New Directions for Design-Space Exploration of Low-Power Hardware Accelerators

by

Kumud Nepal

B.S., Trinity College; Hartford, CT, 2009

Sc. M., Brown University; Providence, RI, 2011

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in The School of Engineering at Brown University

PROVIDENCE, RHODE ISLAND

May 2015

This dissertation by Kumud Nepal is accepted in its present form
by The School of Engineering as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date_____ _____

Iris Bahar, Ph.D., Advisor

Date_____ _____

Sherief Reda, Ph.D., Advisor

Recommended to the Graduate Council

Date_____ _____

Pedro Felzenszwalb, Ph.D., Reader

Approved by the Graduate Council

Date_____ _____

Peter Weber, Dean of the Graduate School

Vitae

Kumud Nepal was born in Kathmandu, Nepal in 1985. He graduated from Trinity College, Hartford, CT in 2009 with a Bachelor of Science in Electrical Engineering. He joined the Ph.D. program at Brown University in 2009 and completed his degree requirements for the Master of Science in Engineering in 2011.

Kumud's research interests include low-power Very Large Sale Integration (VLSI) systems and computer architecture, reconfigurable systems, Computer-Aided Design (CAD) flows, Electronic Design Automation (EDA), approximate computing and high throughput systems for image processing/video analytics.

kumud_nepal@brown.edu
Brown University, RI, USA

Acknowledgements

I would like to express my sincere gratitude to my advisors Professor Iris Bahar and Professor Sherief Reda for their invaluable guidance and support during my graduate study at Brown University. Under their tutelage, I have learned to become an independent thinker. I am also thankful to Professor Pedro Felzenszwalb for agreeing to serve as a member of my dissertation committee even at hardship. His comments and questions have made this work better.

Special thanks to my undergraduate Professor David Ahlgren, who helped me become the Engineer I am today. His guardianship at Trinity College during my undergraduate study will always be highly appreciated. I would like to thank my friends and colleagues for making the work place exciting and my stay at Brown University a pleasant experience. Thank you Cesare Ferri, Rotor Le, Nuno Alves, Elif Alpaslan, Yiwen Shi, Marco Donato, Onur Ulusel, Dimitra Pappagianapolou, Christopher Picardo, Octavian Biris, Ryan Cochran, Abdullah Nowroz, Kapil Dev, Yueting Li, Xin Zhan, Reza Azimi and Soheil Hashimi.

This work would not have been possible without the relentless support, love and encouragement from my family. I would like to thank my beautiful wife Pragya for her love, understanding and unwavering support through this journey. Finally, a special thanks to my parents for always standing by me and teaching me the value of education.

Abstract of " New Directions for Design-Space Exploration of Low-Power Hardware Accelerators " by Kumud Nepal, Ph.D., Brown University, May 2015

Hardware accelerators and custom computing platforms are being widely used in high-throughput computation domains. With the wide usage, there is also an increasing need to operate these hardware accelerators in resource-constrained environments.

In this dissertation work, we address two different ways of exploring possibilities for producing low-area and low-power hardware accelerators. First, we propose an analytical modeling of design metrics such as area, power, and throughput for hardware accelerators. Accelerators have various tunable implementation parameters, both at the algorithm and hardware levels. Physically synthesizing every possible design permutation to discover optimum implementations is highly inefficient given the exponentially large search space. We propose sampling a small fraction of the design space and using these samples to train analytical models that will accurately predict area, power, throughput for other unexplored parameter combinations. By doing this analytically, we take away a bulk of the time-consuming synthesis process. We also use these mathematical models to accurately formulate various multi-objective optimization trade-offs between area, power, and output accuracy.

For the second part of this dissertation, we pursue an idea we learned from the first part that computational accuracy of circuit implementations within certain application domains can be traded-off to make accelerators consume less area and power. We introduce a novel idea for automated synthesis of approximate circuits directly from their behavioral descriptions. By applying transformations that introduce a controlled amount of error, we make a number of approximate variants of the original circuits. We also realize a streamlined design space exploration process

so that the creation of newer approximate variants does not cause an exponential increase in the design space.

Finally, we study various techniques to expand the search scope for better approximate designs and achieve significantly better power savings without additional loss in output accuracy. We also propose a first-of-its-kind methodology to perform targeted approximations on timing-critical paths of a custom circuit hardware accelerator so as to reduce its delay. We use this delay reduction to use standard voltage scaling and make additional savings on power consumption.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Hardware accelerators and custom computing circuits are becoming widely used in real-time image processing, video analytics, machine learning and signal processing. Their use in surveillance, scientific research, smart camera technologies and automotive industries is becoming ubiquitous [8, 7, 16]. Traditional microprocessor designs have reached diminishing returns on instruction-level parallelism. Similarly, simply increasing clock frequencies to speed up software implementations is becoming increasingly difficult because of the dangers of thermal and power runoff [44, 55, 80]. While multi-core implementations of algorithms are possible in software with threaded applications, there are more benefits today to shifting the programming model for high-throughput to other computing platforms such as Graphic Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs) and/or custom Application Specific Integrated Circuits (ASICs) or Application Specific Signal Processors (ASSPs).

GPUs have become increasingly programmable and are very suited for data-level

parallelism and high bandwidth data proessing and transfer. Compared to Intel Core i7's with 2000 MHz DDR3 RAMs which have a peak bandwidth of 48GB/s, GPUs are capable of reaching over 150GB/s bandwidth with over 1TFlops/sec peak performance [80]. Similarly, dedicated ASICs remove the slow and generic architectural limitations of software-based systems. ASSPs exploit the parallelism and efficiencies of the digital signal processors (DSPs) and hardwired application-specic accelerators, usually managed by a standard controller core like ARM, or PowerPC [65]. Similarly, FPGAs are ideal for inexpensive platforms to implement high-throughput solutions. Their reconfigurability allows for iterative refinement and validation of a design implementation until desired goals are achieved. With programmable logic elements, registers, lookup tables (LUTs), Block RAMs (BRAMs), DSP blocks, and digital clock managers, FPGAs, by themselves or as parts of a heterogeneous system, have the capability of parallelizing algorithms on various hardware modules, making them superior in instrumenting tasks that require high throughput.

This prolific use of different hardware accelerator platforms leads to an increased demand for even higher computational capabilities and necessitates the need for processing larger data sets at higher throughput. This becomes more challenging because most of these high-performance platforms are also used in highly resource constrained environments where reduced power consumption becomes imperative. A lot of custom hardware accelerators in the form of heterogeneous systems or independent solutions are being increasingly used in embedded systems/mobile platforms where circuit area and energy efficiency both become imperative. Therefore, adding more hardware resources to solve the throughput problem may not lead to a feasible solution.

In this thesis, we turn our focus on two different hardware accelerating platforms - FPGAs and custom ASICs and explore options of how we can make use

of these accelerators with as little power and area cost as possible. We observe that hardware accelerators in these forms, specially those that can be used for image/signal processing or machine learning applications offer many algorithmic and hardware design parameters. Carefullly chosen combination of these parameters can lead to outcomes with the desired throughput, power, design area and arithmetic accuracy. Faster design-space exploration techniques as well as smarter synthesis methodologies can help designers explore options to get to optimum hardware accelerator implementations with low area footprints and reduced power consumption. These implementations can then be used to either increase accelerator throughput at the same area/power specifications or lower area/power numbers at the same throughput levels, based on the preferences and needs of the designers.

In the following sections, we discuss some of the approaches we develop and implement in order to do efficient search and synthesis of hardware accelerators.

1.1 Fast Design Space Exploration of Hardware Accelerator Implementations

Accelerating a particular algorithm in hardware is a challenging task as the large number of possible algorithmic and hardware design parameters lead to different accelerator variant implementations, each with its own metrics such as performance, area, power, and arithmetic accuracy characteristics. A design that is efficient in throughput can have large area and higher power consumption implications. Similarly, a design that takes care of area and power constraints could face accuracy and performance setbacks. Given the different configurations of hardware and software parameters possible for implementation, finding the 'sweet spot' for design imple-

mentation that meets all design requirements can be a time-consuming process.

As a result, designers are interested in exploring tens of algorithmic and hardware design parameters without doing explicit enumeration of each permutation. Works have been done in trying to evaluate different combinations of design parameters to see their impact on circuit behavior and topology — some involve reducing number of possible parameter configurations and doing experiments [28, 72], while others propose following an analytical methodology to model the design space mathematically [17, 36, 46, 71]. Predicting design metrics like power for various architectures and micro-architectures has been commonly done in a lot of work as well [31, 37, 45, 48, 49, 50, 57, 64, 74, 82] .

Regardless of the methodology used in implementing fast design space exploration, intelligent ways to discover the impact of hardware/software design parameters and the inherent interactions between them as a means to study their true impact on final design metrics, are still lacking. Sensitivity analysis of design parameters as a means to reaching optimality in making efficient hardware accelerators still remains an area relatively less explored. For any fast design space exploration methodology, it is imperative that an accurate relationship be established between each of the design parameters and their power, throughput, area and arithmetic accuracy implications. Typically, with existing research, designers need to make educated guesses about the interactions between different variables to identify appropriate variables for use in exploration purposes.

In this thesis work, we try to tackle this problem of doing accurate parameter sensitivity and parameter-interaction analysis, and thereby eliminate the guesswork on the designers' part. We adopt mathematical modeling methodologies or efficient design search and automate the process of discovering dependencies between design

parameters - thus making analytical design metric model predictions closer to actual measurements. The proposed approach involves sampling the large design space and then using regression models and statistical inference from the samples to create mathematical models that estimate the target metrics (such as area, power, throughput) over the entire design space. The analytical model generation process involves automated parameter-interaction analysis and the selection of the most important design variables whose impact individually or in interaction with other variables of the accelerator implementation affects design metrics the most.

The effectiveness of this methodology is demonstrated by applying it on two image processing algorithms implemented on an FPGA platform and evaluating both hardware and algorithm parameters for design space exploration. Only a few samples from the design space are physically implemented on the FPGA and their design metrics then used for training analytical models eventually used as scalable models for accurate design metric prediction.

1.2 Approximate Custom Computing Circuits as Low-Area/Power Hardware Accelerators

In search of designs that are optimum in the multi-faceted way, researchers are developing new circuit topologies and methodologies for increasing efficiency as well as reducing energy consumption. A relatively newer methodology is the development of approximate circuits and computers that produce good-enough computations for applications and algorithms that may not require 100% computational accuracy, with the idea of either increasing circuit throughput or reducing area and power consumption.

Our own work on fast design-space exploration techniques have shed some light on the possibility of using circuit accuracy as a parameter for design optimization. For instance, we were able to use arithmetic accuracy of a circuit implementation as a tunable design parameter for design optimization by studying truncation of certain number of bits from some computational kernel variables.

Similar ideas of sacrificing some accuracy for design optimization in terms of throughput and/or area/power have become popular lately. Often, it is the case that trading-off reasonable arithmetic accuracy in circuit implementations of algorithms, specially in the fields of machine learning, signal processing and computer vision can yield low area/power variants within acceptable output correctness. Many algorithms from these domains tend to show inherent error resiliency and their integrity is not compromised by controlled perturbations in their underlying computations. In this thesis, we leverage this circuit resiliency in implementations from certain domains by giving up a controlled amount of arithmetic accuracy to obtain less complex circuits, thereby reducing area footprints and power consumption.

Approximate computing circuits are more relevant today from two aspects [54]:

- With aggressive transistor scaling, devices are being pushed to the limits of their reliability. In addition, ways to improve silicon die yield is always being explored. Approximate computing circuits can be a good solution in these situations where error-resilient parts of circuits (or parts of them) can be operated on the unreliable parts of the chip and the more error-prone parts can be implemented on the reliable parts. This can certainly come in handy in imminent dark-silicon scenarios in the chip-manufacturing industry.

- Many of the applications in emerging mobile and embedded device market

can be approximate - mostly because of their use in imaging/video applications where there is inherent noise/redundancy and computation precision loss, which they are inherently tolerant to. Also, most of these applications incorporate human perception and there is no perceptual difference in small deviations in outputs for a lot of these applications [13, 24].

Most existing works in approximate computing can be broadly categorized in to two categories. There has been a lot of research on run-time methodologies where circuit inaccuracy is a result of run-time parameters like supply voltage [9, 11, 29, 59, 77]. More controlled design-time methodologies where approximations are intentionally introduced on a specific part/component of the circuit have also been proposed [4, 14, 30, 32, 78]. Whether it is run-time approach or design-time approximation, there are some limitations. Specific quantitative error control for run-time methodologies like standard voltage scaling can be non-trivial. For the other approach where inaccuracies are introduced by design, designers are usually required to have at least some degree of familiarity with the circuit functionality and/or its response to different run-time and design-time decisions.

In this thesis, we devise new techniques for approximate computing as a means to optimize designs for area and power, and eliminate some of the problems seen in existing approximation methodologies. We work directly on behavioral descriptions that capture the algorithmic intent of the circuit; and thus raising the level of abstraction enables a larger range of approximations that are not possible to apply at low-level design specifications. This enables us to work on more global and scalable approximation efforts without needing to have any prior knowledge about the algorithm or its circuit implementation. We draw inspiration for our work from the recent advances in software engineering targeted for automatic bug identification

[81]; however, we investigate methods that are suitable for hardware designs.

1.3 Multi-Objective Exploration and Optimizations

We extend the idea of regularized design space exploration and approximate computing to do smarter design search and multi-objective optimizations. With analytical models derived from L_1 regularization, we formulate non-linear optimization methodologies that enable us to optimize accelerator designs with respect to certain selected metrics while imposing constraints on other metrics. These chosen metrics and their values are selected to optimize designs within specifications of their target deployment.

We define cost functions and objective functions for various design metrics and derive different optimization formulations like: i) optimizing an accelerator's power under accuracy constraints; ii) optimizing an accelerator's design area within a given power budget; iii) minimizing accelerator architectural area under accuracy constraints; and iv) optimizing accelerator design area under throughput constraints.

There have been some work in optimizing design metrics after co-exploration, specially for throughput or power. For instance, works have been done to explore variety of architectural as well as algorithmic design choices to find optimum parameters for area and throughput constraints [12, 33, 70, 75]. However, most approaches presented in these work need to explore a large set of the design space for each platform to be able to perform their optimization via interpolation of the measured data. Our approach hopes to solve this problem by being able to analytically optimize design metrics by making use of the relative small design space we need to sample,

thanks to the previously-mentioned contributions of the L_1 regularization processes.

We also investigate the approximate computing methodologies proposed earlier and explore ways to make the automation as well as design search process smarter and better. We study different methodologies to evaluate fitness of approximate designs and compare their impact on power and area savings as well as output accuracy. We first instrument an iterative evolutionary approach to approximating a circuit design and propose an efficient way to choose design(s) from a particular iteration that is amenable to more transformations without deviating much from the original circuit correctness.

Inspired by the Non-dominated Sorting Genetic Algorithm II (NSGA2) we also implement a selection methodology at each iteration of our approximation flow so that Pareto-optimal points are constantly searched for from the design variants generated [19, 18, 25, 73]. Different non-dominated fronts of optimality and design fitness are created based on whether design points are dominated by other design points generated during an iteration, and also based on their spread from each other. Non-dominated design points represent designs that are optimal in the multi-objective sense, whereas the spread measure helps in identifying uniqueness of designs and serves as a better 'genetically diverse' feed for future approximations. This process ensures a global optimum selection from all rounds of approximations selecting a parent design that is multi-objectively 'optimal' instead of being trapped on a local optimum during the iterative refinement. We compare this global-optimality aware approach with an individual fitness ranking instrumentation where fitness is defined as a linear combination of weighted objectives like accuracy and power savings.

In addition to refining the design selection proces for approximate variants generation, we also introduce a novel approach where we apply targeted approximations

on timing-critical paths of circuits so as to reduce their circuit delay. Once circuit delay is reduced, we can use the net timing slacks to make use of standard voltage scaling processes to futher optimize approximate designs for better savings in power consumption.

1.4 Thesis Contributions

To summarize, this thesis makes the following contributions:

- We develop regression-based techniques to train mathematical models for various design metrics such as power, area, arithmetic accuracy and throughput for accelerator platforms like the FPGA. We propose the use of L_1-regularized least squares method to assess all possible interactions between the input design variables and use only the relevant ones to design the best model for power, area and accuracy under these variables. Our method completely automates the process of identifying the best model for each metric and is able to obtain 92% model accuracy by exploring as little as 0.42% of the vast design space.
- Given this automated design exploration methodology, we also develop formulations for multi-objective optimization by leveraging the best models developed from L_1 regularization for power, arithmetic accuracy and area in order to show different important design optimizations, such as minimizing power consumption under maximum arithmetic error tolerance, minimizing area for a given power budget, minimizing area under error limit restrictions, and maximizing accuracy over throughput constraints. Finding best configurations using modeling and non-linear optimization allows for a 340× speed-up over a manual-brute force design space exploration.

- We develop a novel toolflow called *ABACUS*, which automatically generates approximate designs from input behavioral descriptions of circuit implementations. Behavioral descriptions capture the algorithmic structure of the circuits, and thus, *ABACUS* transformations are of global nature rather than limited to a particular sub-circuit. This ensures that no prior knowledge of the design or the algorithm be needed for our approach to work efficiently. With *ABACUS*, we make the process of generating the variants transparent to the design flow, and thus, different design flows (e.g., ASICs or FPGAs) can be used and subsequently all standard synthesis optimization techniques are applicable on the approximate variants.

- We introduce ways to automatically identify timing-critical paths in a circuit and apply prioritized approximations on them first. This enables generation of approximate circuits that are not just smaller and use less area, but also run faster. This timing enhancement is coupled with a voltage scaling scheme with the *ABACUS* flow for additional savings in power consumption.

- We demonstrate that *ABACUS* can be integrated with complementary methods for approximate circuit generation (e.g., using an approximate adder, or even using fault-prone circuitry in implementation). By exploring local and global optimality aware fitness evaluation approach like NSGA-II and implementing some more one our own, we show a huge design-space can be explored in an efficient matter to generate circuits that consume up to 80% less area and power.

1.5 Organization of the Thesis

This thesis is organized as follows. Chapter 2 provides background and related work on established design-space exploration and circuit approximation methodologies. Several methods in which some of the current research work choose design parameters and use them to explore the design space are presented. Similarly, related work on currently accepted and used approximate circuit generation techniques are studied. In Chapter 3, we introduce our L_1 regularized methodology for design-space exploration and show the benefits of using this approach over brute-force enumeration as well as other commonly used analytical modeling works. We also study some of the multi-objective optimization formulations we discussed in Section 1.3. In Chapter 4, we present the high-level approximation methodology we have developed in generating optimal circuit and systems directly from their high level behavioral descriptions. We detail the implementation of our methodology called *ABACUS* and show why this methodology is superior over other works in approximate computing. We show the effectiveness of our approach by testing it on three different test benches from three domains, *viz.* machine learning, signal processing, and computer vision. In Chapter 5, we show how the design selection methodologies in Chapter 4 can be refined for more inclusive design space exploration. We also introduce our novel timing-critical path prioritized approximation techniques and its possible coupling with standard voltage scaling in Chaper 5. Finally, Chapter 6 presents our conclusions and future work.

Chapter 2

Background and Previous Work

In this chapter, we will present some fundamentals of design-space exploration and review various techniques proposed in prior literature. We will start out by describing some of the metrics that are important from the design point of view. We will then address specific methodologies to obtain designs that are considered optimal in terms of these design metrics. We will look for specific cases of optimization done so far for hardware accelerators, with greater focus on analytical modeling of the design space as well as inexact circuits and approximate computing as a means for obtaining low area/power circuit alternatives.

2.1 Important metrics for consideration in digital design

2.1.1 Power Consumption

Power consumption of any circuit is an important factor to consider while exploring the design space in the pursuit of optimal circuit designs. The optimality in regards to power, or energy, has become critical with the advent of smartphones, tablets and mobile computing platforms with limited battery lives and limited means to cooling devices. As a designer, it is imperative to know the fundamentals of power/energy consumption and methodologies to reduce it. Thus, we will begin by first reviewing the sources of power consumption in digital circuits:

CMOS power has two main components, namely, dynamic and static power:

$$\begin{aligned} P &= P_{dynamic} + P_{static} \\ &= P_{internal} + P_{switching} + P_{static} \end{aligned} \tag{2.1}$$

where P is the total power consumed by a circuit, $P_{dynamic}$ is the power consumed when there are logic switching activities, and P_{static} is the power consumed when the circuit is idle. Dynamic power, $P_{dynamic}$ can be further divided into short-circuit power or internal power, $P_{internal}$ and switching power, $P_{switching}$.

The switching power, $P_{switching}$, arises from charging and discharging of load capacitances when there are transitions between logic '0' and logic '1' levels, and is

usually the dominating component of power consumption. It is computed by:

$$P_{switching} = \alpha C V^2 f \tag{2.2}$$

where α is the activity factor or simply the fractions of the circuit components switching; C is the switching capacitance of the circuit, V is the supply voltage and f is the operating frequency. As is clear from Equation 2.2, switching power is affected quadratically by the supply voltage and linearly by the operating frequency of the circuit. The short-circuit component of dynamic power arises from the short period of time when both the pull-up and pull-down network in a logic gate are open, creating a direct short-circuit path from VDD to GND. This, from the design point of view is usually negligible provided that gates are sized properly such that rise/fall delays have similar magnitudes.

With aggressive technology scaling and packing of more and more transistors in a single die, static power has become a dominant factor in power consumption. Static power is a function of static current and the supply voltage, and can be expressed as:

$$P_{static} = I_{static} V \tag{2.3}$$

The static current itself depends on various factors like process parameters such as transistor gate-oxide thickness, threshold voltage as well as external parameters like temperature. All these factors can lead to sub-threshold channel current leakage:

$$I_{leakage} = I_0 e^{\frac{V_{gs}-V_T}{nV_T}}[1 - e^{-\frac{V_{ds}}{V_T}}] \tag{2.4}$$

where $I_0 = \frac{W\mu_0 C_{ox} V_{th}^2 e1.8}{L}$, $V_T = \frac{KT}{q}$ is the thermal voltage, V_{th} is the threshold voltage, V_{ds} and V_{gs} are the drain-to-source and gate-to-source voltages respectively. W and L are the effective transistor width and length, respectively. C_{ox} is the gate oxide capacitance, μ_0 is the carrier mobility and n is the sub threshold swing coefficient. It is important to note that static power is a growing concern among designers as increasing number of transistors are placed on a single die due to shrinking devices and more heat generated by these devices due to their switching at rather high frequencies. The dissipated heat translates to higher operating temperatures, which eventually lead to higher leakage current, as shown in Equation 2.4 by the exponential relationship of current with temperature. In short channel devices, source and drain depletion regions advance into the channel influencing the electrical field around the terminals. The phenomenon is called short-channel effects and indirectly contributes to threshold voltage drop thus also contributing to further leakage currents.

As a designer, there are many things to consider for design optimality and it is extremely important to devise power saving techniques that can be incorporated well in to smart design space exploration methodologies. In methods we propose in this thesis work, we try to reduce both dynamic and static power by mostly reducing the number of gates in a design, or simply, by reducing the logic complexity of a design. This eventually contributes to both dynamic and static power reduction by reducing number of active gates and switching capacitance.

2.1.2 Design Area

Design area, like power, is an important factor to consider during design space exploration. The advantages of considering low-area designs are multi-folds:

- Smaller circuits are correlated with low power consumption. As explained in the previous section, static power is a function of number of transistors/devices in a circuit, among others. This can be minimized substantially by considering smaller circuit designs with fewer transistors. Also, smaller area implies smaller switching capacitance, and thus minimizes dynamic power consumption as well.
- Smaller circuits have potentially faster execution times because of lower switching capacitance and smaller critical path delays.
- A smaller area design is economical from a fabrication and production point of view. Area determines manufacturing cost and determines the yield of production. Research works shows that die costs are proportional to the fourth power of the area [2]. Thus, it is evident that optimizing circuit area will reduce costs while providing more computation per unit area of the wafer.

In light of the above-mentioned factors, we will use area as an important constraint in the design space exploration processes to help find optimal design points.

2.1.3 Timing and Throughput

Circuit delay or timing is inherently related to the area of a design — larger designs with high gate-counts will potentially have a longer critical path, thus increasing

delay in a particular circuit operation. We will use this idea as a corollary — approximate circuits compared to their accurate counterparts are more likely to have superior timing characteristics because they have fewer gates needed to compute a signal or value and consequentially take less time to do computations.

Similarly, circuit delay is related with the supply voltage of a system. Complementary Metal Oxide Semiconductor (CMOS) technology used widely in today's circuits can be accurately modeled as lumped resistances and capacitances. This implies that performance of the circuits are directly dependent on the inherent resistances and capacitances in the circuit.

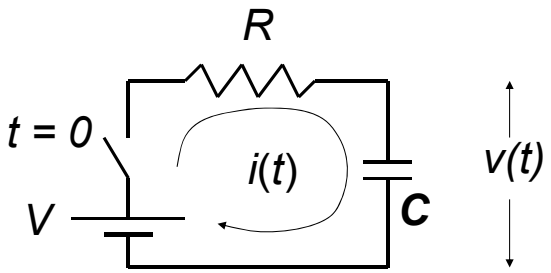

Figure 2.1: Simple RC network modeling of a circuit.

If a circuit is modeled as a simple resistor-capacitor network as seen in Figure 2.1, the voltage, $v(t)$, across capacitor C at time t, can be modeled as seen in Equation 2.5:

$$v(t) = V[1 - e^{\frac{-t}{RC}}] \tag{2.5}$$

R and C are the device resistance and capacitance. The product, RC is also called the τ, or the time constant, and is the measure of circuit delay or specifically, the time taken for the capacitor to charge by about 63.2% of the supply voltage.

A higher supply voltage translates to a lower effective resistance because there is more current and hence makes circuits faster. Thus, as a designer it is imperative to take into account device resistances, capacitive loads (which increase with number of circuit components) as well as given supply voltage to meet good timing behavior for a design.

Overall circuit performance, as measured by overall timing, is also directly related with circuit throughput. Throughput in the simplest terms can be defined as the number of computations per unit time. Throughput is usually dependent on and affect other circuit parameters and metrics like area and power. More area (more devices running in parallel) could potentially increase the throughput of the circuit but it also comes at a cost of increased area and power. Similarly, increasing the frequency of a device is likely to increase the throughput of a system but that increases design power consumption.

It is important, as well as difficult to find the sweet spot where all three metrics are optimized in a design. That is where a smart design space exploration comes in handy. We will talk about that next.

2.2 Design Space Exploration

Design space exploration is essentially exploration of different viable options for a design prior to its actual physical implementation. As we mentioned in Section 1.1, for any particular design implementation, there could be hundreds, thousands or even millions of ways of doing the physical implementation. Enumerating over every single possibility for its feasibility as a final implementation does not make much

sense. This is where doing an effective design space exploration comes in handy. A good space exploration can help a designer in many ways [38], including:

- Rapid prototyping: Design space exploration can be used to generate a set of prototypes and generation of these prototypes and the design parameters used are helpful in understanding design dynamics, such as the weight of each parameter on various design metrics, their importance and interdependence.
- Optimization: Design space exploration can be used to optimize designs for various design metrics - area, power, throughput, timing etc. Eliminating bad design choices from the very beginning of the exploration process can lead to a very cost-effective way of reaching optimality goals.
- System Integration: Design space exploration can also be used to find best configurations of various components within an inclusive system to meet a global design constraint.

Our attempt at design space exploration in this thesis will consider all three advantages. That is, we will use smart design space exploration techniques that will pay special attention to interdependence of design variables and use them to assemble various components on hardware accelerators to carry out multi-objective optimizations. Our methodologies offer a new approach compared to those found in the literature.

2.2.1 Synthesis and Compiler Level Techniques

There have been a few work in prior literature about high level synthesis tools coupled with compiler techniques used for automated design space exploration. These

techniques use the compiler's awareness complimented with manual programmer intervention to produce various possible implementations of a single design. Automation in design space search process implemented at the compiler level may lead to exploration options, that otherwise may not be seen by the designer or are simply tedious to implement manually.

A common example of where compiler level space exploration is achieved is in hardware implementations that incorporate a number of coded loops. In pursuit of a good hardware implementation that exploits parallelism to produce a good computation throughput, designers manually apply loop transformations such as loop-unrolling [72]. For example,

```
always @(posedge clk)
begin
   for (i=0; i<65; i++)
   begin
      for (j=0; j<65; j++)
      begin
         out[i] <= out[i] + (in1[i+j] * in2[i]);
      end
   end
end
```

can also be written as:

```
always @(posedge clk)
begin
   for (i=0; i<65; i=i+3)
   begin
      for (j=0; j<65; j=j+3)
      begin
         out[i] <= out[i] + (in1[i+j] * in2[i]);
         out[i+1] <= out[i+1] + (in1[i+j+1] * in2[i+1]);
         out[i+2] <= out[i+2] + (in1[i+j+2] * in2[i+2]);
      end
   end
end
```

Loop unrolling exposes instruction-level parallelism at the expense of hardware resources. In the example shown above, unrolling exposes operator parallelism to high-level synthesis by allowing all the multiplications to be performed in parallel. However, unrolling also increases the amount of data a computation requires and can be memory intensive. Thus too much unrolling can lead to a heavy memory bound implementation and leave computation resources idle for a lengthy period of time. It is hard to come up with a good space-time tradeoff just by looking at a loop implementation.

Tools where the programmers must manually determine the unroll factor and location and impact of loop unrolling on design metrics are available [53]. Other automated methodologies that discover which loops are beneficial to be transformed (among other compiler optimization techniques - like scalar replacement, dead-code elimination) by collaborating between parallelizing compiler technology and high-level synthesis tools have been developed [72]. For such automated implementation, estimates derived from behavioral synthesis of a small sample of the huge design space are used to predict space-time tradeoffs for the design and to figure out when it is beneficial to devote more resources to storage or computation. Results show that a mere 0.3% exploration of the design space was able to derive a design that closely matches the best performance within the design space and was smaller than other designs with comparable performance.

2.2.2 Pareto Optimal Techniques

For design space exploration processes involving multi-objective optimization, there may be more than one optimal design. While one design may be better in one particular design objective, there may be another or multiple others that could be better

in other explored objectives. Therefore, optimality for these kinds of optimization can be represented as a *Pareto front* that represents points which are better than all others in the design space in at least one design objective. A simple illustration of pareto optimality in a design space with two design objectives is shown in Figure 2.2. The line connecting the outer set of points represents the optimal set of designs. There are several works in prior literature that make use of pareto optimality techniques for design space exploration. We discuss some of them hereafter.

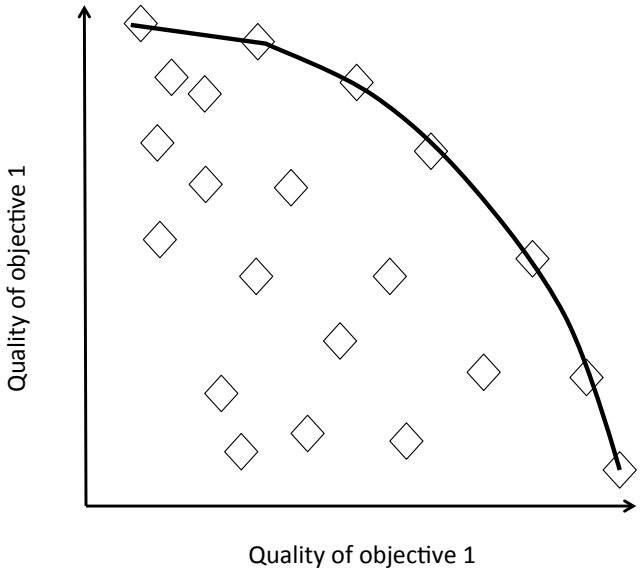

Figure 2.2: Pareto optimality in a design space.

Single Factor Analysis

An approach using Single Factor Analysis (SFA) is presented [60]. SFA changes design parameters in an isolated fashion - one at a time until all possible parameter assignments for each parameter are obtained. This process is repeated for each design variable and results of SFA can be used to create a regression model to predict optimal designs for given design budget. The effectiveness of this process would depend highly on the kind of models derived from initial isolated experiments

with individual parameters.

This approach has been extended to generate Pareto optimal points in a design. Pareto points in a design space are those points which are better than all others in at least one design objective. In their work, authors first determine the significance of each parameter is by observing the maximum change seen for a given parameter, which is then normalized to a value between 0 and 1 [67]. This is done for each parameter of interest and the parameters are eventually sorted by their normalized weight values. To take interdependence between design parameters into account, two highest-impact parameters are considered and all possible interactions between them are generated. Results are filtered for Pareto points and the resulting set of intermediate Pareto points are then used with the third most impactful parameter to generate a new set of design points. Pareto points are constantly pruned after each parameter consideration.

While the Pareto generation methodology using SFA is helpful, it may fail to see all possible interactions between parameters because it is primarily focused on generating Pareto optimality near base configurations chosen by the designer. Some other intricate interdependencies may be ignored.

Parameter Interdependency Graph

Other works have been done in pre-identifying interdependencies among design parameters and finding all Pareto-optimal configurations of parameterized System-on-Chip (SOC) architectures [28, 27]. An architecture consisting of a processor, two caches, buses, and memory, with each component having numerous parameters, is explored. While there are many interdependencies between parameter groups, some groups have only a few possible assignments. Each of these parameter groups is

searched exhaustively and local Pareto points are identified, thus avoiding an exhaustive global exhaustive search. From these local Pareto searches, the designer can identify parameter interactions represented in what is called a Pareto Interdependency Graph (PIG).

This work is shown to be much faster than SFA, searching less than 1% of all possible configurations; however, one big problem still exists: PIGs are to be implemented manually, requiring the designer to identify and choose interactions between design parameters. Another limitation with this approach is that all dependencies are given equal significance and in reality, this is likely not true.

Randomized Approaches

Several other randomized search approaches have also been developed to find Pareto points faster without having to know interdependencies beforehand [68]. Quite a few of these approaches start with a random subset of configurations and start exploring the design-space with the goals of identifying Pareto-optimal points. The idea is to start out at a few possible configurations and then use some guidance towards optimal Pareto points [3, 5, 6]. This guidance can be evolutionary in process, first choosing a few good designs and then combining features from them to further evolve the design space, much like genetic algorithms. Some researchers have ompared [21] multiple genetic algorithms like the Strength Pareto Evolutionary Algorithm (SPEA2) [85, 84] and Non-dominating Sorted Genetic Algorithm (NSGA-II) [19] in terms of performance and runtime to come up with Pareto optimal points.

Similarly, Pareto Simulated Annealing and Pareto Reactive Tabu Search are some other randomized search methodologies used in Pareto optimal design points for effective design space exploration. While these methods are quite effective in producing

good Pareto optimal points, their execution runtimes are long.

Design of Experiments Paradigm

Design of Experiments (DoE) was first presented as a set of experiments that gives information about design parameter interactions and their impact on the output [61]. With these sets of experiments, a few things can be found:

- whether each parameter affects output positively or negatively
- which parameters contribute to the output
- how these important factors interact with one another.

DoE can be used with Pareto generation methodologies for smart design space exploration [68]. In this multi-phase approach, the first phase automatically generates a parameter interdependency graph with weighted edges. Running algorithms such as Plackett-Burman [61], a set of experiments are conducted to evaluate parameter interdependency. For example, if setting parameter A to its high level increases runtime by 10ms, and doing the same to parameter B increases it by 15 ms, the estimated runtime for when both are high would be 10+15=25ms. A physical implementation of the design is done setting the parameters A and B to these same values to see the validity of the estimated values. If implemented and estimated values are different, it suggests that there is some interdependency between the two parameters; the difference is stored as 'edge error' which essentially serves as the weighted edge value for parameter interdependence. Next, the second phase starts with the pair of parameters that has the highest edge value and a subset of these parameter values is used to generate exhaustive data for that particular pair of configurations. Local

Pareto points are selected out of these configurations. If a parameter has more than three possible configurations, intermediate configurations are explored by generating local Pareto regions and guiding the designer towards configurations that would be of more interest.

Again, as effective as the DoE methods may be in doing design space exploration, they can be slow because of the inherently slow processes of generating parameter interdependencies, creating local Pareto fronts and filling in intermediate configurations from these information.

2.2.3 Regression based Machine Learning Techniques

Use of analytical models using design parameters to evaluate design metrics for a large design space have also been explored. Methodologies that relate accelerator parameters like speed and other metrics of the circuit analytically using statistical analysis and similar mathematical modeling have been developed and studied [17, 36, 46, 71].

These studies propose ways to formulate a design metric as a function of the input design parameters and try to estimate the metric function to fit the actual data as closely as possible.

Normally, macro-models for power, area, throughput expressed in a mathematical form are characterizations of these metrics derived from a few sample configurations of the design parameters. A small number of designs from the huge design space is usually sampled and their corresponding impact on the design metric is logged. Finally these logged real metric measurements along with their parameter configurations are analyzed by statistical regression using curve fitting or machine learning

tools to get the estimation function for the metric.

Generating accurate analytical models is important but not easy. To realize a statistical regression model, it is first important to establish a relationship between each of the design parameters and the various metrics used to define the 'goodness' of a design, such as power, throughput, area and arithmetic accuracy implications. Typically, with an understanding of the sensitivity to each of these metrics, ad-hoc models are formulated using experimental observations.

Let's say a design metric y depends on four different input parameters x_1, x_2, x_3, and x_4. These parameters reflect the dependency of the output metrics on algorithm and/or hardware design parameters. A *linear model* representing the relationship between these input variables and the design metric is given by $y = c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$; a *pure quadratic model* is given by $y = c_0 + c_1x_1^2 + c_2x_2^2 + c_3x_3^2 + c_4x_4^2 + c_5x_1 + c_6x_2 + c_7x_3 + c_8x_4$; and a *quadratic model with interaction terms* is given by $y = c_0 + c_1x_1^2 + c_2x_2^2 + c_3x_3^2 + c_4x_4^2 + c_5x_1x_2 + c_6x_1x_3 + c_7x_2x_3 + c_8x_1 + c_9x_2 + c_{10}x_3 + c_{11}x_4$, where $c_0, \ldots, c_n$ are coefficients that give the weights of each of the n terms in the model. Cubic and interactive cubic models are similar to quadratic and interactive quadratic except for the added degree in the model polynomial function.

The model that best represents the design metric characterization depends on which parameters are more sensitive than others and which parameters interact to affect the output. If and when these parameter sensitivity and interactions are carefully chosen, analytical models can lead to very fast and accurate design space exploration. This is exactly what we explore in this thesis — an analytical approach to doing fast and smart design space exploration by taking advantage of the reconfigurability of hardware accelerator platforms like the FPGA.

2.3 Approximate Computing

Like we mentioned in Section 1.2, not all designs require 100% accuracy. Many algorithmic implementations in the domains of computer vision, machine learning and computer graphics can allow controlled error in their underlying computations and still perform within reasonable accuracy boundaries. This gives designers ways to complement other design space exploration methodologies by considering approximate computation as a part of it as well.

The benefits of studying and exploring approximate computing can help designers in multiple ways:

- Low Area/Low Power Circuits: Approximate Computing can perform approximations on data-types, data-structures, arithmetic operators and such, thereby making the circuits less complex and in effect reducing area footprints and power consumption

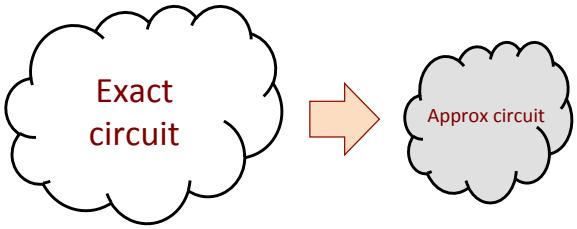

Figure 2.3: Illustration of a possible low-area approximate circuit.

- High Throughput Circuits: If area and power are not tightly constrained, the designer can focus on increasing circuit throughput by using multiple instances of these smaller, less complex approximate versions in parallel.

In this thesis, we look at approximate computing as a way of creating less complex custom computing circuit implementations by introducing approximations by design.

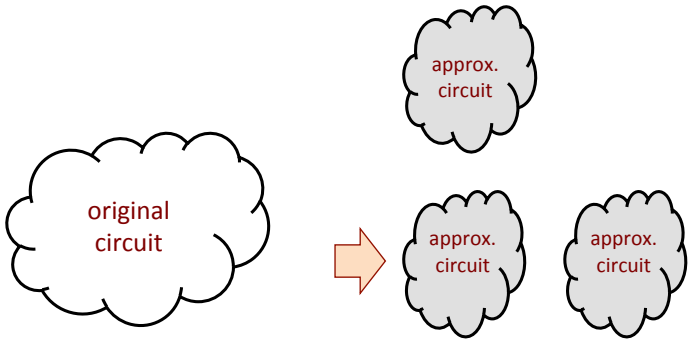

Figure 2.4: Illustration of a possible high-throughput approximate system.

Our focus will be on reducing power and area, but it is also possible to target this technique for reducing circuit delay. There have been two circuit approaches when it comes to doing approximate custom computing, *viz.* run-time approximation and design-time approximation. We discuss these two approached in the next two subsections.

2.3.1 Run-time approximation techniques

A very popular run-time approximation technique is Dynamic Voltage Scaling (DVS), where as the name suggests, the supply voltage is dynamically adjusted in order to save power [11, 62, 29, 63, 59]. Recall from Equation 2.2 that dynamic power dissipation of a system is directly proportional to the square of the supply voltage provided to it. As can be seen from discussion in Section 2.1.1, by scaling down the voltage, there is a large potential for power saving. However, lowered supply voltage will also affect circuit timing as discussed in Section 2.1.3. Circuit delay increases with lowered power supply and this may lead to functional as well as timing errors in the system, which can translate to circuit behavior that is only approximately correct. From a designer's perspective this would be the same as an approximate circuit that has some defects otherwise and produces functional errors within possibly an acceptable range.

The potential disadvantage to this method is that reduction in supply voltage may cause intermittent functional errors that are not easy to quantify. There has been research focused on ways to quantify and mitigate such errors that come from supply voltage reduction, but the solutions usually come at the expense of runtime and additional area.

For example, a technique named RAZOR is used to dynamically detect and correct timing errors caused by dynamic voltage scaling [22]. The key idea behind RAZOR is to tune the supply voltage by monitoring error rate during operation, thereby eliminating the need for voltage margins. As shown in Figure 2.5, a RAZOR flip-flop is introduced where pipeline data are double sampled, once with a fast clock, and again with a time-borrowing delayed clock. A comparator, that is tolerant to metastability then compares the two data samples and in case of discrepancy, uses a misspeculation recovery mechanism to restate correct behavior for any circuit.

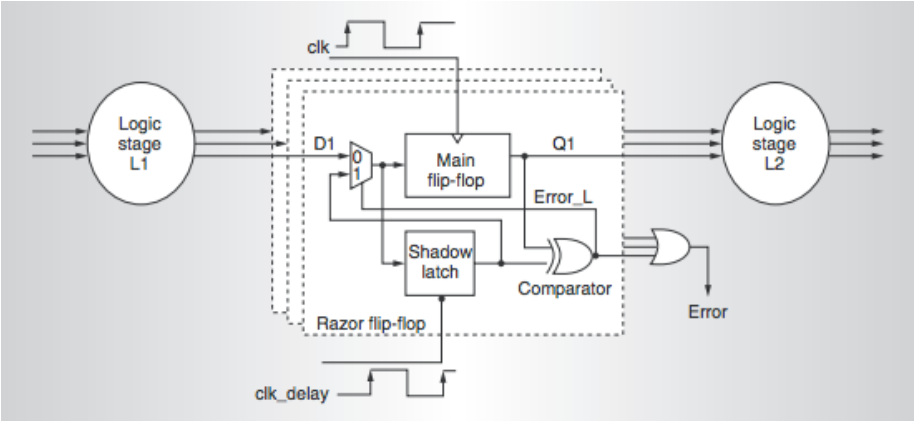

Figure 2.5: RAZOR implementation [22].

2.3.2 Design-time approximation techniques

In more recent years, direct logical approximation has been studied as an alternative approach, where designers intentionally introduce functional inaccuracies in their approach as a means of saving power or area. This approach is fundamentally different from the aforementioned run-time approach where functional errors are introduced as a by-product of not meeting timing specification. Today, most approximation work on circuit design is done by deducing some sort of a 'short-cut' logical derivation so as to reduce either circuit delay and/or area and power consumption. Arithmetic functions such as adders and multipliers may also be approximated (for delay, area, and/or power advantages) by making changes at the algorithmic, logic, or transistor levels. Some of these methodologies are described in the subsections to follow.

Micro-architecture level approximation

Researchers propose data value speculation schemes as opposed to value prediction schemes for use in processors [39]. Processors commonly use branch prediction to speculatively execute instructions and while various schemes are used for branch prediction, there are not many works on prediction and approximation of arithmetic values. The authors discuss two basic designs for arithmetic approximating units, using a ripple carry adder as an example, and ultimately include these modifications on a MIPS architecture model. They design 'incomplete' adders for computation of results earlier than the worse-case completion time, and study the effects of this on the probability of errors, as well as design a 'temporarily' incomplete adder which is clocked at a rate that will violate worst-case design. Same authors also consider the use of these approximate adders and other execution units in speculative execution of instructions for possible enhancement in system performance by increasing

instructions per cycle.

Similarly, approximation concepts have been applied to various stages of a superscalar processor like execution, rename logic and issue logic, by implementing approximate adders and Booth multipliers [51]. Similar approximation techniques like implementing adder and multiplication logic for the average case situation rather than worst case scenarios are used to save circuit delay and increase frequency.

Algorithm-level approximation

Approximation by reducing complexity has been explored in several works at the algorithm level. Researchers have implemented an H.264 video encoder in software and have instrumented a real-time scheme for determining computational complexity [34]. A fast Mode Decision (MD) method is presented which reduces computational complexity by simply adjusting and approximating some of the encoding parameters. By defining a single quality loss metric based on both bitrate loss and distortion, it is shown that complexity parameters can be dynamically tuned so as to achieve faster encoding.

Other works have proposed a similar approximation for Motion Estimation (ME) scheme used in H.264 /MPEG-4 encoding. A Run-Time Adaptive Predictive Energy Budgeting (enBudget) scheme for energy-aware ME is presented that predicts energy budget for different video frames and different Macroblocks (MBs) in an adaptive manner considering run-time changing scenarios and available energy, video frame characteristics, and user-defined coding constraints while maintaining a reasonable computational accuracy [66].

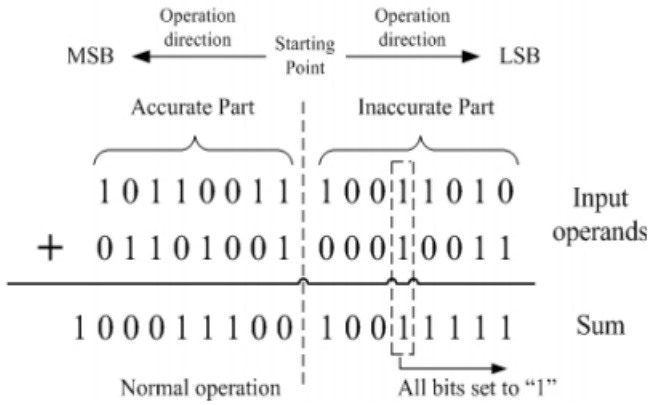

Figure 2.6: Error Tolerant Adder II from [79].

Logic synthesis approximation

Over the past few years, there have also been some work targeting approximation at the logic synthesis or optimization level of abstraction. A fast adder is proposed where for two n bit integers added, the longest sequence of propagated signals is approximated as log_n on average [83]. This approximation leads to inaccurate but faster adders. A similar approximation scheme is presented where addition is performed by splitting the input operands into two parts: an accurate part that includes several higher order bits and the inaccurate part that is made up of the remaining lower order bits as shown in Figure 2.6 [79]. While the accurate part performs normal operation, the inaccurate sets of bits go under a parallel set of addition operations where no carry signal is generated or taken in at any bit location, thus avoiding the carry propagation delay. To minimize overall error, every bit position from left to right is checked for the inaccurate part and if both input bits at that location are '0' or different, normal addition is performed. However, if both input bits of a particular portion are logic '1', then the checking process is stopped and all output bits to the right of that bit location are approximated to '1'. Therefore by reducing the carry chain and performing addition in two separate parts, overall circuit delay is greatly reduced.

A common technique among researchers for generating approximate adders is setting the carry bits across accurate and inaccurate segments to zeros. This approach may have some weakness, however — using a constant '0' as the carry-in at the cutoff point of the critical path, leads to negatively-biased errors because '0' is always an underestimation of the carry-in bit. Similarly, using a '1' produces positively-biased errors because of over-estimation. The proposed solution is to take in the carry-in-bit from the bit immediately before the propagate chain. If the inputs are random, every bit has a 50% chance of being either a '0' or a '1' and ultimately an unbiased additive result will be produced at the output [32].

Another approximation work proposed a 2×2 inaccurate multiplier, where its logic functionality is altered by modifying the Karnaugh Maps (K-map) [43]. For example, by representing the result of 3 * 3 using three bits (111) instead of four (1001), it is possible to significantly reduce the complexity of the circuit, thus reducing area and power consumption. These changes are made with small probabilities for larger output functions thus ensuring circuit correctness for the most part. The authors show an area reduction of 50% could be achieved.

A similar logic synthesis approach has been proposed where approximate is done by complementing some of the minterms in a logic function so that they can be minimized using a K-map minimization technique [69]. This helps in reducing the number of literals from the original minimum cover for a logic function and thus reduces circuit area. A heuristic is developed to perform exhaustive search for one or two minterms that can be complemented in a logic function description. About 9.43% literals could be reduced for a 1% error rate threshold using this approach.

In a slightly different approach, researchers first derive a Quality Constraint Circuit (QCC) to formulate the problem of approximate synthesis [78]. The original

circuit, approximate circuit and a Quality function, Q, form the QCC. The error constraints that are to be satisfied for a successful generation of an approximate circuit are encoded in Q. The QCC takes in primary inputs to the original circuit as its own inputs. In the QCC primary outputs from the approximate circuit that do not cause the Q function to report a '0' are considered valid. The authors then try to find out a set of input values for which the primary output of the approximate circuit is unaffected, meaning it doesn't change the Q function. These values are called Approximate Don't Cares (ADCs). By synthesizing the logic using the ADC information without violating the Q function, acceptable approximations for logic synthesis is derived. Power savings from 1.15×-1.75× for 1% accuracy degradation and 1.3×-5.25× savings for a 20% degradation is reported, while area savings between 1.1×-1.85× and up to 4.75× is reported for similar tight and relaxed error constraints respectively.

Gate-level approximation

Gate-level approximation techniques have been equally explored for approximation. A "Lower-Part-OR-Adder" is proposed where a p bit addition is divided into m and n bit additions where $m + n = p$ [52]. As seen in Figure 2.7, from the p bits from each operand for the adder, the m most significant bits are passed throughout a conventional adder and the lower n significant bits are 'added' using an OR gate. Essentially, by performing the bit-wise OR operation, the addition is underestimated. Again, logic simplification this way leads to area as well as power savings. Use of these imprecise adders in fuzzy and neural network fields work well enough for needed computational accuracy at a much reduced area, power and delay cost.

In other work, researchers have redesigned data path modules using approxima-

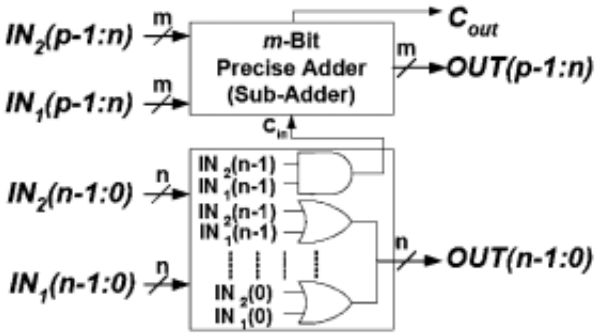

Figure 2.7: Lower-Part-OR-Adder from [52].

tion techniques in order to reduce circuit delay and increase manufacturing yield [20]. The idea is to accept a reasonable amount of error in the system, introduced by design, to increase yield of chips which would otherwise may have been 'defective'. Especially chips containing data path modules are used in applications such as images, video, audio, graphs, games, etc and are considered for approximation. Gate-level simplifications such as simply transforming an AND gate to logic '0' and transforming a two-input X-OR gate to a two-input OR gates are done for approximation purposes. These approaches are applied in synthesis of bigger ripple carry adders, carry-skip adders, and carry look-ahead adders and reductions of up to 10.3%, 6.4%, and 42% are achieved for delay, area and yield improvement respectively. Similar approximation to adders where lower significant bits are truncated so that output accuracy is not greatly affected is another idea discussed as 'Sloppy Addition' [4].

Transistor-level approximation

Approximation is also done by simplifying logic complexity at the transistor level. A conventional mirror adder is simplified by reducing the number of transistors and internal node capacitances and five different approximate versions are proposed to ensure minimal errors in the full adder truth table [30]. These approximate full adder cells are then used in larger implementations as building blocks of other DSP

systems. Approximate adders, in the form of ripple carry adders and carry save adders are used to replace their accurate counterparts for only the least significant bits.

2.4 Improvements for Design Approximation

We have seen from the previous sections that design approximation is an emerging topic of interest in the research field. It has been studied at various levels of abstractions with positive results in circuit delay, area and power consumption improvements. As signal processing, machine learning or data mining applications continue to go through rapid development and grow to be much more complex and power consuming, the benefits that these conventional methodologies can offer have become somewhat limited. Hence the effort of circuit optimizations has been shifted to exploring the possibility of multi-level approximate circuits or architecture-level approximation [14, 78]. Nevertheless, these efforts are either confined to logic approximation either at the Boolean or gate levels or demand high application-specific knowledge.

There truly is a lack of methodology that can work directly on the behavioral description of a circuit, without having to know any information about its functionality beforehand. This is where our work becomes useful. We explore the possibilities of circuit optimizations and smart design space exploration by looking at approximate computing from an angle not explored before — working directly at the behavioral level description of a circuit as an attempt to exploit higher-level transformations and their potential positive effects on design metrics. The work proposed in this thesis can be generalized to all circuits and will not focus on improving a particular arithmetic unit or a particular data-path. It works rather at a much higher level

of abstraction exploring opportunities to approximate arithmetic units, data-types, circuit code structure, among others to generate scalable imprecise circuits while still performing at a reasonable accuracy margin.

Chapter 3

Analytical Method for Fast Design Space Exploration

In Chapters 1 and 2, we discuss the need for a fast way to do design space exploration of hardware accelerators. We also present some of the analytical methods that are used to do so. However, there are still unanswered questions about how these models are chosen or how to accurately identify and define parameter sensitivities, interactions and their impact on the final design objectives. In this chapter, we aim at making the analytical modeling and parameter selection and weighing process automated.

We turn our focus on FPGAs as our hardware accelerator platform; however, our general approach is applicable to other accelerators as well. FPGAs are becoming widely used in real-time image processing, which is used in surveillance, scientific research, smart camera technologies and automotive industries [7]. Their reconfigurability gives them an edge over other platforms when it comes to both prototyping and implementation of algorithms involving high-throughput computation. We observe that FPGA-based accelerators, especially those that are used for image processing

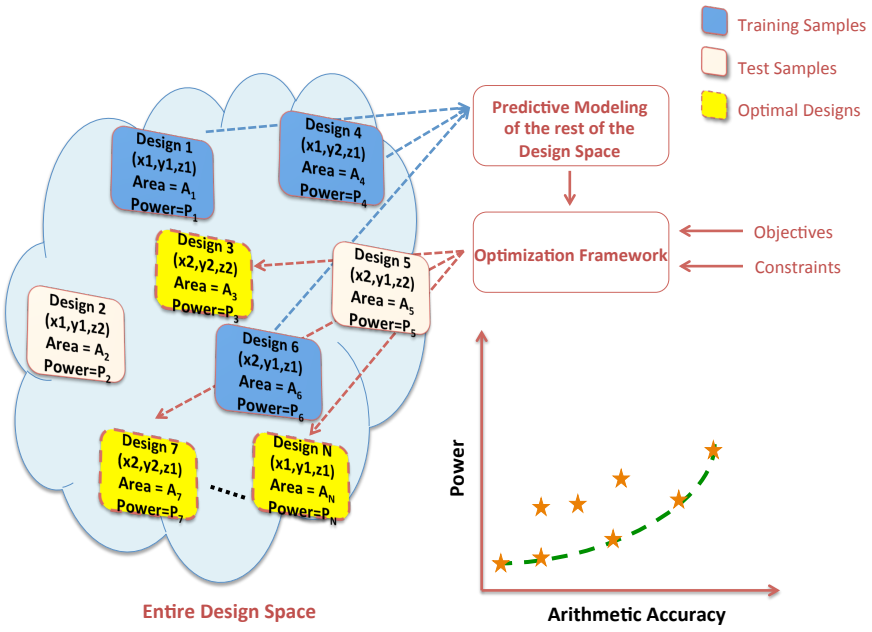

Figure 3.1: Illustration of the idea of using regression based modeling for design space exploration and finding important designs based on objectives and constraints. Each star on the graph on the right represents a design variant and the dashed line represents the Pareto frontier. Designs shown in dashed yellow boxes represent optimal designs given by the optimization framework while the ones in blue represent the training set.

and similar applications, offer many algorithmic and hardware design parameters, which when properly chosen, can lead to outcomes with the desired throughput, power, design area and arithmetic accuracy. However, identification of the right configuration of parameters to obtain optimal designs may not be an easy process. While reconfigurability offers opportunities for fast and iterative prototyping, the process to discover optimum design implementations can still be time-consuming if every possible design point were to be physically synthesized and characterized.

We propose a fast design space exploration methodology possible on reconfigurable accelerator platforms. We demonstrate how such methodology can be used to converge accurately and swiftly on design points that represent design optimality across dimensions like design power, accuracy, area and throughput. The main ob-

jective of this work is to be able to give the designers a faster and more efficient way of automatically selecting the best configurations of the most impactful parameters in a system.

A simplified illustration of our problem statement and proposed solution is presented in Figure 3.1, where we have a system with three design parameters: x, y and z. The total design space for this system consists of any permutation of these three design parameters. Each design dissipates power differently and has certain arithmetic accuracy. Assume we are interested in figuring out which design gives us the best trade-off between power and accuracy, i.e. we want to design an accelerator that dissipates as little power as possible while its accuracy is still maintained at an acceptable level. Toward that goal, we first identify accurate models for the design metrics. Thereafter, to find the optimal design variants, we feed the results from the predicted accuracy and power metrics from regression modeling into an optimization framework. The optimization framework presents a subset of those variants that create a Pareto frontier (dashed green line on Figure 3.1), where the frontier points do not dominate each other in both accuracy and power, but dominate other non-frontier points. The Pareto frontier points represent the optimal trade-off between arithmetic accuracy and power. It is up to the designer to pick from these optimal designs (marked as dashed yellow boxes in the design space) depending on the allowed accuracy and/or power budget. By doing all this automatically, we take the guesswork out of the designer's part and greatly reduce the time and resource needed for an efficient design space exploration.

Before moving forward, let's define some of the terms that are commonly used in the discussion in subsequent sections.

1. *Design metrics*: Measureable output of a design such as area, power consumption, throughput, and arithmetic accuracy.

2. *Design parameters/variables*: Adjustable design choices that can have various configurations. Design choices can be at the algorithm level or at the hardware architecture level.

3.1 Analytical Modeling of Design Metrics

To speed up design exploration, we propose to sample the large design space and then use regression models and statistical inference from the samples to obtain analytical models that accurately describe the different characteristics of the system as it pertains to power, area and throughput. In this way, our approach is similar to that proposed in prior works [36, 47]; however, our specific applications are different. To evaluate the design metrics for any combination of parameters, the models can be *queried* with any possible values of the parameters involved. In the following sections, we will outline our model generation and validation methodology as well as the optimization framework that uses these models to solve multi-objective optimization problems.

3.1.1 Design Sampling and Characterization

Following DoE principles as discussed in Section 2.2.2, it is important to sample a subset of the design space but in a uniform way to capture the essential features of the design.We do this by selecting design combinations randomly from within the design space. We incorporate possible minimum and maximum configurations of

each of the parameters in our training samples so that we consider the full range of the design space. In this way, the models output predictions that span the range of possible designs such that our optimization framework can identify the configurations that lead to optimal designs.

These sample combinations are implemented in the design and the resultant metrics characterized from real measurements and/or from synthesis tool results.

3.1.2 Training Set Generation and Regression Analysis

The characterized results will help train the model generation process towards achieving accurate analytical models. To realize a statistical regression model, it is first important to establish a relationship between each of the design parameters and their power, throughput, area and arithmetic accuracy implications. Typically, with an understanding of the sensitivity to each of these metrics, ad-hoc models can be formulated using experimental observations. Let's say a design metric y depends on four different input parameters x_1, x_2, x_3, and x_4. A particular model could be used to express the dependency of the output performance metrics on the various parameters.For instance, a *linear model* representing the relationship between these input variables and the design metric is given by $y = c_0 + c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$; a *pure quadratic model* is given by $y = c_0+c_1x_1^2+c_2x_2^2+c_3x_3^2+c_4x_4^2+c_5x_1+c_6x_2+c_7x_3+c_8x_4$; and a *quadratic model with interaction terms* is given by $y = c_0+c_1x_1^2+c_2x_2^2+c_3x_3^2+c_4x_4^2 + c_5x_1x_2 + c_6x_1x_3 + c_7x_2x_3 + c_8x_1 + c_9x_2 + c_{10}x_3 + c_{11}x_4$, where $c_0, \ldots, c_n$ are coefficients that give the weights of each of the n terms in the model. Cubic and interactive cubic models are similar to quadratic and interactive quadratic except for the added degree in the model polynomial function

To train the models and obtain the coefficients, the mathematical models are fit to the measured samples using least squares estimation. Note that while the model could be non-linear in its variables, it is linear in its coefficients. If k design space points have been sampled, the design choice values are plugged into corresponding variables in the model to get a design matrix $\mathbf{X}$. For instance, if the chosen model is $y = c_0 + c_1 x_1^2 + c_2 x_2^2 + c_3 x_3^2 + c_4 x_4^2$, then

$$\mathbf{X} = \begin{bmatrix} x_1^2{}_{|x_1=a_{11}} & x_2^2{}_{|x_2=a_{12}} & x_3^2{}_{|x_3=a_{13}} & x_4^2{}_{|x_4=a_{14}} & 1 \\ x_1^2{}_{|x_1=a_{21}} & x_2^2{}_{|x_2=a_{22}} & x_3^2{}_{|x_3=a_{23}} & x_4^2{}_{|x_4=a_{24}} & 1 \\ \vdots & & & & \\ x_1^2{}_{|x_1=a_{k1}} & x_2^2{}_{|x_2=a_{k2}} & x_3^2{}_{|x_3=a_{k3}} & x_4^2{}_{|x_4=a_{k4}} & 1 \end{bmatrix}$$

is the data set for different combinations of design variables, where a_{ij} is the value of the j^{th} term resultant from the algorithm-design parameter values at the i^{th} sample point. The k measured values for the desired objective y are set up in a vector, $\mathbf{y} = (y_1, y_2, \ldots, y_k)$. Thus, the entire set of samples can be represented as

$$\mathbf{X} \begin{bmatrix} c_1 \\ \vdots \\ c_4 \end{bmatrix} + \epsilon = \mathbf{Xc} + \epsilon = \mathbf{y},$$

where ϵ is a vector that gives the impact of non-modeled phenomena or noise on the outputs of tool estimates and measurements. Then the model coefficients, $\hat{\mathbf{c}}$, that minimize the total squared error (i.e., $||\mathbf{y} - \mathbf{Xc}||_2$) is given by

$$\hat{\mathbf{c}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y},$$

where $\mathbf{X}^T$ is the transpose of $\mathbf{X}$.

3.1.3 Use of L-1 Regularization in Model Generation

While regression techniques like the one discussed above have been used by previous works in design exploration, we identified a number of shortcomings:

- If there are interactions between two or more variables, a model derived from merely the individual relationships between isolated design parameters and the design metric would not be accurate.
- To capture the interactions between different algorithm and design parameters, the designers might need to make educated guesses on the interactions between the variables to identify the appropriate terms in the model. These educated guesses are usually guided by design-of-experiment methods. In all cases, a large number of models with different interaction terms are generated, trained, and the closeness of each model to the measured metrics is evaluated.

To address this limitation and to automate the process of identifying the closest model to the measurements, we propose the use of L_1*-norm based regularization*. In this case, we start with a model that captures all possible interactions between the algorithm-design parameters. Then we can solve for $\hat{\mathbf{c}}$ by minimizing

$$||\mathbf{y} - \mathbf{X}\mathbf{c}||_2^2 + \lambda||\mathbf{c}||_1,$$

where $||\mathbf{y} - \mathbf{X}\mathbf{c}||_2^2$ is the total squared error, and $||\mathbf{c}||_1$ represents, the L_1-norm or the summation of the absolute value of all coefficients of $\mathbf{c}$, i.e., $\sum_{i=1}^{n} |c_i|$. The minimization of the L_1 norm of $\mathbf{c}$ attempts to *sparsify* the coefficients $\mathbf{c}$ [41]. Coefficients

that get relatively small numerical values indicate interaction terms that are not important towards estimating the target metrics. By suppressing interactions or features that are irrelevant to the model during training, we avoid the problem of overfitting the model to the given training set of samples. Overfitting reduces the accuracy of the model when queried with non-training samples. Alternatively, an L_2 regularization approach could be considered where $\hat{\mathbf{c}}$ is solved for by minimizing

$$||\mathbf{y} - \mathbf{X}\mathbf{c}||_2^2 + \lambda||\mathbf{c}||_2^2,$$

This approach however minimizes the L_2 norm, essentially penalizing coefficients with higher magnitude square terms more. This does not filter out the noise in a model and keeps sparse features in the model selection process which may result in model over-fitting. Using a regularization parameter λ, which penalizes all coefficients equally, the unwarranted complexity of the design matrix that stems from these irregularities can be reduced. With L_1 regularization, if λ is zero, then the $\mathbf{c}$ coefficients in the design matrix are not regularized. However, if we vary λ over a range of values, the regularization process is able to suppress insignificant terms. The coefficients at each λ are used to cross-validate the predicted model outputs with a different set of measured values in order to compute the error for a design objective. As shown in Figure 3.2, the λ which gives the smallest percentage error in this process is the optimal value to be chosen for maximum suppression of irrelevant terms. This makes the model generation process much more intelligent and removes any guesswork on the part of the designer.

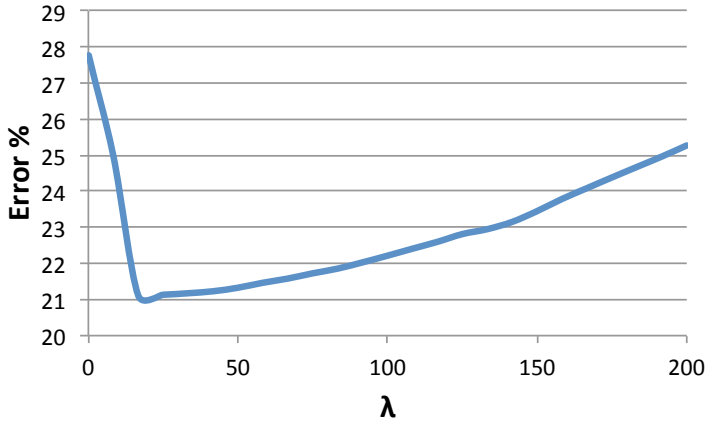

Figure 3.2: Relationship between the error for a design objective and λ, least error observed from coefficient set generated at $\lambda = 17$.

3.2 Multi-Objective Optimization Framework

Once the best model representing the design objective is obtained, we will be able to:

1. Plug in different design parameters to estimate design metrics, with each design metric (e.g., power, area, arithmetic accuracy) having its own model. The scalability and accuracy of the design models can be controlled and smoothly tuned by adjusting the number of samples for the model.

2. Incorporate the model into non-linear optimization formulations with an objective and under one or more model constraint(s). If a designer is studying two metrics, say y_A and y_B, these formulations will enable him/her to carefully design an architecture with focus on one or the other design variable, making it more efficient in the direction of either objective.

Multi-objective optimization problems mentioned above can be solved using standard non-linear optimizing techniques, as presented in [10]. With an objective func-

tion for y_A (e.g, power) to be optimized under a constraint function y_B (e.g., arithmetic inaccuracy) bounded by constant M, our optimization could be expressed as

$$\text{minimize } y_A(\mathbf{x}) \text{ subject to } y_B(\mathbf{x}) \leq M, \mathbf{x} \in \mathbb{R}^n$$

We solve this optimization formulation using *interior point* algorithms [26]. In interior point methods, the constraint and the objective functions are first translated into a single unconstrained problem using a logarithmic barrier function. Then the optimal design parameters for the objective function are obtained by searching for points where the gradient of the barrier function is zero, using first and second order partial derivatives and a Lagrange multiplier.

We will specifically solve these multi-objective problems to identify the optimal values for the algorithm and design parameters of our case study accelerator under various power, area, and arithmetic accuracy objectives and constraints.

3.3 Test Cases

To evaluate our methodology we consider two cases of hardware accelerators that can be and are used in a wide range of image processing and video analytics applications — one for image deblurring and the other for block matching using sum of absolute differences. Both accelerators were built in-house as a part of the research work in collaboration with colleague Onur Ulusel. The accelerator implementations are based on algorithms which involve high-throughput computation and processing of High Definition (HD) or bigger image sets. More details on the algorithm themselves and the accelerator architectures as well as the design parameters can be found in

our previously published work [76]. A brief description is given in sections hereafter.

3.3.1 Image Deblurring

Image deblurring is performed by a filtering operation over the image, which is one of the fundamental operations of image processing applications. The accelerator is developed for deployment within a real-life image processing system mounted on a unmanned air vehicle system for surveillance. The real-life setting of the accelerator has put tight requirements on its throughput, power, area, and arithmetic accuracy, which motivated the need for our proposed modeling and multi-objective optimization methodology.

The filtering operation in our implementation is performed as

$$I_D(i,j) = \sum_k \sum_l I_0(i+k, j+l) H(k,l),$$

where $I_D(i,j)$ and $I_0(i,j)$ are the deblurred and original pixel intensities at coordinate (i,j) and $H(k,l)$ is the deblur filter value at index (k,l). Our implementation uses input (I_0) and output (I_D) images with 12-bit pixels and deblur filters (*kernel*) of varied sizes and fixed-point bit-widths.

Our accelerator has a number of algorithmic and hardware design parameters, the values of which determine its final metrics (e.g., power, design area, and arithmetic accuracy). Parameters that are chosen by the designer are expected to have an impact on the constraint metrics and thus their selection requires an understanding of the inherent nature of the algorithm and the design (e.g., parameters that affect

the number of critical resources in the hardware or the accuracy of the algorithm in the software). However, the designer does not need to understand exactly how these parameters may affect the design constraints. Our goal is to simplify the process of selecting optimal parameters and interactions by enhancing the least squares based modeling methodology by an L_1 regularization process. L_1 regularization, as we have previously mentioned, suppresses irrelevant parameter and interaction terms between them and selects only those that have an impact on the constraint metrics, thereby highlighting design choices that may not have been obvious to the designer.

We chose two algorithmic parameters and two architecture parameters for our exploration study for the image processing test-case. We used kernel bit-width and kernel window size as our algorithmic choices and the DSP adder structure (and the number of registers used to synchronize inputs between the adders) as well as the running DSP block frequency as the tunable hardware parameters. As previously mentioned, the testbench accelerator development was a collaboration between the author and his colleague and implementation particulars can be found in greater detail in previously published work [58, 76].

3.3.2 Block Matching

Similarly, a block matching algorithm was implemented on an FPGA platform to accelerate the process of finding motion vectors between two frames of a video sequence and matching pixel blocks between the frames. Block matching is a sliding window operation performed over video sequences and is commonly used in motion estimation and video compression applications as well. Block matching partitions a given frame into non-overlapping $N \times N$ rectangular blocks and tries to find the block from the reference frame in a given search range that best matches the cur-

rent block. The measure of similarity between the blocks is computed by Sum of Absolute Differences (SAD). For our design, we perform full search block matching over a search window in a reference frame to determine the best match for a block in a current frame. The search for the best matching block is based on the search criterion of minimum SAD.

We used this particular implementation as a platform for design exploration as well. There were three algorithm parameters and two architectural ones. The algorithm parameters included pixel bit-width length, search window size, and the size of the non-overlapping rectangular blocks between the two frames. We studied the number of processing elements (PEs) as the sole architecture parameter in this particular implementation.

3.4 Experimental Results

Our hardware accelerator prototypes use a 40 nm Xilinx XC6VLX240T FPGA with 240,000 logic elements and 768 DSP blocks. Xilinx ISE Design Suite 12.4 is used for physical synthesis and Mentor Graphics Modelsim 10.1b is used for functional and timing simulations of the design. MATLAB is used for regression and optimization. To evaluate our accelerator performance for the image deblurring system, we use a number of sample images that are captured from the aerial vehicle platform. For evaluation of the block matching architecture, results are computed using sample video sequences obtained online [1]. The design *metrics* are estimated as follows:

- **Throughput:** For the deblur design, throughput is measured as the number of pixels deblurred per cycle. In all our design variants, we ensure the design meets

an operating frequency of 125 MHz and 8 pixels/cycle deblur. We relax this limitation for the block matching algorithm and make throughput a variable that depends on design parameters. Throughput for this particular example is measured in terms of frame rate — how many 720p HD frames we can perform block matching on, per second.

- **Area:** Area for the deblur example is measured by the number of DSP blocks used by the accelerator, since DSPs are the most critical resource. For the block matching architecture, since DSPs are not used, we measure the area metric using total number of look-up tables (LUTs). Each logic element in the FPGA used is composed of 8 registers and 4 LUTs. For purpose of uniformity, we convert the number of registers to equivalent LUTs and use the total number of LUTs as our measurement for area of the design.

- **Accuracy:** To estimate the accuracy of a particular deblur accelerator variant, we compute the *mean square error* (MSE) between a sample image data and its deblurred result from the accelerator. The MSE is the average of the squared differences between the image pixels and its deblurred result as produced from the accelerator variant. In the case of the block matching algorithm, we use MSE between reference block and the current block relative to the results obtained from the base implementation: 32×32 window size, no pixel truncation and 64 PEs.

- **Power:** To estimate the power dissipation of the accelerator, we followed two approaches. The first approach uses Modelsim on the routed design to estimate signal activity and then provides this result to the Xilinx XPower tool to estimate power. The second approach measures the *incremental power consumption* of our prototype board directly using an external digital multimeter (e.g., Agilent 34410), where the incremental power is the difference between

the reset state power and the execution state power of the design. The first approach estimates true power dissipated by the architecture only, while the second approach accounts for all the additional system power (e.g., FPGA and memory) that is associated with the computations of our accelerator. The incremental system power is the real cost that the end user incurs. For variation, we use the XPower results with our block matching architecture analysis and the board results for the deblur architecture. Consistency within each example ensures the validity and accuracy of our methodology is not compromised.

3.4.1 Modeling Results

Image Deblurring

For the image deblurring accelerator, we implement the design parameters as mentioned in Section 3.3.1. We use factors of 1, 2, and 4 for time-division multiplexing which correspond to a DSP clock frequency of 125, 250 and 500 MHz. The ability to implement time-division multiplexing for the data stream feeding into DSPs is what essentially allows them to run at up to 4 times the system frequency. We quantify the DSP adder structure by using what we call pipeline depth. A DSP pipeline depth is calculated by dividing the total number of DSPs used in all pipeline blocks by the number of blocks used. We make four different choices of average DSP pipeline depths between 3.3 and 11.5, each representing a unique arrangement of the DSPs to perform a multiply-accumulate operation. For kernel bit-width, which are originally represented as 18 bit signed fixed point coefficients, we vary the parameter from 8 bits to 18 bits. We also pick four random kernel sizes between 5×3 and 13×7. The combinations of parameters create a design space with $3 \times 8 \times 11 \times 45 = 11{,}880$ possible design points that potentially lead to different accelerator design variants.

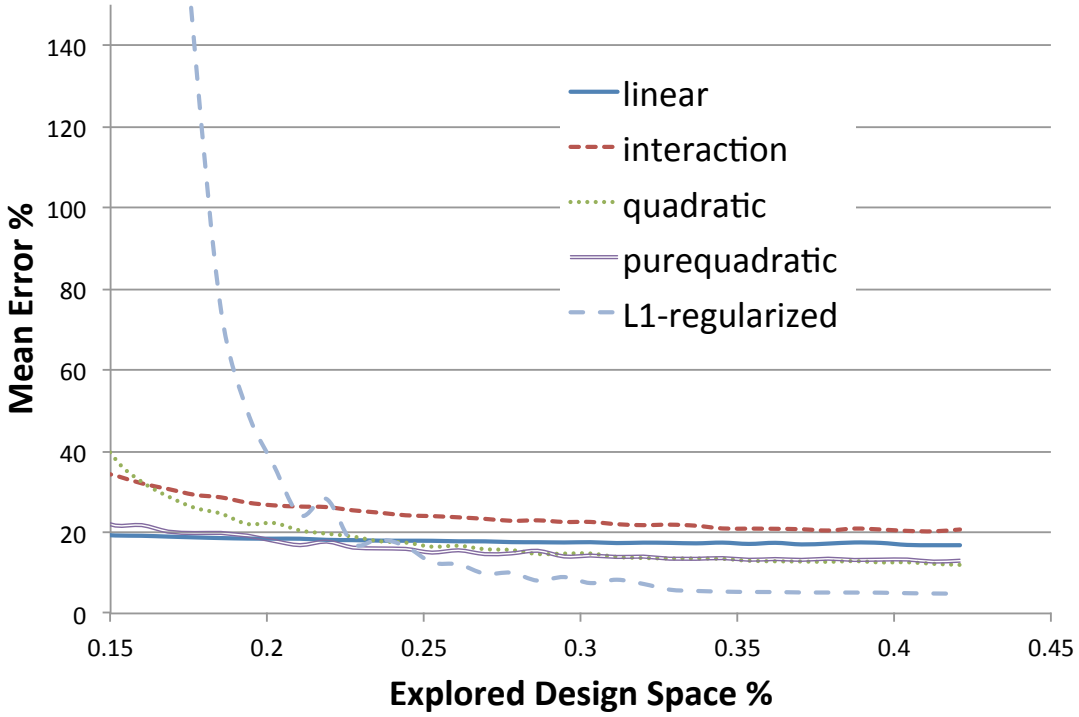

Figure 3.3: Error percentage of power model over explored design space percentage.

Full physical synthesis (which includes placement and routing) of a particular accelerator design takes about two hours on our quad-core based system, which puts limitations on the ability to execute a brute-force exploration of all accelerator variants. This motivates the need for fast design space exploration and optimization. To obtain our samples, we fully synthesize and implement 50 deblur accelerator variants with different parameter permutations; i.e., we only sample $\frac{50}{11880} = 0.42\%$ of the entire design space. As outlined in Section 3.1.1, these design points are selected randomly across the entire design space with the condition that the minimum and maximum configuration for each parameter is used at least once. This guarantees that any data point estimated by our predictor lies within the space covered by the training set, and that the training set is representative of the entire design space.

We first analyze the closeness between the results of different regression models

for power, area, arithmetic accuracy, and throughput against the measurements we obtained from our samples. We show the comparison of the accuracies of the models as an illustration of how our approach stands against traditional regression models (linear, quadratic, etc.). All these models without L_1 regularization have similar run times (i.e., less than 0.2 seconds). The additional step of finding the right λ parameter that minimizes the prediction error in the L_1 regularization methodology involves searching from a range and trying out all possibilities before a specific value is chosen. In our approach, we varied λ from 0.0001 to 1000 and randomly picked 25 values within this range. Of these 25 values, we chose the one that leads to the lowest error. This whole process took about 25 mins; however, it should be noted that this a one-time overhead required for accurate model generation. Once the model is derived, the process of physically implementing and synthesizing a design can be eliminated — thus allowing for what would take two hours to be completed in less than a second.

To train and evaluate the aforementioned model, we split our samples into two subsets: a *training subset* is used to learn the model parameters, and a *query subset* is used to validate the closeness of the model to the true measurements by taking the average absolute error between the model predictions and the actual measurements. For evaluation of the results, we follow the *repeated random sub-sampling* validation methodology [42] and repeat our training and query set selection 100 times so that any training bias is eliminated. For the purpose of this particular implementation, we randomly chose 35 samples as training and the remaining 15 as query for each iteration. Evaluation of predicted values from the model is validated by averaging over these 100 runs.

To gain further insights into the effectiveness of various models, we evaluate the relation between the mean error generated by the different models as a function

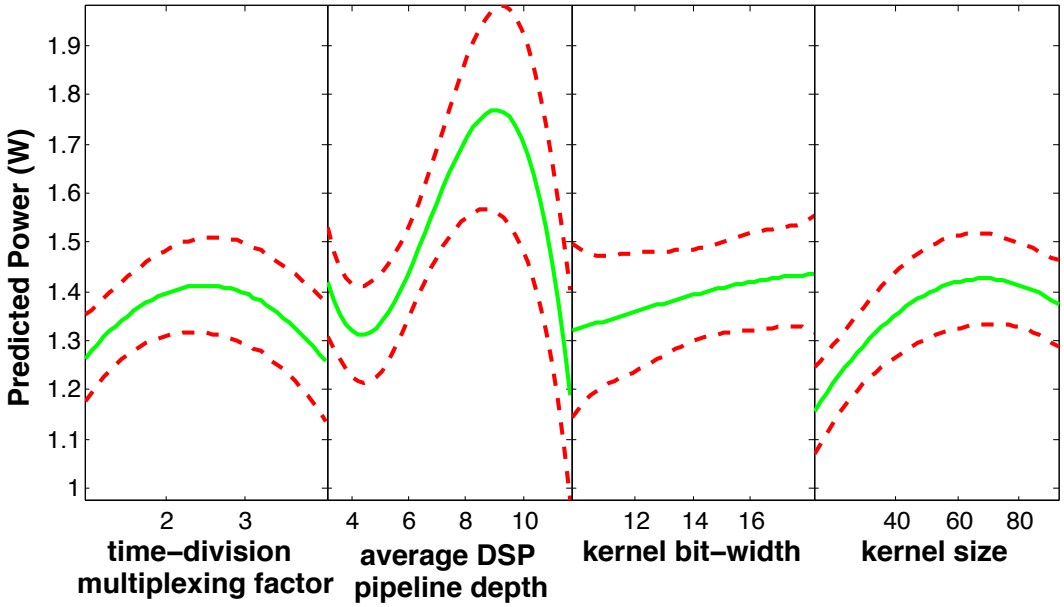

Figure 3.4: Sensitivity of different parameters over the power estimation for the deblurring test case.

of the training subset size. For example, we plot the results for the power model used in the deblur example in Figure 3.3. The plot shows that the model obtained using L_1-regularization requires a slightly larger training set to stabilize due to the presence of higher order terms. However, it performs better and stabilizes after a certain percentage of the design space is explored for both accelerator setups. This stability is reached after exploring 0.3% of the design space.

The model coefficients obtained from L_1-regularization reveal the impact and significance of different algorithm-design variables on the final outcome of his/her design. However, because of the presence of quadratic terms and interactions, numerical comparisons are not straightforward. To carry on an accurate evaluation, we evaluate the *sensitivity* of the L_1 model to different variables using a response model tool. We plot the results, again as given for the power model for the deblur example in Fig. 3.4. The solid line represents estimated power for each parameter variation, given that all other parameters are kept constant. The dotted lines show

the 95% prediction bands for the power value estimated, showing that if the prediction was repeated with different samples, the estimated value would lie within the range specified 95% of the time.

Sensitivity analysis pertaining to each parameter can help the designer understand important parameter impacts on the design. In the case of the deblur test-case, it can be inferred from the picture that average DSP pipeline depth has the highest sensitivity on power dissipation as the power values vary the most for this parameter. The trend observed for DSP pipeline depth reflects the trade-off obtained by varying the parameter as the smaller pipeline depths requires larger DSP groups to perform the same number of computations with fewer delay registers for synchronization. Therefore both small and large DSP pipeline depths benefit from this trade-off from different ends in terms of power. The time-division multiplexing factor affects power the most after average DSP pipeline depth, followed by kernel bit-width and

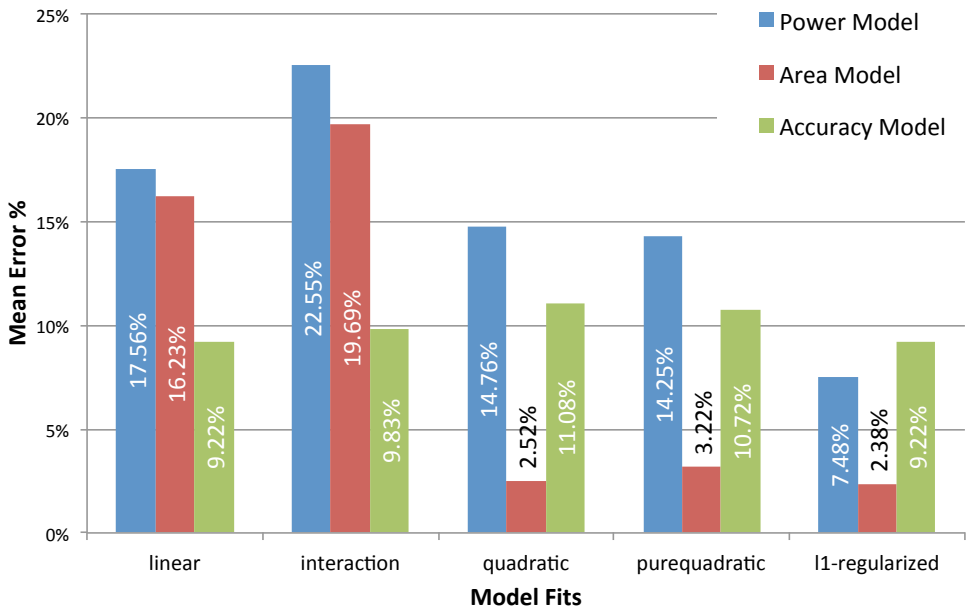

Figure 3.5: Comparison of mean error percentage using different model fits for power estimation, area and arithmetic accuracy models for the image deblur algorithm.

size, which have similar impacts as time-division multiplexing. Time-division multiplexing has a quadratic relationship with respect to power which is caused by the trade-off between the number of DSPs used and their running frequencies. Lower values require more DSPs, while larger time-division multiplexing factors require fewer DSPs running at higher frequencies. As expected, kernel bit-width has a linear interaction with power, since larger complexity in pixel arithmetic always results in higher power dissipation. Kernel size also has a quadratic relationship with power and power saturates for very large kernel sizes. The sensitivity results also align with the individual trends we observed in our measurements.

The estimation accuracy of the different models used for power, area and arithmetic accuracy metrics is given in Fig. 3.5 for a training size of 36, 9, 23 samples. The results show that the models obtained from L_1 regularization outperform other models.

Block Matching

A similar experiment with the block matching algorithm gives us better results for modeling done with L_1-regularization. Given the fact that the L_1-regularization process sparsifies the model and just keeps relevant parameters and interactions, results come out expectedly better than using other models. As seen in Figure 3.6, regularized models are superior and predict closer values to XPower generated data for all metrics (power, area, arithmetic accuracy and throughput) than any other model tried. Again, details of the implementation and the modeling results can be found in out previously published work [76].

L_1-regularized model coefficients provide insight to the interaction of defined parameters with the constraint metrics.This demonstrates the usefulness of our method-

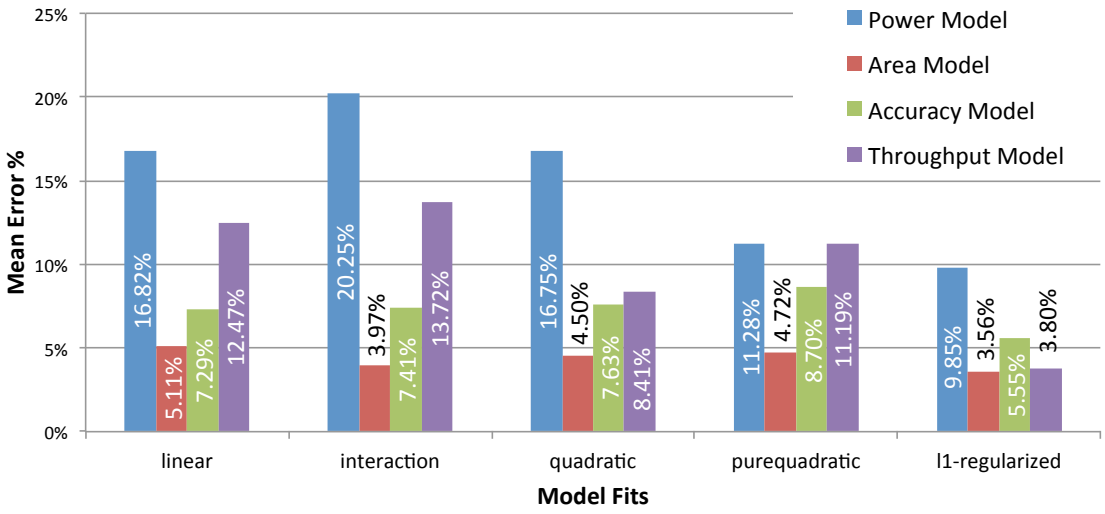

Figure 3.6: Comparison of mean error percentage using different model fits for power estimation, area and arithmetic accuracy and throughput models for the block matching algorithm.

ology at automatically detecting the combined effects of parameters on design constraints without relying on user input.

3.4.2 Multi-Objective Optimization Results

The mathematical models obtained through L_1 regularization enable us to create a numerical optimization framework to optimize the accelerator designs with respect to certain selected metrics while imposing constraints on other metrics. These chosen metrics and their values are selected to optimize the design within the specifications of its target deployment.

We consider four optimization formulations, two each from either test applications: (1) optimizing the deblur design's power under accuracy constraints; (2) optimizing the deblur design's area with a given power budget; (3) minimizing the block-matching architecture area under accuracy constraints; and (4) optimizing the block-matching design's area under throughput constraints.

Minimizing deblur design power under accuracy constraints We set up the objective function to be equal to the mathematical model for power obtained from L_1 regularization, and similarly we set a constraint function based on the mathematical model of arithmetic accuracy. We experiment with setting different accuracy values over a range. For each constraint value, we solve the numerical optimization problem as discussed in Section 3.2 using MATLAB. The results from our experiments are given in Figure 3.7, where we plot the resultant power of the system as a function of arithmetic accuracy. We label solution points with the identified design parameters. The results from the numerical optimization are intuitive as they show that relaxing the accuracy constraint leads to reduced power dissipation. While it is impossible to verify the optimality of these implementations without brute-force exploration, we show *high fidelity* of our optimization results by implementing the designs with the identified optimal parameters and evaluating their actual measurements. The dotted red line in Figure 3.7 gives the results from the actual implementation strongly indicating the validity of our optimization framework.

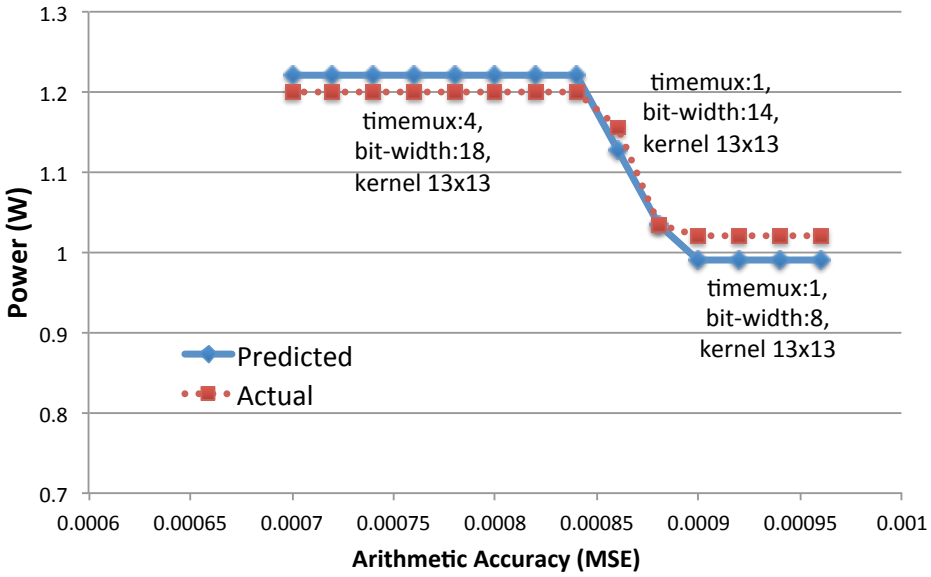

Figure 3.7: Trade-off between power and accuracy of the deblur system.

Minimizing deblur design area within a power budget In this second formulation we use the mathematical model for the design area as an objective function, and use the power function as a constraint. The area for the deblur design is defined by the number of DSPs used and the model does not necessarily apply to the relationship between logic elements and power. We use different numerical values over a range for the power constraint. We plot the results in Figure 3.8. The results from numerical optimization show the trend that *minimum* number of DSPs reduces as we relax the allowed power threshold. Although achieving a reduction in area resources at the cost of increased power is not intuitive, the impact of the average DSP pipeline depth makes it possible. In this particular case, all the design parameters other than average DSP pipeline depth are aligned with their corresponding global minimums. An increase in average DSP pipeline depths implies using fewer DSPs, at the cost of additional synchronization registers, which results in increased power dissipation compared to having smaller average DSP pipeline depth. For Figure 3.8, the number of slice registers used ranges from 8812 to 7786 while the number of

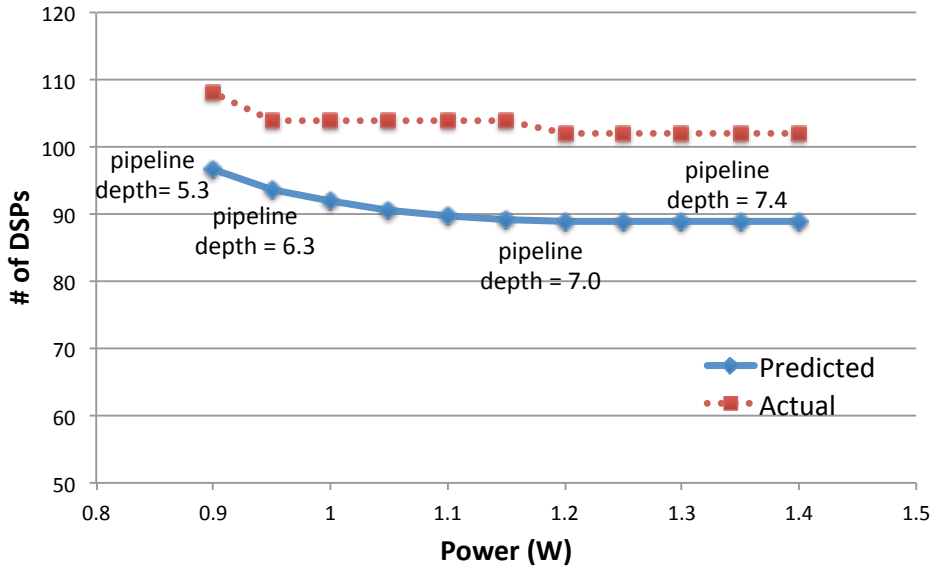

Figure 3.8: Trade-off between area and power of the deblur system.

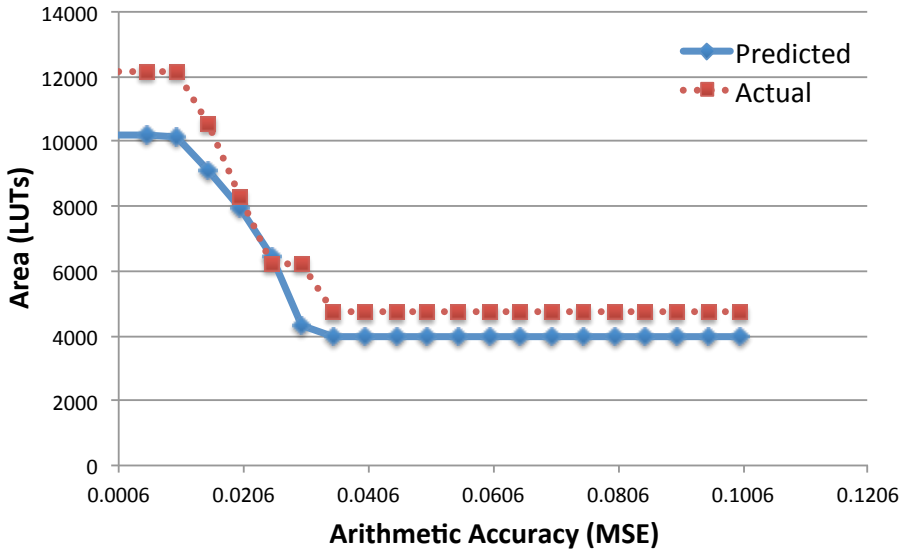

Figure 3.9: Trade-off between area and accuracy for the block matching implementation.

DSPs ranges from 102 to 108. While the total number of DSPs is based on both the time-division multiplexing factor and the DSP pipeline depth, it is seen that the *minimum* number of DSPs becomes solely a function of average-pipeline depth for the given range for power. We also evaluate the measurements obtained from implementing the identified optimal designs and plot the results by the dotted red line. The results again show a high-fidelity trend as the results from our optimization formulations.

Minimizing block matching area under accuracy bounds Here, we use the area model for the block matching implementation as the objective function and try to constrain it under a range of accuracy bounds. We vary our error in the system from 0 to 0.1 relative MSE and observe the minimum area values we can achieve for these constraints. As shown in Fig. 3.9, we see that minimum area required for the design increases as we make accuracy requirements tighter. To achieve more accuracy, the design requires more area because of minimal or no pixel truncation.

The predicted results from the optimization formulation again align with the actual area results of the designs and show high fidelity.

Maximizing block matching arithmetic accuracy given throughput constraints For accuracy under throughput constraints, the optimal solution is independent of throughput and hence we get a constant minimum solution for arithmetic accuracy at varying ranges of minimum throughput set for the system. This makes sense just intuitively because throughput, which is associated with speed of computations is unrelated with the accuracy of computations. The predicted minimum error of 0.0297 is always close to the theoretical minimum of 0 relative MSE.

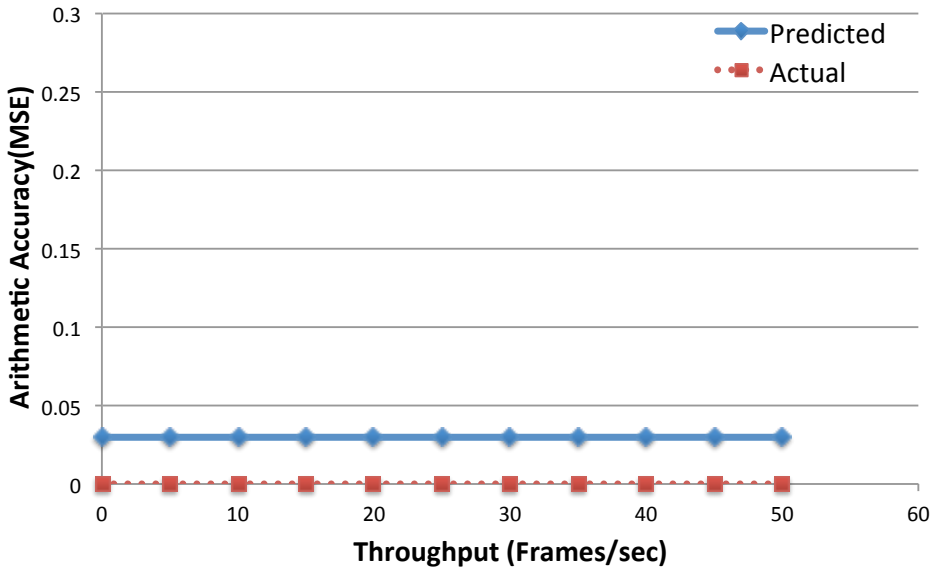

Figure 3.10: Constant minimum error for a range of variable framerate for the block matching algorithm indicating the independence between arithmetic accuracy and throughput.

3.5 Summary and Discussion

In this chapter, we explored the idea of using analytical modeling for design metric characterization and prediction to be able to facilitate fast and accurate design space exploration for hardware accelerators. We tested and demonstrated the validity of our idea on two test cased implemented on an FPGA platform. The reconfigurability of various parameters and the options they provide can lead to tens of thousands of possible design implementations and choosing an optimal one without a methodical process may be time-consuming. We have shown that by using regression modeling and some machine learning techniques, we can accurately predict design metrics such as area, power, throughput of a design, while sampling less than 1% of a vast design space, and thus significantly cut down on exploration time. More importantly, we have demonstrated that the use of L_1 regularization gives us key benefits in establishing the relevance and impact of the various parameters within a design. Compared to previous regression techniques proposed in the literature, L_1 regularization brings about the following benefits:

- L_1 regularization enables automatic discovery of the accurate mathematical dependencies between the variables and the desired model outcome with no need for guesswork from the designers. This accurate dependency results in improved closeness between the results of the model and the actual measurements.

- L_1 regularization together with sensitivity analysis enables designers to assess the relative impact of different changes in design variables on the final outcome, which can help designers focus their design optimization efforts.

These benefits of automated model generation and the resulting closeness of our models to the measurements enables us to use this model to query directly for non-sampled design points attaining a dramatic speed-up in design space exploration. Since a full synthesis, place and route execution for our deblur design takes two hours, and our model achieves its least error using only 35 samples (0.3%) of the full 11880 points design space, our L_1 based model is able to achieve approximately a 340× speedup in design exploration with estimation errors of 7.48%, 2.38% and 9.22% for power, area, and accuracy respectively. With the block matching algorithm, we use only 18 samples from the entire design space for training, so our speedup is approximately 90×. We report speedup for both examples as the ratio of the time it would have taken to implement and synthesize the entire design space to the time it takes to implement and synthesize only a few sample points for training and query. The non-linear optimization framework implemented in MATLAB after the models are generated for each metric takes about 0.1 seconds to run and is almost negligible in comparison to the runtime taken for synthesis of the designs.

Thus, our mathematical models enable us to substitute the entire physical synthesis and simulation flow and estimate directly the final accelerator metrics as a function of different algorithm and hardware design parameters with speed and accuracy. In addition to this, the experiments and methodology presented in this chapter also open up newer avenues in the direction of design space exploration. We made use of some tunable algorithmic parameters that directly impacted computational accuracy of a design. In Chapter 4, we will present an idea of using computational accuracy as a modifiable design parameter in order to realize designs that are significantly better in terms of their area and power consumption.

Chapter 4

Design Space Exploration using Behavioral Synthesis of Approximate Circuits

In the previous chapter, we presented an approach for design space exploration using analytical models for estimating design metrics such as area, power, thoughput. These models were derived by first selecting only a small number of samples from the vast design space and then using statistical inference and regression analysis to converge onto accurate characteristic models. What we noticed during these experiments was that for a certain class of applications (i.e.,multimedia, graphics, computer vision and signal processing), algorithmic accuracy could be used as a design choice as well. Since these applications have some inherent error tolerance, adjusting the algorithm accuracy may be acceptable if it is within a certain error margin as was discussed in Chapter 2.

In this chapter, we will discuss ways in which we can manipulate computational accuray in various circuits and systems and leverage an algorithm's tolerance to output error to perform design space exploration in search of optimal designs. More specifically, we will study the idea of approximate computing as a means to create

circuit and system designs that perform within an acceptable range of accuracy and yet show significant savings in power consumption and area footprint.

As discussed in Chapters 1 and 2, our methodology of creating approximate variants of circuits *directly from their behavioral RTL descriptions* is the first of its kind. Working on thhe higher level abstractions has its benefits — the designer does not have to be familiar with the Boolean, gate, or transistor level implementations of the ciruits. The appproach also allows for a higher-level view of transformation and approximation that may have otherwise been missed by the designer. Given these merits, our approach to using approximate computing as a design exploration tool is appropriately called *ABACUS*, for Automated Behavioral Approximate Ciruit Synthesis.

4.1 Behavioral Synthesis Flow

We discussed in Chapter 2 how most existing circuit approximation techniques aim at generating approximate variants for standard sub-circuits (e.g., adders), given their Boolean descriptions, or through manual modifications for specific applications (e.g., DSPs). In contrast, we aim to generate approximate circuit variants for any system from its high-level behavioral description.

In a basic design flow, the behavioral or register transfer level (RTL) code is first generated by the designers from the design specifications. The code is then simulated functionally using a number of representative testbenches and the simulation results are evaluated to verify correctness of operation. The code is then compiled and synthesized to a netlist using a design compiler, which also takes as input a standard cell library for ASICs or the look-up table and cluster architecture for FPGAs. The

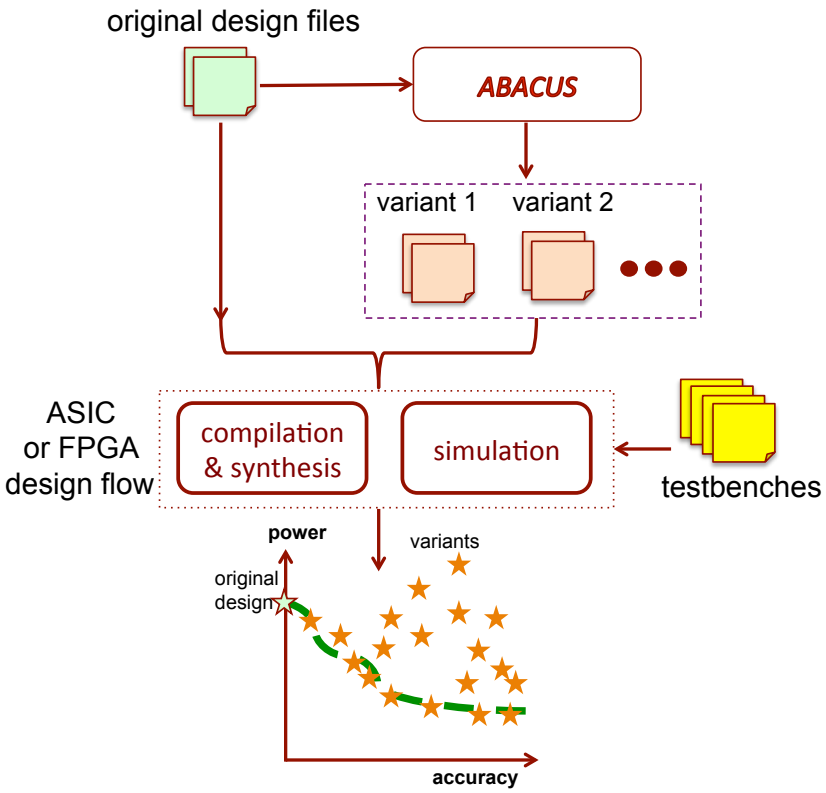

Figure 4.1: Incorporation of *ABACUS* within standard design flows.

netlist is afterwards placed and routed to get the final area, timing, and power metrics. We integrate *ABACUS* with traditional ASIC/FPGA flows but instead of synthesizing a single exact code, multiple approximate code variants are also synthesized at the RTL/behavioral level. As illustrated in Figure 4.1, these variants are pushed through the standard design flow, and the approximate outcomes are compared with the original exact design in terms of functional accuracy and hardware design metrics such as power, area and timing. The evaluated outcomes are then plotted and a *Pareto frontier* is computed to identify the approximate designs that give the optimal accuracy-power trade-off.

To achieve our goal of generating approximate behavioral variants, we propose (i) to capture the exact RTL or behavioral hardware description language (HDL) design in an Abstract Syntax Tree (AST) structure; (ii) create the approximate

design variants through transformations to the AST; and (iii) finally write back the modified AST into readable RTL or behavioral HDL design. The new approximate design is then pushed through the standard ASIC/FPGA flow for evaluation in terms of accuracy and other design metrics. The AST is a convenient intermediate format to work with as we apply approximations directly on the design and allows us to easily go back and forth between a modifiable data structure and readable RTL code.

4.1.1 RTL Parsing

An integral part of doing high-level approximate circuit synthesis is our approach to take in an RTL convert it to a manipulative data structure(s). Approximations are done over this data structure and the final results are read out in the form of readable RTL itself. This is all done by capturing the behavioral description or the RTL into what is called an AST. AST conversion is an important step that allows our approach to be efficient in time, and coverage. The entire transformation process is shown in Figure 4.2.

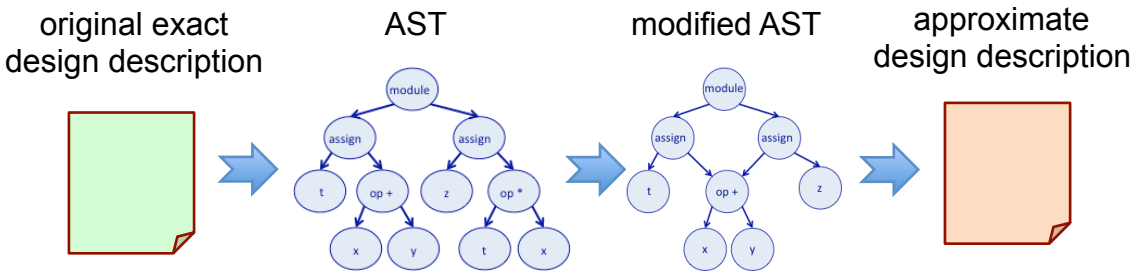

Figure 4.2: Overall methodology of *ABACUS*.

In an AST, each node represents an action to be taken by the behavioral code, or an object to be acted upon [56]. Building an AST for a HDL syntax automatically

captures all the concurrency that is in the design. Compared to a regular parsing tree, the AST captures the logical structures of the statements and shows less of the grammar structure, which makes it a better candidate to analyze and transform the code. Compared to control flow graphs, ASTs are easier to use to produce readable code.

Nodes in an AST are created out of the RTL or behavioral-level semantics. The graph structure that is composed represents the semantics of the behavioral language and not actually any inherent data/control dependencies. Consider the following Verilog code for an adder:

```
always @(posedge clk)
begin
    out <= in1 + in2;
end
```

Such a syntax tree, allows us to easily make transformations that may reflect approximation of a circuit. For example, we can approximate the adder shown in Figure 4.3, by replacing the add operation by bitwise-OR operation. An adder inherently has an XOR functionality with added capabilities for carry generation and propagation. XOR is probabilistically very similar to an OR gate. It can therefore be expected that an adder changed fully or partially (only some bits) to a bit-wise OR operation may produce a reasonably accurate result(depending on the application), but at a lower cost for area and power consumption.

The adder to bit-wise OR transformation at the syntax tree level would produce an AST as shown in Figure 4.4. The RTL code from this tree would then be regenerated as:

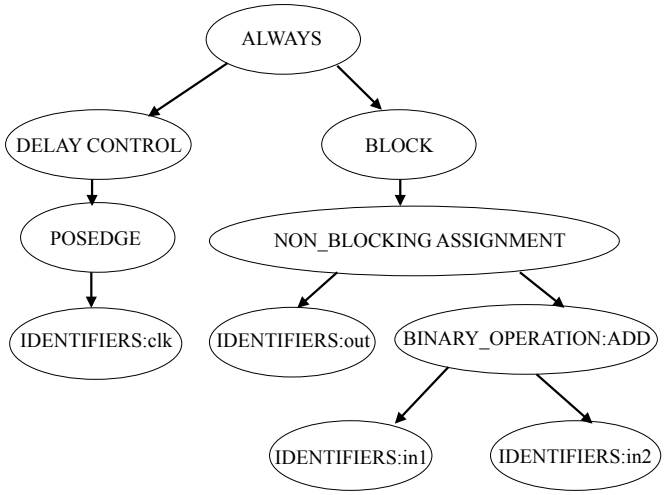

Figure 4.3: Example AST created out of a simple vVerilog code for an adder.

```
always @(posedge clk)
begin
    out <= in1 | in2;
end
```

The initial front-end parsing of the RTL to an AST is done using an open-source tool called ODIN-II [35] which is primarily written in C with additional parser/-compiler tools like Lex, Yacc, Flex and Bison. We added all of the tool's backend capabilities to perform RTL transformation and regenerate the modified approximate circuit as readable synthesizable RTL.

The result is a novel tool that can read behavioral/RTL code, modify it to approximate the underlying circuit representation and produce this new version in readable behavioral format. The novelty in our *ABACUS* approach allows us to make automated transformations to any generic HDL design without the need to have any *a priori* knowledge of the functionality or the semantics of the design. Hence, a

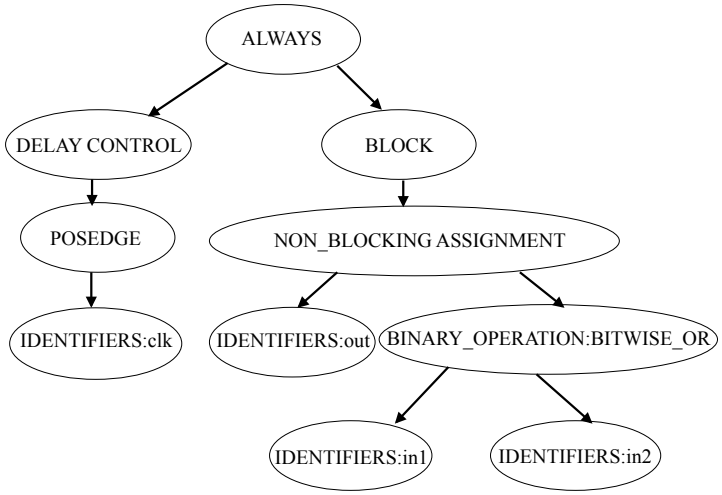

Figure 4.4: Example AST showing change from an adder to a bit-wise OR operation.

much broader range of transformations can be explored at the high level, leading to a superior design compared to prior approaches.

4.1.2 Generating HDL-based Approximate Transformations

Since our technique works directly at the behavioral level, it is important to understand what kind of transformations are possible and how these transformation may affect circuit accuracy. Here, we present a set of *transformation operators* that can be applied to the original HDL-based AST to yield meaningful approximate designs for error-resilient applications. As described in Section 4.1.1 , whenever any of these transformations is invoked, *ABACUS* automatically traverses through the AST and searches for places in the AST where the change could be applied. We propose and implement the following five transformation operators in *ABACUS*:

1. Data Type Simplifications: For applications dealing with massive data, truncating the size of intermediate signals may be a good way to achieve power, delay and/or area savings, since it reduces the requirements for the underlying hardware,

especially for fixed-point arithmetic operations. *ABACUS* is capable of performing truncation in two ways: first, by setting a number of the least significant bits to zero, and second by truncating a certain number of significant bits for operands during binary arithmetic operations and then shifting the result of the operation to get the approximation. The latter transformation yields more significant power and area savings. An example of this kind of simplification would be:

```
out <= in1 + in2;
```

transformed to:

```
out <=in1 + {in2 [7:3], 3'b000};
```

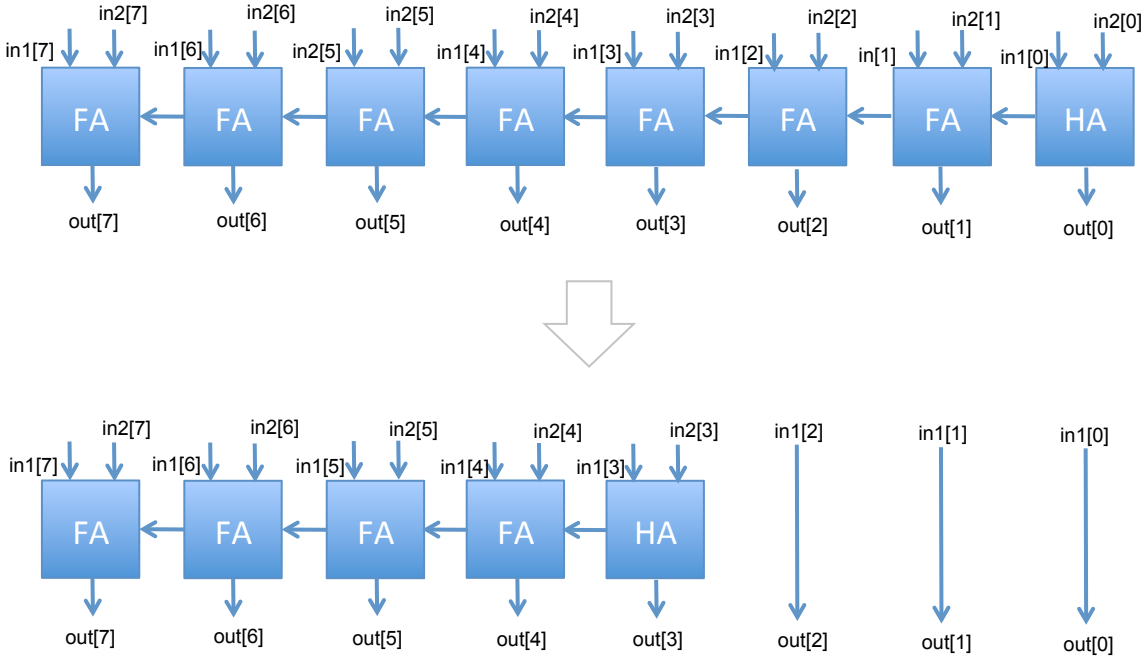

Figure 4.5: Illustration showing how truncation can reduce hardware complexity of an 8-bit adder. FA represent full adders and HA represent half adders.

What effectively happens here, as seen in Figure 4.5, is that the adder shrinks by three bits. If $in1$ and $in2$ are both eight bits, the original statement when synthesized would be represented as an array of seven full adders and a half-adder for the least significant bit. When the bit-truncation is applied on the last three bits, the statement synthesizes as three full-adders for the most significant three bits, and one half-adder for the fourth significant bit. Since $in2$ is truncated by three bits, the three bits of $in1$ will be directly registered at the output register at its least three significant bit locations. This shrinking of the adder directly impacts circuit area and power.

2. Operation Transformations: We may choose to substitute an arithmetic operation with one or more arithmetic operations that use less power and hardware area. For example, arithmetic additions could be replaced by bitwise ORs as shown in Section 4.1.1, or a multiplication could be replaced by shifts and an addition or by simply addition. Also, a standard adder or multiplier could be replaced by an approximate unit from the ones proposed in the literature [32, 4, 30]. Thus, our behavioral-based approach can easily leverage approximate Boolean arithmetic circuits. One example of an operator transformation could look like:

```
out <= in1 * in2;
```

transformed to:

```
out <= in1 + in2;
```

which could go through further transformation as:

```
out <= in1 | in2;
```

3. Arithmetic Expression Transformations: There are cases where near similar arithmetic structures appear in the same statement description. Through a transformation, these near-similar structures could be transformed to similar structures so that they may be shared and simplified. For instance, we can approximate the expression

```
(w_i * x_i) + (w_j * x_j)
```

with substitutions to the variables or the constants, such as substituting x_j by x_i leading to

```
x_i * (w_i + w_j)
```

or substituting w_j by w_i leading to

```
w_i * (x_i+x_j)
```

thus saving one multiplier. We can simplify computations by sharing common or similar operands and get good approximations. We can also delete or swap some of the AST nodes to get newer or smaller arithmetic expressions that could potentially lead into smaller sub-circuits than the original one.

4. Variable-to-Constant Substitution Transformations: Simulation results of the original design contain useful information about the numerical characteristics of the design variables. This information can guide the transformation operations. For instance, if an intermediate variable derived from a certain arithmetic operation appears to be a constant or has a small standard deviation in the simulation results,

then we can substitute it with a constant based on its average value, thus saving unnecessary computation. *ABACUS* implements this feature by reading simulation results from the original exact design to identify design variables that are constant or are within a 10% standard deviation. These design variables are candidates for substitution by a constant based on their simulation results.

5. Loop Transformations: *ABACUS* automatically unrolls loops in behavioral descriptions. Loop unrolling is typically used as a compiler transformation technique; however, in our case we use automatic unrolling in the pre-compiler phase of the behavioral description of an algorithm. By unrolling loops, we uncover instances where we may be able to apply operator and/or data-type simplification transformations. In addition, the unrolling can be done in an approximate way by skipping certain iterations and substituting the outcomes of these iterations from the results of prior iterations. A simple example of loop unrolling is given below:

```
sum[0]=result[0];
for (i=1; i<=5; i++)
begin
    sum[i] = sum[i-1] + result[i] ;
end
```

transformed to:

```
sum[0] = result[0];
sum[1] = sum[0] + result[1];
sum[2] = sum[1] + result[2];
sum[3] = sum[2] + result[3];
sum[4] = sum[3] + result[4];
sum[5] = sum[4] + result[5];
```

which now allows for futher transformation such as:

```
sum[0] = result[0];
sum[1] = sum[0] + {result[1][7:3], 3'b000};
sum[2] = sum[1] + result[2];
sum[3] = sum[2] + result[3];
sum[4] = sum[3] | result[4];
sum[5] = sum[4];
```

4.1.3 Effective Design Space Exploration

Application of the proposed transformation types can lead to a combinatorial explosion in possible approximate design variants — there are multiple transformation types possible, and each operator can be applied at several locations. Also, the AST resultant from one set of transformations can be used as input for another set of transformation in what can be a series of additive transformations. A permutation/combination and the addition of all these possible transformations can result in tens of thousands of possibilities to choose from for identifying optimal designs. To effectively explore and identify the Pareto frontier of optimal trade-off designs, we propose the following *iterative stochastic greedy algorithm* that continuously evolves the approximate designs by doing multiple iterations of transformations to identify the Pareto frontier that gives the optimal trade-off between accuracy and power.

Algorithm *Approximate Design Space Exploration*

Input: original exact design

Output: approximate design variants

1. Let O_1 = original design
2. **while** $i \leq N$
3. **while** $j \leq M$
4. **do** pick a transformation operator at random;
5. apply the operator to O_i to yield A_j;

6. evaluate accuracy of A_j over input data sets
7. **if** accuracy of A_j is within threshold
8. **then**
9. synthesize A_j;
10. $V = V \cup A_j$;
11. Use results from Steps 6 and 9 to evaluate the fitness, F_j, of A_j;
12. **else** goto step 4;
13. identify A_k, $k \in \{1, \ldots, M\}$, with best F_j;
14. let $O_{i+1} = A_k$;
15. **return** V

We couple a simulation and a synthesis tool with *ABACUS* to evaluate accuracy and design metrics respectively. The algorithm goes through N iterations, where each iteration attempts M transformation operators. In steps 4-6 above, a transformation operator is picked at random with some probability and applied to the current design and the results are evaluated for accuracy. Accuracy of an approximate design is averaged over a number of input training data sets. Approximate designs still need to retain a minimum accuracy threshold for all input training data sets. If the average accuracy measure meets an accuracy threshold then the design is considered a valid variant and passed on to the synthesis tool in Step 9. The accuracy threshold is pre-set to filter out bad design variants from the good approximate ones. A design with accuracy less than this threshold will not be synthesized and is not further considered for use with the tool. Using the accuracy results from Step 6 and the synthesis results from step 9, the design variant is evaluated for fitness, which is defined as

$$\mathit{fitness} = \alpha \times \mathit{accuracy} + (1 - \alpha) \times \mathit{power} \tag{4.1}$$

In step 13, all M variants are ranked and the design with the highest rank for fitness is used in the subsequent iterations as the parent design for transformations as given in Step 14. *ABACUS* repeats these generations of transformations greedily to get the best area and power saving results withing the prescribed accuracy constraint until it reaches the defined limit for number of generations.

Whether an operator transformation is applied or not is dependent on a probability function to ensure no bias towards a particular operator. Furthermore, for a given operator, the location where an operator is applied is also randomized with a probability to ensure no bias towards a particular location. Thus the sequence of applied transformations is stochastic in nature. During the iterative greedy procedure, *ABACUS* keeps a log of accuracy, area and power values of each design variant used along the way. A wide range of optimal designs at various accuracy and power savings can be obtained in this manner and help in creating the Pareto frontier for tradeoff between accuracy and other metrics. The designs that pass the final generation of mutations are also ranked for fitness; the highest ranked design represents the behavioral description that has the lowest cost in terms of power dissipation and area utilization while still meeting accuracy constraints.

Our methodology avoids explosive design space exploration by constricting the design choices using a greedy heuristic. If every variant generated per generation were to be used as an originating design for further design transformations, the runtime needed for a full evaluation would increase with the number of generations as a geometric series. Lets say, there are N iterations and M designs per generation. This would mean that without the greedy approach, the number of designs to be used as transformation seeds would increase as $M^0 + M^1 + M^2 + M^3 + ... + M^N$, or mathematically, $\frac{(1-M^{N+1})}{(1-M)}$. With *ABACUS*, the number of designs increases linearly with N so the final number of designs to be used for exploration would be MN. The

speedup in runtime would hence be $\frac{(1-M^{N+1})}{MN(1-M)}$. So for 20 iterations with 5 variants per generation, the speedup would be about 1.2×10^{12}.

4.2 Test Cases

We instrument our approximation techniques in three test-cases representative of three different domains that are amenable to inexact computing. We have a finite impulse response (FIR) filter implementation from the signal processing domain, a perceptron classifier from the machine learning domain and a Block-Matching algorithm that is extensively used in computer vision and graphics. These test-cases, which are all written in Verilog HDL, are described in more detail hereafter.

1. FIR filter: We implement a 25-tap FIR filter that takes in an audio signal and convolves it with a low-pass filter coefficient array, essentially creating a 1-D filtering effect. The quality of an approximate version of this design is assessed using Mean-Squared Error (MSE) on the amplitude of the audio signals generated. The MSE for the original FIR filter circuit is computed with an 16-bit sign-extended fixed point coefficient and image input compared against a floating point implementation done in software.

2. Perceptron Classifier: A perceptron classifier is a commonly used application in machine learning. A perceptron takes an input data, denoted by vector x, and predicts the class (e.g., -1 or +1) of x by computing $sign(w^T x)$, where w is the weight vector and $sign(\cdot)$ is a function that outputs 1 if its argument is positive and -1 otherwise. The perceptron essentially defines a hyperplane to separate the training data into two classes. Perceptrons are also capable of classifying non-linearly separable points by mapping the input points to another space where they are linearly separa-

ble, i.e., $sign(w^T\phi(x))$, where $\phi(\cdot)$ is the mapping function. Our perceptron test case uses a quadratic function to map the input space. The input data set consisted of 1000 randomly generated two-dimensional points from two classes. Classification results are compared against the ground-truth and hence, the total percentage change in classification outputs is considered as the accuracy metric.

3. Block Matcher: Block matching is a technique commonly used in motion estimation and video compression applications. Block matching partitions a given frame into non-overlapping rectangular blocks and tries to find the block from the reference frame in a given search range that best matches the current block. The measure of similarity between the blocks is computed by sum of the differences. For our design, we perform full search block matching over a search window in a reference frame to determine the best match for a block in a current frame. Our particular test case works on 16×16 block sizes from a 352×288 frame sequence. The quality of a design is assessed using Peak Signal to Noise Ratio (PSNR).

The main hardware design characteristics and quality of these test benches are summarized in Table 4.1.

Design	Class of Application	#Lines	Area (um^2)	Power (mw)	Quality Measure	Quality
FIR filter	Signal Processing	265	37776.96	2.74	MSE	99.55%
perceptron	Machine Learning	188	40389.12	6.89	classification error	83.55%
block matching	Computer Vision	1277	80272.44	30.41	PSNR	30.54 dB

Table 4.1: Characteristics of test cases used.

4.3 Experimental Results

We have implemented our *ABACUS* tool and integrated it with a standard industrial-strength flow comprised of Synopsys Design Compiler and Mentor Graphics ModelSim. We used commercial 65 nm technology libraries.

For each design, a total of six data sets were used to train and evaluate *ABACUS*. Three data sets were used to generate the approximate designs as described in Section 4.1.2, and another three were used to assess the accuracy of *ABACUS* for the experiments of this section. Using different data sets in the experimental results eliminates possibility of generating approximate variants overfitted for one particular set of input data. In all experiments, we report the average accuracy of the three data sets. We used weight of 0.6 as α in Equation 4.1 for fitness evaluation. *ABACUS* was applied to the computational data-path parts of the designs, but the control signals in the designs were not modified. Using *ABACUS*, we were able to automatically apply multiple iterations of transformations on each test bench to identify the approximate designs with optimal trade-offs between accuracy and power.

For all these testbenches, *ABACUS* generates readable approximate variants and makes the entire code evolution process transparent to the designer. Consider the code snippet below, taken from the original behavioral code for the perceptron example.

```
always @(*)
begin
 sum_temp[0]=result[0];
 for (i=1; i<nSVs; i=i+1)
 begin
  sum_temp[i] = sum_temp[i-1] + result[i];
 end
end
```

This code snippet goes through behavioral approximations outlined earlier and transforms as follows:

```
always @(*)
begin
 sum_temp[0]=(sum_temp[2] | result[3]);
 sum_temp[1]=(sum_temp[0] | result[1]);
 sum_temp[2]=(sum_temp[1] | result[2]);
 sum_temp[3]=({result[0][15:3], {3'b000}});
 sum_temp[4]=(sum_temp[3] + result[4]);
 sum_temp[5]=(sum_temp[4] + result[5]);
end
```

Figures 4.7, 4.6, and 4.8 plot the accuracy vs. power saving results for all approximate designs generated by *ABACUS* for the FIR, perceptron, and block-matching designs, respectively. The x-axis gives accuracy and the y-axis gives power savings. A subset of all these points create a Pareto Frontier (solid red line), where the frontier points do not dominate each other in both power and accuracy. The red star indicates the original exact design in the population. The runtimes of *ABACUS* are mostly dominated by the runtime of the ASIC design flow. On a cluster computing system with nodes operating at a maximum frequency of 2.8 GHz, 3 CPUs and 24 GB RAM, it took 92 seconds, 20 seconds and 56 seconds for generating one instance of the FIR, perceptron and the block matching benchmarks respectively. A total of 8 generations with 10 iterations were run for all test cases.

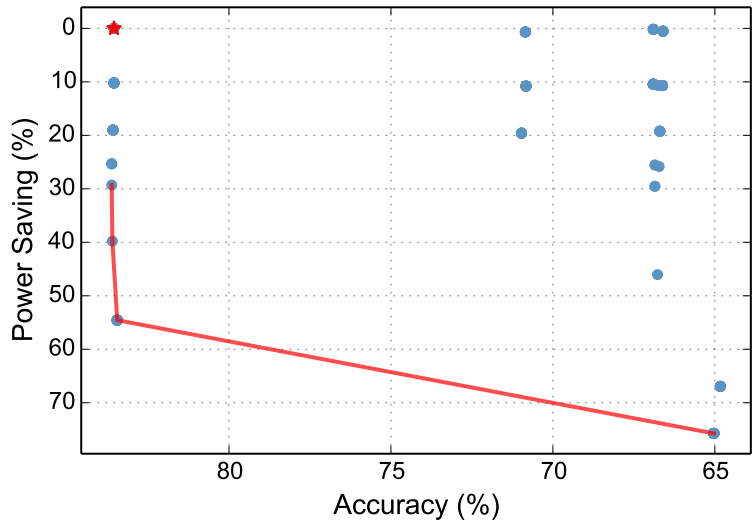

Figure 4.6: Results from various approximate designs and the Pareto Frontier for the perceptron test bench.

In Table 4.2, we highlight results from the best approximate designs that kept within a maximum of 8% degradation in accuracy compared to the original designs. The results show that we are able to obtain significant savings in power consumption (upto to 76%) with these approximate designs with very modest degradation in

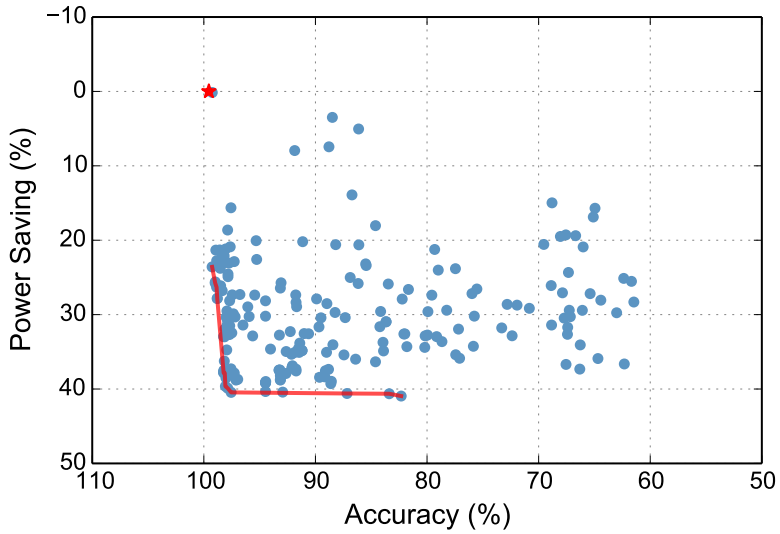

Figure 4.7: Results from various approximate designs and the Pareto Frontier for the FIR test benches.

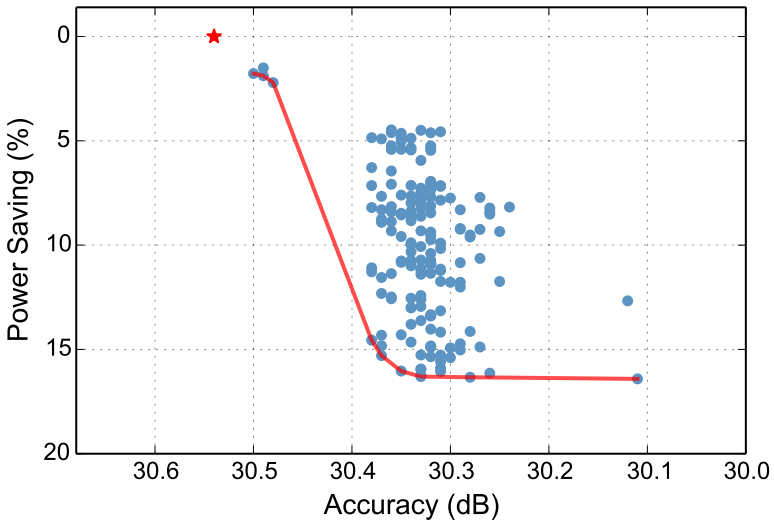

Figure 4.8: Results from various approximate designs and the Pareto Frontier for the block-matching test bench.

accuracy. Figure 4.9 illustrates the results from the perceptron approximate design of Table 4.2. Figure 4.9.a gives the true classification of the data points into the two classes (class A and class B). Figure 4.9.b gives the classification of both the original and approximate hardware (HW) designs on the same data points. The true-true case is when both HW designs correctly predicted the classes of the data points, while the false-false case is when both HW designs incorrectly predicted the classes of the data points. The false-true case is when the original design predicted incorrectly, but the approximate design predicted correctly. Finally, the true-false case is when the original design predicted correctly, but the approximate design incorrectly. The figure shows very few points in the true-false case, and almost balanced in number by

Design	#Iter	Accuracy Threshold	Accuracy Achieved	Power Saving	Area Saving
perceptron	8	76.9%	83.6%	41.6%	38.7%
FIR	8	91.6%	91.0%	32.5%	50.3%
block matching	8	28.1 dB	30.3 dB	12.7%	12.5 %

Table 4.2: Results from *ABACUS* for the three test benches for an allowed 8% degradation to accuracy.

points in the false-true case. As a result the approximate design attains comparable accuracy to the original, while achieving 47.6% power savings.

Finally, Figure 4.10 shows a comparison between an original implementation and an approximate counterpart for a motion vector field created during the block-matching algorithm. Original motion vectors are shown in yellow while the motion vectors in error stemming from an approximate algorithm that allows a 13% error in output are shown in red.

We also compared the results generated by *ABACUS* to a commonly used technique in approximation as described in Section 2.3, where approximate versions of standard components are used in place of the accurate ones. We truncated 3 bits off a set of multipliers for the perceptron and block matching implementations and 3 bits off a set of adders for the FIR implementation. Results are shown in Figure 4.11 for designs that maintain the same accuracy. Our method obtains superior results for all three benchmarks. This supports our claim that *ABACUS* is capable of mak-

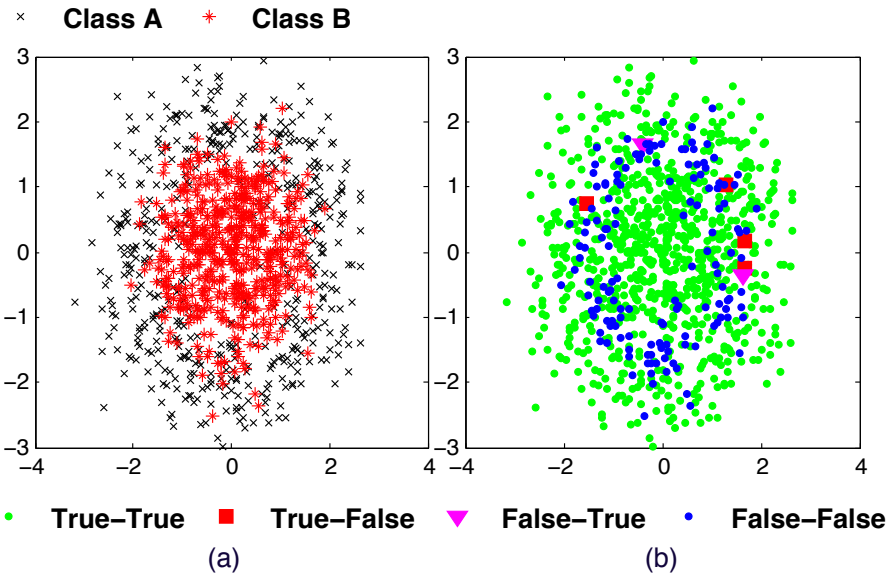

Figure 4.9: (a) Classification of input data into two classes, (b) comparison between original and approximate designs.

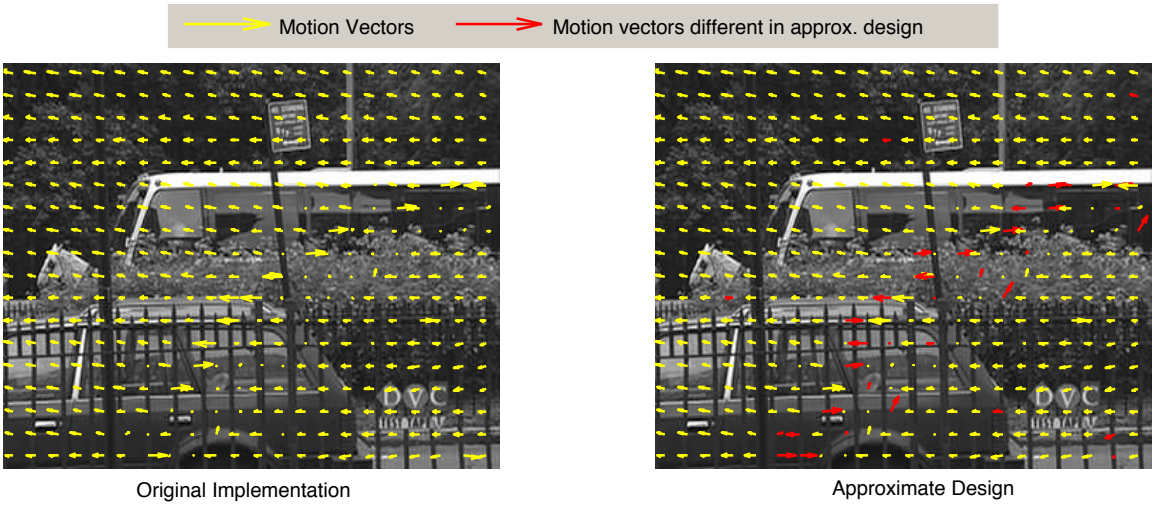

Figure 4.10: Motion vectors from original and approximate block-matching circuits.

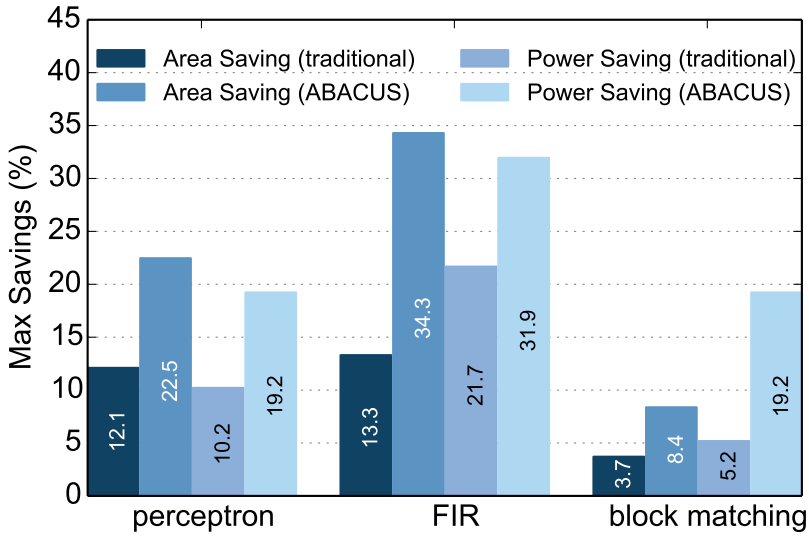

Figure 4.11: Results for use of conventional technique by use of approximate multipliers compared to results from *ABACUS*.

ing global arbitrary approximations that may not be obvious to the designer but produce better results. Finally, Table 4.3 breaks down the amount of data types, operators and arithmetic expressions transformed by *ABACUS* for the three designs in Figure 4.11. The wide distribution of approximations carried out certainly result in better power savings than applying a single approximation technique.

Design	Data Types	Operators	Arithmetic Expressions
perceptron	7	2	0
FIR	21	10	10
block matching	34	59	12

Table 4.3: Number of some of the major transformations made in the three benchmarks.

4.4 Summary and Discussion

In this chapter, we explore the idea of using computational accuracy as a relevant parameter for design-space exploration. The idea stemmed from our work in Chapter 3 where we used the algorithmic accuracy as a design choice to achieve low area, low power and even high throughput circuits. In this chapter, we present a novel way of doing effective design-space exploration of circuits and systems using the idea of approximate computing and leveraging the error resiliency of certain algorithms. We present a behavioral synthesis toolflow that is capable of generating and synthesizing circuits with reasonable error tolerance and significantly less area consumption and power dissipation. While the exact values for these metrics can be highly application dependent, we are able to show with our three test cases that we can get up to 60% savings on both power and area for some designs. Our tool, *ABACUS*, requires no application domain knowledge and applies global transformations that may not be obvious to a designer. These features make it unique compared to previous work in the field of approximate circuits. We also demonstrate the advantages our approach provides over traditional techniques (such as simply using an approximate version of an arithmetic operator).

In the next chapter, we will implement different methodologies and algorithms for design space exploration and fitness selection as an enhancement to the iterative greedy heuristic methodology discussed in this chapter. With better selection

techniques and fitness-ranking mechanism for approximation, we will show that the process of design space exploration using approximation of computational accuracy can be further streamlined and can result in bigger savings in both area and power. We will also discuss how our methodology can be extended to target timing-critical paths optimization and approximation to be able to reduce circuit delay and thus incorporate complimentary power-saving techniques by doing dynamic voltage scaling.

Chapter 5

Exploration Refinements

In Chapter 4, we studied the idea of behavioral synthesis of approximate circuits as a means to achieve significant power and area savings in hardware accelerator implementations. Our novel contribution in the form of the *ABACUS* toolflow validated the applicability of approximating circuits directly at their behavioral level. Working at the highest level of abstraction on arbitrary algorithms and circuit implementations gave us an edge over other related works in the field that require the designer to work at Boolean or gate levels or at least have some prior knowledge of the inherent properties of the algorithm.

In this chapter, we will explore optimizing and enhancing some of the methodologies we instrumented in Chapter 4 for design space exploration and design selection for iterative approximation. Alongside different exploration techniques, we also introduce a novel critical-path aware approximation technique that makes it possible for complementary and well-tested power saving techniques like standard voltage scaling to save extra power.

5.1 Fitness Selection Algorithms

As mentioned in Section 4.1.3, the various transformations applied to a behavioral code can lead to a combinatorial explosion in the number of possible approximate design variants. To streamline the process of design exploration and keep the number of design variants space from increasing exponentially, we proposed a linear-scaled approach for fitness selection. We used the best-ranked individual from each generation of iterations as the parent design for the evolutionary process propagation. This poses a potential problem where design exploration could follow a locally optimum path and not search for points outside a local population pool that could have potentially given better results. In this chapter, we explore and analyze a comparative study of some other design search techniques and fitness ranking methodologies. We test these various schemes on the same three test-benches using similar procedures as for the linear-scaled methodology described in Chapter 4.

5.1.1 Non-Dominated Sorting Genetic Algorithm based Selection

The linear-scaled fitness ranking scheme introduces unnecessary limitations because it picks only one 'best' design as the new transformation seed from each round of transformations, which could potentially waste savings that could have resulted from other unexplored points. Furthermore, considering the best design that is local to a particular generation rather than for the entire population may lead to sub-optimal exploration.

To resolve these issues, we explore a second fitness selection methodology based on the Non-dominated Sorting Genetic Algorithm (NSGA-II) [19, 18, 73]. NSGA-II

is a popular technique used to select multiple optimal parents for crossing-over in genetic algorithms. The fact that this methodology can have multiple individuals with the same fitness rank makes it suitable for our own study — we can use more than one design point as transformation seeds each iteration to widely explore the design space. The NSGA-II process comprises of two main phases:

1. assigning fitness to population members based on non-dominated sorting; and
2. preserving diversity among solutions with same level of fitness.

The first phase involves classifying the population under study into multiple *Pareto fronts*, where each design point of the population is assigned a front with a fitness rank. Designs that do not have any other designs dominating them in at least one of computational accuracy or power savings will all be given the same highest fitness rank, say r_1. Points dominated only by r_1 designs will form the second front with a rank of r_2 and so on. This procedure continues until all design points in the entire population are given their respective fitness ranks.

Once all possible 'best' designs from the entire population — the ones with the highest rank, r_1, are identified, they are further filtered for uniqueness. The uniqueness of a design is computed based on its average difference in accuracy and power savings from its adjacent neighbors. In other words, uniqueness is a measure of how dense the population is around a particular design point. Two designs are considered not unique to each other if their accuracy and power savings are similar. In the second phase of the NSGA-II selection process, individuals that have the same fitness rank but are still far away from each other in terms of uniqueness are favored over individuals that are close to each other. This allows for selection of unique approximate designs with the same fitness that will subsequently produce unique

mutants.

As with the linear-scaled selection scheme, the design space exploration process starts with the single original design as the current design, shown in the algorithm below. Then in steps 2 - 14, it proceeds to generate $M \times N_{sel}$ total variants for N generations, identifying best design candidates using the NSGA-II selection process in step 11. In steps 13 and 14, these multiple best designs are stored and then used one by one as transformation seeds for subsequent generations.

Algorithm *Global NSGA-II fitness based exploration (nsga2-global)*

Input: original exact design

Output: approximate design variants

1. Let $N_{sel} = 5$ number of parent designs to pick after each NSGA-II slection
2. **while** $i \leq N$
3. **while** $s \leq N_{sel}$
4. Let O_s = original design
5. **while** $j \leq M$
6. **do** pick a transformation operator at random;
7. apply the operator to O_i to yield A_j;
8. evaluate accuracy of A_j over input data sets
9. **if** accuracy of A_j is within threshold
10. **then**
11. synthesize A_j;
12. $V = V \cup A_j$;
13. Use results from Steps 9 and 12 and NSGA-II selection to evaluate the fitness, F_j, of A_j;
14. **else** goto step 7;

15. identify A_S, $S \subset \{1, \ldots, MN\}$, with best F;

16. let $O_{s+1} = A_k$, $k \in \{1, \ldots, S\}$;

17. **return** V

By considering multiple best designs in the entire population and using only the most unique among them as transformation seeds, we hope to make our design space exploration more effective and inclusive. However, given we can now use multiple parent transformation seeds per generation, we need to be careful with the expansion of the possible design search space. To keep the exploration process in check, we select five designs with the best fitness ranks from each generation for transformation propagation as shown in the above algorithm. Instead of generating, for example, ten new design points from *one* local best point as with the linear-scaled selection scheme, we generate two new design points from five globally best designs in the NSGA-II based selection scheme. So the total number of possible points generated per generation is the same for both processes and by that virtue, runtime of the NSGA-II based exploration will be the same as the linear-scaled methodology.

Figures 5.2, 5.1, and 5.3 show the entire design population as well as the pareto-tradeoff between power savings and accuracy for the selection and exploration process. The global fitness selection methodology performed poorly compared to the original linear-scaled methodology (Figures 4.7, 4.6, 4.8) in terms of power savings. This can also be seen in Table 5.1. The poor result can possibly be attributed to the fact that the global NSGA-II selection policy always selects the best designs from the global population pool generated *up to* a certain point to avoid getting stuck in local optima. This could, however, mean that if globally best points come from earlier generations, the selection process could keep picking the same designs and this may lead to the design not going through many rounds of transformations to

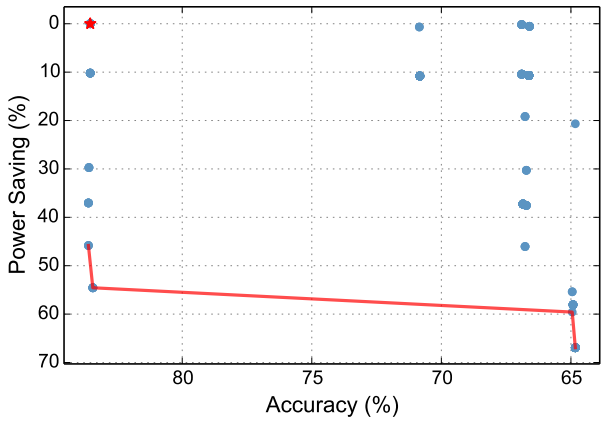

Figure 5.1: Results from various approximate designs and the Pareto Frontier for the perceptron test bench using the global NSGA-II selection scheme.

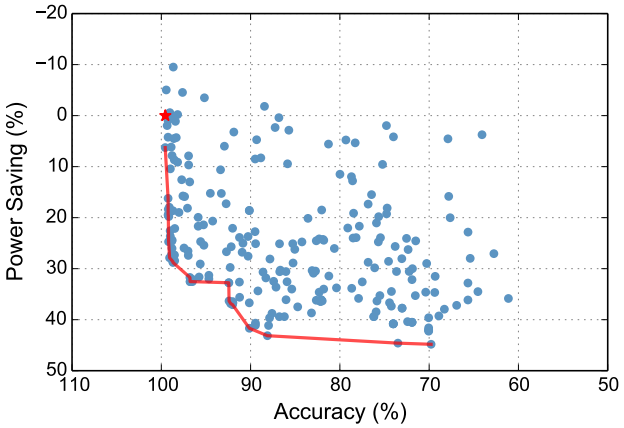

Figure 5.2: Results from various approximate designs and the Pareto Frontier for the FIR test bench using the global NSGA-II selection scheme.

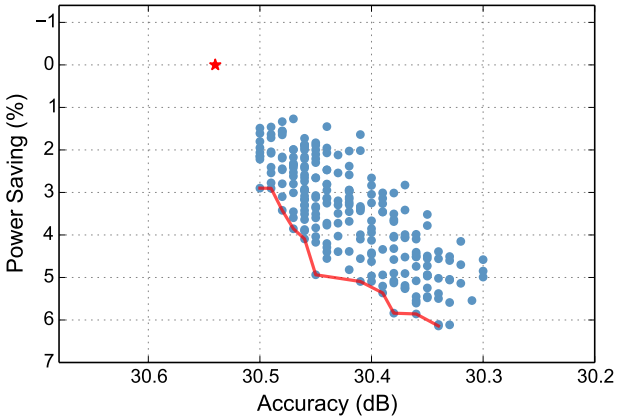

Figure 5.3: Results from various approximate designs and the Pareto Frontier for the block matching test bench using the global NSGA-II selection scheme.

	perceptron	FIR	block matching
linear-scaled	61.87	40.29	14.34
nsga2_global	56.93	42.41	5.43

Table 5.1: Highest power savings(%) for the nsga2-global algorithm compared to linear-scaled.

give a particularly higher power saving.

As for generating a better spread of designs in terms of accuracy, however, the NSGA-II selection methodology performs better than the original linear-scaled approach. We had all experiments run for 8 generations of transformations attempting to generate 10 variants per generation, with the whole process repeated three times. Given the possible 240 unique variants, the linear-scaled selection methodology was able to produce only 212, 233 and 165 unique variants, while the NSGA-II produced 237, 234 and 226 respectively for the FIR, perceptron and the block matching test benches. Remember the *ABACUS* technology is designed to produce a unique variant only if it passes a certain accuracy threshold; if a parent design repeatedly generates unacceptable designs, the tool will pass on that iteration and move on to the next. That NSGA-II consistently produces more unique variants than linear-selection supports the advantage of using multiple best designs which are unique themselves. A corollary of this advantage is that it takes less time for the NSGA-II methodology to generate as many acceptable unique approximate variants as generated by the linear-scaled selection mechanism.

5.1.2 Hybrid Selection Methodology

The linear-scaled and the global NSGA-II based fitness selection methodologies described earlier have their own strengths and weaknesses. The linear-scaled methodology with local optimum selection ensures that a design from a particular generation is representative of all the rounds of changes up to that point. In other words, a design generated in the last generation will have transformations added up from all the generations before it. This will greatly reduce the logical complexity of the original circuit resulting in greater area and power savings. However, as discussed earlier, the local scope used for selection of the best design can still cause the design search process to get stuck on a local optimization path. The NSGA-II based selection scheme solves this problem by considering the global population and choosing more than one parent design to be used as transformation seeds for future generations. However, choosing globally optimum points for population generation and propagation can have its own problems — if good approximate designs are generated in the initial generations of transformations, the selection algorithm may repeatedly select these globally optimum points as transformation seeds, limiting the scope of potential improvements arising from designs with more cumulative changes from all generations.

In light of that, we propose a third fitness selection and exploration scheme that is a combination of both the linear-scaled fitness ranking scheme and the NSGA-II selection scheme. This approach will select *one* best design from a local population pool and additional designs from the overall population generated up to that point. Both local and global optima design points will be considered for use as transformation seeds every generation.

With this hybrid methodology, we generate four new parent designs every gen-

eration, three of which will be the best designs selected using NSGA-II scheme. The remaining one will represent the best design from that particular generation and will be selected using the linear-scaled fitness scheme. This single locally optimum design will generate four approximate variants in the subsequent generation while the three selected by the NSGA-II process will generate three, two and one approximate variants respectively based on their uniqueness ranks. This way, a fair diversification is maintained in the population among designs generated from the local optima and the variants generate from the global optima.

For all schemes, we run *ABACUS* flow for same number of generations with the same number of possible variants in each, thus keeping design space exploration runtimes consistent across all of them. Figures 5.4, 5.5, and 5.6 show the design space explored and results generated for the hybrid selection methodology. Table 5.2 shows the highest savings recorded with the same methodology. In terms of spread of the variants, the hybrid methodology produces 235, 227, and 185 unique variants out of a possible 240 for the FIR, perceptron, and block matching algorithms respectively. These spread numbers as well as maximum power savings are generally better than the linear-selection methodology. Given it contains the virtues of both the linear-scaled as well as the NSGA-II selection schemes, the hybrid methodology gives the designer a good spread of designs to select from as well as good power savings numbers, thus providing for a better and meaningul design trade-off. Also, as with the NSGA-II selection, the hybrid methodology generates acceptable designs within an accuracy margin faster than linear-scaled methodology because of the multiple transformation seed selection.

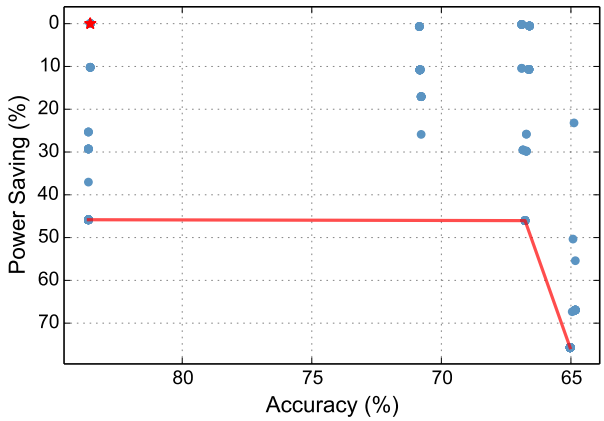

Figure 5.4: Results from various approximate designs and the Pareto Frontier for the perceptron test bench using the hybrid selection scheme.

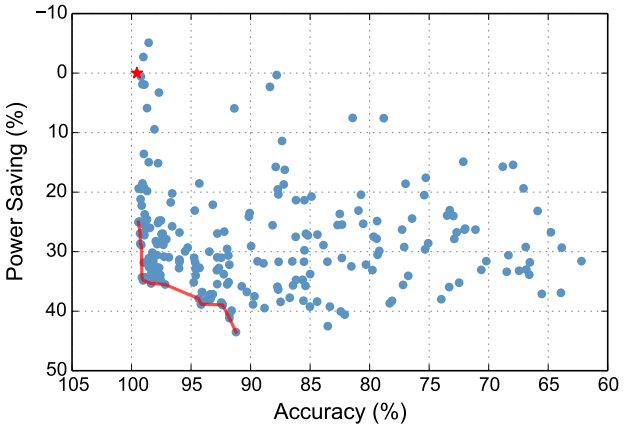

Figure 5.5: Results from various approximate designs and the Pareto Frontier for the FIR test bench using the hybrid selection scheme.

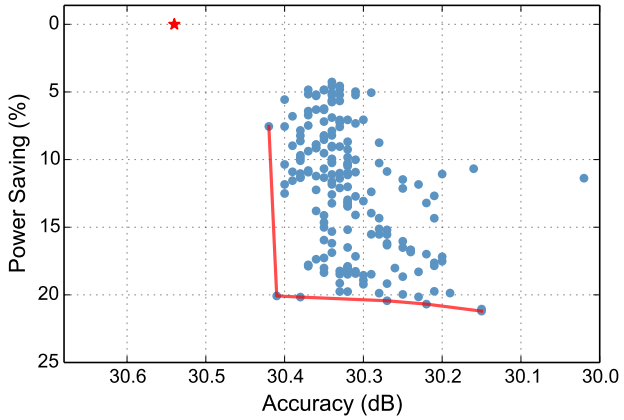

Figure 5.6: Results from various approximate designs and the Pareto Frontier for the block matching test bench using the hybrid selection scheme.

	perceptron	FIR	block matching
linear-scaled	61.87	40.29	14.34
hybrid	70.00	41.62	16.91

Table 5.2: Highest power savings(%) for the hybrid selection algorithm compared to linear-scaled.

5.1.3 Variants of Exploration Methodologies

To make a comparative case-study about various exploration methodologies and study the effects of their inherent properties, we carried out multiple variants of the NSGA-II selection and hybrid methodologies. By altering some properties of the methodologies we explained in Sections 5.1.1 and 5.1.2, we look at what changes may be brought about in the overall exploration process and accuracy-power saving trade-offs obtained from it. We explain some of the variant approaches we realized below.

1. NSGA-II local selection (nsga2-local): In this approach, we implement the NSGA-II selection methodology but instead of selecting multiple best designs from the global population pool, we select multiple best designs from the local generation after each iteration. This algorithm is similar to the linear-scaled methodology in terms of its local scope but chooses more than one best designs from each generation.

2. Weighted NSGA-II global selection (weighted-nsga2-global): A second variant of the NSGA-II algorithm is implemented by weighing the multiple best designs selected from the entire global population. The best designs from the entire population are selected by assigning them to ranked Pareto fronts and then further filtered by their uniqueness. But instead of selecting five parent design after each generation and generating equal number of variants from each as in the global NSGA-

II approach presented in Section 5.1.1, we select four parent designs and weigh the best designs based on their linear-scaled fitness (as in Equation 4.1). From these four parents, we generate four, three, two and one variants respectively according to their ranks. This ensures the number of variants generated every generation remains the same as with earlier methodologies.

3. Weighted NSGA-II global selection with no crowdedness measure (weighted-nsga2-global-nodist): This selection methodology is very similar to the previous one, but the global NSGA-II selection algorithm is stripped off its second property where uniqueness of a design is measured based on its closeness to its adjacent neighbors. The uniqueness of design is an important factor in genetic algorithms where you may want to cross two unique and different parents together for cross-over. However, in an evolutionary algorithm like ours where mutations are made randomly on an individual, two or more parent designs with similar accuracy and power savings could produce approximate variants with entirely different accuracies based on the location of transformations made. Hence, we experiment with an exploration method where multiple best designs are selected from ranked Pareto fronts but without being filtered for their uniqueness factor. Instead, they are all ranked by the linear-scaled fitness formula which combines weighted accuracy and power savings for each design into a fitness rank. Four best-ranked designs from the best Pareto front each generate four, three, two and one variants respectively based on their linear-scaled fitness.

4. Weighted NSGA-II local selection (weighted-nsga2-local): This is the same as variant 2 but instead of selecting best transformation seeds from the global population, multiple best designs are selected from the population local to each generation. In addition to the Pareto ranking, designs from same Pareto front are ranked according to their weighted combination of accuracy and power savings and

four best ranked designs are used as parent designs in the subsequent generation.

5. Weighted NSGA-II local selection with no crowdedness measure (weighted-nsga2-local-nodist): This methodology is the same as variant 3 but it selects best designs from the local population after every generation instead of the global population up to that point. Also, filtering by uniqueness factor is taken out of the algorithm as in variant 3.

6. Hybrid methodology with no crowdedness measure (hybrid-nodist): . This methodology is a variant of the hybrid methodology presented in Section 5.1.2, but as with some of the other variants described above, we take away the property of the NSGA-II selection algorithm to filter for designs with less crowdedness. Instead best designs selected from the global population pool are ranked according to their fitness given by the combined weighted accuracy and power savings. The single local optimum point selected for use as transformation seed after every generation remains unaffected.

7. Savings per error based selection methodology (slope): This methodology is a distant variant of the original linear-scale selection methodology. Instead of ranking every design by the linear combination of its weighted computational accuracy and power savings, we rank it by power savings per computation error, which we call its slope for simplicity. For example, an approximate design with 40% power savings with 30% error will be ranked lower than a design with 20% power savings with 2% error.

From the results shown in Tables 5.3, 5.4, and 5.5, it is seen that the hybrid methodology which has the merits of using multiple best designs including both local and global optimum points as transformation seeds performs consistently well

Methodology	Max. Power Savings(%)	Highest Accuracy Achieved(%)
linear-scaled	61.63	83.50
nsga2-global	56.93	83.56
nsga2-local	72.8	83.54
weighted-nsga2-global	21.37	83.58
weighted-nsga2-global-nodist	14.75	83.58
weighted-nsga2-local	15.61	83.57
weighted-nsga2-local-nodist	12.68	83.57
hybrid	72.9	83.60
hybrid-nodist	71.99	83.48
slope	71.23	83.55

Table 5.3: Comparison of maximum power savings and highest computational accuracy achieved for different exploration methods used for the Perceptron test bench, for error tolerance up to 40%.

Methodology	Max. Power Savings(%)	Highest Accuracy Achieved(%)
linear-scaled	40.29	96.13
nsga2-global	42.41	92.3
nsga2-local	42.71	91.80
weighted-nsga2-global	39.97	95.47
weighted-nsga2-global-nodist	39.32	92.99
weighted-nsga2-local	41.74	91.41
wighted-nsga2-local-nodist	44.15	92.53
hybrid	41.62	92.39
hybrid-nodist	39.17	95.23
slope	36.76	93.30

Table 5.4: Comparison of maximum power savings and highest computational accuracy achieved for different exploration methods used for the FIR test bench, for error tolerance up to 10%.

Methodology	Max. Power Savings(%)	Highest Accuracy Achieved(dB)
linear-scaled	14.34	30.45
nsga2-global	5.43	30.47
nsga2-local	3.53	30.49
weighted-nsga2-global	15.79	30.45
weighted-nsga2-global-nodist	13.12	30.46
weighted-nsga2-local	14.76	30.46
weighted-nsga2-local-nodist	16.13	30.47
hybrid	16.91	30.45
hybrid-nodist	16.47	30.47
slope	12.21	30.47

Table 5.5: Comparison of maximum power savings and highest computational accuracy achieved for different exploration methods used for the block matching test bench, for PSNR $\geq$ 30.45.

yielding both good maximum power savings and highest computational accuracy achieved. It also seems like filtering designs for uniqueness according to their closeness to adjacent neighbors produces generally good results. Compared to the linear-scaled methodology proposed in Chapter 4, the hybrid selection process is a significant improvement. The savings from the two methodologies discussed in Section 5.1.1 and 5.1.2 compared to the linear-scaled methodology is shown in Figure 5.7. Results are averaged for the three sets of experiments run for 8 generations and 10 possible variants per generation.

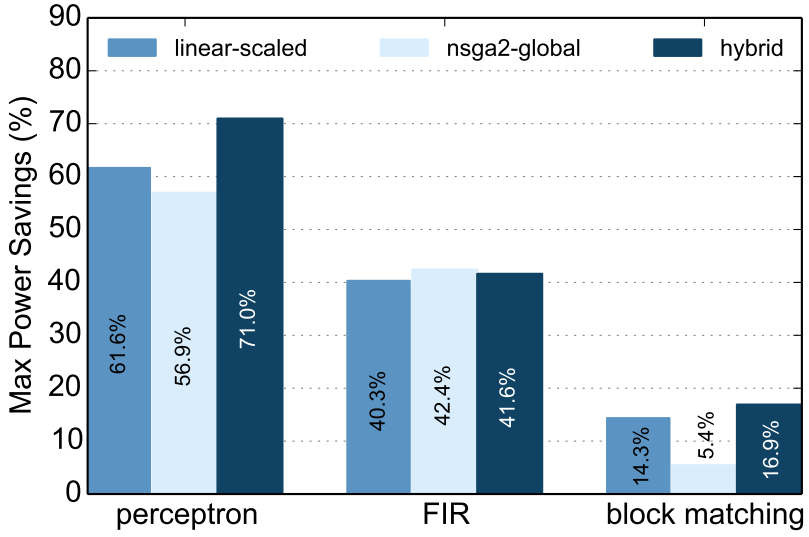

Figure 5.7: Power Savings for the testbenches given by the linear-scaled, NSGA-II selection based and hybrid selection based design-exploration methodologies.

5.2 Techniques for Additional Power Savings

One of the biggest features of our approximation methodology used for design space exploration is that complementary power saving techniques can be applied on top of the *ABACUS* flow to obtain bigger power savings. As discussed in Section 2.3.1, common run-time power saving techniques like standard voltage scaling can be cou-

pled with our approximation techniques. Such run-time optimization methodologies can allow designers to obtain even better savings in power consumption in the post-design phase of an ASIC development cycle. In this section, we explore the possibility of using approximation for reduction of circuit delay, which will make way for the use of standard voltage scaling to further reduce power consumption.

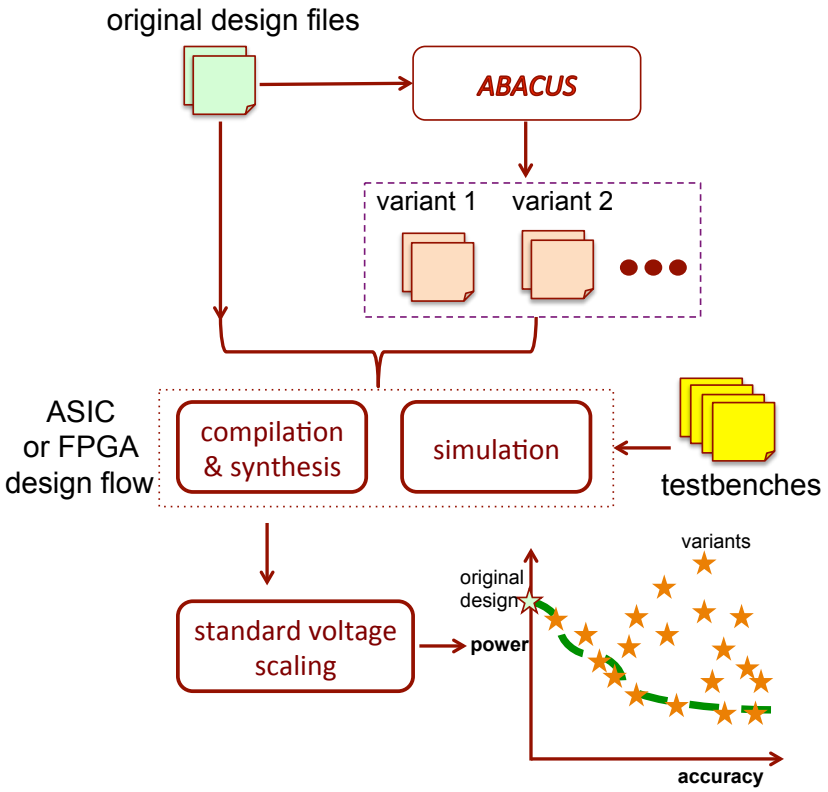

Figure 5.8: The new *ABACUS* flow with voltage scaling added in the post-design phase.

5.2.1 Targeted Critical Path Approximation

Along with our goals of reducing circuit area and power consumption, we also make special optimizations on approximate circuits so that they run faster than their original counterparts. To ensure circuit delay is reduced, we implement a technique that applies targeted approximations on the timing-critical paths of an implementation. Approximations are usually done on random locations of a circuit based on transfor-

mation probabilities as discussed in Section 4.1.3. However, to realize circuit variants with better timing, we make changes to this approach and prioritize approximations along the timing-critical paths in a circuit first. By aggressively approximating the critical path, we hope to obtain circuits that have less logic complexity because of the transformations applied and also demonstrate positive timing slacks during the synthesis process.

Our approach to critical path prioritized approximation involves reading the timing report after the original circuit synthesis and then parsing out the relevant paths with the longest data arrival times. We extract information about the longest paths in the circuit and apply aggressive transformations on all arithmetic operations involved in these paths. Regular transformations are then performed based on randomized probabilities on rest of the circuit. Prioritizing approximations on the timing-critical paths is highly application dependent — if a circuit implementation has relatively large number of parallel running timing-critical paths, aggressively approximating all the paths to reduce circuit delay may cause a big hit in output accuracy given majority of the circuit paths will now have been approximated. However, it opens the possibility of approximate circuits with less power consumption. As far as we know, our targeted critical path optimization approach to approximate computing is a first of its kind.

5.2.2 Standard Voltage Scaling

If targeted critical path optimization contributes to positive timing slacks in circuit implementations, the circuit can be synthesized against all available voltage corners keeping other synthesis parameters constant. First, regular circuits are synthesized at a nominal supply voltage to generate netlists that are then resynthesized at other

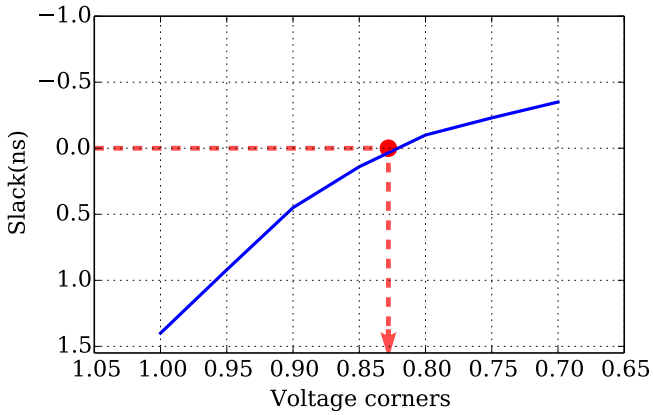

Figure 5.9: Illustration of how a lower voltage value at which the circuit can still run without additional timing errors can be obtained from available synthesis results and interpolation.

available voltage corners. Using the same netlist across different supply voltages ensures that the circuit remains exactly same during resynthesis. If there is positive timing slack at nominal voltage, timing slacks at all voltage corners are recorded and a simple curve fit and interpolation is used to determine a lower supply voltage point at which the circuit has close to zero timing slack as shown in Figure 5.9. This process allows us to run the approximate circuits with no timing errors but with lower supply voltage, thereby contributing to a lower dynamic power consumption.

5.2.3 Results from Critical Path Approximation and Voltage Scaling

By exploring more inclusive exploration algorithms and critical path targeted approximation as well as dynamic voltage scaling based on timing slacks obtained from approximations, we are able to improve on power as well as area savings compared to our own work presented in Chapter 4.

Using the hybrid methodology complemented with critical path approximation

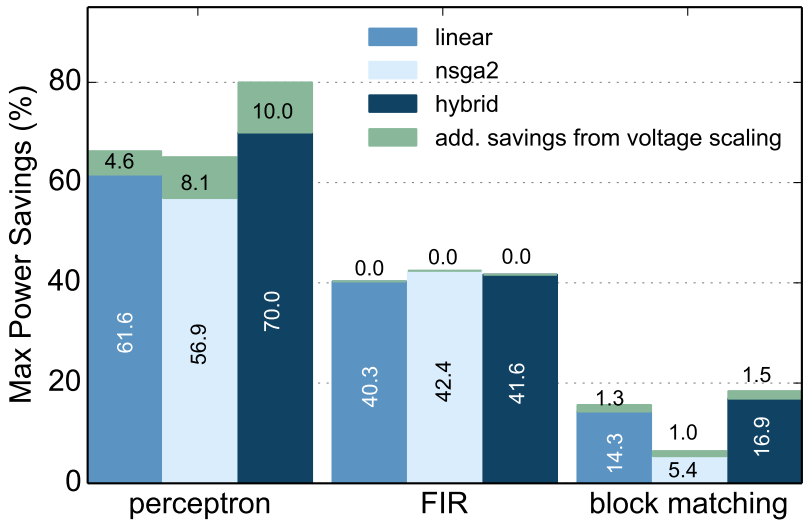

Figure 5.10: Power savings for the testbenches given by the linear-scaled, NSGA-II based selection and hybrid selection methodologies. Additional power savings obtained from voltage scaling also shown.

and standard voltage scaling, we were able to achieve up to 10% additional power savings on our testbenches. This is shown in Figure 5.10.

Table 5.6 details this improvement in power savings from using the newer hybrid exploration methodology that combines the merits of the linear-scaled as well as the NSGA-II based selection, over simply linear-scaled selection presented in Chapter 4. It also demonstrates additional percentage savings that comes from voltage scaling. We achieve 3.35%, 29.75%, and 31% improvement in power savings for perceptron, FIR and block matching testbenches respectively.

Design	Linear-scaled	Linear+VS	Hybrid	Hybrid+VS	Improvement
perceptron	61.63	66.62	70.00	79.97	29.75%
FIR	40.29	40.29	41.64	41.64	3.35%
block matching	14.30	15.61	16.91	18.381	31.00%

Table 5.6: Improvement on maximum power savings from using an enhanced selection and exploration methodology as well as standard voltage scaling (VS) over a linear-scaled selection methodology.

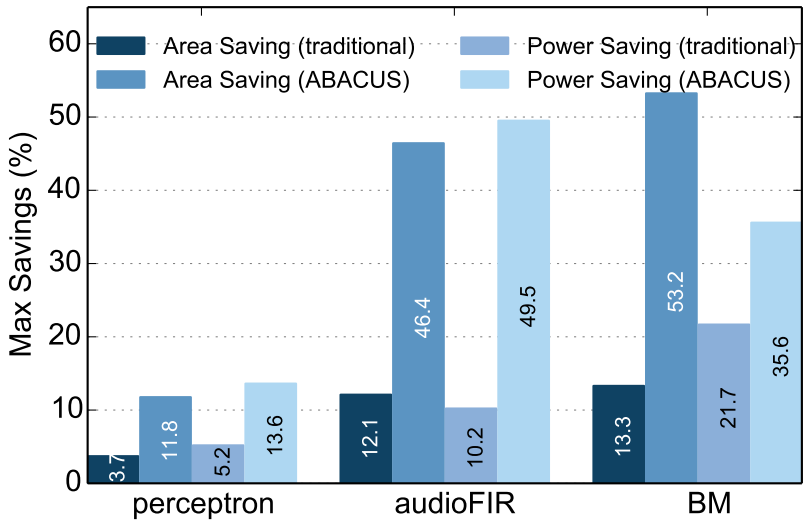

Figure 5.11: Results for use of conventional technique by use of approximate adders and multipliers compared to results from *ABACUS*.

As in Chapter 4, we also compare results generated by *ABACUS* with the new hybrid methodology and voltage scaling to the same traditional technique where approximate versions of standard components are used in place of the accurate ones. Results are shown in Figure 5.11 for designs that maintain same accuracy. Expectedly, our method obtains superior results for all three benchmarks.

5.3 Summary and Discussion

In this chapter, we explored some new fitness selection algorithms for the iterative evolutionary methodology we developed for approximating circuits at the behavioral level. In addition to this, we developed and added a feature to our *ABACUS* tool that automatically identifies timing-critical paths in a circuit and applies targeted approximations on them first. This enables generation of approximate circuits that are not just smaller and use less area but also run faster. We couple the timing

enhancement achieved from this approach with standard voltage scaling to run approximate circuits at a lower supply voltage, thus giving us extra power savings.

All these features make our methodology unique compared to some of the previous work in the field of design space exploration of hardware accelerators. We used approximate computing as a pathway to explore the design space, and showed that our methodology works with applications from various domains like signal processing, computer vision and machine learning. With negligible degradation to output accuracy, we were able to achieve power savings of up to 80%. We also developed the platform for standard voltage scaling by applying approximation optimizations on timing-critical paths in circuit implementations. This targeted approximation approach allowed us to get as much as 10% additional savings. Coupled with better scoped and inclusive exploration process, we demonstrated our work presented in this chapter yielded up to 31% better savings in power consumption compared to methodologies presented in Chapter 4.

Chapter 6

Summary of Dissertation and Possible Future Extensions

As hardware accelerators and custom computing circuits are becoming increasingly popular to meet the demands of high throughput computing and real-time data processing, the need for solutions that perform in highly resource-constrained environments is increasing as well. The use of these high performance computing platforms and accelerators in embedded devices, airborne systems and mobile platforms has called for designs to be optimized for area and power.

In this thesis, we turned our focus on different accelerator platforms like the ASIC and the FPGA and explored options on how we can make multi-objective optimizations on such designs. We explored various design exploration methodologies and methodically tuned algorithmic as well as architectural design choices to derive efficient paths towards design optimality.

6.1 Summary of Results

In Chapter 3, we presented novel ways to perform fast design space exploration with examples tested on an FPGA platform. We explored the idea of using analytical modeling for design metric characterization. With various architectural and algorithmic parameters, each tunable to different settings, the total design space that arises from these parameter perumatations is exponentially large. Physically implementing each of these possible design variants is virtually impossible. As a solution we proposed to use regression modeling with L_1 regularization to sample less than 1% of the vast design space and accurately predict the various design metrics like area, power, throughput of a deisgn for unexplored design points.

We implemented two test cases — first, an image deblurring algorithm and second, a block matching algorithm representative of the designs we wanted to optimize. From these designs, we identified algorithmic as well as hardware parameters that were tunable to get different variants of the design with different power, area, and sometimes even different computational accuracy characterizations. We used L_1 regularization-based regression methodology to automatically use and identify the relevant design parameters that affect these various design metrics. By sampling a meager 0.3% of the possible design space, we showed that analytical models obtained were consistently accurate over 90% for the rest of the unexplored design space. The fact that less than 1% of the design space had to be physically explored to get such accurate results meant we also achieved a 340× improvement over a brute-force physical implementation of every single design point possible. Lastly, we also used these accurate models for area, power and computation accuracy to formulate multi-optimization frameworks and answer imporatant questions about optimizing design objectives under various constraints. We investigated how we can

tackle design questions such as minimizing design power under accuracy constraints, or minimizing design area within a power budget. By using mathematical models for the various design objectives we wanted to optimize, we were able to replace the long and time-consuming synthesis process in to a fast analytical approach.

More importantly, Chapter 3 gave us unique ideas about using computational accuracy of an algorithm as a negotiable design metric that could be used for novel design exploration techniques. In Chapter 4, we used this motivation and pursued the idea of trading off output accuracy of algorithms to achieve low area and less power consuming circuits and systems. We showed that certain domains like multimedia and graphics, signal processing, machine learning, among others are naturally amenable to introduction of inaccuracies in their underlying computations. Using this fact to our advantage, we developed a first of its kind tool called *ABACUS* that produces approximate variants of a circuit automatically from its behavioral/RTL descriptions.

With the proposed tool, we first parsed a behavioral circuit description to an Abstract Syntax Tree (AST) and then applied reasonable transformations that reduced the circuit complexity at the cost of reduced computational accuracy. Accuracy thresholds were defined beforehand and the tool performed within prescribed error margins. To streamline the production and exploration of new designs, we realized an iterative greedy algorithm and performed approximation changes on the original design iteratively for a number of predefined generations. A linear-scaled fitness ranking system based on weighted combination of the computational accuracy and power savings of each individual approximate design is used to select the 'best' design for use as transformation seed for subsequent generations. We demonstrated that our automated approximation methodology worked successfully on three test benches — a perceptron classifier, an FIR filter, and a video block matching algo-

rithm implementation. For one of the test benches, we were able to achieve over 40% savings in power and area consumption with as little as 8% accuracy degradation. What makes our methodology really stand out among related work is the fact that it performs better than approximating just a set of adders or multipliers and implements a more global scope of transformations without the designer having to know anything about the algorithm beforehand. Behavioral descriptions capture the algorithmic intent of the circuit; and thus raising the level of abstraction enables a larger range of approximations that are not possible to apply at low-level design specifications.

While in Chapter 4 we presented the novel idea of using behavioral approximate synthesis as a way to do effective design exploration, we focused on fine-tuning exploration techniques and the approximation toolflow in Chapter 5. We implemented a few different methodologies to improve the fitness ranking system for approximate designs presented in Chapter 4. We introduced a Pareto frontier based fitness selection methodology derived from the NSGA-II genetic algorithm that chose more than one best design per generation, thus expanding the scope of exploration. We also presented a hybrid methodology that included the merits of both the linear-scaled fitness selection methodology based on selecting the local optimum from a population and the NSGA-II methodology that chose multiple best designs from the entire global population. This methodology by itself turned out to have 8.4%, 1.3%, and 2.3% better power savings than the linear-scaled methodology of Chapter 4 for the perceptron, FIR and block matching algorithms respectively.

In addition to this more inclusive search and fitness selection approach, we also introduced a novel idea of performaing targeted approximations on the timing-critical paths of a circuit to reduce its delay. Performing agressive critical path approximations allowed us to use circuits with less delay at lower supply voltages to take

advantage of extra power savings obtained from standard voltage scaling. We obtained as much as 10% power savings just from critical path approximation and voltage scaling on one of our test benches. Coupled together with the hybrid fitness selection methodology, we were able to gain up to additional 31% power savings compared to the approximation and exploration methodologies of Chapter 4.

As a whole, we introduced a novel set of ideas such as analytical modeling and multi-objective optimization of design metrics, as well as approximate computing as potentially very high yield design space exploration techniques. Hardware accelerators that operate in various favorable domains and have to operate in resource-constrained environments can certainly benefit a lot from our proposed and validated ideas in this dissertation.

6.2 Future Work and Possible Extensions

The work presented in this dissertation can be extended in several directions. By combining the analytical modeling presented in Chapter 3 and automated synthesis of approximate circuits presented in Chapter 4, we can speed up the process of design space exploration of optimal hardware accelerators even more. For approximate circuit synthesis, we can investigate empirical regression models that estimate the outcomes from the physical design flow in terms of area, timing, and power, without executing the actual design flow. By modeling a physical outcome (e.g. power, accuracy) as a mathematical function, we can use some of the design variables used in approximate circuit synthesis process like number of iterations unrolled in a behavioral code and locations of transformations applied. These functions can be validated for accuracy by using a number of samples to identify the best pa-

rameters that minimize the total square error between the function estimates and the actual outcomes first, and then cross-validating later. The functions can then substitute the entire design flow for the other non-sampled designs, speeding up the design space exploration process significantly.

In the future, we can also make the approximate circuit generation process smarter by pre-processing the behavioral code for accuracy hot-spots. There are many locations in a behavioral code description of the circuit that are more amenable to transformations than others. While we implemented a similar idea of prioritizing approximations over areas that are timing-critical to reduce circuit delay, we can also realize similar methodologies for reduction of power consumption or computational error caused. By keeping a history of transformations made and their impact on power and area consumption and computational accuracy, code hot-spots can be discovered and can be used to make smarter choices about where and when to apply approximations.

Another area of possible work in the future could be in the use of directives to guide approximations through various parts of a behavioral HDL code. Behavioral circuit descriptions can have both control logic as well as data path descriptions. Our current methodology is favorable for changes in the data path but will still work if changes are made in the control logic. Like we mentioned earlier, our methodology does not require any application domain or architecture knowledge. Errors caused by a change in the control logic of an implementation are very likely to be large enough to be detected by our accuracy checking mechanism and a variant with such a transformation would be automatically discarded. Nevertheless, some input from the designer may be useful in guiding the approximate circuit generation. By inserting directives in the behavioral code to indicate to our tool which areas can withstand changes and which may be untouchable, we can generate acceptable variants faster

and save a lot of time in design space exploration.

Our methodology for fast and automated behavioral synthesis of approximate circuits can be used in various circuits and systems optimization frameworks. Here we provide a brief overview of some of the possible project extensions we can pursue in the future.

Use in dynamic power management

There could be cases, for example, when the system may not require computation with the highest possible accuracy. An approximate computing circuit component can be deployed for low power mode operation and can be switched with an exact version of it in a simple coarse-grain fashion. For example, when low-power mode is enabled, the outputs from the approximate circuits are selected and the exact circuit is power-gated by using sleep transistors, and vice versa as shown in Figure 6.1. Switching to the approximate version of the circuit component can help save both dynamic and static power. This may become more and more relevant in the

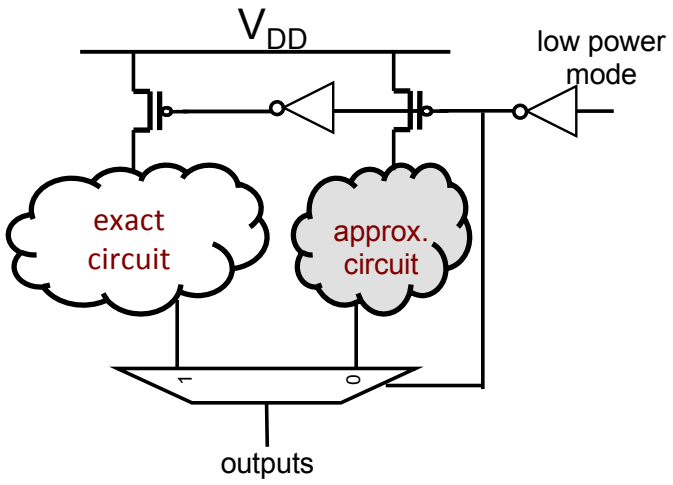

Figure 6.1: Illustration of a possible power management system using approximate circuits for low power consumption

imminent Dark Silicon era in the semiconductor industry. Dark Silicon is essentially the number of transistors that will have to be switched off in a multi-processor system to keep it under given power budget [23]. Dark Silicon has become relevant and inevitable way of controlling leakage power given aggressive technology node scaling and the packing of more transistors per chip.

The disadvantage of coarse-grain multiplexing is that it can incur a significant area overhead due to the need to implement exact and approximate circuits entirely. However, this can be minimized if the exact and approximate circuits can share many common modules or subcircuits that are identical to both of them. Using the similarities between the two versions that arise from the operation of our synthesis algorithm, we can leave portions of the code unmodified if they are deemed critical to accuracy.

Use in redundancy circuits

Using redundant approximate circuits for concurrent error detection and correction can lead to less area and power overhead while still producing outputs that are close to original circuits [15, 40]. Figure 6.2 shows an approximate version of a circuit used as an error checker running in parallel with the exact version. The same concept can

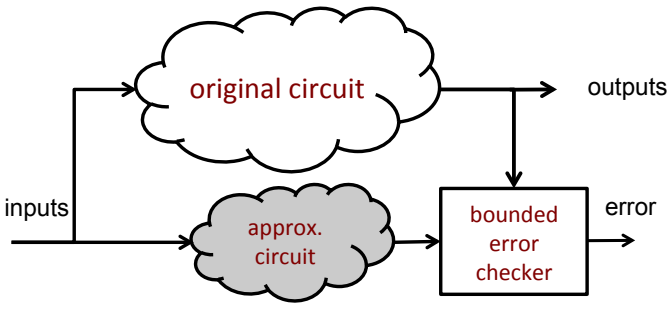

Figure 6.2: Illustration of an approximate system used for redundancy robustness

be followed to use approximate circuits in modular redundancy circuits. As long as the outputs of the original and approximate versions differ only within a reasonable bound, the ciruit could be considered error-free.

If an error is indeed discovered, in case where the exact circuit and approximate circuit vary beyond an acceptable threshold, the error needs to be corrected. Standard error correction techniques with possible voter mechanisms can be realized to deal with such a case.

Bibliography

[1] YUV Video Sequences. `http://trace.eas.asu.edu/yuv/`.

[2] *Digital Integrated Circuits : A Design Perspective*. Prentice Hall electronics and VLSI series. Pearson Education, 2 edition, 2003.

[3] G. Agosta, G. Palermo, and C. Silvano. Multi-objective Co-exploration of Source Code Transformations and Design Space Architectures for Low-Power Embedded Eystems. In *ACM Symposium on Applied Computing*, pages 891–896, 2004.

[4] P. Albicocco, G.C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re. Imprecise Arithmetic for Low Power Image Processing. In *Asilomar Conference on Signals, Systems and Computers*, pages 983–987, 2012.

[5] G. Ascia, V. Catania, and M. Palesi. A Framework for Design Space Exploration of Parameterized VLSI Systems. In *IEEE Asia and South Pacific Design Automation Conference*, pages 245–250, 2002.

[6] G. Ascia, V. Catania, and M. Palesi. A Multiobjective Genetic Approach for System-Level Exploration in Parameterized Systems-on-a-Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):635 – 645, 2005.

[7] S.S. Bhattacharyya B. KisaCanin and S. Chai. *Embedded Computer Vision*. Springer, London, 2009.

[8] R. Bekkerman, M. Bilenko, and J. Langford. *Scaling up Machine Learning: Parallel and Distributed Approaches*. Cambridge University Press, 2011.

[9] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, 19(4):23–29, 1999.

[10] R. Byrd, J. Gilbert, and J. Nocedal. A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming. *Mathematical Programming*, 89:149–185, 1996.

[11] A.P. Chandrakasan and R.W. Brodersen. Minimizing Power Consumption in Digital CMOS Circuits. *Proceedings of the IEEE*, 83(4):498–523, 1995.

[12] D. Chen, J. Cong, Y. Fan, and Z. Zhang. High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs. In *IEEE Asia and South Pacific Design Automation Conference*, pages 529–534, 2007.

[13] V.K. Chippa, S.T. Chakradhar, K. Roy, and A. Raghunathan. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In *IEEE/ACM Design Automation Conference*, pages 1–9, 2013.

[14] V.K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S.T. Chakradhar. Scalable Effort Hardware Design: Exploiting Algorithmic Resilience for Energy Efficiency. In *IEEE/ACM Design Automation Conference*, pages 555–560, 2010.

[15] M.R. Choudhury and K. Mohanram. Approximate Logic Circuits for Low Overhead, Non-intrusive Concurrent Error Detection. In *IEEE/ACM Design, Automation and Test in Europe*, pages 903–908, 2008.

[16] C. Claus, J. Zeppenfeld, F. Müller, and W. Stechele. Using Partial-run-time Reconfigurable Hardware to Accelerate Video Processing in Driver Assistance System. In *IEEE/ACM Design, Automation and Test in Europe*, pages 498–503. EDA Consortium, 2007.

[17] J. Das, S.J.E. Wilton, P. Leong, and W. Luk. Modeling Post-Techmapping and Post-Clustering FPGA Circuit Depth. In *International Conference on Field Programmable Logic and Applications*, pages 205 –211, 2009.

[18] K. Deb. Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *IEEE Transactions on Evolutionary Computation*, 7:205–230, 1999.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[20] S. Doochul and S.K. Gupta. A Re-design Technique for Datapath Modules in Error Tolerant Applications. In *Asian Test Symposium*, pages 431–437, 2008.

[21] C. Erbas, S. Cerav-Erbas, and A. D. Pimentel. Multiobjective Optimization and Evolutionary Algorithms for the Application Mapping Problem in Multiprocessor System-on-chip Design. *IEEE Transactions on Evolutionary Computation*, 10(3):358–374, 2006.

[22] D. Ernst, S. Das, Seokwoo Lee, D. Blaauw, T. Austin, T. Mudge, Nam Sung Kim, and K. Flautner. Razor: Circuit-level Correction of Timing Errors for Low-Power Operation. *IEEE Micro*, 24(6):10–20, 2004.

[23] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *ACM International Symposium on Computer Architecture*, pages 365–376, 2011.

[24] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture Support for Disciplined Approximate Programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 301–312, 2012.

[25] C.M. Fonseca and P.J. Fleming. Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization. In *International Conference on Genetic Algorithms*, pages 416–423, 1993.

[26] A. Forsgren, P.E. Gill, and M.H. Wright. Interior Methods for Nonlinear Optimization. *SIAM review*, 44(4):525–597, 2002.

[27] T. Givargis and F. Vahid. Platune: A Tuning Framework for System-on-a-Chip Platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(11):1317–1327, 2002.

[28] T. Givargis, F. Vahid, and J. Henkel. System-level Exploration for Pareto-optimal Configurations in Parameterized Systems-on-a-Chip. In *IEEE/ACM International Conference on Computer Aided Design*, pages 25–30, 2001.

[29] R. Gonzalez, B.M. Gordon, and M.A. Horowitz. Supply and Threshold Voltage Scaling for Low Power CMOS. *IEEE Journal of Solid-State Circuits*, 32(8):1210–1216, 1997.

[30] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[31] C. Hsieh, M. Pedram, G. Mehta, and F. Rastgar. Profile-driven Program Synthesis for Evaluation of System Power Dissipation. In *IEEE/ACM Design Automation Conference*, pages 576–581, 1997.

[32] J. Huang, J. Lach, and G. Robins. A Methodology for Energy-quality Tradeoff Using Imprecise Hardware. In *IEEE/ACM Design Automation Conference*, pages 504–509, 2012.

[33] Ali Irturk, Bridget Benson, Shahnam Mirzaei, and Ryan Kastner. GUSTO: An Automatic Generation and Optimization Tool for Matrix Inversion Architectures. *ACM Transactions on Embedded Computing Systems*, 9:32:1–32:21, 2010.

[34] Y.V. Ivanov and C.J. Bleakley. Real-time H.264 Video Encoding in Software with Fast Mode Decision and Dynamic Complexity Control. *ACM Transactions on Multimedia Computing, Communications and Applications*, 6(1):5:1–5:21, 2010.

[35] P. Jamieson, K.B. Kent, F. Gharibian, and L. Shannon. Odin II - An Open-source Verilog HDL Synthesis tool for CAD Research. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 149–156, 2010.

[36] T. Jiang, X. Tang, and P. Banerjee. Macro-models for High Level Area and Power Estimation on FPGAs. In *Proceedings of the 14th ACM Great Lakes symposium on VLSI*, pages 162–165, 2004.

[37] N. Julien, J. Laurent, E. Senn, and E. Martin. Power Consumption Modeling and Characterization of the TI C6201. *IEEE Micro*, 23(5):40–49, 2003.

[38] E. Kang, E. Jackson, and W. Schulte. An Approach for Effective Design Space Exploration. In *Monterey Conference on Foundations of Computer Software: Modeling, Development, and Verification of Adaptive Systems*, pages 33–54, 2011.

[39] D.R. Kelly and B.J. Phillips. Arithmetic Data Value Speculation. In *Asia-Pacific Conference on Advances in Computer Systems Architecture*, pages 353–366, 2005.

[40] E.P. Kim and N.R. Shanbhag. Soft N-Modular Redundancy. *IEEE Transactions on Computers*, 61(3):323–336, 2012.

[41] K. Koh, S. Kim, and S. Boyd. An Interior-point Method for large-scale L1-regularized Logistic Regression. *Journal of Machine Learning Research*, 8:1519–1555, 2007.

[42] R. Kohavi et al. A study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *IJCAI*, volume 14, pages 1137–1145, 1995.

[43] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading Accuracy for Power with an Underdesigned Multiplier Architecture. In *International Conference on VLSI Design*, pages 346–351, 2011.

[44] T. Kuroda. CMOS Design Challenges to Power Wall. In *International Microprocesses and Nanotechnology Conference*, pages 6–7, 2001.

[45] V. Lapinskii and F. Jacome. Algorithms for Compiler-Assisted Design Space Exploration of Clustered VLIW ASIP Datapaths, 2001.

[46] B.C. Lee and D. Brooks. Roughness of Microarchitectural Design Topologies and its Implications for Optimization. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 240 –251, 2008.

[47] B.C. Lee and D.M. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. *SIGOPS Operating Systems Review*, 40:185–194, 2006.

[48] B.C. Lee and D.M. Brooks. Illustrative Design Space Studies with Microarchitectural Regression Models. In *IEEE International Symposium on High Performance Computer Architecture*, pages 340–351, 2007.

[49] D. Lee, M. Wolf, and H. Kim. Design Space Exploration of the Turbo Decoding Algorithm on GPUs. In *ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 217–226, 2010.

[50] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP Design Space Exploration subject to Physical Constraints. In *International Symposium on High-Performance Computer Architecture*, pages 17–28, Feb 2006.

[51] S. Lu. Speeding up Processing with Approximation Circuits. *IEEE Transactions on Computers*, 37(3):67–73, 2004.

[52] H.R. Mahdiani, A Ahmadi, S.M. Fakhraie, and C. Lucas. Bio-Inspired Imprecise Computational Blocks for Efficient VLSI Implementation of Soft-Computing Applications. *IEEE Transactions on Circuits and Systems*, 57(4):850–862, 2010.

[53] Mentor Graphics. *Mentor Graphics Monet ^{TM} User's Manual*, 1999.

[54] A.K. Mishra, R. Barik, and S. Paul. iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing. 2014.

[55] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *ACM International Conference on Supercomputing*, pages 35–44, 2002.

[56] S. S. Muchnick. *Advanced Compiler Design & Implementation*. Morgan Kaufmann, 2003.

[57] A. Muttreja, A. Raghunathan, S. Ravi, and N.K. Jha. Automated Energy/Performance Macromodeling of Embedded Software. In *IEEE/ACM Design Automation Conference*, pages 99–102, 2004.

[58] K. Nepal, O. Ulusel, I. Bahar, and S. Reda. Fast Multi-Objective Algorithmic Design Co-Exploration for FPGA-based Accelerators. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pages 65–68, 2012.

[59] L.S. Nielsen, C. Niessen, J. Sparso, and K. van Berkel. Low-power Operation using Self-timed Circuits and Adaptive Scaling of the Supply Voltage. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):391–397, 1994.

[60] Robert Petersen. *Design and Analysis of Experiments*. Mercel Dekker Inc., 1985.

[61] Roger G. Petersen. *Design and Analysis of Experiments*. Statistics Series. Taylor & Francis, 1985.

[62] P. Pillai and K.G. Shin. Real-time Dynamic Voltage Scaling for Low-power Embedded Operating Systems. In *ACM Symposium on Operating Systems Principles*, pages 89–102, 2001.

[63] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic Voltage Scaling on a Low-power Microprocessor. In *ACM International Conference on Mobile Computing and Networking*, pages 251–259, 2001.

[64] G. Qu, N. Kawabe, K. Usarni, and M. Potkonjak. Function-level Power Estimation Methodology for Microprocessors. In *IEEE/ACM Design Automation Conference*, pages 810–813, 2000.

[65] D. Rossi, C. Mucci, F. Campi, S. Spolzino, L. Vanzolini, H. Sahlbach, S. Whitty, R. Ernst, W. Putzke-Roming, and R. Guerrieri. Application Space Exploration of a Heterogeneous Run-Time Configurable Digital Signal Processor. *IEEE Transactions on Very Large Scale Integration Systems*, 21(2):193–205, Feb. 2013.

[66] M. Shafique, L. Bauer, and J. Henkel. enBudget: A Run-Time Adaptive Predictive Energy-Budgeting scheme for Energy-aware Motion Estimation in H.264/MPEG-4 AVC Video Encoder. In *IEEE/ACM Design, Automation Test in Europe*, pages 1725–1730, 2010.

[67] D. Sheldon, R. Kumar, R. Lysecky, F. Vahid, and D. Tullsen. Application-Specific Customization of Parameterized FPGA Soft-Core Processors. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 261–268, 2006.

[68] D. Sheldon and F. Vahid. Making Good Points: Application-specific Pareto-point Generation for Design Space Exploration using Statistical Methods. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 123–132, 2009.

[69] D. Shin and S.K. Gupta. Approximate Logic Synthesis for Error Tolerant Applications. In *IEEE/ACM Design, Automation and Test in Europe*, pages 957–960, 2010.

[70] L.C Sing and H. Yajun. Design Space Exploration for Arbitrary FPGA Architectures. In *IEEE International Conference on Embedded Software and Systems*, pages 269–275, 2005.

[71] A.M. Smith, S.J.E. Wilton, and J. Das. Wirelength Modeling for Homogeneous and Heterogeneous FPGA Architectural Development. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 181–190, 2009.

[72] Byoungro So, Mary W. Hall, and Pedro C. Diniz. A Compiler Approach to Fast Hardware Design Space Exploration in FPGA-based Systems. In *ACM Conference on Programming Language Design and Implementation*, pages 165–176, 2002.

[73] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *IEEE Transactions on Evolutionary Computation*, 2:221–248, 1994.

[74] T.K. Tan, A. Raghunathan, G. Lakshminarayana, and N.K. Jha. High-level Software Energy Macro-modeling. In *IEEE/ACM Design Automation Conference*, pages 605–610, 2001.

[75] K.H. Tsoi and W. Luk. Power Profiling and Optimization for Heterogeneous Multi-core Systems. *SIGARCH Computure Architecture News*, 39:8–13, 2011.

[76] O. Ulusel, K. Nepal, I. Bahar, and S. Reda. Fast Design Exploration for Performance, Power and Accuracy Tradeoffs in FPGA-Based Accelerators. *ACM Transactions on Reconfigurable Technoly and Systems*, 7(1):4:1–4:22, 2014.

[77] K. Usami and M. Horowitz. Clustered Voltage Scaling Technique for Low-power Design. In *ACM International Symposium on Low Power Design*, pages 3–8, 1995.

[78] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan. SALSA: Systematic Logic Synthesis of Approximate Circuits. In *IEEE/ACM Design Automation Conference*, pages 796–801, 2012.

[79] AK. Verma, P. Brisk, and P. Lenne. Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design. In *IEEE/ACM Design, Automation and Test in Europe*, pages 1250–1255, 2008.

[80] R. Weber, A. Gothandaraman, R.J. Hinde, and G.D. Peterson. Comparing Hardware Accelerators in Scientific Applications: A Case Study. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):58–68, 2011.

[81] W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen. Automatic Program Repair with Evolutionary Computation. *ACM Communications*, 53(5):109–116, 2010.

[82] M. Wu, Y. Sun, G. Wang, and J.R. Cavallaro. Implementation of a High Throughput 3GPP Turbo Decoder on GPU. *Journal of Signal Processing Systems*, 65(2):171–183, 2011.

[83] N. Zhu, W. Goh, and K. Yeo. An Enhanced Low-Power High-Speed Adder For Error-Tolerant Application. In *International Symposium on Integrated Circuits*, pages 69–72, 2009.

[84] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report, 2001.

[85] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999.