.1, 0.11, 0.22, 0.33]$

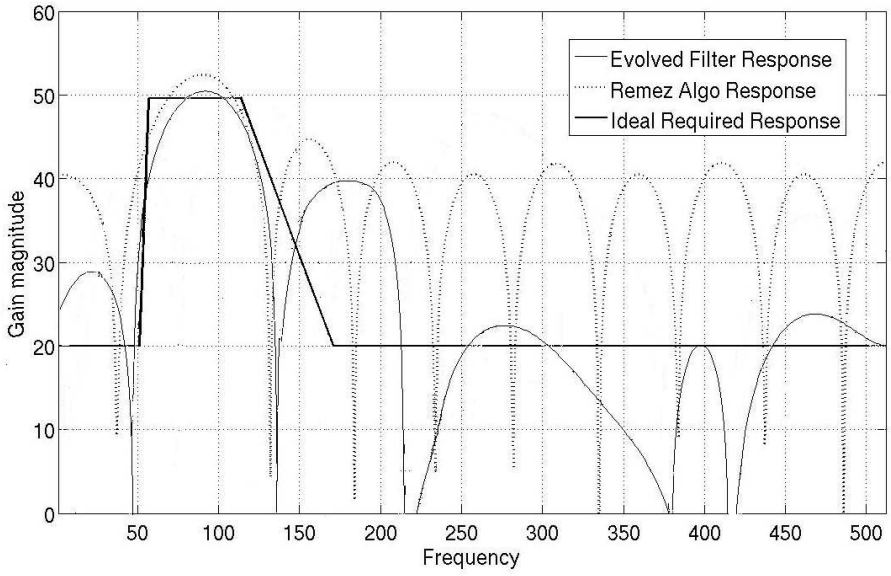


Fig. 4.17: Frequency response evolution progress. Test case with p : 21, $\delta_{s/p}$:$[10, 300, 300, 10]$ and $f_{s/p}$:$[0.1, 0.11, 0.22, 0.33]$

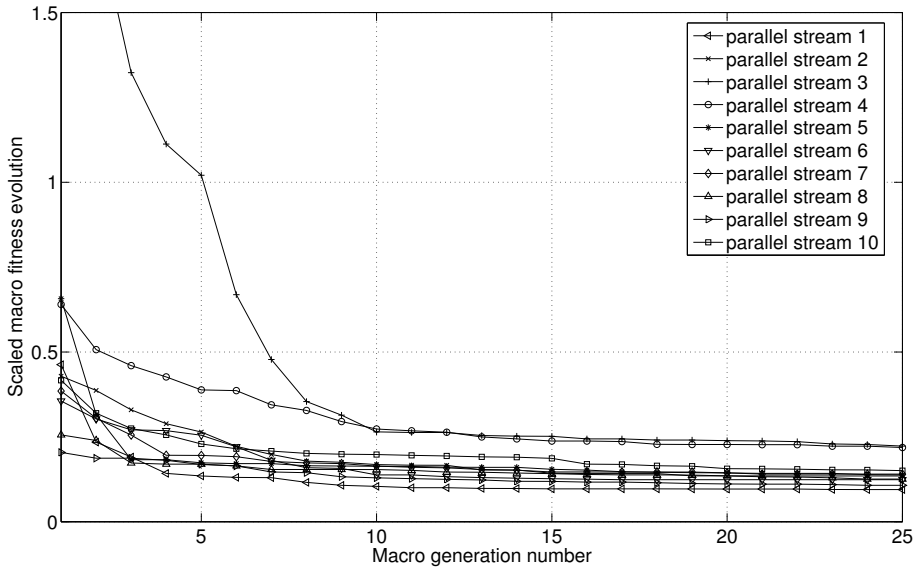


Fig. 4.18: Macro fitness evolution progress. Test case with $p : 21$, $\delta_{s/p} : [10, 300, 300, 10]$ and $f_{s/p}$:$[0.1, 0.11$,$0.22, 0.33]$

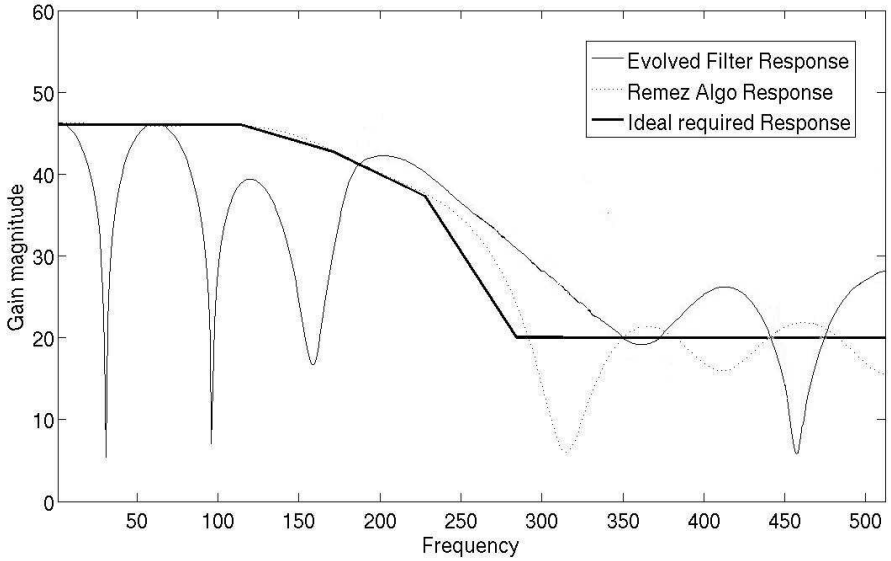


Fig. 4.19: Frequency response evolution progress. Test case with $p : 21$, $\delta_{s/p}$:$[199.9873$,$136.6667, 73.3460, 10]$ and $f_{s/p} : [0.22, 0.33, 0.44, 0.67]$

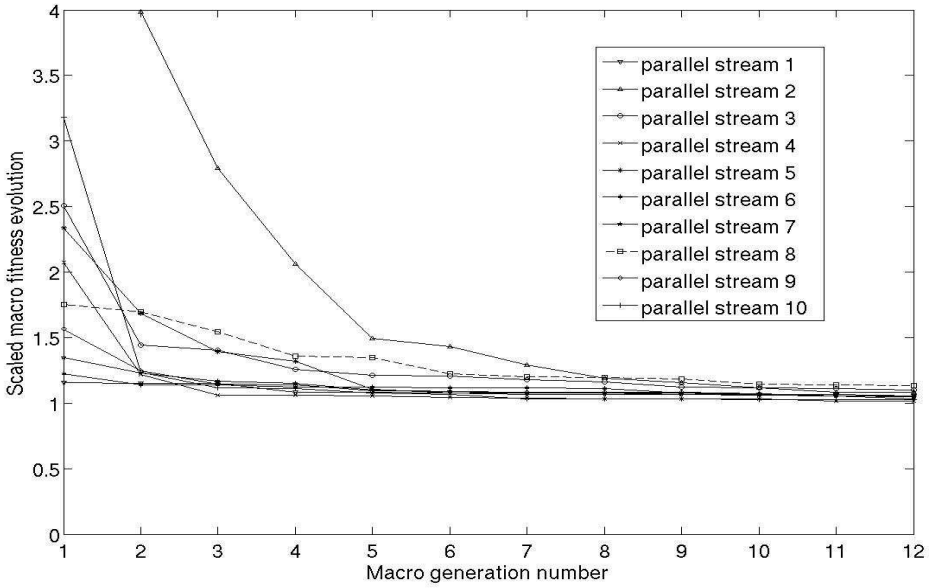


Fig. 4.20: Macro fitness evolution progress. Test case with p : 21, $\delta_{s/p}$:[199.9873, 136.6667,73.3460, 10] and $f_{s/p}$:[0.22, 0.33,0.44, 0.67]

would require very short reconfiguration times with an acceptable convergence precision. Table.4.1 presents the coefficient sets for varying filter lengths and specifications before and after evolution. The table also gives comparisons with the software evaluated coefficients using the approximate remez-exchange algorithm in Matlab. The Parks-McClellan algorithm uses the Remez exchange algorithm and Chebyshev approximation theory to design filters with an optimal fit between the desired and actual frequency responses [Rabiner *et al.* (1975)]. The algorithm maximizes the error between the desired frequency response and the actual frequency response characteristic. Filters designed this way exhibit an equiripple behavior in their frequency responses and are sometimes called equiripple filters [Rabiner *et al.* (1975)].

Fig.4.13 and Fig.4.14 show the evolution of the filter fitness over the genetic iterations for some sample filters. They demonstrate the evolution speed of the filter coefficients. From the results obtained, we conclude that the filter evolution performs best for sharp cut-off filters. Arbitrary response landscapes are evolved to a far better precision than other approximate algorithmic designs. The major strength in the entire ESoC heuristic however lies in the practicality and robustness of the design which can provide on-line self adaptation within the hardware with reduced iterative effort. The implemen-

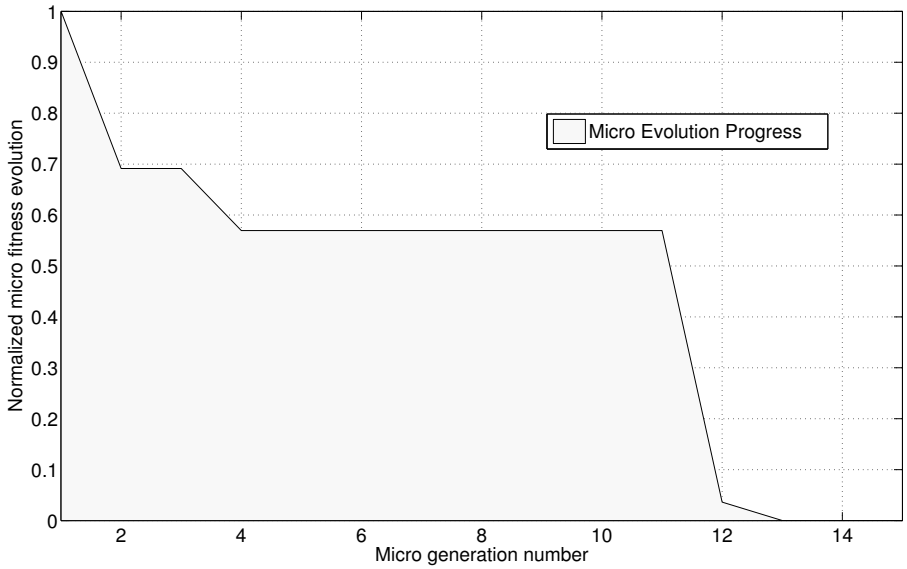


Fig. 4.21: Macro fitness evolution progress. Test case with $p : 21$, $\delta_{s/p} : [10, 300, 300, 10]$ and $f_{s/p} : [0.1, 0.11, 0.22, 0.33]$

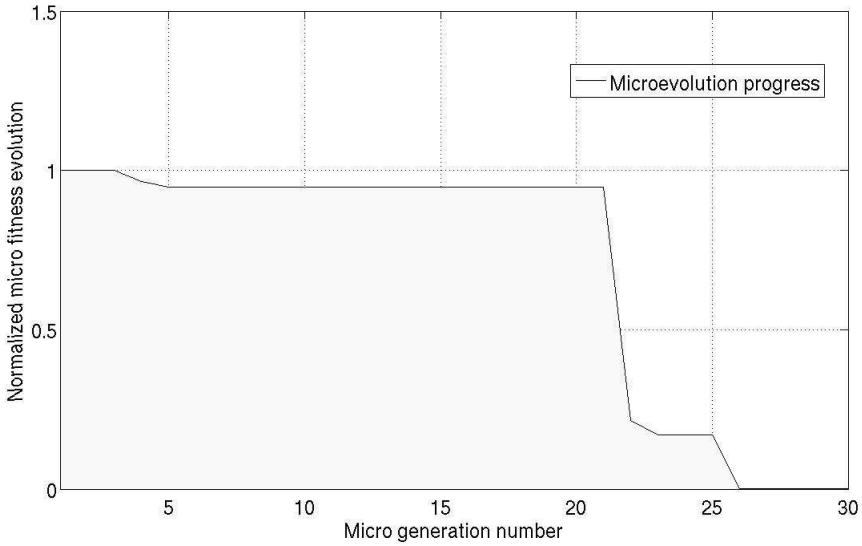


Fig. 4.22: Micro fitness evolution progress. Test case with $p : 21$, $\delta_{s/p} : [199.9873, 136.6667, 73.3460, 10]$ and $f_{s/p} : [0.22, 0.33, 0.44, 0.67]$

tation of the genetic method described in this work is practically feasible since it has simpler operations. This is easily implementable when compared to complex transformations and trigonometric function evaluations in other approximate algorithms like the Parks-McClellan. The complex operations pose a major challenge to the implementation feasibility of these other approximate algorithms.

An extension of this work may involve an evident method of reducing the hardware costs by restricting the filter coefficients to integers. This simplifies the filter design without mitigating its performance. This design approach has been explored in many recent research works [Sundaralingam and Sharman (1997)][Suckley (1991)][Redmill and Bull (1998)]. This has also involved exploration of the design using evolutionary algorithms. The current work has furthered this design simplification and approach. *The ingenuity of this work lies in the deterministic methods presented for generation of the genomic population and mutative variations while keeping faster convergence speeds for filter coefficients making them practical design choices with low implementation complexities and reduced latencies*. Also the hardware architecture (ESoC) presented in Sec.4.4 has a high degree of parallelism and scalability. In [Torresen and Vinger (2002)], a reconfigurable switching architecture for filters is presented which could be effectively used in tandem with the ESoC design of Sec.4.4 to make the evolutionary switching more effective. In [Haseyama and Matsuura (2006)], Haseyama and Matsuura present a novel method for tuning the filter coefficients to greater floating point precision. The method incorporates Simulated Annealing (SA) with the GA for coefficient quantization. The results demonstrated in Sec.4.5 from the ESoC design could be bettered by cascading a second stage of the GA-SA tuning routine presented in [Haseyama and Matsuura (2006)] if the ESoC filter needs to be extended to floating point precision. This may be seen as a design enhancement to the ESoC methodlogy.

4.6 Summary

The designs and results set out in this chapter clearly describe the implementation techniques for self-adaptive FIR filters with an arbitrary response. It proposes the use a modified genetic algorithm for the same. Deterministic heuristics in filter evolution at various

Table 4.1: Coefficient sets for filter test cases

$filt_{testcase}^{(1-5)}$	$h_j, halfset$	p
h_{evo}^j	[26,28,31,0,-1,25,38,-16,-59,-11,84]	21
h_{rem}^j	[-1,2,2,0,0,0,-6,-7,16,55,84]	21
h_{evo}^j	[14,-1,17,18,20,22,25,0,-1,85,105]	21
h_{rem}^j	[-126,62,57,58,62,68,74,79,84,87,108]	21
h_{evo}^j	[61,-41,-1,-36,-76,47,84,26,-130,-20,187]	21
h_{rem}^j	[1,-2,-2,1,0,1,10,11,-24,-84,186]	21
h_{evo}^j	[0,22,33,21,-24,-39,0,-10,-26,0,39]	21
h_{rem}^j	[55,-11,-25,-39,-43,-35,-18,1,17,26,38]	21
h_{evo}^j	[140,-50,157,-77,-33,-90,200,-92,0,-134,282]	21
h_{rem}^j	[-19,4,9,13,15,12,6,0,-6,-9,290]	21

Table 4.2: Coefficient evolution results over mac/mic generations for test filter cases - $f_{s/p}$ = stop/pass band cutoffs and $\delta_{s/p}$ = stop/pass band gains

$filt_{testcase}^{(1-5)}$	$f_{s/p}^{(norm)}$	$\delta_{s/p}$	$N_{Mac/Mic}$
h_{lim}^j	0,0.22,0.33,0.44,0.67,1.0	200,199.99,136.67,73.4,10,10	-
h_{evo}^j	0,0.22,0.33,0.44,0.67,1.0	200,199.99,136.67,73.4,10,10	65/26
h_{rem}^j	0,0.22,0.33,0.44,0.67,1.0	200,199.99,136.67,73.4,10,10	-
h_{lim}^j	0,0.1,0.1111,1.0	400,400,10,10	-
h_{evo}^j	0,0.1,0.1111,1.0	400,400,10,10	46/7
h_{rem}^j	0,0.1,0.1111,1.0	400,400,10,10	-
h_{lim}^j	0,0.22,0.33,0.44,0.66,1.0	10,10,106.67,203.3,300,300	-
h_{evo}^j	0,0.22,0.33,0.44,0.66,1.0	10,10,106.67,203.3,300,300	84/5
h_{rem}^j	0,0.22,0.33,0.44,0.66,1.0	10,10,106.67,203.3,300,300	-
h_{lim}^j	0,0.1,0.11,0.22,0.33,1.0	10,10,300,300,10,10	-
h_{evo}^j	0,0.1,0.11,0.22,0.33,1.0	10,10,300,300,10,10	70/13
h_{rem}^j	0,0.1,0.11,0.22,0.33,1.0	10,10,300,300,10,10	-
h_{lim}^j	0,0.1,0.11,0.22,0.33,1	300,300,200,200,300,300	-
h_{evo}^j	0,0.1,0.11,0.22,0.33,1	300,300,200,200,300,300	126/98
h_{rem}^j	0,0.1,0.11,0.22,0.33,1	300,300,200,200,300,300	-

stages show a praticality of implementation with fast convergence rates. A heuristic based limitation of the initial search space for the filter coefficients(tap weights) reduces the convergence latency. The maximum (mean) coefficient value, the filter cut-off and sampling frequencies and a $\frac{1}{x}$ functional model is used to restrict this search domain. The Genetic Algorithm (GA) used, incorporates a mutation of the strategy parameter based on a random normal perturbation which is modulated based on the coeffficient number with a sign selecting switch. The Evolutionary System on Chip (ESoC) design presented in Sec.4.4 provides a clear, detailed and structural architecture for the implementation of the evolutionary architecture on a standalone hardware platform. The ESoC architecture provides a high level of parallelism and scalability which typically lack in the design of evolutionary systems. The need for practical fault-tolerant adaptive filters is also addressed. The solution presented here is a complete hardware evolution model which is robust and speedily self-adaptive. With the advent of modern safety critical systems, ESoC might be the right direction to take. In this context, the proposed technique has significantly large practical relevance in many real life applications.

CHAPTER 5

A SEU TOLERANT DISTRIBUTED CLB RAM FOR IN-CIRCUIT RECONFIGURATION

5.1 SEUs and FPGAs

Reconfigurable computing and adaptive hardware is an emerging technology for many performance critical applications such as in defense and aerospace. After a discussion of the application of genetic algorithms to VLSI design in the previous chapters, we provide insights into the design optimizations using redundant logic elements within a specific reconfigurable hardware platform with a lot of promise to provide fast and reliable on chip-memories for many critical applications. Increasing use of Field Programmable Gate Arrays (FPGAs) in these has necessitated an enhanced degree of fault tolerance due to harsh operation environments. Many high reliability systems today entail error detection and correction as an integral component. A Single Event Upset (SEU) [Actel (December 2002)] is a common fault in SRAM-based FPGAs that operate in the presence of external radiation. These errors are particularly troublesome for memory elements as they store and propagate bits throughout the circuit [Mocanu and Oliver (1999)]. Besides error resilience, high performance memories demand faster operation speeds. Use of on-chip Configurable Logic Block (CLB) buffers for high speed single-port and dual-port RAMs is one popular solution for reduced latencies in memory design [XAPP464 (March 2005)][Reddy *et al.* (2005)]. CLBs are functional elements in FPGAs for constructing logic circuits [XAPP151 (October 2004)]. A Xilinx Virtex CLB is made up of *Slices* which contain logic cells, the interconnection and configuration of which provides a specific functionality. Blocks of RAM within CLBs can be used to implement on-chip synchronous memories [XAPP464 (March 2005)] for high operation speeds and reduced routing overheads. Fault tolerance of these Distributed RAMs (DRAMs) is a vital requirement in mission critical systems [Miron Abromovici (2001)]. Hardware redundancy for

fault tolerance is a resource intensive solution. This involves duplication of logic known as Doubling with Comparison (DWC) which has a redundant area investment, power consumption and pin count. Time redundancy, which involves repetition of a computation on the same circuit, is effective only in the detection of highly transient errors (with a performance penalty) and hence cannot be used to detect errors in storage elements[Reddy *et al.* (2005)]. Error detection codes such as Hamming codes are used in Xilinx Virtex Devices [XAPP645 (February 2004)] to detect SEUs in SRAM configuration memory. However in [XAPP645 (February 2004)] there is no consideration of the fact that SEUs affecting the Hamming logic itself may make the entire system unreliable. In this chapter we evaluate a technology dependent optimization that exploits the presence of unused blocks of RAM (BlockRAM) and Tri-State Buffers (BUFTs) in the Xilinx Virtex architecture. The aim is to provide a *high speed, SEU-tolerant, on-chip memory using redundant elements on the Virtex FPGA*. Single Event Transients (SETs) are short duration metastable glitches which tend to corrupt logic states. SETs have an effect on combinational outputs based on load capacitances. Transient pulses higher than half V_{dc} (the rail voltage) tend to propagate in buffers. However the probability of their occurance and propagation in limited by the critical charge induction of 0.02pC or a radiation intensity of $2Mev - cm^2mg$ for the worst case [Course (2002)]. BUFTs in the Virtex architecture, although impervious to SEUs are susceptible to SETs but with a very low probability. These are a topic of transient analysis and short duration stability and are not considered in our discussion of buffer (BUFT) radiation stability in this chapter. SEUs or soft errors have a ground level upset rate of 2×10^{-12} upsets/bit-hr [Normand (December 1998)] and probabilities of Multiple Event Upsets (MEUs) tend to down shoot from this low value to sub-atto rates. BUFT configuration is hardwired with no stored configuration bits. Hamming is a technique for detecting and correcting single bit soft errors in transmitted data [Ou (2004)]. Hamming Error Correction Control (ECC) requires that j parity (p) bits (or check bits) be transmitted with every i data (d) bits. The algorithm is called a $(i + j, i)$ hamming code, because it requires $(i + j)$ bits to encode i bits of data. The goal of the Hamming code is to create a set of parity bits that overlap such that a single-bit error (the bit that is logically flipped in value) in a data bit or a parity bit can be detected and corrected. Appendix. C describes the hamming code and ECC in detail. In the following sections we describe a fault tolerant memory design incorporating the Hamming ECC followed by

its applications in Reconfigurable hardware.

5.2 Distributed CLB RAMs

BlockRAM in a Xilinx Virtex FPGA is a 18Kb dual-port RAM which can be configured with various data/address aspect ratios such as $16K \times 1$ bit, $4K \times 4bits$, $2K \times 9bits$ or $512 \times 36bits$. Xilinx Virtex-II Pro FPGAs incorporate 444 blocks of such RAM [XAPP151 (October 2004)] that can be used as lookup tables to store data and serve as on-chip memory. With two BlockRAMs occupying as much area as 24 CLBs or 192 LUTs [XAPP151 (October 2004)], the BlockRAM cells may not seem to be a silicon-area efficient solution for on-chip memories at first. However, since 444 of these BlockRAMs are anyhow integrated on the chip die but not used in typical designs, they can be considered free [XAPP258 (Januray 2005)]. FPGA Tri-State Buffers (BUFTs) in the Virtex architecture are hard wired AND-OR logic elements which are often unused in logic designs, Fig.5.2. In this chapter, we demonstrate a cross connection of these BUFTs to produce SEU proof Hamming functions for Error Correction Control of DRAM memories. The advantage of using BUFTs for designing the ECC logic is that their functionality is not based on the contents of any SRAM cells (no configuration bits), which may get upset. The only aspects of the BUFTs which are controlled by configuration memory cells are the routing pips which connect them together [C Carmichael and Caffrey (1999)]. Upsetting one of these cells would only result in temporarily disconnecting one of the inputs or outputs of the BUFTs. Such an upset would not affect the output of the ECC logic. In fact, this ECC design is completely impervious to single upset (SEU) failure. Use of these BlockRAMS for on-chip buffers and BUFTs for ECC is the key contribution of this chapter. These do not reduce the logic or RAM in the circuit as the entire methodology is based on unused BlockRAMs and BUFTs. As a byproduct of this fault tolerant memory design, we also propose a novel in-circuit reconfiguration methodology for Fast Dynamic Reconfiguration of FPGA Devices made feasible by the re-programmability and read-back capabilities of the Xilinx Virtex architecture. This run-time reconfiguration makes FPGA designs fault tolerant and bestows them with functional evolution. An application of this methodology could be in Network on Chips (NoCs) which require a large number

of high speed on chip buffers.

5.3 Fault Tolerant DRAM

The Error Correction Control (ECC) functions described in our DRAM model use hamming codes. This involves transmitting data with multiple check bits and decoding the

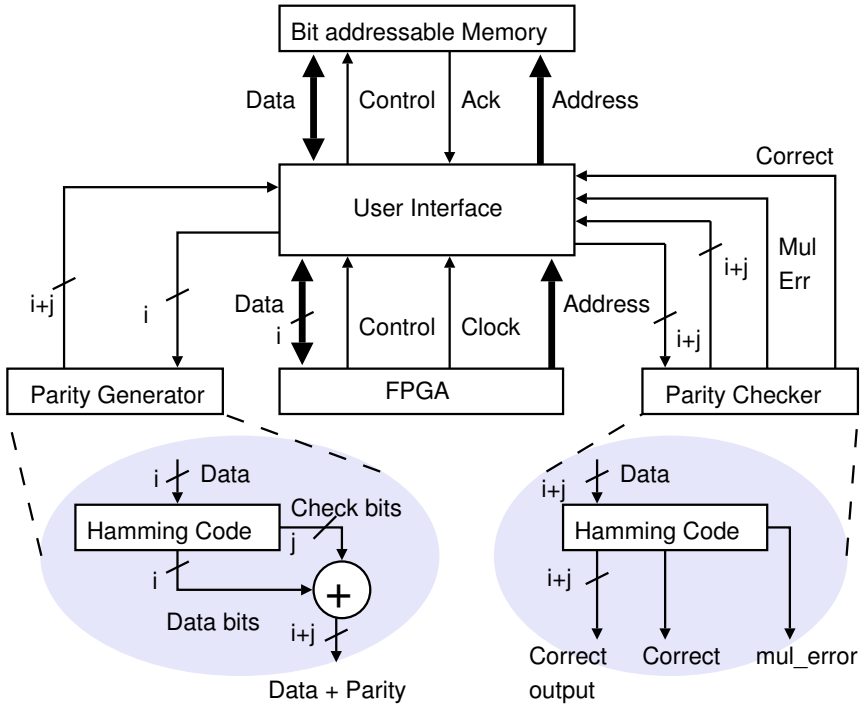


Fig. 5.1: The Proposed Hamming ECC Fault Tolerant DRAM Memory Model

associated check bits when receiving data to detect errors. The check bits are parity bits generated in parallel by XORing certain bits in the original data word. If bit error(s) are introduced in the codeword, several check bits show parity errors after decoding the retrieved codeword. The combination of these check bit errors display the nature of the error. In addition, the position of any single bit error is identified from the check bits. Fig.5.1 shows the system architecture of the proposed memory design. The *bit addressable memory* can read/write a single bit data from the given address. This incorporates the use of CLB RAMs. The *parity generator* takes as input i bits of data and gives back a $(i + j)$ bits of a data+parity word. The parity checker takes a $(i + j)$ bit word and checks it for errors. If no errors are detected, the *correct pin* is set. In case of a single bit

error, the pin is reset and the corrected data is sent to the user as well as to the memory for correction. For multiple errors, the *correct pin* is reset, mul_error is set and the data output is set to high impedance.

Modeling with Buffers: In any self correcting mechanism, the error correcting component should not be susceptible to errors. As described in Sec.5.2, buffer gates in the FPGA (BUFTs) are not susceptible to soft errors. We have used buffer gates entirely to model the parity generator and the hamming parity checker. Buffer gate $notif1$ was used to construct all the other required gates. The buffer gate $notif1$ and the wor construct in verilog were used to implement the nor gate, which is universal. Using this, all other gates were constructed.

Two input NOR gate construction: Let the inputs be a,b. To obtain $(a+b)'$, we pass $a+b$ to $notif1$, with the control being 1. The verilog code is as follows.

```
module my$_$nor(a,b,c)
     input a,b;
     output c;
     wor temp;
     assign temp = a;
     assign temp = b;
     notif1 n1(c,temp,1);
end module
```

The If-Else construct: The if-else construct is needed to perform toggling of bits if found to be erroneous. $notif1$ gates are used and a wor verilog construct is required. The verilog code is as follows.

```
module my$_$if(a,b,c) // if (~b)
     input a,b;       // then return
     output c;        // a else
     wor c;           // return (~a)
     wire na,nb;
     notif1 n1(na,a,1);
```

```
    notif1 n1(nb,b,1);
    notif1 n1(c,a,nb);
    notif1 n1(c,na,nb);
end module
```

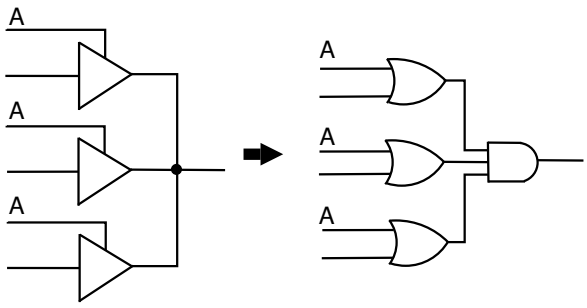
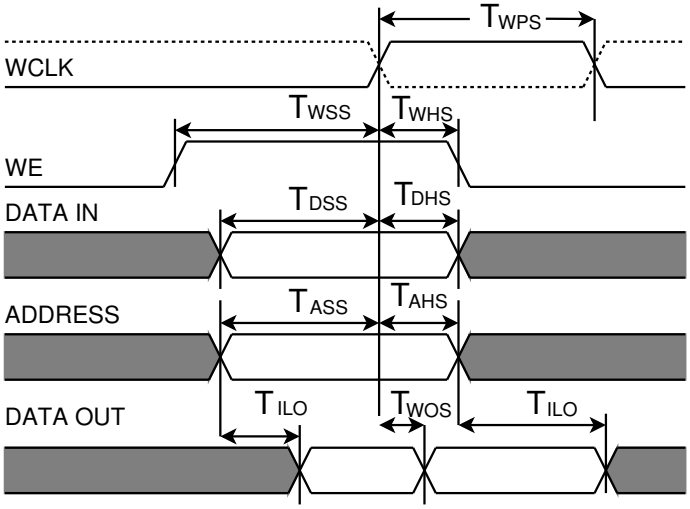


Fig. 5.2: The Virtex Tri-State Buffers: Equivalent to Wired AND-OR Logic.

Operation: The model proposes a fault tolerant memory system that can detect and correct single bit errors and detect multiple errors. The user interface provides an abstraction of the bit addressable memory as a byte addressable memory and takes care of error detection and correction. The whole process is transparent to the user. The basic operations are read, write and correct. The order of priority is $write > correction > read$. Whenever the user wants to write to memory, the address and data are supplied and the $write\_enable$ pin is set. The parity generator supplies the $(i + j)$ bit word (data + parity) to be written into memory. For a read operation, the user provides the address and sets the $read\_enable$ pin. Now if the memory is not busy. i.e no write/correction is being performed. A $(i + j)$ bit-word is read. The parity checker finds out the validity of the word. In case of no error, the correct pin is set, the i bit data is extracted and given to the user. If a single error has been detected, the correct pin is reset, the correction process is triggered and then the user is supplied with the corrected value. For multiple errors, the correct pin is reset, mul_error pin is set. As the user should not be provided with the wrong value, the output is set to high impedance. The read-write timing diagram for the fault tolerant memory model is shown in Fig.5.3.



Configure dev with encoded bit stream. Copy it into Mem: i [D]

Compute Hamming ECC and Syndrome Matrix for Correction.

Fig. 5.3: The DRAM ECC Cycle Clock Timing Diagram.

5.4 Application: In-Circuit Reconfiguration

A key application of the proposed fault tolerant memory design is an in-circuit reconfiguration methodology for LUT based FPGA designs. Applications which abundantly employ reconfigurable systems, for example space applications, demand a severe necessity for fault tolerant architectures. Once in space, FPGAs are susceptible to several kinds of errors like: Single Even Upset (SEU), Single Hard Error (SHE), Single Event Burnout (SEB), Single Event Gate Rupture (SEGR), Single Event Effect (SEE) and Multiple Bit Upset (MBU). Except SEUs and MBUs, all the others cause permanent damage to the hardware. As the probability of multiple errors occurring in the same system is infinitesimal[Normand (December 1998)], the objective of our in-circuit reconfiguration model is to correct a single bit error (SEUs) and to detect multiple errors using the memory model described in Sec.5.3. For mitigating SEUs in FPGAs, numerous models have been proposed in the literature, one of the most popular among them is the Triple Modular Redundancy (TMR) [Kastensmidt *et al.* (2005)] which creates two extra copies of every data element. To verify the correctness, a voter circuit is used which outputs the value that is stored by at least two of the three copies. In case two of the copies are wrong, the result of the voter circuit is erroneous. Thus, this model is unable to detect multiple

errors in two different modules. Moreover, having two extra copies is costly. Another model (Cluster-based Detection of SEU-caused Errors in LUTs of SRAM based FPGAs) uses simple parity to detect errors. Though this is the cheapest method to detect errors, it can detect odd-errors only [Reddy *et al.* (2005)]. Yet another model [XAPP645 (February 2004)] performs single error correction and double error detection using Hamming Codes. In this model, the errors detected are not corrected back into the memory. If another error occurs in the same word, the model will detect it as a case of multiple error though this condition can be prevented. A major problem with all these designs, as mentioned in Sec.5.2, is that the ECC logic itself may not be tolerant to radiation upsets. This necessitates a robust ECC design impervious to SEUs. Our In-circuit Reconfiguration design (Fig.5.4) which uses the BUFT and BlockRAM memory model described in Sec.5.3 achieves this error resilience. The Hamming Error Correcting Code (ECC) [Appendix.C] is capable of correcting single bit errors and detecting multiple errors. The above process of handling error(s) is done by using additional check-bits that are clubbed together with data-bits. An 8 bit data word, for example, requires 4 parity bits to handle single bit errors and an extra bit to detect multiple errors. An example of the basic algorithm is as follows - All bit positions that are powers of two are used as parity bits (positions $1, 2, 4, 8, 16, 32, 64$ etc). All other bit positions are for the data to be encoded (positions $3, 5, 7, 9, 10, 11, 12, 13, 14, 15, 17$ etc). The parity bit at position 2^k check bits in positions having bit k set in their binary representation [Hamming (2008)]. For example, 1011 is encoded into 01100110. Even parity is followed in the system description to follow. The ftDRAM with the BUFT ECC which is resilient to SEU errors is used to store FPGA device configuration data. The stored ftDRAM configuration is error tolerant as the memory is protected with a hamming ECC built using hardwire BUFTs in the FPGA. This stored configuration bit stream is compared with the read back data from the FPGA device at regular intervals. In case of an SEU in the FPGA device logic, the Hamming ECC routes corrected configuration data to be written back to the device. The time utilized for the reading of the device configuration, its correction and writing back is called the Mean Time To Repair (MTTR). This In-Circuit Reconfiguration methodology is simple as well as error tolerant.

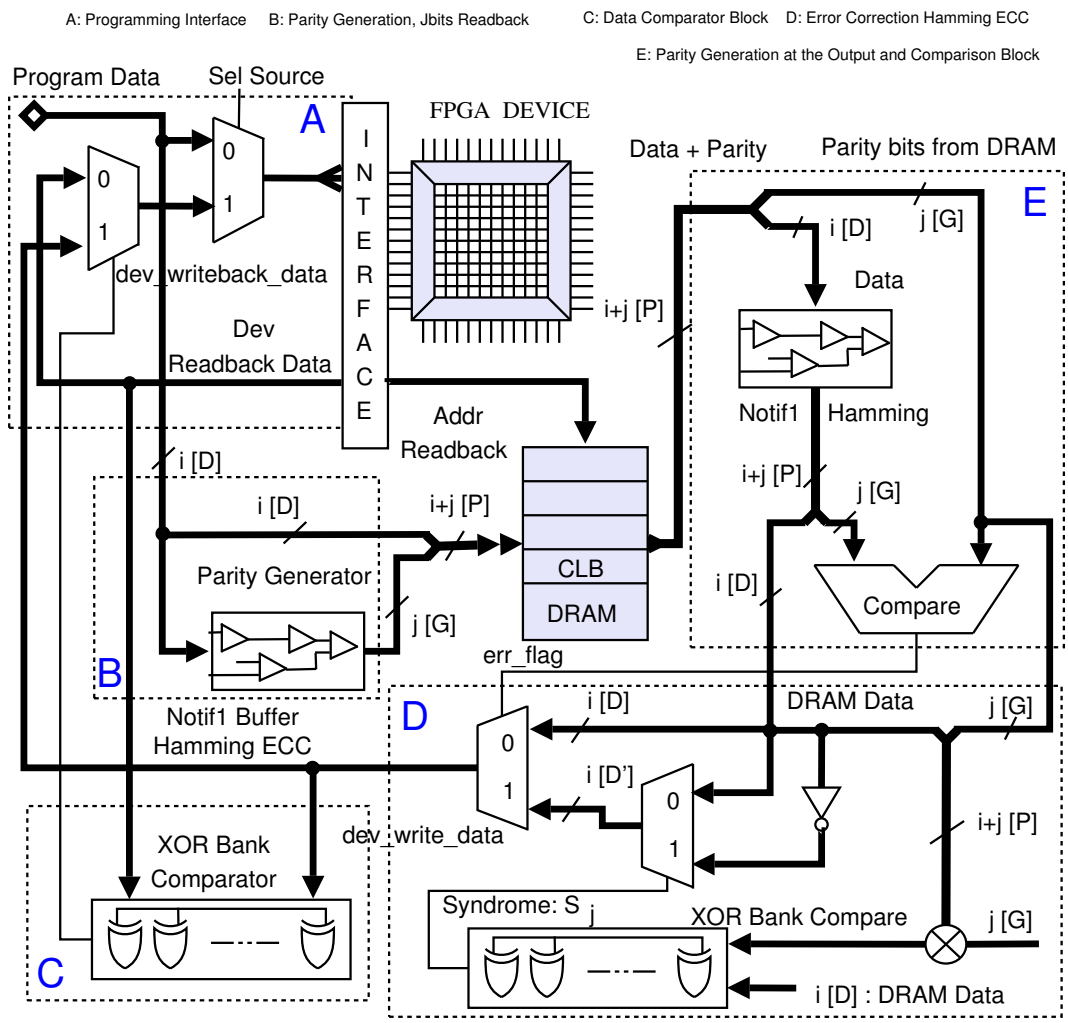


Fig. 5.4: The In-Circuit reconfiguration methodology using the ftDRAM.

5.4.1 Design Specifics

Fig.5.4 shows the system design of the in-circuit reconfiguration of FPGAs for fault tolerance. FPGA configuration bits are read back for SEU correction using In-Circuit Reconfiguration (IR) as described in [Hoflich (2005)][Bolchini *et al.* (2007)] and [Lima *et al.* (2003)]. *Block A* in the design distinguishes fresh FPGA configuration from IR for SEU mitigation. The hamming ECC uses a codeword that is an (un)ordered union of the parity and the data bit vectors. It is described by the ordered set $(i+j, i)$. As an example, let us consider a hamming codeword with i data bits and j parity bits concatenated as $(d_i, d_{i-1} \ldots d_1, p_j, p_{j-1} \ldots p_1)$.

Parity bits Given the data set $D = (d_i, d_{i-1} \ldots d_1)$ of i elements, we can define a j element parity set $P = (p_i, p_{i-1} \ldots p_1)$ with a minimum number of check bits for single bit error-correction constrained by Eq.(5.1).[XAPP645 (February 2004)]

$$i + j + 1 \leq 2^j \tag{5.1}$$

The parity set may be defined as a modulo-2 sum of a preemptive subset of randomly chosen k elements $(d_x, d_y, d_z \ldots k\ elements)$ from D where $\binom{i}{k} \geq j$. This parity bit generation and regeneration is done in *Blocks B and E* respectively in Fig.5.4.

Encoding The parity matrix, $[P]$, which encodes a given data of i bits into a codeword, is expressed in Eq.(5.2)

$$[P] = [D] \bullet [G] \tag{5.2}$$

where, $[D]$ is the i bit data matrix and $[G]$ is the generator matrix. The $[G]$ matrix consists of an identity matrix $[I]$ and a creation matrix $[C]$ of order $i \times (i+j)$ and is given by $[G] = [I : C]$. The creation matrix is a concatenation of column vectors containing a 1 in the row corresponding to each of the k data bits included in its computation and a 0 is all other rows. The encoded data is stored in the CLB RAM designed in Sec.5.3 without the ECC logic.

Decoding Upon receiving a dynamic reconfiguration request by the FPGA after a given number of operation cycles, the Hamming codeword stored in the DRAM is compared with the read back configuration data in *Block C*. Decoding of the Hamming ECC in a situation without errors is straight forward. Disregarding the parity bits leaves us with the data. For example, if an an encoding produces 1011010 as the hamming vector $[P]$ in a $(7, 4)$ encoding scheme we have after pruning out the first 3 parity bits, 1010 as the 4 bit data. In case of an error, which is detected by recomputing the parity bits, we construct a $j \times (i + j)$ parity check matrix $[H]$ such that the j^{th} row contains a 1 in the j^{th} column corresponding to the j^{th} parity bit and all of the k data bits included in its parity calculation. The parity check matrix is ordered as $[H] = [C : I]$. Multiplying the matrix $[H]$ of order $j \times (i + j)$ with the data code matrix of order $(i + j) \times 1$ produces a matrix $[S]$ of order $j \times 1$ called the *syndrome*. In other words, the data code vector multiplies with the transpose of the generator matrix to produce the syndrome vector. This is shown in Eq.(5.3)

$$[S] = [D, P] \bullet [G^{'}] \tag{5.3}$$

If all the elements of the syndrome vector are zeros, no error is reported. Any other non-zero result reflects the column in which the error has occurred in $[H]$. If the syndrome vector elements correspond to the elements of the q^{th} column of $[H]$, it indicates an error in the $(q - j)^{th}$ data bit which is corrected by inverting the vector corresponding to the syndrome as shown in *Block D* in Fig.5.4. This corrected vector is passed on to the XOR comparator to compare with the read back device bit stream. In case of an SEU error, the DRAM configuration data is written back to the FPGA in the following device configuration cycle else the existing FPGA configuration data is retained.

The Reconfiguration algorithm using the fault tolerant DRAM is shown illustrated in 5.4 and described in Algorithm. 5. The algorithm requires a FPGA read back capability and a configuration map scan using an SelectMAP/ JTAG interface. A Mean-Time-to-Repair (MTTR) requirement of at least $m + n + p$ should also be ensured where, n is the number of clock cycles to read back data from the FPGA device, m to read BlockRAM data and p to write back device configuration data. A $capture\_enable$ flag is raised to initiate correction and the $config\_data$ pointer holds the address of the last memory location containing the device configuration data.

Algorithm 5 DYNAMICPARTIALRECONFIGURATION(n)

Require: In-circuit FPGA Configuration Read back.

Ensure: SelectMAP/ JTAG Boundary Scan/ μController.

Ensure: $MTTR \geq m + n + p$

for all x such that $0 \leq x < capture\_enable$ **do**
  for all y such that $y \leq config\_data$ **do**
    $[D]_y$ = [$d_i, d_{i-1} \ldots d_1$]
    $[G]_y$ = [$p_j, p_{j-1} \ldots p_1$]
    $[P]_y$ = [$D_y : G_y$]
    mem_data_y = $[P]_y$ //write memory
  end for
  for all z such that $z \leq mem\_data$ **do**
    if $[G]_z = G_z^{mem}$ **then**
      return $err\_flag = 0$
      return Break;
    end if
    $err\_flag = 1$
    S_z = $P_z \bullet G_z^T$
    for all y such that $y \leq i$ **do**
      if $[S]_z == D_z$ **then**
        return Break;
      end if
      $dev\_write\_data = D_z$
    end for
  end for // ECC Cycle
  if $err\_flag = 1$ **then**
    return $dev\_write\_back = dev\_write\_data$
    return Break;
  end if
end for

mem_data_y holds the parity matrix. z is also iterated upto the $config\_data$ pointer and raises an error flag if there is a mismatch in the generator matrix calculated by the BUFT ECC (G_z) and that stored in the memory (G_z^{mem}). In case of an error, if the syndrome matrix, S_z, generated from the read back data matches the stored device configuration data, there is no need for correcting that memory location. In case there is a mismatch, dev_write_data is updated to D_z the original configuration data. This forms the correction part of the algorithm (ECC cycle). After error correction, if the err_flag is set to 1 at the end of the ECC Cycle ($i.e$ at the end of $m+n$ clock cycles), new data dev_write_back to be written back to the device is updated to the corrected write data dev_write_data.

5.5 Experimental Results

Table 5.1: Test cases with variable data word lengths for 1KB ftDRAM on Xilinx Virtex II XC2VP2-7FG256 Device

D-Width	# Slices	# Slice FF	4-LUTs	BUFTs	$T_{WCLK-min}$	$T_{CLKH-min}$
8 Bits	588	351	1022	30	6.237 nS	3.522 nS
16 Bits	645	365	1193	46	6.485 nS	3.370 nS
32 Bits	708	389	1307	78	6.876 nS	3.407 nS
48 Bits	756	414	1349	106	7.203 nS	3.422 nS
64 Bits	780	431	1382	118	7.082 nS	3.432 nS

Table 5.2: Test cases for variable memory sizes with and without error checks

Mem Size	ERR CHK	Slices	Slice FF	4 i/p LUTs	Min Period	Max T_{hold}
8×1KB	no	458	286	831	5.206 nS	3.290 nS
8×1KB	yes	588	351	1022	6.237 nS	3.522 nS
8×2KB	no	766	484	1384	5.727 nS	3.290 nS
8×2KB	yes	911	547	1663	7.370 nS	3.508 nS
8×4KB	no	1462	768	2132	5.854 nS	3.290 nS
8×4KB	yes	1543	940	2793	7.376 nS	3.522 nS

The fault-tolerant memory model using BlockRAMs and BUFTs was simulated on a Xilinx Virtex II Pro platform FPGA Device XC2VP2-7FG256. The Hamming ECC was

implemented on the device using BUFTs and Table.5.1 shows the BUFT usage and clock speeds for different data word sizes used by the ECC to generate parity information. For word lengths beyond 64 bits, the BUFTs on the Virtex device were insufficient. However, a 64 bit data word which constitutes about 16 BlockRAM addresses for a $4K \times 4bits$, can be used to generate parity information for clustered memory addresses. This provides error tolerance for data clusters in about 16 memory locations. Assuming this kind of operation speed we have a typical clock period, T_{WCLK}, of $7.082ns$. Fig.5.5 shows the operation cycles (device read, memory write, memory read and device write cycles) of a typical In-circuit Reconfiguration (IR) model. Assuming an asynchronous memory read, based on this operation cycle delays, we can estimate a Mean Time To Repair (MTTR) of about $p + 2$ clock cycles, where p is the time taken to write back the data into the device. Not including the programming time p in the MTTR computation, we arrive at an approximate MTTR order of about $14.164ns$. This repair time is quite less compared to a configuration power cycle of about $20ms$ [C Carmichael and Caffrey (1999)] for typical FPGA configurations. Also, typical off-times in FPGA space circuits can have a tolerance of a few nano-seconds[Normand (December 1998)]. This quick MTTR cycle would mean SEU tolerant FPGA logic which can be repaired at regular intervals to ensure correct operation. This is built with the help of unused device elements (BlockRAMs and BUFTs) along with lower off-times. Table.5.2 shows FPGA area occupancy and device operation speeds for varying memory sizes on the FPGA. It shows a memory operation with a clock period of about $7.376ns$ for a $4kb \times 8bit$ DRAM.

5.6 Future Work

The technique proposed in this chapter is limited to the hugely popular Xilinx Virtex families of FPGAs which has flexible read back capabilities, large amount of unused BlockRAM and radiation immune buffers (BUFTs)[C Carmichael and Caffrey (1999)][Xilinx (2005)][XAPP258 (Januray 2005)]. The technique seeks to address SEU mitigation in radiation intensive environments. Multiple upset (or MEU) probabilities in practical environments are infinitesimal [Normand (December 1998)]. However, they have a finite probability of occurrence. The in-circuit reconfiguration technique proposed in this work

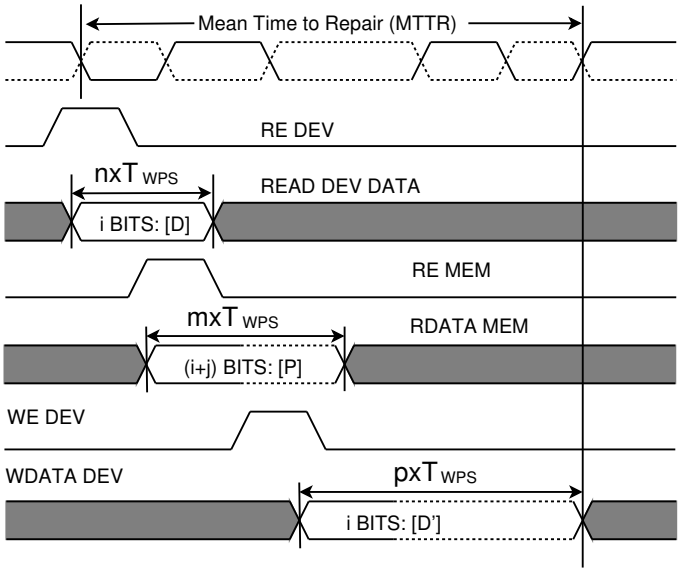


Fig. 5.5: The In-circuit Reconfiguration Timing Diagram.

helps in the detection of these using the ECC to raise an error flag which may be further used to do offline correction. The choice of Hamming codes for an ECC procedure is owing to the fact that this provides MEU detection and a simpler XOR implementation using the BUFT gates. Comparison of different ECC implementations, online MEU correction and porting of the technique to various FPGA platforms is the future direction of this work.

5.7 Summary

This chapter describes the top-down design of a robust, SEU tolerant Distributed RAM (ftDRAM). The key benefit of the proposed memory design is that the it is built using BUFTs and BlockRAMs which are resources left unused in typical FPGA designs. We make use of the BUFTs to construct the Hamming corrector used an ECC the ftDRAM design. This makes the ECC logic SEU proof unlike other designs in the literature which make use of radiation prone CLBs to construct the ECC logic. Moreover, the use of BlockRAMs within Configurable Logic Blocks (CLBs) for storing the FPGA configuration data provides a very fast on-chip memory. As an application of the ftDRAM model, an In-Circuit Reconfiguration methodology for SEU mitigation in systems exposed to high intensity radiations is demonstrated. The stored configuration data in the ftDRAM is

compared with device bit streams read back from the FPGA at regular intervals of time. An error in the device configuration is detected based on this comparison. This reconfiguration model also enables us to perform online error correction by writing back the correct configuration data from the ftDRAM. It is also demonstrated that the In-Circuit methodology to has an estimated Mean Time to Repair (MTTR) of about $14.164ns$ for a $4kb \times 8bit$ DRAM.

CHAPTER 6

EPILOGUE

6.1 Wrap Up

The chapters in this thesis have looked at a popular optimization technique called the Genetic Algorithm (GA). After an introduction about the basic methods in GAs *viz.* IP Generation, Variation (mutation and crossover), Evaluation and Selection, an emphasis on the problem identification and modeling was placed in Chap.1. The importance of reliable and accurate designs was outlined and the application of the genetic methods to optimization at various levels of abstraction was demonstrated using the first three problems:

1. **Genetic transistor level optimization:** A genetic approach to gateless custom VLSI design flow.
2. **Genetic gate level optimization:** Vector parts and genetic search methods for PDPE in digital circuits.
3. **Genetic system level optimization:** Hardware based genetic evolution of FIR filters.
4. **System level Reconfigurable Design:** A SEU tolerant distributed CLB RAM for in-circuit reconfiguration.

Each of the these problems were modeled appropriately for a heuristic GA and circumstantial approximations were imbibed. Having given different design schemes for the GA, the thesis wrapped up with a fourth optimization aside using reconfigurable hardware. In the fourth problem, novel insights into exploitation of FPGA hardware was demonstrated. The intent was to provide a reliable and robust on-chip memory system for many critical hardware applications. The upside to this design was the usage of redundant logic elements in the FPGA platform for the memory design as well as the error correction control.

6.2 Conclusions

Individual conclusions from the project parts are analyzed in their respective chapters. An overall value addition of this work is in providing new methodologies for design at different abstraction levels. The various concepts and examples presented in this thesis demonstrate two major facts

1. *Genetic Algorithms (GA) can provide reliable and optimal solutions to various problems in VLSI design using problem specific heuristics.*
2. *Exploitation of redundant hardware resources on reconfigurable platforms can provide useful logic functionalities which may often be robust and fast.*

Problem abstraction and modeling appropriately for a GA can be a major challenge in evolutionary design optimization. Also, a choice of an appropriate fitness function is quintessential for an optimal convergence of the GA. These two facts were exemplarily shown using genetic cmos designs and the fir filter in Chap.2 and Chap.4 respectively. These and the power virus problem in Chap.3 show that a heuristic variation aptly suited for the problem at hand can speed up the GA. It can also lead to the exploration of a larger search space eventually leading to the avoidance of getting stuck at local optima. Besides the problem demographics and hardware dynamics exploited by the GA, designs on reconfigurable platforms (like FPGAs) can be made more reliable and robust using redundant on-chip logic elements which are unused in typical designs. The SEU tolerant distributed CLB RAM described in Chap.5 has shown such a potential.

6.3 Future Work

Design exploration of the GA at the transistor, circuit and the system levels could be leveraged more beneficially if the problem heuristics, models and evaluations are directed by manufacturing and fabrication parameters. A hardware application of a GA is strictly dependant on these factors for accurate modeling. Future work on this may account for realistic inputs from foundries and fields. The genetic cmos design and fir filters could be enhanced by inputs from these. The firGA problem in Chap.4 relies heavily on a random normal simulator which has been designed on hardware. As an extension of this work

other probability distribution functions could be used in the variation step. Also the evaluation methodology used in the firGA design relies on peak detection and approximate IO peak value averaging for gain computation. This was to avoid the fourier transforming and inverse transforming of the filters response for gain computation. A new accurate, yet a simple methodology could be developed for the gain computation in the genetic design.

The power virus problem in Chap.3 can be extended by the application of the proposed techniques at different abstraction levels *viz* the RTL and high level designs. Early and late power estimates may benefit hugely from the proposed partitioning and genetic methods. In Chap.5, the proposed ftDRAM design uses a simple hamming code to keep hardware usage under control. The hamming ECC takes up a larger number of bits for encoding compared to a few other existing ECC techniques. As an extension of this work, the fault tolerant RAM design could be extended by using better error correction control mechanisms.

APPENDIX A

CMOS LOGIC DESIGN AND THE IRSIM SIMULATOR

A.1 Digital CMOS Logic Design

CMOS refers to both a particular style of digital logic design and a set of manufacturing processes used to implement the circuitry on an integrated circuit (IC). CMOS circuitry dissipates less power and is denser than other implementations having the same functionality. As its advantages grow over other process technologies, CMOS has seen more and more of industrial focus and adaptation [CMOS (2008)].

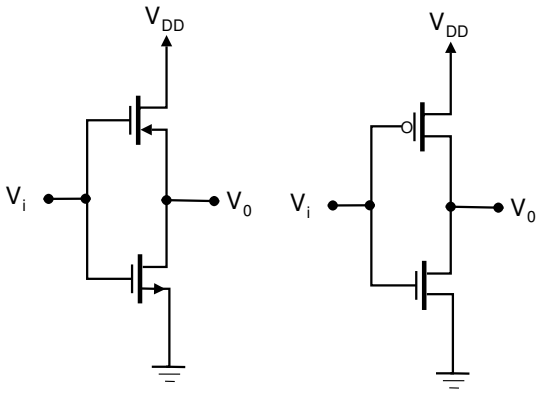


Fig. A.1: Static inverter using different n and p transistor representations

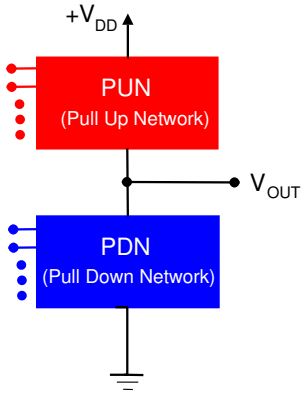


Fig. A.2: Pull up-down networks in the static inverter

A.1.1 Static CMOS Logic Design

CMOS gates are all based on the fundamental inverter circuit shown in Fig.A.1. Note that both transistors are enhancement-mode MOSFETs; one N-channel with its source grounded, and one P-channel with its source connected to $+Vdd$. Their gates are connected together to form the input, and their drains are connected together to form the

output. The two MOSFETs are designed to have matching characteristics. Thus, they are complementary to each other. When off, their resistance is effectively infinite; when on, their channel resistance is about 200 Ω. Since the gate is essentially an open circuit it draws no current, and the output voltage will be equal to either ground or to the power supply voltage, depending on which transistor is conducting [Bigelow (1996)].

When input A is grounded (logic 0), the N-channel MOSFET is unbiased, and therefore has no channel enhanced within itself. It is an open circuit, and therefore leaves the output line disconnected from ground. At the same time, the P-channel MOSFET is forward biased, so it has a channel enhanced within itself. This channel has a resistance of about 200 Ω, connecting the output line to the $+Vdd$ supply. This pulls the output up to $+Vdd$ (logic 1). When input A is at $+Vdd$ (logic 1), the P-channel MOSFET is off and the N-channel MOSFET is on, thus pulling the output down to ground (logic 0). Thus, this circuit correctly performs logic inversion, and at the same time provides active pull-up and pull-down, according to the output state. This is as shown in Fig.A.2. The

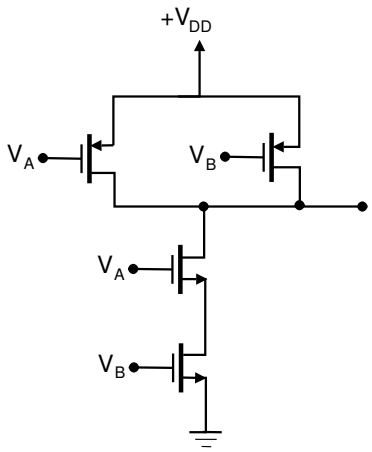


Fig. A.3: Static CMOS NAND

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

A	B	$\overline{A}$	$\overline{B}$	$\overline{A+B}$	$\overline{A.B}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

Fig. A.4: Truth table for NAND

CMOS concept can be expanded into NOR and NAND structures by combining inverters in a partially series, partially parallel structure. The circuit in Fig.A.3 is a practical example of a CMOS 2-input NAND gate and that in Fig.A.5 is a 2-input NOR gate. In Fig.A.3, if both inputs are low, both P-channel MOSFETs will be turned on, thus providing a connection to $+Vdd$. Both N-channel MOSFETs will be off, so there will be no ground connection. However, if either input goes high, that P-channel MOSFET will turn off and disconnect the output from $+Vdd$, while that N-channel MOSFET will turn

on, thus grounding the output. The truth table for the NAND and NOR gates are shown in Fig.A.4 and Fig.A.6 respectively [Bigelow (1996)]. This type of logic design is what

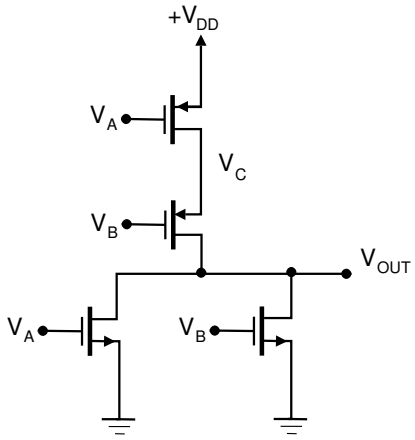


Fig. A.5: Static CMOS NOR circuit

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

A	B	$\overline{A}$	$\overline{B}$	$\overline{A.B}$	$\overline{A+B}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Fig. A.6: Truth table for NOR

constitutes the static CMOS logic design. A variation of this is the dynamic or clocked CMOS logic design. This style of logic design relies on a clock for its operation and is explained in the following section.

A.1.2 Dynamic CMOS Logic Design

Dynamic CMOS logic is a clocked logic design style based on the CMOS logic family. Although static CMOS logic is widely used for its high noise margins and relative ease of design, it is limited at running extremely high clock speeds. For applications requiring the fasted circuit speeds possible, dynamic CMOS logic has numerous advantages over static CMOS including not only higher speeds but also significantly reduced surface area. The advantages do not come without a cost however. Due to the nature of dynamic CMOS logic, undesired effects can occur within the circuit unless extra effort is put into the engineering design[Knoth (1997)]. The basic idea of dynamic logic is depicted in Fig.A.7 the output node (V_{OUT} is pre-charged by a PMOS transistor during the LOW-phase of the clocking signal clk. When the clock (ϕ) goes LOW the output is discharged by the n-block or not, depending on its logic function. Due to the parasitic and interconnect capacitances the output remains stable for some time also in the HIGH state. During the evaluations cycle of the logic, the output is evaluated based on the transistor interconnection, very similar to the static CMOS case. Unfortunately, cascading such dynamic gates would

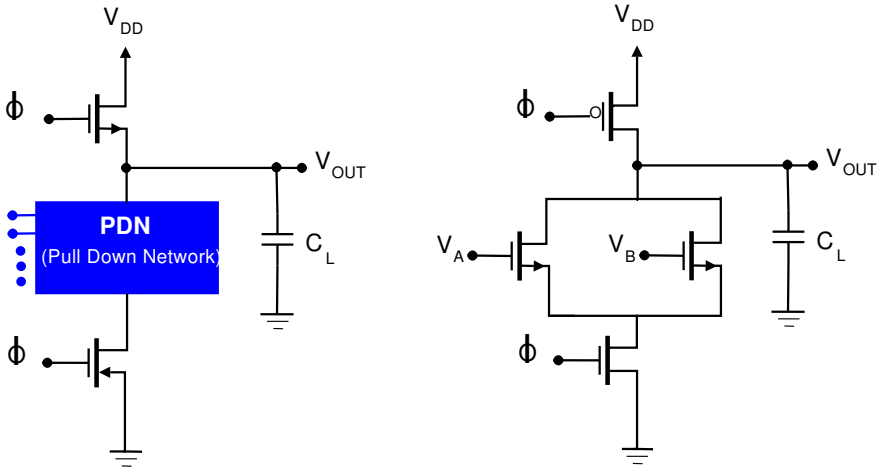


Fig. A.7: Dynamic CMOS PDN

not work because, due to the pre-charging of the preceding stage, the inputs may be still HIGH at the rising edge of the clock, which results in an erroneous discharge of the following stage. Ways to circumvent exist [Jan M. Rabaey and Nikolic (2003)], [Schrom (1998)] but are not discussed as a part of this appendix. However, the big advantages

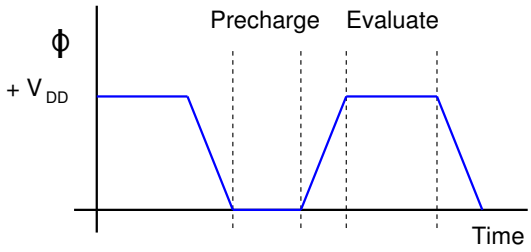


Fig. A.8: Dynamic CMOS Clock

of this type of logic is that the inputs are connected only to NMOS transistors so that the input load capacitance is much smaller. Therefore, dynamic logic is faster than static CMOS. Furthermore, for complex functions the transistor count is almost halved. The big disadvantages of this type of logic is the need for repeated charging and discharging even when the inputs do not change their state. Therefore, dynamic logic consumes more power than static CMOS despite the lower transistor count. Another problem is the sensitivity to leakage current of the n-block (in the order of I_{off} when the output is HIGH (during the clock-HIGH phase). This imposes a lower limit to the clock frequency. Also, as the ratio $\frac{I_{off}}{I_{on}}$ is limited by the supply voltage V_{DD} must be above a certain limit, which also limits the power efficiency of dynamic logic. These problems can be alleviated to some extent by using a positive feedback with a narrow PMOS transistor at the output buffer, which

then works like a latch. The extra charge needed to change the switch the latch causes a usually small speed penalty. The precharge-evaluate operation cycles of the circuit are shown in Fig.A.8 [Schrom (1998)].

A.2 The IRSIM Switch Level Simulator

IRSIM is a tool for simulating digital circuits. It is a “switch-level ”simulator; that is, it treats transistors as ideal switches. Extracted capacitance and lumped resistance values are used to make the switch a little bit more realistic than the ideal, using the RC time constants to predict the relative timing of events. IRSIM shares a history with magic, although it is an independent program. Magic was designed to produce, and IRSIM to read, the “.sim ”file format, which is largely unused outside of these two programs. Parts of Magic were developed especially for use with IRSIM, allowing IRSIM to run a simulation in the "background while displaying information about the values of signals directly on the VLSI layout. For “quick ”simulations of digital circuits, IRSIM is still quite useful for confirming basic operation of digital circuit layouts. The addition of scheduling commands put IRSIM into the same class as Verilog simulators. The current version of the simulator is 9.7 [Edwards (2007)].

A.3 Modeling details

The first step in creating a .sim file for a circuit is to label all the nodes. Label power V_{DD} and ground as GND. IRSIM is not case sensitive and the cases will work interchangeably. one can label the other nodes anyway he want. For the circuit in Fig.A.3, the inputs are labeled V_A and V_B and the output node is labeled V_{OUT}. The internal nodes are arbitrarily labeled V_C. When using IRSIM, it is helpful to have the labeled circuit schematic available so that one can know the names of the nodes that one want to probe. The irsim_logic.sim file for the above circuit is shown in Table.A.1.

In the net file, the transistor types occur first in the list followed by the gate, source and drain nodes in that order. At the end of the list, the length and the width of the transistors

Table A.1: An example of a .sim file for the IRSIM simulator

line	text					
1	| units: 100 tech: scmos					
2	|					
3	| type	gate	source	drain	length	width
4	| ———	———	———	———-	———	———-
5	| p	V_A	V_{DD}	V_C	$L_{V_{DD}V_C}$	$W_{V_{DD}V_C}$
6	| p	V_B	V_C	V_D	$L_{V_CV_D}$	$W_{V_CV_D}$
7	|					
8	| n	V_A	V_{OUT}	GND	$L_{V_{OUT}GND}$	$W_{V_{OUT}GND}$
9	| n	V_B	V_{OUT}	GND	$L_{V_{OUT}GND}$	$W_{V_{OUT}GND}$

are recorded. This input format provides a clean way of a genetic representation of the netlist. The interconnection is manipulated according to the GA to obtain the required functionality among a set of transistors as described in Sec.2.2.2. A detailed tutorial about the operation and working of the simulator with typical designs can be found at [Edwards (2007)].

APPENDIX B

PDPE ESTIMATES USING THE GENETIC SEARCH AND PARTITIONING METHODS

The modified genetic search 0and the vector partitioning method were described in Chap.3. In this appendix, we present the results obtained from the experiments conducted on the ISCAS'85 benchmark circuits. We elaborate on the results presented in Sec.3.5. The specification for the combinational benchmark circuits were also presented in the same section.

B.1 Modified Genetic Search Method

B.1.1 Results for the Zero Delay Model

Table B.1: Power virus vectors and genetic search progression - zero delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
c17	98	13	01001	10110
C432	984	224	0010010011011011	1101001110100100
			1111101111001101	0100010000111010
			1011	1110
C499	234	306	1110101110001110	0001010001110000
			0001110010110111	1111001101001010
			101011100	000110011

Table B.1: Power virus vectors and genetic search progression - zero delay model for the ISCAS'85 benchmark circuits

Circuit	**# Generations**	**# Pk Togls**	**IP Vector 1**	**IP Vector 2**
C880	988	479	0000100100010010	1111111011111111
			1111110110000000	1100000101100111
			1011000110000100	0100010001110011
			111001101111	101110010000
C1355	1016	596	1101010111101100	0110111000010001
			1101001111111100	0010110001001011
			000110000	111001011
C1908	252	964	0001110011000111	1110001100111000
			0111001110001010	1000110000111111
			1	0
C2670	510	1380	1010111001110000	1001010111001111
			0001000111011111	0110111011100100
			1100101111101000	0111010110010110
			1001010000101111	0010110111010000
			0010011111101000	1111100010110000
			0010001011001010	1101100000111101
			1011101010100100	1010010101000001
			1101101001000111	0001001110001000
			1000010000001001	0110001001000110
			1101111101100011	0011000001001100
			0110101011010100	1101110100101011

Table B.1: Power virus vectors and genetic search progression - zero delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
			0111000011010101	1010101110100011
			0001001100111110	1110110111100001
			0111111111000100	1100100010111010
			101011111	000110000
C3540	146	1633	1111011000111110	0000000110001001
			0011101101101001	1101010010010100
			1001111011000101	0001001100101110
			11	00
C5315	788	2815	0111101101111101	1010011110001010
			0110101000010101	1001011001111010
			1000111111000100	1111100000110110
			0100011111010101	1011100111100111
			1010101100100101	1001010001001010
			0001100010011010	1110010001100111
			1101000011001000	1010111101110111
			1001100110000011	0111011101111100
			0100000010000010	1010111110111101
			0011001001000011	1101110110011100
			0110100001111110	1000011110000001
			11	00
C7552	207	3541	0111010011011000	0010001100001011

Table B.1: Power virus vectors and genetic search progression - zero delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
			0011011100000100	1101111100110110
			0001101011101110	0100101100010111
			1011110110111010	1100011010100100
			0111001111000100	0101111111000111
			1010101000100000	1011100111110110
			0110111001000101	0101010001000101
			1010000011011001	1111010101010111
			1110010001110101	0100010000100101
			1011010100111110	0100001011100110
			0011111000100111	0111011000101110
			1010111110101010	0000000010001101
			001010001101000	010110110010100

Presented in Table.B.1 is the number of generations required for the genetic method in the zero delay model. The table also shows the final power virus vector pair which results in the number of toggles reported in column 3. The ISCAS circuits are shown in column 1.

B.1.2 Results for the Unit Delay Model

This section presents the results for the modified genetic method using the unit delay model for the benchmark circuits. The power virus vector pairs are shown in column 4 and column 5 of Table.B.2. The successive application of IP Vector 1 and IP Vector 2

for the benchmark circuits results in the peak number of toggles shown in column 3. The number of generations required for the genetic method to converge are shown in column 2.

Table B.2: Power virus vectors and genetic search progression - unit delay model for the ISCAS'85 benchmark circuits

Circuit	**# Generations**	**# Pk Togls**	**IP Vector 1**	**IP Vector 2**
c17	67	16	10001	01110
C432	1788	363	0010010001010100	1101101010100011
			1000110100100100	0111001110111011
			0110	1001
C499	851034	631	0000100110101011	1111011101000100
			0101111100110111	1010100011101000
			101011100	110100011
C880	23600	791	10101010010011111	1101010111100000
			0000101011111010	1111010110001001
			0101100100010100	1010111001101111
			011101111101	100011001110
C1355	54	757	10110010010001101	1110010110011001
			1100101101000100	0010010000111011
			111100111	101111000
C1908	644	1756	00011111001100000	10000000110011111
			11000001001000100	00111010010010010
			0	0
C2670	142	2580	10011100001100000	01101011000011111

Table B.2: Power virus vectors and genetic search progression - unit delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
			1011000001101001	0101100111010110
			0010010011110011	1110111101100000
			0011110011101000	1100001100000111
			0100101010110001	0001010101001011
			0101110001011000	0010000010111011
			1100111000100011	1101000011001101
			1100000011111100	0010011001100011
			0010110110100101	1111101010001010
			0100000110100001	0011111000011110
			0011100001000111	0101011110111000
			1101000101011101	0010001101100010
			1101010000001100	0010101100010011
			1111101100010111	0000000111101001
			101111111	011000100
C3540	28	2717	0110011111100011	1011100010011000
			1011100000111011	0100010111000001
			1010101001101000	1101110011110111
			01	10
C5315	3454	5197	1000111001000101	0111010111110010
			1110011010001100	0011101011111111
			1111110101101101	0100001010110100

Table B.2: Power virus vectors and genetic search progression - unit delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
			0010100100111100	1101010101010011
			0101000000011101	1010010111100110
			1111101111010000	0010011000101101
			0000101101101101	1111010010010110
			1001000101001010	0110110011110101
			1100110100111000	1011111011010111
			0010001011111000	1101100101001100
			1001010110000010	0110101001101101
			10	00
C7552	25744	7883	0011101011100010	1100011100111100
			1110011011110101	0001100000001010
			0111000011011011	1000111101000101
			1010100110001001	1001011101111110
			1110100110000011	0001011101111100
			0110111010101011	0001110101010101
			1011010011000001	1010101100100010
			0011000111100011	1100101001001000
			0101110101111010	1011001001000101
			0100111010000011	0010110101111100
			1000110001100100	0111101101101011
			0111101011010000	1000000110101100

Table B.2: Power virus vectors and genetic search progression - unit delay model for the ISCAS'85 benchmark circuits

Circuit	# Generations	# Pk Togls	IP Vector 1	IP Vector 2
			101111110011101	001000001100111

B.2 Vector Partitioning Method, Zero Delay

The results for the genetic method on the ISCAS'85 benchmark suite with the zero and unit delay models was presented in Sec.B.1.1 and in Sec.B.1.2 respectively. The vector partitioning method was proposed in Sec.3.4.1 of Chap.3.

This section presents the results for the partitioning method using the zero delay model for the benchmark circuits. The power virus vector pairs are shown in column 4 and column 5 of Table.B.3. The successive application of IP Vector 1 and IP Vector 2 (from columns 4 and 5 respectively) for the benchmark circuits results in the peak number of toggles shown in column 3. The partition splits at the primary inputs (simple uniform) are shown in column 2. This can be enhanced by the cone based algorithm [Saucier *et al.* (1993)] described in Sec.3.4.1 in Chap.3

Table B.3: Power virus vectors and partitioning iteration - zero delay model for the ISCAS'85 benchmark circuits

Circuit	IP Partitions	# Pk Togls	IP Vector 1	IP Vector 2
c17	$(< 0, 2 >, < 3, 4 >)$	12	00001	11110
C432	$(< 0, 5 >, < 6, 11 >$	242	1100010001000100	1011101110101011
	$\ldots < 30, 35 >)$		1100011101101100	0011101010011000
			0100	1011
C499	$(< 0, 7 >, < 8, 15 >$	249	0101111101000101	0011001001010110

Table B.3: Power virus vectors and partitioning iteration - zero delay model for the ISCAS'85 benchmark circuits

Circuit	**IP Partitions**	**# Pk Togls**	**IP Vector 1**	**IP Vector 2**
	$\ldots < 32, 40 >)$		0100000011010011	1001110101001000
			011101100	110011000
C880	$(< 0, 9 >, < 10, 19 >$	498	1111101010011010	1000011000111111
	$\ldots < 50, 59 >)$		0011011101111111	1011100001000101
			1110001111101111	1001101100001000
			011111111100	101001100011
C1355	$(< 0, 7 >, < 8, 15 >$	509	1000111111111001	0011101100101100
	$\ldots < 32, 41 >)$		0001111010100110	0100000000001101
			000000000	000000000
C1908	$(< 0, 5 >, < 6, 11 >$	886	0100110011101001	1001001100001010
	$\ldots < 25, 31 >)$		0111101001001001	0110100100100110
			0	0
C2670	$(< 0, 5 >, < 6, 11 >$	1266	1011111001001111	0010101101010101
	$\ldots < 25, 31 >)$		1101100111111110	1010101000010001
			1001001000111100	1001110000010000
			0000010100011111	1000100100100001
			1001011101001101	0010100001010000
			1010110001100111	0101101011011000
			0100111001000011	1111101110111011
			1100000110110000	0001100001000001
			0111010111101010	1000101100010100

Table B.3: Power virus vectors and partitioning iteration - zero delay model for the ISCAS'85 benchmark circuits

Circuit	**IP Partitions**	**# Pk Togls**	**IP Vector 1**	**IP Vector 2**
			0100001110101001	1011110110010011
			1011110110010000	0100111001111010
			1001001000010010	0101100101000001
			1001011111011100	0010011000100011
			0100111111000100	0001000000101111
			001111110	110001010
C3540	$(< 0, 9 >, < 10, 19 >$	1395	1111100001110110	1100010011001010
	$\ldots < 40, 49 >)$		1111100111001111	0100110011110000
			0101001110000010	0011110111111101
			00	00
C5315	$(< 0, 21 >, < 22, 43 >$	2673	0011011110110110	1100011001010011
	$\ldots < 154, 177 >)$		1100001000111111	0011110111000100
			0110000101001000	1000010100010101
			0101100010010010	0001111101001101
			1110000111110110	1000001100001001
			0000110100111010	1111010001001111
			1011110011100100	0111101000011100
			1000110000000001	0001101101101111
			1011001100101110	1100001110011011
			1011010101100110	1110001100101001
			0010001110101101	1110000101111000

Table B.3: Power virus vectors and partitioning iteration - zero delay model for the ISCAS'85 benchmark circuits

Circuit	**IP Partitions**	**# Pk Togls**	**IP Vector 1**	**IP Vector 2**
			01	11
C7552	$(<0,17>,<18,35>$ $\ldots<180,206>)$	3431	0110101011100111	0010011010110000
			1000010100010110	0111110101101001
			0000111101100000	0000001110010111
			1111101110010111	0001001001110010
			1101011111010111	1000110111011000
			1110000100111110	1100111011000010
			1111001110000110	0001010101011101
			0101000100010100	0001011011101101
			0001101111110000	1000010110100100
			1001000100111100	0000001010100100
			0001101101010011	0011011001100000
			0000101101000110	1000111001101101
			111110101100000	110110101101011

APPENDIX C

HAMMING CODE AND ECC

C.1 Error Correcting Codes(ECC)

Codes that correct errors have a mature, difficult, and mathematically oriented theory but this appendix describes a simple and elegant code, discovered in 1949. Thanks to Prof. Neal Wagner at the University of Texas at San Antonio from whose book draft on "The laws of cryptography", portions of this appendix are borrowed [Wagner (2002)].

C.2 Hamming Codes

Richard Hamming found a beautiful binary code that will correct any single error and will detect any double error (two separate errors). The Hamming code has been used for computer RAM, and is a good choice for randomly occurring errors. The Hamming code was used in Chap.5 and its implementation was described in Sec.5.4.1. This appendix provides insights into technical details of the same.

The Hamming code uses extra redundant bits to check for errors, and performs the checks with special check equations. A parity check equation of a sequence of bits just adds the bits of the sequence and insists that the sum be even (for even parity) or odd (for odd parity). In this appendix, we describe its use based on even parity. Alternatively, one says that the sum is taken modulo 2 (divide by 2 and take the remainder), or one says that the sum is taken over the integers mod 2, Z_2. A simple parity check will detect if there has been an error in one bit position, since even parity will change to odd parity. (Any odd number of errors will show up as if there were just 1 error, and any even number of errors will look the same as no error). One has to force even parity by adding an extra parity bit and setting it either to 1 or to 0 to make the overall parity come out even. It is important to realize that the extra parity check bit participates in the check and is itself

Table C.1: Parity Checks for the first 17 bits of the Hamming code

Pos	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	12	13	14	15	**16**	17
BRep	**1**	**10**	11	**100**	101	110	111	**1000**	1001	1010	1011	1100	1101	1110	1111	**10000**	10001
Chk1	**x**		x		x		x		x		x		x		x		x
Chk2		**x**	x			x	x			x	x			x	x		
Chk4				**x**	x	x	x					x	x	x	x		
Chk8								**x**	x	x	x	x	x	x	x		
Chk16																**x**	x

checked for errors, along with the other bits. The Hamming code uses parity checks over a portion of the positions in a block. Suppose there are bits in consecutive positions from 1 to $n-1$. The positions whose position number is a power of 2 are used as check bits, whose value must be determined from the data bits. Thus the check bits are in positions $1,\ 2,\ 4,\ 8,\ 16, \ldots$ to the largest power of 2 that is less than or equal to the largest bit position. The remaining positions are reserved for data bits.

Each check bit has a corresponding check equation that covers a portion of all the bits, but always includes the check bit itself. Consider the binary representation of the position numbers: $1 = 1_2, 2 = 10_2, 3 = 11_2, 4 = 100_2, 5 = 101_2, 6 = 110_2$, and so forth. If the position number has a 1 as its rightmost bit, then the check equation for check bit 1 covers those positions. If the position number has a 1 as its next-to-rightmost bit, then the check equation for check bit 2 covers those positions. If the position number has a 1 as its third-from-rightmost bit, then the check equation for check bit 4 covers those positions. Continue in this way through all check bits. Table.C.1 summarizes this. In the table, the check bits are in positions $1,\ 2,\ 4,\ 8,\ and\ 16$, and are marked in bold. Table C.2 shows an example assuming a data bit word 1101101. The check equations in Table.C.1 are used to determine the values for check bits in positions $1,\ 2,\ 4,\ and\ 8$, to yield the word 11101010101 [Wagner (2002)]

Intuitively, the check equations allow one to "zero-in "on the position of a single error. For example, suppose a single bit is transmitted in error. If the first check equation fails, then the error must be in an odd position, and otherwise it must be in an even position. In other words, if the first check fails, the position number of the bit in error must have its

Table C.2: Parity Check Example for a 7 bit data word

Position	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11
Bin Rep	**1**	**10**	11	**100**	101	110	111	**1000**	1001	1010	1011
Word	**1**	**1**	1	**0**	1	0	1	**0**	1	0	1
Check 1		**1**	1			0	1			0	1
Check 2	**1**		1		1		1		1		1
Check 4				**0**	1	0	1				
Check 8								**0**	1	0	1

Table C.3: Single Error detection using the Hamming ECC

Pos	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	**Chk**
BRep	**1**	**10**	11	**100**	101	110	111	**1000**	1001	1010	1011	
Word	**1**	**1**	1	**0**	1	0	1	**0**	1	0	**0(Er)**	
Chk1		**1**	1			0	1			0	0	**1** fail
Chk2	**1**		1		1		1		1		0	**2** fail
Chk4				**0**	1	0	1					**-** pass
Chk8								**0**	1	0	0	**8** fail

rightmost bit (in binary) equal to 1; otherwise it is zero. Similarly the second check gives the next-to-rightmost bit of the position in error, and so forth. Table.C.3 shows the result of a single error in position 11 (changed from a 1 to a 0). Three of the four parity checks fail, as shown below. Adding the position number of each failing check gives the position number of the error bit, 11 in this case. This shows how to get single-error correction with the Hamming code. One can also get double-error detection by using a single extra check bit, which is in position 0. (All other positions are handled as above.) The check equation in this case covers all bits, including the new bit in position 0. In case of a single error, this new check will fail. If only the new equation fails, but none of the others, then the position in error is the new 0^{th} check bit, so a single error of this new bit can also be corrected. In case of two errors, the overall check (using position 0) will pass, but at least one of the other check equations must fail. This is how one detects a double error. In this case there is not enough information present to say anything about the positions of the two bits in error. Three or more errors at the same time can show up as no error, as two

Table C.4: Hamming ECC block sizes

Check Bits	Max Data Bits	Max Total Size
3	1	4
4	4	8
5	11	16
6	26	32
7	**57**	**64**
8	120	128

errors detected, or as a single error that is "corrected"with a bogus correction. Notice that the Hamming code without the extra 0^{th} check bit would correct a double error in some bogus position as if it were a single error. Thus the extra check bit and the double error detection are very important for this code. Notice also that the check bits themselves will also be corrected if one of them is transmitted in error (without any other errors) [Wagner (2002)].

C.3 Block sizes for the Hamming Code

The Hamming code can accommodate any number of data bits, but it is interesting to list the maximum size for each number of check bits. Table.C.4 includes the overall check bit, so that this is the full binary Hamming code, including double error detection [Hamming (2008)]. For example, with 64 bits or 8 bytes, one gets 7 bytes of data (plus 1 bit) and uses 1 byte for the check bits (actually, only 7 bits). Thus an error-prone storage or transmission system would only need to devote 1 out of each 8 bytes 12.5% to error correction/detection.

REFERENCES

1. **A. V Oppenheim, S. R. W.** and **J. R. Buck**, *Discrete time signal Processing, 2nd Edition*. Pearson Education, .

2. **Actel, T. R.** (December 2002). Understanding soft and firm errors in semiconductor devices – questions and answers - http://www.actel.com/documents/ser_faq.pdf. URL http://www.actel.com.

3. **Adams, J.** and **J. Willson, A.** (1984). Some efficient digital prefilter structures. *IEEE Transactions on Circuits and Systems*, **31**(3), 260–266.

4. **Adoptech** (2008). Design optimization - http://www.adoptech.com/design-optimization/design-opt.htm. URL http://www.adoptech.com.

5. **Antoniou, A.**, *Digital Signal Processing: Signals, Systems and Filters*. McGraw-Hill, 2005.

6. **Bahuman, A.**, **B. Bishop**, and **K. Rasheed**, Automated synthesis of standard cells using genetic algorithms. *In VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*. 2002.

7. **Bigelow, K.** (1996). Inside logic gates: Cmos logic - http://www.play-hookey.com/digital/electronics/cmos_gates.html. URL http://www.play-hookey.com.

8. **Bolchini, C.**, **D. Quarta**, and **M. D. Santambrogio**, Seu mitigation for sram-based fpgas through dynamic partial reconfiguration. *In GLSVLSI '07: Proceedings of the 17th great lakes symposium on Great lakes symposium on VLSI*. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-605-9.

9. **Bull, D. R.** and **D. H. Horrocks** (1991). Primitive operator digital filters. *Circuits, Devices and Systems, IEE Proceedings G*, **138**(3), 401–412.

10. **C Carmichael, P. B., E Fuller** and **M. Caffrey**, Seu mitigation techniques for virtex fpgas in space application. *In Proceedings of the MAPLD'99 (Poster), page 24*. IEEE, 1999.

11. **Cho, N. I.** and **S. U. Lee** (). Optimal design of finite precision fir filters using linear progression with reduced constraints. *IEEE Transactions on Signal Processing*.

12. **Chou, T.-L.**, **K. Roy**, and **S. Prasad** (1994). Estimation of circuit activity considering signal correlations and simultaneous switching, 300–303.

13. **CMOS** (2008). Complementary metal oxide semiconductor (cmos) - http://en.wikipedia.org/wiki/cmos. URL http://en.wikipedia.org.

14. **Course, M. S.** (2002). Single event transient (set), mapld short course - http://www.klabs.org/.../tutorial/minicourses/radiation_mapld_2002/radiation_course _2002_mapld.hardcopy/f_set.ppt. URL http://www.klabs.org.

15. **Devadas, S.**, **K. Keutzer**, and **J. White** (1992). Estimation of power dissipation in CMOS combinational circuits using boolean function manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **11**(3), 373–383.

16. **Edwards, T.** (2007). Irsim version 9.7 switch-level simulator - http://opencircuitdesign.com/irsim/. URL http://opencircuitdesign.com.

17. **fpga4fun** (2008). How fpgas work - http://www.fpga4fun.com/fpgainfo2.html. URL http://www.fpga4fun.com.

18. **Godlberg, D. E.**, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley SF, 1989.

19. **Greenwood, G. W.** and **A. M. Tyrrel**, *An Introduction to Evolvable Hardware*. IEEE Press, Wiley Interscience, International, 2007.

20. **Hamming** (2008). Hamming code general algorithm - http://en.wikipedia.org/wiki/hamming_code#general_algorithm. URL http://en.wikipedia.org.

21. **Haseyama, M.** and **D. Matsuura** (2006). A filter coefficient quantization method with genetic algorithm, including simulated annealing. *IEEE Signal Processing Letters*, **13**(4), 189–192.

22. **Heusler, L. S.** and **W. Fichtner** (1991). Transistor sizing for large combinational digital cmos circuits. *Integr. VLSI J.*, **10**(2), 155–168. ISSN 0167-9260.

23. **Ho, M. C.**, **S. Leung**, **H. Kurokawa**, and **O. C. Choy**, Digital logic synthesis using genetic algorithms. *In Genetic Algorithms in Engineering Systems: Innovations and Applications, 1997. GALESIA 97. Second International Conference On (Conf. Publ. No. 446)*. Glasgow, UK, 1997. ISSN 0537-9989.

24. **Hoflich, W.** (2005). Using the cx4000 readback capability - http://www.xilinx.com/support/documentation/application_notes/xapp015.pdf. URL http://www.xilinx.com.

25. **Holland, J. H.**, *Adaptation of Natural and Artificial Systems*. University of Michigan Press Ann Arbor, MI, 1975.

26. **Horrocks, D. H.** and **Y. M. A. Khalifa**, Genetically derived filter circuits using preferred valuecomponents. *In Analogue Signal Processing, IEE Colloquium on*. London, UK, 1994.

27. **Hounsell, B. I.** and **T. Arslan**, A novel genetic algorithm for the automated design of performance driven digital circuits. *In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1. 2000.

28. **Hsiao, M. S.** (1999). Peak power estimation using genetic spot optimization for large vlsi circuits, 38.

29. **Hsiao, M. S.**, **E. M. Rudnick**, and **J. H. Patel**, Effects of delay models on peak power estimation of VLSI sequential circuits. *In Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on*. 1997*a*.

30. **Hsiao, M. S.**, **E. M. Rudnick**, and **J. H. Patel**, K2: an estimator for peak sustainable power of VLSI circuits. *In Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on*. 1997*b*.

31. **Hsiao, M. S.**, **E. M. Rudnick**, and **J. H. Patel** (2000). Peak power estimation of VLSI circuits: new peak power measures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **8**(4), 435–439.

32. **Jan M. Rabaey, A. C.** and **B. Nikolic**, *Digital Integrated Circuits, 2/E*. Prentice Hall, 2003.

33. **Jong, K. A. D.** (1975). *An analysis of the behavior of a class of genetic adaptive systems.*. Ph.D. thesis, Ann Arbor, MI, USA.

34. **Kastensmidt, F. L.**, **L. Sterpone**, **L. Carro**, and **M. S. Reorda**, On the optimal design of triple modular redundancy logic for SRAM-based FPGAs. *In Design, Automation and Test in Europe, 2005. Proceedings*. 2005.

35. **Knoth, L. A.** (1997). Eee 425 honors project: Dynamic cmos - http://www.lauraknauth.com/academic/dyncmos.html. URL http://www.lauraknauth.com.

36. **Kodel, D. M.** (1980). Design of optimal finite precision fir filters using linear programming techniques. *IEEE Transactions on Acoustics, Speech and Signal Processing*, (3), 304–308.

37. **Kriplani, H.** (1994). *Worst case voltage drops in power and ground buses of CMOS VLSI circuits*. Ph.D. thesis, Champaign, IL, USA.

38. **Kriplani, H.**, **F. Najm**, **P. Yang**, and **I. Hajj**, Resolving signal correlations for estimating maximum currents in CMOS combinational circuits. *In Design Automation, 1993. 30th Conference on*. 1993.

39. **Lefebvre, M.** and **D. Marple**, The future of custom cell generation in physical synthesis. *In Design Automation Conference, 1997. Proceedings of the 34th*. 1997.

40. **Lim, Y.**, **S. Parker**, and **A. Constantinides** (1982). Finite word length FIR filter design using integer programming over a discrete coefficient space. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, **30**(4), 661–664.

41. **Lima, F.**, **L. Carro**, and **R. Reis**, Designing fault tolerant systems into SRAM-based FPGAs. *In Design Automation Conference, 2003. Proceedings*. 2003.

42. **Lu, W. S.** and **A. Antoniou**, *Two-Dimensional Digital Filters*. Marcel Dekker NY, 1992.

43. **Manne, S.** *et al.*, Computing the maximum power cycles of a sequential circuit. *In Design Automation, 1995. DAC '95. 32nd Conference on Page(s):23 - 28*. 1995.

44. **Mastipuram, R.** and **E. C. Wee** (2004). Soft errors impact on system reliability - http://www.edn.com/article/ca454636.html. URL http://www.edn.com.

45. **Mazumder, P.** and **E. M. Rudnick**, *Genetic algorithms for VLSI design, layout and test automation*. Prentice Hall, 1999.

46. **Miller, J. F.**, An evolvable hardware approach to digital filter design. *In Evolutionary Hardware Systems (Ref. No. 1999/033), IEE Half-day Colloquium on*. 1999.

47. **Miron Abromovici, C. E. S., John M. Emmert**, An integrated approach to on-line testing, diagnosis, and fault tolerance for fpgas in adaptive computing systems. *In Proceedings of The 3rd NASA/DoD Workshop on Evolvable Hardware, pp.73–79*. NASA/DoD, 2001.

48. **Mitchell, M.**, *An Introduction to Genetic Algorithms*. MIT Press, 1998.

49. **Mocanu, O.-D.** and **J. Oliver** (1999). Fault-tolerant memory architecture against radiation-dependent errors: A mixed error control approach. *J. Electron. Test.*, **14**(1-2), 169–180. ISSN 0923-8174.

50. **Najeeb, K.**, **K. Gururaj**, **V. Kamakoti**, and **V. M. Vedula**, Controllability-driven power virus generation for digital circuits. *In VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*. 2007.

51. **Normand, E.** (December 1998). Presentation by e normand to the c-17 avionics group - http://www.boeing.com/assocproducts/radiationlab/publications/. URL http://www.boeing.com.

52. **Optimization** (2008). Mathematical optimization - http://en.wikipedia.org/wiki/optimization_(mathematics). URL http://en.wikipedia.org.

53. **Ou, E.**, Fast error-correcting circuits for fault-tolerant memory. *In MTDT '04: Proceedings of the Records of the 2004 International Workshop on Memory Technology, Design and Testing*. IEEE Computer Society, Washington, DC, USA, 2004. ISBN 0-7695-2193-2.

54. **Parks, T. W.** and **C. S. Burrus**, *Digital Filter Design*. John Wiley NY, 1987.

55. **Pasternak, C., J.H; Salama** (1993). Differential pass transistor logic. *Circuits and Devices Magazine, IEEE*, **9**, 23–28.

56. **Pedram, M.** (1996). Power minimization in ic design: principles and applications. *ACM Trans. Des. Autom. Electron. Syst.*, **1**(1), 3–56. ISSN 1084-4309.

57. **Pittman, J.** and **C. A. Murthy** (2000). Fitting optimal piecewise linear functions using genetic algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(7), 701–718.

58. **Rabiner, L. R.**, **J. H. McClellan**, and **T. W. Parks** (1975). FIR digital filter design techniques using weighted chebyshev approximation. *Proceedings of the IEEE*, **63**(4), 595–610.

59. **Reddy, E. S. S.**, **V. Chandrasekhar**, **M. Sashikanth**, **V. Kamakoti**, and **N. Vijaykrishnan**, Cluster-based detection of seu-caused errors in luts of sram-based fpgas. *In ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation*. ACM, New York, NY, USA, 2005. ISBN 0-7803-8737-6.

60. **Redmill, D. W.** and **D. R. Bull**, Automated design of low complexity FIR filters. *In Circuits and Systems, 1998. ISCAS '98. Proceedings of the 1998 IEEE International Symposium on*, volume 5. 1998.

61. **Rogenmoser, R.**, **H. Kaeslin**, and **T. Blickle**, Stochastic methods for transistor size optimization of cmos vlsi circuits. *In PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*. Springer-Verlag, London, UK, 1996. ISBN 3-540-61723-X.

62. **Sabbir, U. A.** and **A. Antoniou**, Design of digital filters using genetic algorithms. *In 6th International Symposium on Signal Processing and Information Technology*. IEEE, 2006.

63. **Saramaki, T.**, **T. Neuvo**, and **S. K. Mitra** (1988). Design of computationally efficient interpolated FIR filters. *IEEE Transactions on Circuits and Systems*, **35**(1), 70–88.

64. **Saucier, G.**, **D. Brasen**, and **J. P. Hiol**, Partitioning with cone structures. *In ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993. ISBN 0-8186-4490-7.

65. **Schrom, G.** (1998). *Ultra-Low-Power CMOS Technology*. Ph.D. thesis, Adamsgasse 3/3, A-1030 Wien, Matrikelnummer 8326270.

66. **Suckley, D.** (1991). Genetic algorithm in the design of FIR filters. *Circuits, Devices and Systems, IEE Proceedings G*, **138**(2), 234–238.

67. **Sundaralingam, S.** and **K. Sharman**, Genetic evolution of adaptive filters. *In Proceedings of Digital Signal Processing (DSP) UK, pp. 47–53*. IEE, 1997.

68. **Suzuki, M.**, **N. Ohkubo**, **T. Yamanaka**, **A. Shimizu**, and **K. Sasaki** (1993). A 1.5ns 32b cmos alu in douple pass-transistor logic. *Dig. Tech. Papers, ISSCC*, 90–91.

69. **T H Cormen, R. L. R., C E Leiserson** and **C. Stein**, *Introduction to Algorithms, 2nd Edition*. Prentice Hall, 2005.

70. **Torresen, J.** and **K. A. Vinger**, High performance computing by context switching reconfigurable logic. *In Proceedings of the 16th European Simulation Multiconference on Modelling and Simulation 2002*. SCS Europe, 2002. ISBN 90-77039-07-4.

71. **Tufte, G.** and **P. C. Haddow**, Evolving an adaptive digital filter. *In Evolvable Hardware, 2000. Proceedings. The Second NASA/DoD Workshop on*. 2000.

72. **Wade, G.**, **P. Van-Eetvelt**, and **H. Darwen** (1990). Synthesis of efficient low-order FIR filters from primitive sections. *Circuits, Devices and Systems, IEE Proceedings G*, **137**(5), 367–372.

73. **Wagner, N. R.** (2002). The laws of cryptography: The hamming code for error correction - http://www.cs.utsa.edu/w̃agner/lawsbookcolor/laws.pdf. URL http://www.cs.utsa.edu.

74. **Wang, C. Y.** and **K. Roy** (2000). Maximization of power dissipation in large CMOS circuits considering spurious transitions. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on [see also Circuits and Systems I: Regular Papers, IEEE Transactions on]*, **47**(4), 483–490.

75. **Wang, C.-Y.**, **K. Roy**, and **T.-L. Chou**, Maximum power estimation for sequential circuits using a test generation based technique. *In Custom Integrated Circuits Conference, 1996., Proceedings of the IEEE 1996*. 1996.

76. **Wenzel, W.** and **K. Hamacher** (1999). Stochastic tunneling approach for global minimization of complex potential energy landscapes. *Phys. Rev. Lett.*, **82**(15), 3003–3007.

77. **Woods, S.** and **G. Casinovi**, Efficient solution of systems of boolean equations. *In Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*. 1996.

78. **XAPP151** (October 2004). Virtex series configuration architecture user guide - http://www.xilinx.com/support/documentation/application_notes/xapp151.pdf. URL http://www.xilinx.com.

79. **XAPP258** (Januray 2005). Fifos using virtex-ii block ram - http://www.xilinx.com/support/documentation/application_notes/xapp258.pdf. URL http://www.xilinx.com.

80. **XAPP464** (March 2005). Using look-up tables as distributed ram in spartan 3 generation fpgas - http://www.xilinx.com/support/documentation/application_notes/xapp464.pdf. URL http://www.xilinx.com.

81. **XAPP645** (February 2004). Error detection and correction in virtex-ii pro devices - http://www.xilinx.com/support/documentation/application_notes/xapp645.pdf. URL http://www.xilinx.com.

82. **Xilinx** (2005). Xilinx virtex-v capabilities - block ram - http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex5/capabilities/block_ram.htm. URL http://www.xilinx.com.

83. **Yano, K.** *et al.* (1990). A 3.8ns cmos 16x16-b multiplier using complementary pass-transistor logic. *Proceedings of the IEEE Journal of Solid-State Circuits*, **25**, 388–395.

LIST OF PAPERS BASED ON THESIS

Conferences:

1. M. Shoaib, Noor. M, and Kamakoti. V. A Genetic Approach to Gateless Custom VLSI Design Flow. *to appear in the Proceedings of the* 19^{th} *IEEE International Conference on Microelectronics (ICM)*, 29-31 Dec 2007, Cairo, Egypt.

2. Karthik. K. S, Shyam. S, Ramasubramanian. S, Noor. M, Shoaib. M and Kamakoti. V A SEU Tolerant Distributed CLB RAM for Run-Time Reconfiguration. *accepted at the* 12^{th} *IEEE International VLSI Design and Test Symposium (VDAT)*, July 2008, Bangalore, India.

Journals:

1. M. Shoaib, Noor. M and Kamakoti. V Genetic Evolution of Self-Adaptive Arbitrary Response FIR Filters. *submitted to the IEEE Transactions on Evolutionary Computation.*