DESIGN METHODOLOGIES AND ARCHITECTURES OF HARDWARE-BASED EVOLUTIONARY ALGORITHMS FOR AEROSPACE OPTIMISATION APPLICATIONS ON FPGAS

Jonathan Kok

BEng(Elec)

Submitted in partial fulfilment of the requirements for the degree of

Doctor of Philosophy

Australian Research Centre for Aerospace Automation

Science and Engineering Faculty

Queensland University of Technology

Australia

June 2014

This page intentionally left blank.

Keywords

Evolutionary algorithm

Evolutionary computation

Field programmable gate array

Genetic algorithm

Heuristics

Multi-objective evolutionary algorithm

Multi-objective optimisation

Path planning

Travelling salesman problem

Unmanned aerial vehicle

This page intentionally left blank.

Abstract

The motivation for this thesis stems from the interest to address the computational time complexity of evolutionary computation techniques. Investigations into parallel computing concepts through digital hardware-based designs are carried out for improving computation run-time and meeting constraints of highly automated aircraft systems.

Evolutionary algorithm (EA) is an effective evolutionary computation technique that is widely used in many fields of research and development. Fundamentally, EA is a generic population-based metaheuristic optimisation algorithm that employs features inspired by biological evolution. The practical applications of EAs are limited by the heavy computational overhead that arises from the complexity of real-world scenarios, especially when applied to aerospace optimisation problems. EAs are therefore rarely used as an on-board optimisation method for unmanned aerial vehicles (UAVs) or highly automated aircraft systems where flight computer processor power is limited. A few of the common ways to address this issue is to simplify the optimisation problem, run an EA offline or use a compromised algorithm in place of an EA.

The key to realising the full potential of EAs lies in addressing the algorithm design from a lower level. Although EAs were originally designed and intended to run sequentially, they opportunistically have inherent parallelism potentials that are attributed to their population-based characteristics and the low dependency of

individuals in the population. One method for exploiting parallelism of an algorithm is by re-designing it for a hardware circuit implementation. Field programmable gate array (FPGA) is an integrated circuit device that is reprogrammable and allows for concurrent data processing. FPGA technology offers efficient extraction of parallelism through the flexibility of reconfigurable logic resources. Additionally, being compact in size, light in weight, and low in power consumption, FPGAs are ideal computing platforms for UAVs and highly automated aircraft systems where flight computers and processors have to adhere to strict size, weight, and power constraints.

The primary aim of this thesis is to provide knowledge that contributes to the design methodologies and architectures needed to directly map EAs on an FPGA hardware device. Furthermore, the knowledge discovered offers a greater degree of confidence concerning the effectiveness of developing and implementing FPGA-based EAs. One of the key challenges in designing an efficient FPGA-based architecture is the need to directly map the EA onto a hardware design without compromising the original algorithmic integrity, which is not straightforward. The outcomes of this research have produced design methodologies and architectures of hardware-based EAs for solving aerospace optimisation problems on FPGAs. This research investigation encompasses both FPGA-based single-objective and multi-objective EAs. The robustness and effectiveness of FPGA-based EAs have been demonstrated via evaluation across several practical aerospace optimisation applications, which exhibits different problem characteristics, such as path planning, travelling salesman problem, and multi-objective test function. Overall, the proposed FPGA-based EAs offer advantages including meeting physical constraints in aerospace applications and performance speedups without compromising the integrity of the evolutionary technique. This research is a step forward towards the advancement of efficient UAVs and highly automated aircraft systems.

Table of Contents

List of Figures

This page intentionally left blank.

List of Tables

This page intentionally left blank.

List of Abbreviations

AMGA2	archive-based micro-genetic algorithm 2
BFS	breadth-first search
CLB	configurable logic block
CPU	central processing unit
EA	evolutionary algorithm
FPGA	field programmable gate array
GA	genetic algorithm
HDL	hardware description language
IOB	input/output bank
LUT	lookup table
NSGA-II	non-dominated sorting genetic algorithm II
PEGA	parallel elite genetic algorithm
PROM	programmable read-only memory
RAM	random access memory
SPEA2	strength Pareto evolutionary algorithm 2
SRL	set and reset latch
TSP	travelling salesman problem
UAV	unmanned aerial vehicles

This page intentionally left blank.

Statement of Original Authorship

The work contained in this thesis has not been previously submitted to meet requirements for an award at this or any other higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signature: QUT Verified Signature

Date: 18/06/2014.

This page intentionally left blank.

Supervisory Team

Principal Supervisor

Dr Felipe Gonzalez felipe.gonzalez@qut.edu.au

Associate Supervisors

Dr Neil Kelson n.kelson@qut.edu.au

Dr Troy Bruggemann t.bruggemann@qut.edu.au

Prof Duncan Campbell da.campbell@qut.edu.au

This page intentionally left blank.

Acknowledgements

Administrative, office resources and research support used in this work were provided by the Australian Research Centre for Aerospace Automation, Queensland University of Technology (QUT), Brisbane Airport, Australia. Computational resources and services used in this work were provided by the HPC and Research Support Group, QUT, Brisbane, Australia.

This work has been supported by the Cooperative Research Centre (CRC) for Spatial Information, whose activities are founded by the Australian Commonwealth's Cooperative Research Centres Programme. This work was financially supported by the Australian Government International Postgraduate Research Scholarship, the QUT Vice Chancellors Initiative Scholarship, the QUT Postgraduate Research Award, and the CRC for Spatial Information Top-up Scholarship.

This page intentionally left blank.

List of Publications

Journal Article

J. Kok, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Transactions on Evolutionary Computation*, vol. 17, no.2, pp. 272–281, April 2013.

J. Kok, T. S. Bruggemann, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of a micro-genetic algorithm for solving the travelling salesman problem," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2013 (Submitted).

J. Kok, L. F. Gonzalez, and N. A. Kelson, "Multi-objective evolutionary algorithm using FPGA-based pipelining and parallel architecture: design, test, and analysis," *IEEE Transactions on Cybernetics*, 2013 (Submitted).

Conference Paper

J. Kok, T. S. Bruggemann, and L. F. Gonzalez, "An evolutionary computation approach to three-dimensional path planning for unmanned aerial vehicles with tactical and kinematic constraints," in *Proceedings of the 15th Australian International Aerospace Congress (AIAC15)*, 2013, pp. 1–9.

J. Kok, N. A. Kelson, L. F. Gonzalez, and T. S. Bruggemann, "Computational experiments involving population size for FPGA-based implementation of a GA for the TSP," in *Proceedings of the 4th International Conference on Computational Methods (ICCM 2012)*, 2012, pp. 1–6.

G. Rappa, L. F. Gonzalez, J. Kok, and F. Quagliotti, "Hybrid game evolutionary algorithm for mission path planning of aerial survey tasks," in *Proceedings of the 28th International Congress of the Aeronautical Sciences (ICAS 2012)*, 2012, pp. 1–13.

J. Kok, L. F. Gonzalez, N. A. Kelson, and J. Periaux, "An FPGA-based approach to multi-objective evolutionary algorithm for multi-disciplinary design optimisation," in *Proceedings of the 2011 Evolutionary and Deterministic Methods for Design, Optimization and Control (Eurogen 2011)*, 2011, pp. 1–10.

J. Kok, L. F. Gonzalez, N. A. Kelson, and T. Gurnett, "A hardware-based multi-disciplinary design optimization method for aeronautical applications," in *Proceedings of The 4th International Conference on Computational Methods for Coupled Problems in Science and Engineering (Coupled 2011)*, 2011, pp. 1–11.

J. Kok, L. F. Gonzalez, R. A. Walker, T. Gurnett, and N. A. Kelson, "A synthesizable hardware evolutionary algorithm design for unmanned aerial system real-time path planning," in *Proceedings of the 2010 Australasian Conference on Robotics and Automation (ACRA 2010)*, 2010, pp. 1–8.

Conference Item

D. Warne, N. A. Kelson, J. Kok, T. Gurnett, and U. Rueckert, "Experiences with implementing common mathematical operations using field programmable gate arrays," in *Proceedings of the 16th Biennial Computational Techniques and Applications Conference (CTAC 2012)*, 2012, pp. 23–26.

This page intentionally left blank.

Chapter 1

Introduction

1.1 Background

The motivation for this research stems from the interest to address the computational time complexity of evolutionary computation techniques that are effectual but can be computationally intensive. Evolutionary computation techniques such as evolutionary algorithms (EAs) are capable of solving for optimality and are often used to find excellent solutions within acceptable time [1]. EA is a class of population-based metaheuristic optimisation algorithm that uses principles inspired by biological evolution. EAs have been proposed for solving aerospace optimisation problems, which recently include missile guidance control [2], missile fire distribution [3], airline route networks management [4], airline boarding process [5], flight control systems [6], flight navigation planning [7], compound helicopter design [8], aerofoil design [9], and cooperative unmanned aerial vehicles tasking [10].

The theoretical framework of EAs, specifically the genetic algorithm (GA), was formalised and popularised through the early works of John Holland [11]. His pioneering

theory was supported with theoretical and experimental results [12]. It was not until the late 1980s that dramatic increase in CPU computational power allowed for the practical application of this new evolutionary computation technique on desktop computers [13]. The subsequent decades of improved computational power opened the door to further research and establishment of EAs by many others, such as David B. Fogel, David E. Goldberg, Zbigniew Michalewicz, Melanie Mitchell and David J. Schaefer who are now significant contributors to the academic community. The success of EAs is evident in the ever-growing number of EA-based software solvers, academic courses, journals, conferences, books and applications [14]. Today, a simple search of scholarly materials on the Internet using Google's search engine with the terms "evolutionary algorithms" returns over 3,000,000 results.

The solving of complex modelled problems using software-based EAs can result in a time consuming process as the computation process is executed sequentially. Hence, the application of EAs on optimisation models of real-world problems can be computationally intensive. Although EAs were originally designed and intended to run sequentially, they opportunistically have inherent parallelism potentials that are attributed to the population-based characteristics and the low dependency of individuals in the population [15]. The parallelism potentials of EAs allow for the reduction of computational time by means of parallel processing through hardware implementation methods [16].

Although not a new parallel computing concept, recent advancements in field-programmable gate array (FPGA) technology have made it an increasingly attractive means to implement complex digital computations [17]. Unlike an application-specific integrated circuit that is manufactured for a particular use, an FPGA is a reprogrammable integrated circuit device that is manufactured to be reconfigurable for general-purpose

uses [18]. The recent increase in FPGA density, capabilities and speed have allowed for the implementation of larger functions and even full system on chips in which can be advantageous within the aerospace industry. Additionally, FPGAs being compact in size, light in weight and low in power consumption, are well suited for unmanned aerial vehicles (UAVs) applications and highly automated systems where flight computers and processors have to adhere to strict size, weight and power constraints [19].

1.2 Research Problem

The implementation of EAs and other algorithms on FPGA is an active area of research (see Section2.2.3). Past design methodologies and architectures of EAs on FPGA were traditionally approached by the proposal of reengineered EA architectures to suit FPGA logic level implementations. This reengineering methodology was appropriate then, as the logic density of FPGAs in the past did not allow for an entire EA to be directly mapped onto them. However, the notion of reengineering methodology for the hardware implementation of EAs does not genuinely represent the effective building blocks of EAs.

Past FPGA-based implementations of EAs may share some common EA features, but the differences from the original algorithmic structure can have a significant impact on the effectiveness that has been asserted by original EA literature. Most of past implementations with respect to FPGA-based EAs were vague, inconclusive and lacking in detail, as will be discussed in the literature review (see Chapter 2). The integrity of previous works is questionable and hence may exhibit behavioural uncertainty when such designs are applied to larger and more complex real-world applications. Furthermore, without clear and strong foundations of FPGA-based EA literature, the

degree by which current and future researchers can extend and build upon in the area of FPGA-based EAs is limited. This thesis investigates the efficient mapping of EA type of algorithms onto reconfigurable hardware platforms to optimise performance and hardware cost.

1.3 Aims and Objectives

The effectiveness of EAs is well recognised in the fields of both single-objective and multi-objective optimisation [20]. In recent years, advancements in FPGA technology have enabled the manufacturing of high-density FPGAs allowing its use for hardware accelerating EAs. The population-based characteristic of EAs allows potential levels of parallelism that FPGA technology can exploit for concurrent processing. However, the mapping of complex software algorithms, such as EAs, to hardware is not straightforward [21].

The primary aim of this thesis is to provide knowledge that contributes to the theory and application of effective FPGA-based EA architectures. The knowledge discovered offers a greater degree of confidence concerning the effectiveness of developing and implementing EAs on FPGA hardware devices. In this thesis, FPGA implementation methods of single and multiple objective EAs for different types of optimisation problems from aerospace applications are investigated. In particular, the consideration of preserving algorithmic integrity of the original EA is taken into account. To achieve this aim, five objectives are identified as follows:

1. To design and develop FPGA-based EA architectures for solving

   (a) single objective optimisation, and

(b) multi-objective optimisation.

2. To investigate the hardware implications of different EA parameters.

3. To implement the proposed design methodologies and architectures on aerospace optimisation applications, namely,

   (a) path planning problem,

   (b) the travelling salesman problem, and

   (c) multi-objective test function.

4. To evaluate the performance of hardware implementations over software counterparts.

1.4 Scope

This thesis explores the hardware implementation of EAs using FPGA technology. The type of EAs being investigated are limited to the GA and its variations, such as the micro-GA and NSGA-II, because of its simplistic yet effective evolution strategy [22]. The assessments of problems are scoped to encompass only single-objective path planning, single-objective TSP, and multi-objective test function. The hardware programming is coded using VHDL (VHSIC Hardware Description Language). The FPGA devices used for development are limited to those manufactured by Xilinx [1]. The Xilinx ISE design environment was used for simulation and synthesis of the FPGA designs. The experiments were limited to PC and hardware-in-the-loop simulations. Care was taken to provide fair comparisons between hardware and software implementations.

[1] Xilinx company website: www.xilinx.com

No program optimisation (e.g., multi-threading) was done to modify the hardware or software implementations to make any of them work more efficiently.

1.5 Significance

Proof of the significance of this research is the publication of produced research papers by peer-reviewed high impact factor journals and international conferences. The FPGA-based EA design methodologies and architectures generated from this research not only speedup the EA optimisation process but also assert higher confidence in their application to real-world problems.

In the aerospace community, UAVs and highly autonomous flight systems are increasing being deployed for applications where land-based approach may be unsafe, unviable or uneconomical, such as infrastructure survey, disaster assessment, and high-precision terrain mapping. The global UAV market alone is estimated to worth just over USD$89 billion in the next ten years [23]. The on-board flight systems are often faced with high processing needs that are limited by the computing capabilities of their on-board computer. The physical attributes of FPGAs being compact in size, light in weight, and low in power consumption, along with reconfigurable and real-time processing capabilities make FPGA technology well suited for these applications. Thus, design methodologies and architectures for FPGA-based solutions will significantly contribute to the knowledge field and effective deployment of FPGA-based flight systems.

This research is also significant to the high-stakes industry of finance and economics, where EAs are used to rapidly analyse and evaluate large quantity of market data [24]. The development of FPGA-based EAs that are significantly faster than its original software equivalent will play a significant role in stock market trading where money

is made and lost in milliseconds. In the academic and scientific community, this investigation also provides researchers and operators with useful design and evaluation insights in the overlapping fields of hardware design, metaheuristics, and optimisation problems.

1.6 Account of Research Progress

This section details the list of research papers authored during the PhD candidature period and describes the linkage between them. Individual research papers are presented in the body of this thesis as standalone chapters and are included "as submitted or published" without editing as per QUT thesis by published papers guidelines. All research papers are formatted as that of a data paper containing an introduction, methodology, results, and discussion sections.

1.6.1 Linkage of Research Papers

The linkage of research papers in this thesis is illustrated in Figure 1.1. The account of research progress is divided into two studies: single-objective and multi-objective optimisations. For both studies, the effectiveness of evolutionary algorithms is established and assessments of FPGA applications to optimisation problems are evaluated. The FPGA implementation methodologies proposed are built upon the basic fundamentals of the EA algorithmic framework. Hence, evaluating the applications of implementing EAs on FPGAs are the common topics throughout the research papers.

The first study, which involves single-objective optimisation, investigates the effectiveness of FPGA-based EAs for solving the path planning problem and TSP. Chapter 3 and Chapter 4 propose two novel hardware implementation methods of a modified

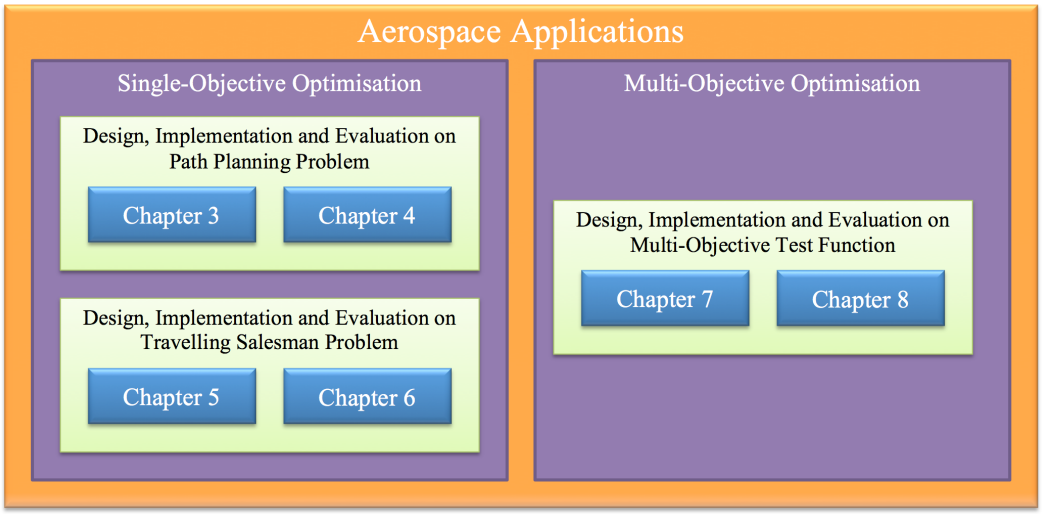


Figure 1.1: Linkage of research papers.

EA for three-dimensional flight path planning. The initially proposed architecture is presented in Chapter 3 and an improved architecture is subsequently presented in Chapter 4. Results from the first architecture showed that the proposed FPGA-based EA achieved speedup of up to 52,000 times faster than its software equivalent. Results from the second architecture demonstrated the performance of the FPGA-based EA to meet the 10 Hz update frequency of a typical autopilot system.

Chapter 5 investigates the effects population sizes of EA have on FPGA resource utilisation and solution quality for solving the TSPs. Results indicated that a GA with small population size is sufficient to obtain quality solutions, thereby permitting relative resource efficient hardware implementations on FPGAs. With the findings, the FPGA implementation of micro-GA for solving the TSP is proposed in Chapter 6. New hardware-based implementations of two popular genetic operators for combinatorial problems are also proposed. Results show that the FPGA-based approach achieved speedups averaging 70 times faster when compared to an equivalent software version

and 26 times faster when compared to the powerful Concorde TSP solver.

The second study, which involves multi-objective optimisation, investigates the effectiveness of FPGA-based multi-objective EAs on multi-objective engineering problems. FPGA implementation methodologies of the widely used NSGA-II [25] for engineering problems are presented in Chapter 7 and an improved architecture with pipelining features is presented later in Chapter 8. Results from both architectures showed that the FPGA-based implementations achieved speedup of approximately 1,300 times and 1,987 times faster than the software versions, respectively.

Table 1.1 illustrates the summary of objectives (see Section 1.3) met with respect to each body chapter of this thesis.

Table 1.1: Summary of objectives met with respect to each body chapter of this thesis, where each chapter consist of a single published or submitted research paper.

Research objectives	**Chapter (research paper) in which respective objectives are met**					
	3	4	5	6	7	8
1. (a)	✓	✓	✓	✓		
1. (b)					✓	✓
2.			✓			
3. (a)	✓	✓				
3. (b)			✓	✓		
3. (c)					✓	✓
4.	✓	✓		✓	✓	✓

1.6.2 List of Research Papers

The full citation details of the research papers contributed by this thesis are listed in Table 1.2.

Table 1.2: List of research papers.

Chapter	Citation
Chapter 3	J. Kok, L. F. Gonzalez, R. A. Walker, T. Gurnett, and N. A. Kelson, "A synthesizable hardware evolutionary algorithm design for unmanned aerial system real-time path planning," in *Proceedings of the 2010 Australasian Conference on Robotics and Automation (ACRA 2010)*, 2010, pp. 1–8.
Chapter 4	J. Kok, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Transactions on Evolutionary Computation* [1], vol. 17, no.2, pp. 272–281, April 2013.
Chapter 5	J. Kok, N. A. Kelson, L. F. Gonzalez, and T. S. Bruggemann, "Computational experiments involving population size for FPGA-based implementation of a GA for the TSP," in *Proceedings of the 4th International Conference on Computational Methods (ICCM 2012)*, 2012, pp. 1–6.
Chapter 6	J. Kok, T. S. Bruggemann, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of a micro-genetic algorithm for solving the travelling salesman problem," *IEEE Transactions on Circuits and Systems I: Regular Papers* [2] , 2013 (Submitted).
	Continued on next page

[1] JCR impact factor = 4.81, Eigenfactor^{TM}score = 0.00837, Article Influence^{TM}score = 1.768
[2] JCR impact factor = 2.24, Eigenfactor^{TM}score = 0.0254, Article Influence^{TM}score = 1.074

Table 1.2 – continued from previous page

Chapter	Citation
Chapter 7	J. Kok, L. F. Gonzalez, N. A. Kelson, and J. Periaux, "An FPGA-based approach to multi-objective evolutionary algorithm for multi-disciplinary design optimisation," in *Proceedings of the 2011 Evolutionary and Deterministic Methods for Design, Optimization and Control (Eurogen 2011)*, 2011, pp. 1–10.
Chapter 8	J. Kok, L. F. Gonzalez, and N. A. Kelson, "Multi-objective evolutionary algorithm using FPGA-based pipelining and parallel architecture: design, test, and analysis," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* [1], 2013 (Submitted).

[1] JCR impact factor = 3.236, Eigenfactor™ score = 0.01749, Article Influence™ score = 1.446

This page intentionally left blank.

Chapter 2

Literature Review

This thesis evaluates the application of FPGA-based EAs over single-objective and multi-objective optimisation problems. This chapter provides a critical review of relevant literature and highlights the knowledge gaps being addressed. Furthermore, a self-contained literature review relative to each investigation is included as part of each respective research paper.

2.1 Optimisation Theory

Optimisation is implicitly considered in decision making everywhere; human nature is subtly inclined to covet for the best result with the least amount of effort. In an industrial context, the need for optimisation becomes explicit, where optimal solutions can result in minimum fabrication cost, maximum profit margin, and other benefits. Significant outcomes from optimal solutions elevate the importance of optimisation algorithms across many domains, such as science, engineering, management and business.

The general formulation of an optimisation problem can be represented in the

following form:

$$\left.\begin{array}{lll} \text{Minimise/Maximise} & f(\mathbf{x}), & \\ \text{subjected to} & g_j(\mathbf{x}), & j = 1, 2, \ldots, J; \\ & x_n^{(L)} \leq x_n \leq x_n^{(U)}, & n = 1, 2, \ldots, N. \end{array}\right\} \tag{2.1}$$

where f, are the objective functions to be minimised or maximised, $\mathbf{x} = (x_n, \ldots, x_N)$, $n \in \{1, \ldots, N\}$, is the "optimisation vector" of N design variables that are individually restricted by lower $x_n^{(L)}$ and upper $x_n^{(U)}$ bounds, and g_j, $j \in \{1, \ldots, J\}$, are constraint functions.

Optimisation problems can be categorised into two classes: *single-objective optimisation*, and *multi-objective optimisation*. The former relates to optimisation problems involving only one objective function, whereas the latter to more than one objective function. It is noteworthy that most real-world optimisation problems are generally subjected to multiple objectives. The steps commonly involved in an optimisation process for a given optimisation problem is shown in Figure 2.1. The typical optimisation process for a given problem is described as follows: a *problem formulation* defines the selection of decision variables, objectives, and constraints into an *optimisation model* in which is processed by an *optimisation algorithm* to obtain a set of *optimised solutions*. A breakdown of the *optimisation model* and *optimisation algorithm* are described in the following sub-subsections.

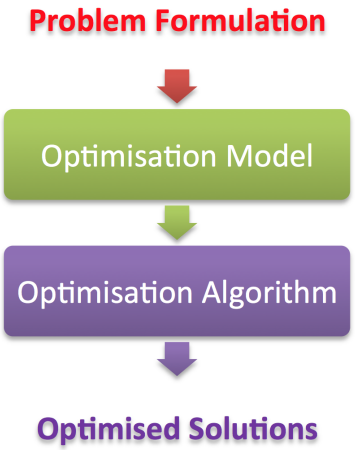


Figure 2.1: The optimisation process in obtaining a set of optimised solutions.

2.1.1 Optimisation Model

The different families of optimisation models that are typically used for problem formulation are illustrated in Figure 2.2. This partitioning is not exclusive and different classes can be possibly overlapping. *Mathematical programming* models problems using mathematical concepts and languages [26]. *Combinatorial optimisation* is class of optimisation problems characterised by discrete decision variables bounded by a finite search space, although discreteness is not obligatory for the objective functions and constraints [27]. One of the most popular combinatorial optimisation problem is the travelling salesman problem (TSP) [28]. *Constraint programming* is a generic class of optimisation problems approached by mathematically constraining the decision variables [29]. The three aforementioned optimisation models are *analytical*, this means that an explicit mathematical formulation is derivable from the given optimisation problem. In cases when the optimisation problem is *non-analytical*, the evaluation will depend on *physical* or *simulation* models [30].

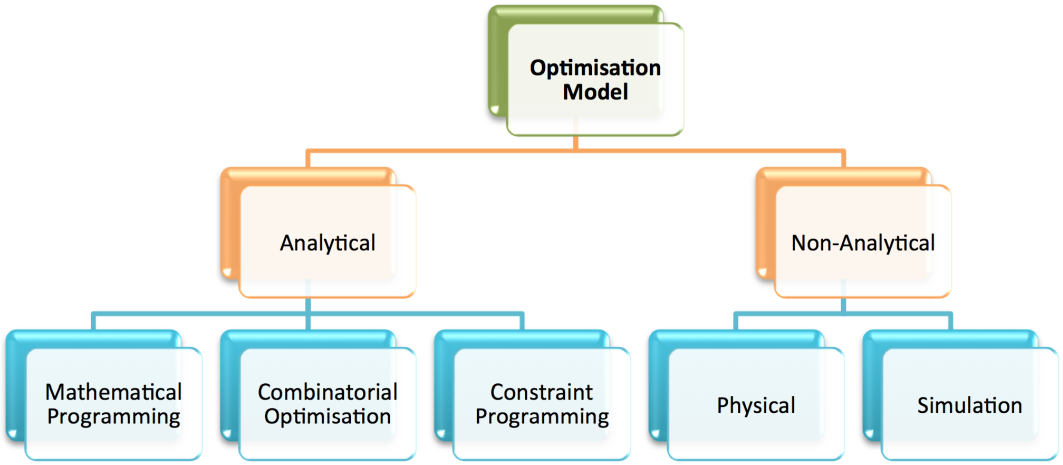


Figure 2.2: The different families of optimisation models that optimisation problems can be formulated by.

2.1.2 Optimisation Algorithm

One classification of optimisation algorithms for solving optimisation problems is depicted in Figure 2.3. *Iterative* algorithms explore attractive regions of the search space by mathematical procedures that gradually improve the solutions, such examples include sequential quadratic programming [31], Jacobi method [32], conjugate gradient methods [33], and generalised minimal residual method [34]. *Heuristic* algorithms effectively approach large and complex optimisation problems by means of iterative experience-based improvement techniques, however optimality of solutions are without guarantee. Heuristic algorithms can be divided into two families: *problem-specific heuristic* and *metaheuristic*. Problem-specific heuristics are customised for a specific problem case, whereas metaheuristics are tailored for general-purpose application.

One of the many classifications for metaheuristics is *individual-based metaheuristics* versus *population-based metaheuristics*. Individual-based metaheuristics (e.g., local search [35], simulated annealing [36], tabu search [37]) intensify a single solu-

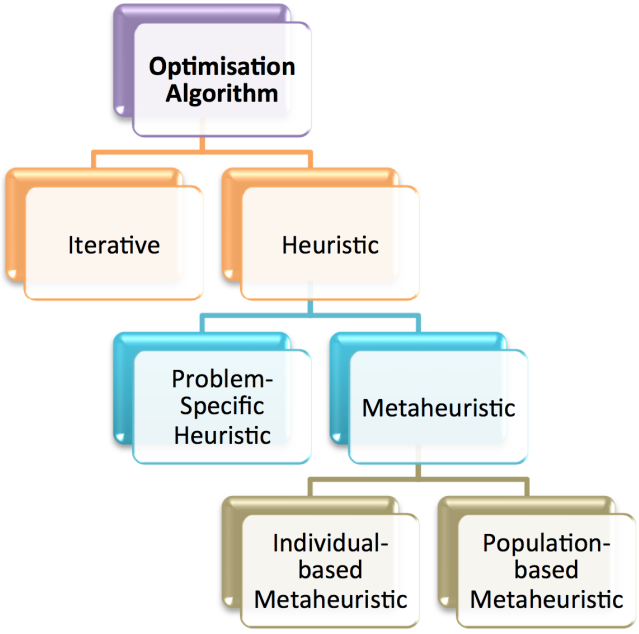


Figure 2.3: The different classifications of optimisation algorithms for solving optimisation problems.

tion during the optimisation process while in population-based metaheuristics (e.g., evolutionary algorithms [38], ant colony optimisation [39], particle swarm optimisation [40]) a population of solutions is maintained for exploration and exploitation. This research particularly considers the application of the widely used EA population-based metaheuristics.

2.2 FPGA Technology

An FPGA is a reprogrammable semiconductor device that is designed to be reconfigurable after development by the system designer or consumer. FPGA technology continues to gain momentum since its invention by Xilinx in 1984; from initially utilised as simple glue logic to modern applications for high performance embedded computing.

The success of FPGA technology is attributable to technological capabilities and advantages, such as pipelining, parallelism, dynamic reconfigurability, and rapid prototyping development cycle.

The basic FPGA architecture is depicted in Figure 2.4. The matrix of configurable logic blocks (CLBs) can be individually configured as a set and reset latch (SRL), random access memory (RAM), or lookup table (LUT). The input/output banks (IOBs) provide an interface bridge into the internal system. The design software, such as the Xilinx Vivado Design Suite, interconnects signals between CLBs and IOBs to create a custom designed system on the FPGA device. The number of logic gates in FPGAs has been increasing since the 1980s, from thousands, to ten thousands, to hundred thousands, and now millions. It is currently possible to implement larger and more complex algorithms onto FPGAs, such as fast Fourier transforms [41].

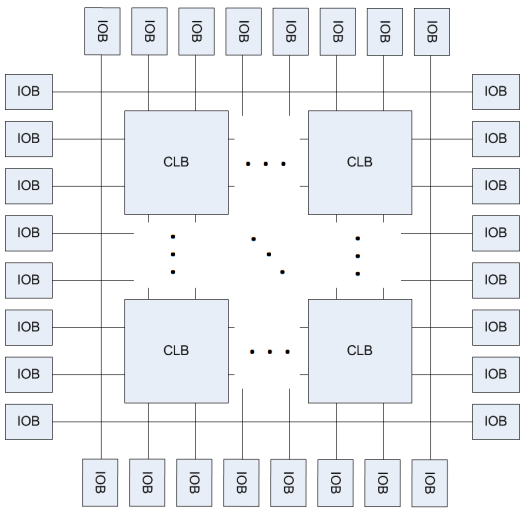


Figure 2.4: A basic FPGA device architecture.

2.2.1 FPGA Design Flow

The standard FPGA design flow, as shown in Figure 2.5, comprises the following steps: design entry, design synthesis, design implementation, and download to FPGA device. Design verification techniques are carried out across different stages of the design flow. This techniques include functional simulation, timing analysis, and in circuit verification.

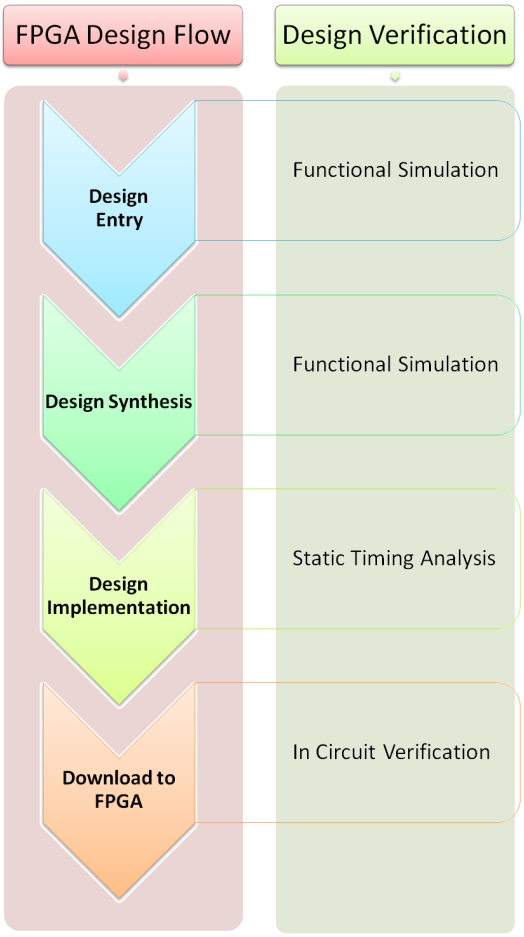


Figure 2.5: The standard FPGA design flow.

Design Entry and Synthesis: A complete description of the intended behaviour of the system to be developed is created by using a vendor-supported schematic editor, a hardware description language (HDL), or both. This design entry must be synthesized for the targeted FPGA device. The synthesis process translates behavioural information and user constraints into a structural netlist for the specified FPGA device. The functionality is verified through simulation before the design is implemented. The simulation environment is not running in real-time rather the behaviour of internal signals over each time period are computed and simulated.

Design Implementation and Download: This step maps the logical design to the targeted FPGA device. The mapped circuit design is subsequently placed and routed. A configuration bitstream (a binary file with .BIT extension), is generated for the FPGA device configuration using the placed and routed information. The generated BIT file can either be downloaded into a FPGA device or formatted into a programmable read-only memory (PROM) file for storage on non-volatile memory.

2.2.2 Properties for Efficient FPGA Implementations

The design of an efficient dedicated hardware computing architecture, such as an FPGA, requires the forethought of three implementation properties: parallelisation, parameterisation, and communication. These implementation properties are presented in Figure 2.6.

Parallelisation: Parallelisation can be extended to a optimisation *algorithm* level, which includes multiple optimisation algorithms processing independently or cooperatively; an *iteration* level, which handles the population of solutions concurrently; and

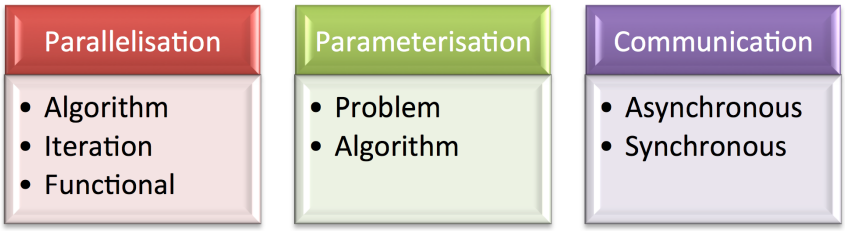


Figure 2.6: Properties for efficient implementation of hardware designs.

functional level, which synchronises the execution of partitioned functions and data.

Parameterisation: Finite resources on a given FPGA device limits the allowed variable bits for the setting of parameters in both the *problem* and *algorithm*. In regards to designing FPGA-based EAs for optimisation problems, the parameters that depend on the problem include decision variables and boundaries of each variable while parameters for the EAs include population size and solution resolution.

Communication: The communication level determines the pipelining throughput of the overall system. Sharing of information (e.g., elites, convergence rate, extremes) and operations between internal modules can be either one stage at a time or multiple stages occurring simultaneously, *asynchronous* or *synchronous* respectively [17]. Synchronous communication produces higher throughput but at the expense of design complexity.

Algorithm design for an FPGA implementation starts with the deliberate consideration of such implementation properties. The justification of utilising different sets of properties is dependent on the overall system requirement for the given application, such as algorithm complexity, problem constraints, and resource available.

2.2.3 Algorithm acceleration using FPGAs

The notion of exploiting advantages of FPGA technology for algorithm acceleration is not new. In fact, FPGA technology has matured to the point where they can be effectively applied to complex processing tasks. A list of recently proposed FPGA-based algorithms and their corresponding researchers is shown in Table 2.1. This list is not claimed to be complete but rather illustrates the different varieties of applied research on algorithm acceleration using FPGA technology over the recent five years (2009 to 2013). The list also highlights that FPGA technology has been and still is an active area of ongoing research for algorithm design and development.

Table 2.1: A list of proposed FPGA accelerated algorithms.

Year	Researcher(s)	Algorithm	Reference
2013	K. Anumandla et al.	Differential evolution algorithm	[42]
2013	C. Grigorios et al.	Classification and regression tree algorithm	[43]
2013	S. Pan et al.	Artificial neural network	[44]
2013	S. Cruz et al.	Extended kalman filter algorithm	[45]
2013	X. Liuet al.	Multi-target tracking algorithm	[46]
2012	S. Li et al.	Ant colony optimisation algorithm	[47]
2012	K. Rahimunnisa et al.	Advanced encryption standard algorithm	[48]

Continued on next page

Table 2.1 – continued from previous page

Year	Researcher(s)	Algorithm	Reference
2012	G. Mingas et al.	Scan-matching genetic simultaneous localisation and mapping algorithm	[49]
2012	C. Gonzalez et al.	N-finder algorithm	[50]
2012	C. Gonzalez et al.	Hyperspectral image processing algorithm	[51]
2011	C. Le Lann et al.	Krawczyk algorithm	[52]
2011	N. G. Johnson-Williams et al.	Three dimensional positioning algorithm	[53]
2011	B. Abhishek et al.	Image watermarking algorithm	[54]
2011	S. Dikmese et al.	Beamforming algorithm	[55]
2011	Z. Ding et al.	Parallel transitive closure algorithm	[56]
2010	Z. Xinyi	Modified Goertzel algorithm	[57]
2010	Y. Wang et al.	MD5 hash algorithm	[58]
2010	A. Dinu et al.	Direct neural-network	[59]
2010	A. Annovi and M. Beretta	Clustering algorithm	[60]
2009	A. Jain et al.	Protein structure prediction algorithm	[61]
2009	M. A. Ibarra-Manzano et al.	Stereo vision algorithm	[62]

Continued on next page

Table 2.1 – continued from previous page

Year	Researcher(s)	Algorithm	Reference
2009	C. Claus et al.	Corner detection algorithm	[63]
2009	R. Maes et al.	Helper data algorithm	[64]
2009	J. Cho et al.	Face detection algorithm	[65]

2.3 Single-Objective Optimisation

Single-objective optimisation problem is straightforward where there is only one objective function to optimise with only one optimal solution to be obtained.

2.3.1 EA for Single-Objective Optimisation

In artificial intelligence, EA is a generic population-based metaheuristic optimisation algorithm associated to the field of evolutionary computation. The fundamentals of EA use mechanisms inspired by biological evolution: *selection*, *reproduction*, and *replacement*. A pseudo code describing the working principle of an EA is illustrated in Figure 2.7. A brief description of each operational step through the evolution process of an EA is as follows.

Initialise Population: A *population* refers to a set of candidate solutions. The first generation of population is initialised randomly as the generality of EAs does not require *a priori* knowledge of the optimisation problem.

```
Evolutionary Algorithm()
    Initialise Population()
    WHILE (Termination Criteria NOT MET)
        Selection()
        Reproduction()
        Replacement()
    END
```

Figure 2.7: Pseudo code of an evolutionary algorithm.

Selection: The objective of the *selection* operator is to institute a selection pressure in which fitter solutions in a population have better chance of survival as evolution progresses. Some reputable methods include tournament selection, roulette wheel selection, ranking selection, and truncation selection [66, 67].

Reproduction: The objective of the *reproduction* operator is to genetically improve the population of solutions through exploitation and exploration of the search space by means of crossover operation and mutation operation, respectively. The crossover operation exchanges portions of information to create possibly better solutions from shared traits, whereas the mutation operation randomly alters portions of a candidate solution to investigate other regions of the search space. A range of reproduction operators can be found in [68].

Replacement: The objective of the *replacement* operator is to establish an update scheme for the offspring population with the parent population. There are three fundamental replacement schemes commonly employed: generational replacement, in which offspring population overwrites all of parent population; environmental replacement, in which worst solutions are deleted incrementally until population reaches a predefined minimum size; and elitist replacement, in which best parent solutions are preserved [69].

Termination: The evolution process is repeated until termination criteria are met. Common criteria used are maximum generation, desired solution, maximum computational run-time, convergence plateau, or any combinations of the above. The generation count is incremented each time the termination does its check.

These biological mechanisms of evolution are the fundamental building blocks for EAs. The candidate solutions of an EA population are rendered independently within the evolution processes, which allows for the exploitation of iteration level parallelism. The underlying binary nature of the evolution-based mechanisms is complementary for implementing fast primitive bitwise operations. These two key advantageous features of EAs make them well suited for hardware-based algorithm implementations via FPGA technology.

2.3.2 FPGA-based EA

It is noteworthy to mention that it is important not to associate the methodological use of FPGAs to develop efficient EAs with the research field of *evolvable hardware* where EAs are used for evolving a combinational circuit on an FPGA to efficiently configure specialised architectures without manual engineering [70]. Implementation of EAs on FPGAs have been studied by a number of researchers, including [71–82]. Their works proposed generic FPGA-based EAs highlighting the effectiveness of hardware designed EAs, however they cannot be directly used for path planning or TSP applications. The following sub-subsections present a review of literature with regards to FPGA-based implementations for path planning and TSP applications. The main findings of the literatures reviewed are also outlined.

2.3.2.1 Path Planning

Path planning can be defined as the framework employed to determine the discrete motions for a vehicle traversing from one location to another [83]. Influence of FPGA technology in path planning applications is recent and limited [84–91].

Girau et al. [84] used an FPGA implementation to compute approximated harmonic control functions [92] for making robot navigation decisions. The use of harmonic functions ensures that generated trajectories avoid local optima in cluttered and concaved environments [93]. They argued that the main advantage of their work was not the speedup, as real-time computational speed can be easily reached by software coded harmonic control functions. Instead, they highlighted the potential of embedding FPGAs for online processing needs on low powered mobile robots.

Vacariu et al. [85] proposed an FPGA implementation of a simple breadth-first search (BFS) algorithm [94] applied on a static two-dimensional discrete environment. The BFS technique, which has two degrees of freedom, is based on maintaining a queue of all accessible neighbours, whereby a sequence of directions leading to the goal point is acquired. This search algorithm is complete, that is to say it will find a solution if one exists, but does not guarantee any level of optimality or feasibility. Their results show execution times sped up by factors of hundreds.

Sudha and Mohan [86] designed an FPGA-accelerated path planner based on the Euclidean distance transform [95] of a captured image from an overhead camera. Priya et al. [87] customised an FPGA architecture for path planning based on revised simplex method [96] applied on a pre-constructed visibility graph [97]. Sudha and Mohan argued that their path planning solution is complete as contrasted to Priya et al.work, as a raw binary image of the environment is directly processed in the hardware rather

than a meta-modelled nodes-and-edges visibility graph. On the other hand, results from Priya et al.were in factors of microseconds as compared to milliseconds shown in the results from Sudha and Mohan. The time response were of relative importance as the work by Priya et al.was focused on ballistic missile application, whereas the work by Sudha and Mohan was directed towards ground-based robot navigation.

Hachour [88] proposed an FPGA-based path planning GA for ground-based mobile robots. However, the path planning GA concept and the actual hardware implementation were not described in any detail. Moreover, the results and validated functionality were not reported.

Alliare et al. [89] demonstrated that the possibility of increased UAV navigation autonomy can be achieved by instantaneous on-the-fly replanning via the implementation of a path planning GA on an FPGA for algorithm acceleration. They argued that GA produces higher quality solutions as compared to deterministic algorithms but are disadvantaged due to their extensive computational overhead that is inevitably inherited by the GA's population-based metaheuristics optimisation approach. Their path planning GA implementation details were partially set according to Cocaud's [98] work involving a customised GA specifically for UAV task allocation and path generation. Their results indicate that some mechanisms of the GA running on an FPGA can be sped up by factors of thousands. However, their research was limited to co-simulation between a computer and an FPGA running simultaneously and exchanging information in a collaborative manner.

Huang et al. [90] proposed a hardware/software co-designed parallel elite genetic algorithm (PEGA) for ground mobile robot path planning in a static environment. Their FPGA-based PEGA architecture consists of two path planning GAs [99] of which the evolutionary-influenced selection, crossover, and mutation modules operate

concurrently. Elitism is preserved via a migration module that periodically exchanges the corresponding two best members into the selection pool after a predefined number of generations. However, the computationally dominant fitness evaluation function was executed sequentially on an embedded processor.

Schmidt and Fey [91] implemented marching pixels algorithm [100] applied to a skeleton map [101] on an FPGA device for path planning. The marching pixels algorithm is akin to artificial ants behaving as modelled by cellular automata, where one pixel of an image is represented by a two-dimensional coordinate on the map. Their results show computational effectiveness in factors of milliseconds for image resolutions up to 1024×800. However, their approach inherits the two main disadvantages from the nature of skeleton map sampling: the resulting path solution is non-optimal and sharp turning edges are likely to occur.

2.3.2.2 TSP

In the field of combinatorial optimisation, the TSP has been intensively investigated [102]. The TSP has its uses in a wide range of practical applications, such as manufacturing of microchips [103], cold rolling scheduling [104], automation of disassembly system [105], and routing of shop floor logistics [106]. TSP type problems are common in aerospace optimisation, such examples include the design of global navigation satellite system survey networks and optimal routing of unmanned aerial vehicle applications [107]. Formally stated, the objective of the TSP is to find the Hamiltonian cycle with the least weight in a complete weighted graph (where vertices, edges and weights represent cities, roads and distance of roads, respectively).

In terms of computational complexity theory, the TSP belongs to the class of NP-complete problems such that an optimal solution for even moderate sized problems can

be intractable to solve [108]. Hence, heuristics have been proposed for yielding good solutions within reasonable time. Heuristics such as EAs explore and exploit random regions of the search space for increasingly better solutions instead of performing an extensive analysis on all possible solutions. A comprehensive survey of the TSP in conjunction with exact algorithms and heuristics for solving it can be found in [109], while a detailed literature review of chromosome representation and genetic operators for EAs can be found in [110]. A type of EA known as GA has been argued to be one of the best available heuristics for solving the TSP [111]. In view of the potential advantages, several studies have explored the application of GAs for solving the TSP problem using FPGAs [112–118].

Very early work by Graham and Nelson [112] attempted a hardware implementation of a GA for solving the TSP on the Splash reconfigurable computing platform. However, due to the limited reconfigurable logic resources on FPGAs at that time, separate GA operations had to be distributed amongst four FPGAs for execution. Note that subsequent advances in FPGA technology now feasibly allow for an entire GA system, such as a micro-GA, to be implemented entirely on a single FPGA, as is undertaken in this research. Later work by Skliarova and Ferrari [113] presented and implemented a GA crossover operator on FPGA for solving the TSP.

Vega et al. [114] developed alternative versions of a GA for FPGA implementation using Handel-C (a high level C like programming language that targets low-level hardware instantiation). However, in contrast to this research, no hardware-based architecture was actually proposed, as the focus of that study was on development efforts aimed at improving the Handel-C encoding.

Tachibana et al. [115] proposed a basic island GA architecture suitable for hardware implementation which includes management, crossover, mutation, and evaluation

modules. Parallel implementation of the individual modules was not considered and an immigration module was included to oversee periodic exchange of individuals between separate concurrently running GAs. Their approach also involved implicit handling of selection and update phases through an incorporated management module, as opposed to this research in which explicitly considers distinct selection and update phases to allow for greater modularity.

Deliparaschos et al. [116] proposed the design and hardware implementation of a parameterised GA on an FPGA was proposed and evaluated on the TSP. A control module was implemented to manage each operation asynchronously, which limits the effective throughput of the system. Contrary to that, this research aims to develop on possible pipelining aspects from FPGA-based EA architecture.

Although insufficient design details were provided, two more recent proposals for the TSP using a steady-state GA on FPGA by Santos and Alves [117] and a pipelining structure for GA on FPGA by Zhou et al. [118] can also be mentioned.

2.4 Multi-Objective Optimisation

This subsection highlights the main concepts for multi-objective optimisation. In multi-objective optimisation scenarios, some objectives might prove to be conflictive. This is normally addressed by assigning some parameters to distribute preferences across the objectives, which forms a single composite function. Multi-objective optimisation can thereby be handled as single-objective optimisation. However, this transformation essentially creates a degenerate case of multi-objective optimisation, of which misconceptions and drawbacks are ignored [20]. With respect to all objectives and without additional information, no single solution can be said to be of paramount optimality.

Hence, the main goal of multi-objective optimisation is to obtain the set of optimal solutions that pertains to the trade-offs arising between conflicting objectives.

A multi-objective analytical optimisation model can be formulated as follows:

$$\left.\begin{array}{lll} \text{Minimise/Maximise} & f_m(\mathbf{x}), & m = 1, 2, \ldots, M; \\ \text{subjected to} & g_j(\mathbf{x}), & j = 1, 2, \ldots, J; \\ & x_i^{(L)} \leq x_i \leq x_i^{(U)}, & i = 1, 2, \ldots, n. \end{array}\right\} \tag{2.2}$$

Where M is the number of objective functions, $f_m, m \in \{1, \ldots, M\}$, that are to be minimised or maximised subject to J constraints, $g_j, j \in \{1, \ldots, J\}$. $\mathbf{x} = (x_i, \ldots, x_n), i \in \{1, \ldots, n\}$, is the "optimisation vector" of n decision variables that are individually restricted by lower $x_i^{(L)}$ and upper $x_i^{(U)}$ bounds. The *decision space* is the feasible region where the candidate solution, **x**, is considered as subjected by the boundaries of the decision variables. In contrast with single-objective optimisation, the objective functions in multi-objective optimisation constitute a multi-dimensional space called the *search space*. The mapping between each solution **x** in the decision space and its specific point in the search space is denoted by $f(\mathbf{x}) = \mathbf{z} = (z_1, \ldots, z_M)$, as portrayed in Figure 2.8.

Multi-objective optimisation algorithms use the concept of *dominance* to distinguish if one candidate solution is better than another. That is, solution $\mathbf{x}^{(1)}$ is said to dominate solution $\mathbf{x}^{(2)}$ (denoted by $\mathbf{x}^{(1)} \preceq \mathbf{x}^{(2)}$), if $\mathbf{x}^{(1)}$ is no worse than $\mathbf{x}^{(2)}$ in all objectives and $\mathbf{x}^{(1)}$ is strictly better than $\mathbf{x}^{(2)}$ in at least one objective. Thereby, a solution is considered *Pareto optimal* if it is not dominated by any other solution, as illustrated in Figure 2.9. Note that there exists levels of dominance such as weak dominance, strict

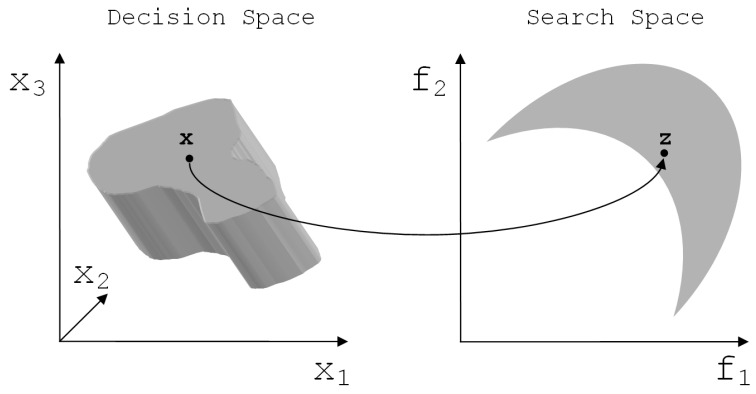


Figure 2.8: Representation of the decision space and the corresponding search space in a multi-objective problem.

dominance, and ϵ-dominance [119]. The set of Pareto optimal solutions generated by the multi-objective optimisation algorithm is called the *Pareto front*.

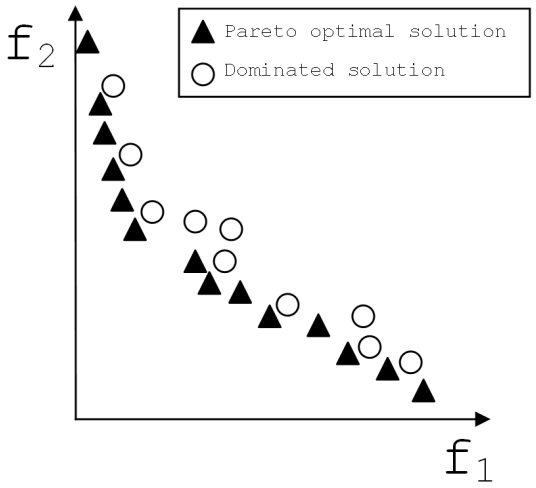


Figure 2.9: An illustration of Pareto optimal (or non-dominated) solutions and dominated solutions in the search space for a two-objective minimisation problem.

The ideal goal of multi-objective optimisation is to obtain the exact Pareto front of a given multi-objective problem, but this is not possible since the exact Pareto front contains an infinite number of points. Therefore, obtaining a good approximation

of the Pareto front from a finite set of Pareto optimal solutions, which exhibit exact *convergence* and uniform *diversity* are its two practical goals [20]. The exact Pareto front can be described as convex or non-convex, continuous or discontinuous, uni-modal or multi-modal. Figure 2.10 illustrates an example of a Pareto front obtained after the multi-objective optimisation process of a welded beam design problem for which formulation can be found in [120]. It shows two extreme optimal solutions and one of the possible tradeoffs in between, in which a designer will make a preferred design selection from. As mentioned earlier, any preference made from the Pareto front are part of a non-dominated solution set, hence optimality is not compromised from the decision making.

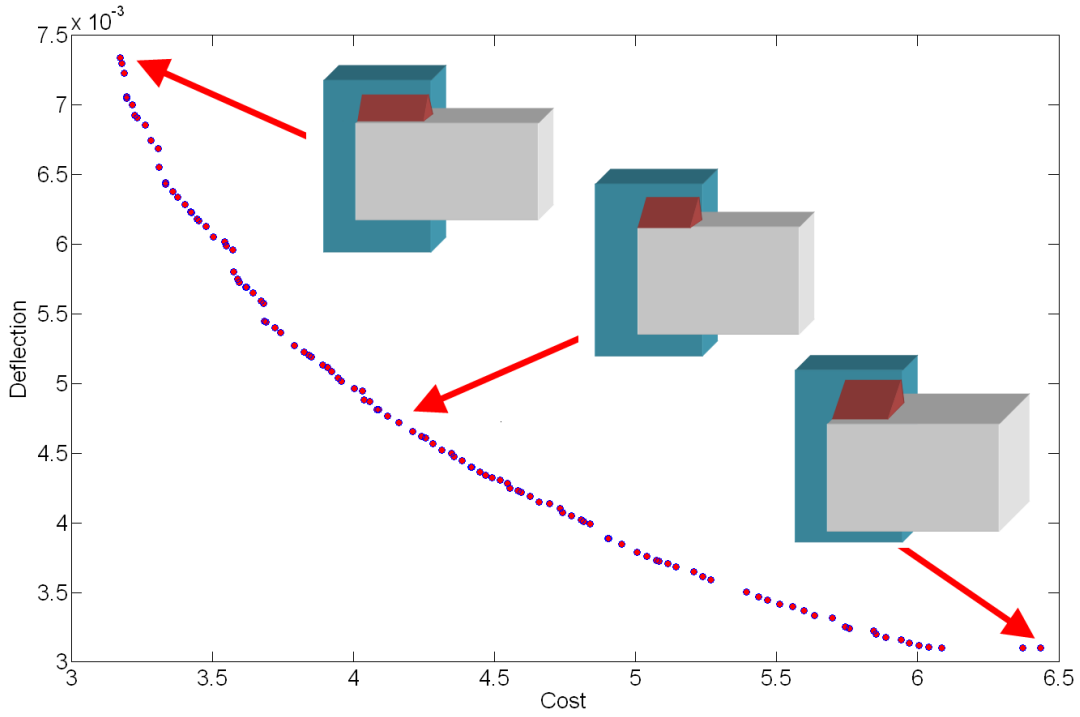


Figure 2.10: An example of the Pareto front solutions for a welded beam design problem.

2.4.1 Multi-Objective EA for Multi-Objective Optimisation

Unlike single-objective EA, where all individuals in the population converge to a single optimal solution, a multi-objective EA has to evolve and obtain an entire population of Pareto optimal solutions. Hence, at the end of the multi-objective optimisation process, any solution of the multi-objective EA's population is an optimal solution with respect to all of the objectives. Although multi-objective EA are fundamentally based on EA mechanisms as well, it still require a few additional components to handle multi-objective problems effectively. This subsection describes the three main search components required in multi-objective EAs to efficiently guide the population towards the two goals of exact convergence and uniform diversity: fitness assignment, diversity preservation, and elitism.

Fitness Assignment: The objective of the fitness assignment is to translate the vector of evaluated objective functions for a given solution into a single qualitative value, which promotes convergence of the Pareto front. Pareto-based schemes [20] use the concept of dominance to guide the optimisation process, while indicator-based schemes [121] rely on a specified quality measure for performance.

Diversity Preservation: The objective of the diversity preservation is to maintain and diversify the population of solutions. Sharing schemes [122] strongly rely on *a priori* knowledge to specify a threshold parameter for measuring a distance metric between two solutions. Contrary to that, crowding schemes [123] quantify the crowding metric of a set of neighbouring solutions independently.

Elitism: The objective of elitism is to prevent the loss of beneficial solutions during the evolutionary process. This concept is achieved by maintaining an archive of non-dominated or preferred solutions, which were discovered earlier, for introduction back into the evolutionary process.

From this set of search components, different multi-objective EAs are designed and implemented by the combination of relevant features. For instance, the NSGA-II incorporates non-dominance sorting, crowding distance assignment and elitism selection process [25]; the strength Pareto evolutionary algorithm 2 (SPEA2) employs fine-grained non-dominance fitness assignment, nearest neighbour density estimation, and archive truncation method [124]; and the archive-based micro-genetic algorithm 2 (AMGA2) integrates non-dominance sorting, modified crowding distance assignment and external archiving [125]. Multi-objective optimisation using NSGA-II, SPEA2, AMGA2 and other multi-objective EA methods have been applied to many aerospace optimisation problems. For instance, Arias-Montano et al. [126] presents a comprehensive review of applications where multi-objective EAs are employed for solving multi-objective optimisation problems in aerospace.

2.4.2 FPGA-based Multi-Objective EAs

Research on multi-objective EA-based FPGAs is limited. To the author's knowledge, there are only two works in literature that attempt to develop multi-objective EAs on FPGAs: Tachibana et al. [127], and Bonissone and Subbu [128].

Tachibana et al. [127] developed and implemented a multi-objective EA on FPGA based on a modified selection and overlap rejection components into a minimal genera-

tion gap model [129], for aiding convergence and diversity respectively. The selection operation compares the fitness value of the objectives between two solutions for superiority, in which the better solution subsequently replaces the worst one. The overlap rejection operation compares the fitness of a given solution against the fitness of all solutions in the population and removes ones with identical fitness. Their experiment compared a simulation of the FPGA implementation with its software version, which resulted in a significant speed improvement from 43 seconds to 10 milliseconds. However, their work has some theoretical drawbacks, firstly the selection operation does not take into account dominance across the entire population, rather it only compares two randomly selected individuals. Thereby, the notion of Pareto optimality is not applied. Secondly the overlap rejection that is aimed at removing solutions with similar fitness implicitly makes the assumption that many-to-one mapping between the decision space and the objective space do not occur, which may not always be the case.

Bonissone and Subbu [128] presented an implementation of a basic multi-objective EA on an FPGA. They proposed the use of a dominance filter to segregate an archive of non-dominated solutions, in which a comparison of each solution against every other solution is carried out. Their experiment compared the FPGA design with its software version, which showed significant speedup by two orders of magnitude. However, there was no analysis or verification conducted on the overall multi-objective EAs behaviour without the fitness assignment and diversity preservation features. Thereby, the implementation could be theoretically flawed and unstable. One other limitation from both Tachibana et al. [127] and Bonissone and Subbu [128] is that they inadequately justified conclusions of efficiency by using only one test problem experiment rather than against a variety of test problem with known solutions.

2.5 Summary

The key findings of the literature review are as follows:

Through the years there has been strong evidence of research interest around the notion of algorithm acceleration using FPGA technology. FPGA-based algorithm implementations offer benefits stemming from both its physical attributes of being compact in size and its delivered performance of significantly reducing computational time. However, the architecture design itself is not a straightforward process with proposed FPGA-based algorithms having to be scaled down or reengineered. FPGA design methodologies for EAs have been continuously proposed since the past 20 years but low in frequency and quality, which indicates that it is not a dead-ended research direction and there are undeveloped areas where there is room for investigating new concepts.

In robotics navigation, path planning is a complex task where navigating across an environment requires a number of considerations within the path planning algorithm, including environment, vehicle, and mission profiles. Although prior research works have established the concept of FPGA implementations for speeding up different computationally intensive path planning algorithms, this research instead contributes a completely embedded path planning system based on EA optimisation algorithms. This is considered in Chapter 3 and Chapter 4 where the proposed FPGA-based EA architectures consider the inclusion of constraints from UAV, environment, and mission profiles. Note that for this research, the UAV was the designated robotic platform but this is not expected to be restrictive.

In the field of combinatorial optimisation, the TSP is an NP-complete problem that has been intensively investigated, which is rightly appropriate with its uses in a

wide range of practical applications. Several studies have explored specialised FPGA-based EA architectures for solving the TSP. The hardware and solution implications of different EA parameters are studied in Chapter 5. A thorough investigation of implementing an efficient FPGA-based EA architecture for solving the TSP is conducted in Chapter 6.

Multi-objective optimisation is complex from the problem and optimisation algorithm perspectives. It has its application in many real-world problems, which requires computationally intensive optimisation algorithms to obtain an effective Pareto front. Multi-objective EAs have a well-established reputation for addressing multi-objective optimisation. This is mainly attributed to their population-based nature that allows for information of candidate solutions to be handled and shared effectively within the EAs, which in return produces increasingly better solutions. However, research in FPGA-based multi-objective EA is limited and does not provide conclusive evidence supporting the interesting combination of coupling multi-objective EA with FPGA technology. Investigations presented in Chapter 7 and Chapter 8 address the explicit preservation of algorithm integrity by adapting the original multi-objective EAs functional behaviour into the proposed architectures.

This page intentionally left blank.

Chapter 3

A Synthesizable Hardware Evolutionary Algorithm Design for Unmanned Aerial System Real-Time Path Planning

In this chapter, the design flow and FPGA-based EA architecture for on-board path planning is investigated. The development of a feasible FPGA-based GA to determine flight path plan for UAVs navigating terrain with obstacle boundaries is detailed. The design architecture includes the hardware implementation of terrain data and EA population memories within the hardware architecture. Simulation results show significant speedup with an average of 52,000 times faster than its software version, suggesting that the present approach is well suited for UAV real-time path planning applications. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(a), 3.(a) and 4.

3.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

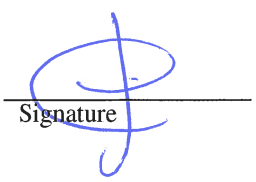
J. Kok, L. F. Gonzalez, R. A. Walker, T. Gurnett, and N. A. Kelson, "A synthesizable hardware evolutionary algorithm design for unmanned aerial system real-time path planning," in *Proceedings of the 2010 Australasian Conference on Robotics and Automation (ACRA 2010)*, 2010, pp. 1–8.

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Rodney Walker	✓	✓	✓	✓
Timothy Gurnett	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ
Name


Signature

18-6-2014
Date

A Synthesizable Hardware Evolutionary Algorithm Design for Unmanned Aerial System Real-Time Path Planning

Jonathan Kok, Luis Felipe Gonzalez, Rodney Walker
Australian Research Centre for Aerospace Automation (ARCAA)
Eagle Farm, Queensland 4009, Australia
j1.kok@qut.edu.au, felipe.gonzalez@qut.edu.au, ra.walker@qut.edu.au
Timothy Gurnett, Neil Kelson,
High Performance Computing & Research Support, Queensland University of Technology (QUT)
Brisbane, Queensland 4001, Australia
timothy.gurnett@qut.edu.au, n.kelson@qut.edu.au

Abstract

The main objective of this paper is to detail the development of a feasible hardware design based on Evolutionary Algorithms (EAs) to determine flight path planning for Unmanned Aerial Vehicles (UAVs) navigating terrain with obstacle boundaries. The design architecture includes the hardware implementation of Light Detection And Ranging (LiDAR) terrain and EA population memories within the hardware, as well as the EA search and evaluation algorithms used in the optimizing stage of path planning. A synthesisable Very-high-speed integrated circuit Hardware Description Language (VHDL) implementation of the design was developed, for realisation on a Field Programmable Gate Array (FPGA) platform. Simulation results show significant speedup compared with an equivalent software implementation written in C++, suggesting that the present approach is well suited for UAV real-time path planning applications.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are widely used in military and civilian contexts. Military missions could involve target and decoy, reconnaissance, combat and logistics operations. In the civilian context, UAVs are being developed for environmental and agricultural purposes such as weather forecasting, storm and bush fire detection, farm field seeding and aerobiological sampling. All these scenarios involve a common task, which currently is determined by a human operator: Path Planning.

One of the main objectives in path planning is to develop optimization techniques which are effective and efficient in terms of computational cost and solution quality [Deb, 2001], [Lee *et al*., 2008]. Traditionally, optimal path plans are found using deterministic optimizers. However, many approaches have a tendency to become trapped in local minima [Zheng *et al*., 2003]. Instead, evolutionary algorithms (EAs) are preferable as the most viable search algorithms for a real-time UAV path planner [Allaire *et al*., 2009]. These are more robust and allow them to find global solutions, but at a large computational expense. Hence of interest is a computationally efficient hardware implementation of a path planning algorithm based on EA running on an FPGA platform. The drawback of having a population based algorithm manipulated sequentially, which is a computationally intensive process, is overcome by exploiting the parallelism processing capability of the FPGA. Earlier work involving partial FPGA implementation of EAs for UAV real-time path planning [Allaire *et al*., 2009] considered EA modules running on an FPGA platform, while the evaluation and simulation phases were performed on a PC. Results indicated that partial FPGA implementation could provide orders of magnitude speedups over software-only execution.

The main objective of this paper is to detail the development of a feasible hardware EA-based design to determine flight path planning for Unmanned Aerial System (UAS) navigating terrain with obstacle boundaries. The design architecture can provide UAS with autonomous real-time path planning capability using an EA entirely implemented on an FPGA platform, and includes hardware implementations of Light Detection And Ranging (LiDAR) source terrain data and EA population memories, as well as

the EA search and evaluation algorithms used in the optimizing stage of path planning. The design will be herein referred to as Hardware-EA. As will be shown subsequently, simulation results for the Hardware-EA show significant speedup compared with an equivalent software implementation written in C++, suggesting that the present approach is well suited for UAS real-time path planning applications.

This paper is organized as follows. Section 2 gives a brief description of related work and the main difference in our approach. Section 3 describes in detail the architecture of the Hardware-EA. Section 4 describes an example of the Hardware-EA design implementation and its details. Section 5 discusses the computation time results of the PC-based EA and the Hardware-EA, and elaborates on the efficiency of the Hardware-EA. Section 6 presents a real world application. Section 7 concludes with a brief summary and possible enhancements for future work.

2 Background and Related Work

Although route planning has been widely researched, less attention has been given to UAV applications [Zheng *et al*., 2003]. Both [Zheng *et al*., 2003] and [Allaire *et al*., 2009] elaborate on the importance of path planning, describe various types of path planning techniques, and argue why Evolutionary Algorithms are preferred as the most viable search algorithms for a real-time UAV path planner. [Lavelle, 2006] also presents a good summary of path planning algorithms, while the theory of circuit design using VHDL and FPGAs is extensive, and can be found in e.g. [Pedroni, 2004].

Previous work into path planning algorithms on hardware is limited but is now being considered by a number of researchers. Implementation of an EA on FPGAs has been explored in studies including those by [Allaire *et al*., 2009], [Aporntewan and Chongstitvatana, 2001], and [Fernando *et al*., 2010], and have shown promising results. [Allaire *et al*., 2009] concluded that FPGA implementation of EA for UAV real-time path planning inherits the EA's optimal search advantage and overcomes the EA's computational disadvantage. However, they did not implement a fully synthesizable hardware design. [Aporntewan and Chongstitvatana, 2001] showed that their hardware Compact Genetic Algorithm (CGA) implementation implemented on an FPGA ran about 1,000 times faster than the software execution of their original code. [Fernando *et al*., 2010] illustrated the use of hardware implementation of EA as an efficient optimization engine for evolvable hardware, having speedup of 5 times over an analogous software implementation. Their research focused mainly on the general theory of EAs being implemented on an FPGA. [Bonissone and Subbu, 2007] proposed a multi-objective EA architecture, but due to the simulator constraint, their research was limited to simulation without synthesizing their design. Their simulation results show a speedup of 328 times over its software counterpart. [Gallagher *et al*., 2004] compares and contrasts a family of CGAs implementations on an FPGA. Their research concluded with recommendations to redesign the dataflow and to introduce systolic array methods to improve efficiency without increasing implementation cost.

The main difference with the approach used here and earlier work is the focus on an EA-based design architecture that is fully synthesizable and implementable on a single FPGA device, with reference to benefits and difficulties of practical application aspects (i.e. UAS path planning). The basic hardware design flow which this research aims to encapsulate is shown in Figure 1. To demonstrate implementation of the design in synthesisable VHDL, a modified version of the [Cocaud, 2006] EA search algorithm for UAV path planning has been used. More complex algorithms such as those given in [Lee *et al*., 2008] could also be implemented and are being considered. Synthesis is important because it involves producing a netlist from VHDL that can be subsequently mapped onto an FPGA; from low level description to an even lower level description.

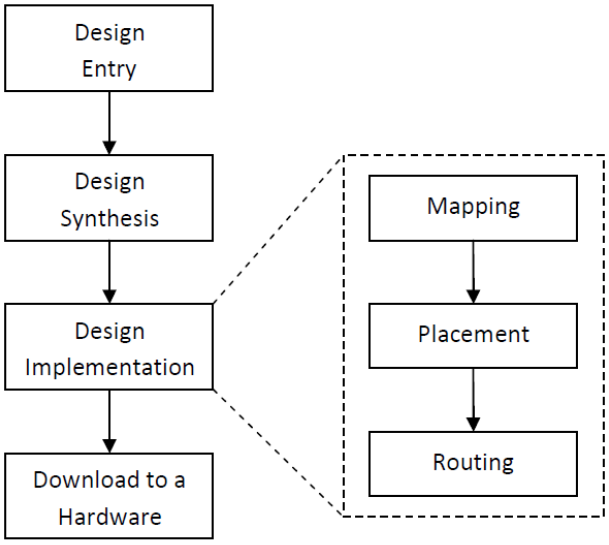


Figure 1: Basic hardware design flow

3 Hardware-Evolutionary Algorithm Design

3.1 The Architecture

The design architecture for the Hardware-EA proposed here is illustrated in Figure 2 and detailed in Figure 3 in the next subsection. The design is intended to fit into a single FPGA module, and includes units corresponding to typical EA tasks (such as selection, crossover, mutation, fitness evaluation, and so forth [Goldberg, 1989; Michalewicz, 1996]) where the functionality of each unit can be set according to the algorithmic requirements of the specific EA under consideration.

The driving component of the Hardware-EA is the Sort/Termination Unit (STU). The STU provides a memory interface, sorting algorithm, population update mechanism, and monitors the termination criterion of the evolutionary algorithm. In doing so, it acts as the main control unit of the Hardware-EA throughout the entire operation and is the Hardware-EA's sole output interface.

Additionally, a Terrain Memory (TM) unit is included where terrain information is stored (e.g. terrain data

from LiDAR source stored in a single FPGA Block RAM). The initial and subsequent populations are then evolved based on this data, where each population member represents one possible path-solution for the UAV to traverse the terrain between designated initial and terminal waypoints.

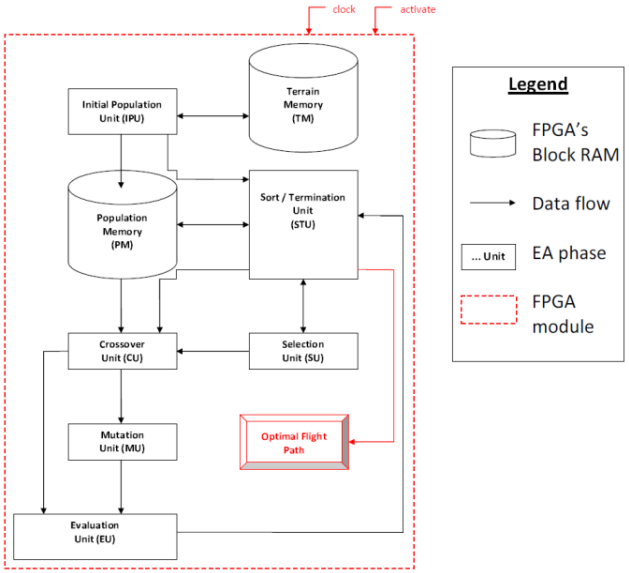


Figure 2: Overview of the Hardware-EA architecture

The design also includes an on-module Population Memory (PM) unit. This initially stores the "parent" population of path-solutions (e.g. stored in another FPGA Block RAM location) generated by the Initial Population Unit (IPU), but is also reused at each step of the EA iterative procedure to store the offspring population. Typically, each path-solution within a population is a bit stream consisting of a binary-code scheme representing the fitness, number of nodes and its transitional waypoints. A simple encoder/decoder is used to convert between this bit stream format and its decimal representation.

The two inputs which drive the Hardware-EA are the clock signal and the activation signal. The clock signal is connected to the embedded global clock of the FPGA, and the activation signal is connected to an input from the outside world.

Although not illustrated in Figure 2, the design includes an input line for initial transfer of terrain data to the TM unit, or subsequent data refresh of the TM unit from an external source to account for changed conditions such as UAV location. There is also an STU signal line where the optimal flight path computed by the module is output to the UAV for subsequent processing.

3.2 Operation

In this section, the flow of execution and communication between the individual units of the Hardware-EA is described and illustrated in Figure 3. Note that during the evolution phases, control is distributed; all units operate autonomously and asynchronously.

To begin the Hardware-EA process, an activation signal is received, initiating transfer and storage of externally generated terrain data into the TM unit. The TM unit interfaces with the IPU via two connections, an incoming address channel and an outgoing data channel. These channels allow the IPU to receive terrain information from the TM unit and generate, influenced by flight parameters such as minimum and maximum UAV elevation, a set of random path-solutions that are to be stored in the PM unit. These solutions are evaluated by the EU, sorted by the STU and then stored into the PM unit. This completes the initial setup of the Hardware-EA.

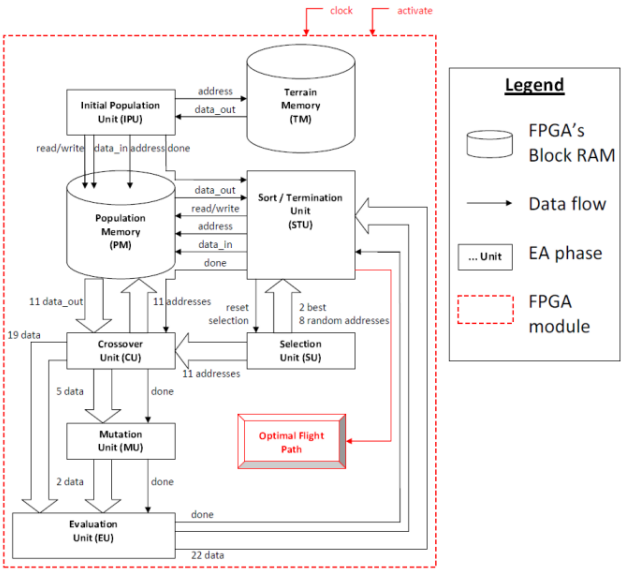


Figure 3: Detailed data path of the Hardware-EA architecture

To commence the first iteration of the Hardware-EA process, the STU notifies the Crossover Unit (CU) and Selection Unit (SU) that the Hardware-EA is ready to begin execution. The task of the SU is to generate random addresses to be used for the CU and population update for the STU. When the CU receives addresses from the SU, it requests these members from the PM unit and starts the crossover operation, creating new members. A selection of the new crossovered members is then passed to the Mutation Unit (MU) to be mutated. Once completed, the mutated members are sent to the Evaluation Unit (EU) for evaluation. The EU determines the fitness of new members generated by the CU and MU. Upon completion, the EU sends the evaluated members to the STU. Finally, the STU writes the new members and their new fitness values into the PM unit.

The above iterative step is repeated until the STU determines that the current Hardware-EA run is finished when the stopping criteria or convergence criterion is satisfied. It then transmits out the optimised flight path-solution, which is the best member decided through the fitness sorting algorithm.

4 Example of Hardware-EA Design Implementation

To explore the feasibility of the proposed design architecture, an implementation of the proposed Hardware-EA in

synthesisable VHDL was undertaken. To facilitate comparison with previous work, the EA used by [Cocaud, 2006] for flight path planning was employed. Additionally, a development platform containing a Xilinx Virtex 4 LX200 FPGA processor was available. This platform was used to explore implementation issues such as the population characteristics and extent of parallelism possible within the design, subject to various constraints including the programmable logic resources available on the FPGA. Implementation details of the various elements and the EA population characteristics are briefly described below.

4.1 EA Population Characteristics

The implementation of the Hardware-EA requires specification of the EA population characteristics and operations. The operations of selection, crossover, mutation and evaluation are involved in the iterative EA process, as shown in Figure 4, and require specification. Selection involves identification of those members to undergo crossover from the current "parent" population of path-solutions.

```
Evolutionary Algorithm ();
      Initialise population;
      Evaluate initial population;
      WHILE convergence criteria IS NOT satisfied, DO
            Selection technique;
            Crossover operations;
            Mutation operations;
            Evaluation technique;
            Update population;
      END-WHILE;
```

Figure 4: Pseudo code of EA

For crossover of the selected path-solutions, all transitional waypoints after the middle of the path are truncated and swapped between the two selected parent path-solutions. The resultants are the offspring (see Figure 5).

Two selected offspring are further subjected to an insert mutation and a delete mutation to promote speed-up of convergence (as illustrated in Figure 6 and Figure 7). The insert and delete mutation takes into consideration the minimum and maximum number of transitional waypoints allowed, hence the population member is not corrupted with an invalid number of transitional waypoints. No swap mutation was considered as the algorithm is optimising a single objective function and the swap mechanism recommended by [Cocaud, 2006] is mainly for multi-objective functions.

The population is then updated using a semi-elitist approach, where a selection of the best path-solutions from the old population is retained and the remaining more inferior members are randomly overwritten by the offspring. Finally, the fitness of each member of the updated population is assessed based on feasibility and shortest distance. The iterative steps just described are repeated until a stopping criterion chosen for this implementation of sixty (60) generations has been completed.

Following a slightly modified [Cocaud, 2006] recommendation, the offspring from the new population is set to 70% of the old parent population. All of the offspring are bred from crossover, and 10% of the crossovered offspring are further subjected to mutation. Thereafter, a population update is performed where 5% of the best path-solutions from the old parent population are retained, and each of the offspring has replaced 70% of the remaining 95% of the old population.

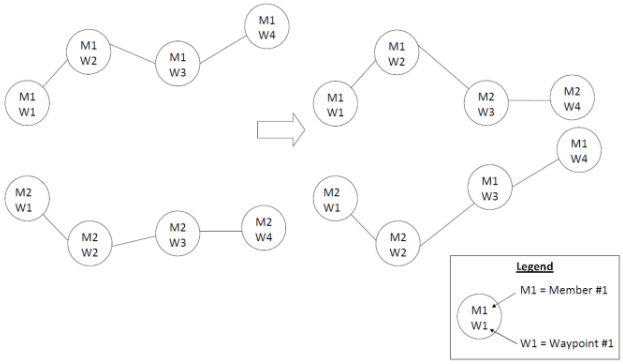


Figure 5: Example of crossover

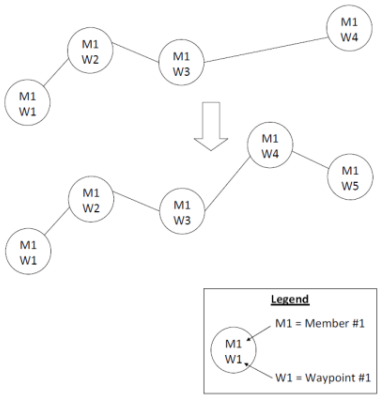


Figure 6: Examples of insert mutation

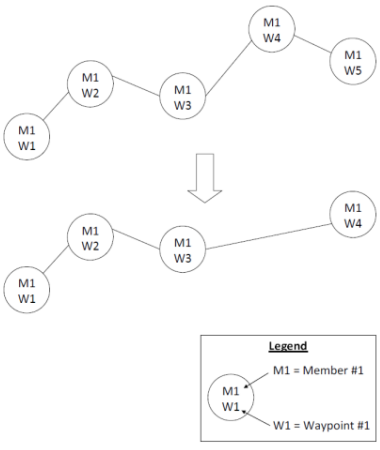


Figure 7: Examples of delete mutation

4.2 Implementation Details

Encoding of EA parameters: One of the first design decisions is determining the encoding of the parameters as this will enhance or hinder the computational time. Here, population members were chosen to correspond to single path-solutions

comprising a minimum of two and a maximum of five transitional waypoints in addition to the starting and ending waypoints. Each transitional waypoint is characterized by its three spatial coordinates. The population size and starting/ending waypoints for all path-solutions are not subjected to change during the entire EA process. The bits encoding for the parameters of each member is depicted in Table 1.

Table 1
Encoding of EA parameters

Parameter	Number of bits	Integer range
Fitness	7 bits	[0,127]
Number of Transitional Waypoints (Excluding starting and ending waypoints)	2 bits	[0,3] ; where 00_2 = 2 waypoints 01_2 = 3 waypoints 10_2 = 4 waypoints 11_2 = 5 waypoints
Transitional Waypoint 1	27 bits = (X+Y+Z) ; where X = Y = Z = 9 bits	X = Y = Z = [0,511] ; where X = longitude Y = latitude Z = altitude
Transitional Waypoint 2	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 3	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 4	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Transitional Waypoint 5	Same as Transitional Waypoint 1	Same as Transitional Waypoint 1
Total Number of bits	144 bits	

Terrain Memory unit (TM): For the example Hardware-EA implementation, sample terrain data to be used for path planning was gathered from a Light Detection And Ranging (LiDAR) source (see Figure 8) and converted, by a in house C++ application, into a bit stream represented by an array of latitude, longitude and altitude coordinates. Subsequently, this bit stream is stored in one of the FPGA's Block RAM modules. The testing environment map is large and consists of 55,556 points. Information for a point includes latitude, longitude, altitude, intensity and classification. The entire terrain is equivalent to 600 KB of data.

Selection Unit (SU): Within one clock cycle, the SU generates 19 different random addresses (11 for the CU and 8 for population update), which change only when instructed by the STU through the incoming control signal connection.

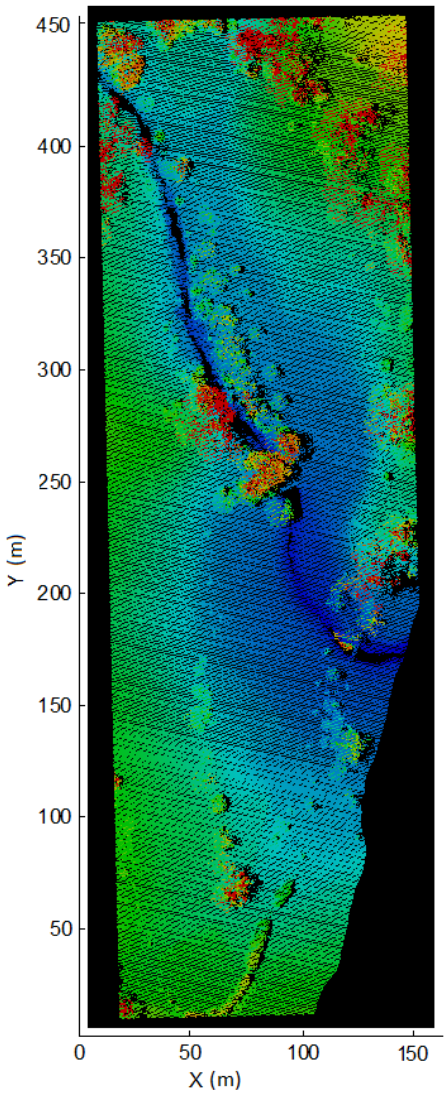


Figure 8: Two-dimensional view of testing environment

Population Memory (PM) & Initial Population Unit (IPU): The PM unit, like TM unit, is implemented using available FPGA Block RAM. As such, there is no shared memory external to the Hardware-EA system. Due to size limitations, the PM unit was fixed to store 32 population members (i.e. path-solutions). Initially, the IPU generates 32 random path-solutions, where each member includes 2 random transitional waypoints, the number of nodes, and the fitness.

Sort / Termination Unit (STU): The STU interface to the PM unit allows for the STU to read, in a single cycle, the entire list of path-solutions, sort them, and overwrite the original contents of the underlying Block RAM with the list of path-solutions now sorted by their individual fitness values.

The STU also interfaces with the EU to receive 22 offspring path-solutions (i.e. ~70% of the population size of 32) from the EU, and is responsible for storing them back in the PM unit using the previously mentioned semi-elitist approach. To these ends, the implementation of the STU interface to the PM unit includes both a data read and data write operation, however all control of this data transfer is initiated from the STU via a read/write control signal. The STU has two outgoing control signals (to the CU and SU) as well as two incoming control signals (from the IPU and EU). These signals determine the current operational state of the evolutionary algorithm. For a given iterative step of the EA process, the internal generation-counter within the STU is incremented if the termination criterion of 60 generations is not met, the SU is notified to generate a new set of random addresses, and the CU is notified to continue the evolution. Otherwise, the path-solution at the top of the PM unit is delivered externally from the FPGA module as the most optimised path-solution.

Crossover Unit (CU): The CU is composed of 11 identical processing modules. Notably, all crossover operations for a single generation are done in parallel. In one clock cycle, the CU utilizes the 11 addresses provided by the SU twice, generating a variety of parental combinations to produce 22 offspring path-solutions (i.e. ~70% of the population size of 32).

Mutation Unit (MU): The MU selects 2 of the offspring path-solutions (i.e. ~10% of the offspring size) from the crossover unit for mutation. One for insert mutation and the other for delete mutation. All of the mutation modules operate in parallel.

Evaluation Unit (EU): The EU evaluates the feasibility of the 22 new path-solutions (19 offspring and 2 mutated offspring) and generates a new fitness value for each. Equation (1) is used to calculate the fitness value, based on the feasible distance travelled;

$$\sum_{i=0}^{N-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2} \quad (1)$$

where N is the total number of transitional waypoints and x, y and z are the latitude, longitude and altitude, respectively. Once again, these operations are all done in parallel.

4.3 Synthesize Details

The setup of the design synthesis is as follows. The design was synthesized with the Xilinx Virtex 4 XC4VLX200 FPGA as the target device, and the design goal was set to "balanced". A "balanced" design implies that no optimization for speed or utilization of resources was considered. Once the design was synthesized successfully, it was the compiled and built for implementation, as shown earlier in Figure 1. This process consists of translating, mapping, placing and routing of the signals. For the design implementation process, no partition was specified and the design was translated and mapped successfully. All signals were placed and routed successfully as well, and all timing constraints were met. Table 2 shows the device utilization summary.

Table 2
Device utilization summary

Logic Utilization	Used	Available	Utilization
Number of Slices	18,415	89,088	20%
Number of Slice Flip Flops	9,051	178,176	5%
Number of 4 input LUTs	32,831	178,176	18%
Number of 18 Kb Block RAM	45	336	13%
Number of Bonded IOBs	146	960	15%
Number of GCLKs	1	32	3%

5 Preliminary Experiments and Results

5.1 Computation Time Comparison

Five experiments were conducted with the C++ executable of the EA and the average timing was recorded. The experiments were run on an Intel® Core™ 2 Duo Core CPU 2.66GHz. The Hardware-EA experiments were run on a FPGA simulator via Xilinx ISE 12.3. The entire terrain consisting 600 KB of data was used. Results for each EA phase are exhibited in Table 3.

Table 3
Timing results per EA cycle

EA phase	PC-based EA	Hardware -EA	Speed improvement
Crossover	4 ms	375 ns	10,000
Mutation	1 ms	1250 ns	800
Evaluation	1,370 ms	1,000 ns	1,370,000
Selection + Population Update	0.501 ms	23,750 ns	20
Total[a]	1,376 ms	26,375 ns	52,000

The tabulated time values are averages of the multiple runs.
[a]The total time is in reference to one complete EA generation.

5.2 Discussion

From Table 3, it can be seen that the hardware implementation provides significant speed improvement for all of the EA phases. The EA with a convergence criterion of sixty (60) generations results in a computational time improvement from 82 seconds to 0.3 milliseconds. This is mainly due to the fact that the PC-based EA had to run sequentially and the processing speed is CPU dependent. In the Hardware-EA design, parallelism incorporated by duplicating modules, and the Selection Unit generating a set of random numbers for the crossover in a single clock cycle, resulted in the significant speed improvement.

6 A Real World Application

6.1 Overview

The concept of having significantly fast algorithms is to realise real-time application. In a potential practical application, the Unmanned Aerial Vehicle (UAV) is flying at low level avoiding terrain while conducting an air sampling mission (see Figure 9). The UAV has to be able to fly the shortest path between certain points, while avoiding obstacles that are present at low altitudes. An external processor or device could be used to generate the TM unit directly from a LiDAR sensor.

Queensland University of Technology (QUT), in conjunction with the Cooperative Research Centre (CRC) for Plant Bio-security, Department of Agriculture and Food, Western Australia's Murdoch University, Queensland Department of Primary Industries and Fisheries, is currently involved in a project to develop an UAV to monitor inaccessible cultivation areas and sample air and look for either unwanted spores or other plant pathogens. An UAV fitted with such data collection system and air sampling device flying an optimal path can monitor and reduce the risk of pest introduction from international trade and, at the same time, will capture a wide range of plant health information in a cost-effective way so as to cover international and domestic market demands.

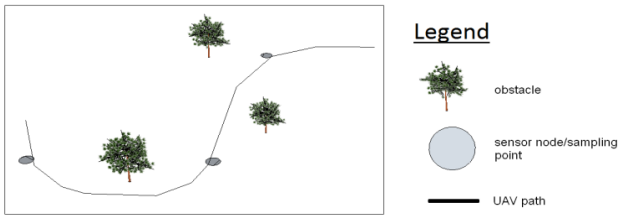


Figure 9: Practical operation scenario: low level flight air sampling mission

6.2 Implementation

The implementation process is best described as a flowchart depicted in Figure 10. The first step is to convert the LAS file format, generated by the LiDAR, to a text file. The second step is to process the points in the text file so that the EA can be applied. Next, the start and end waypoint positions are defined (step 3) and the optimal path is obtained using the EA (step 4). In Step 5, the LIDAR map will be loaded as an image or Digital Elevation Model (DEM) in the UAV ground station software. Step 6 will load the optimal waypoints to the autopilot ground station and simulate the mission. Finally, in Step 7, the mission is executed.

The LiDAR map was converted to a text file, an optimal path was found and subsequently the DEM was loaded to the UAV ground station. Figure 11 shows the three dimensional view of the optimal flight path mission mapped out in MATLAB.

Figure 12 shows the simulation environment of the mission running on Horizon™ MicroPilot flight simulator. Figure 12 also shows that a UAV helicopter is capable of navigating through the terrain via the shortest route and avoiding obstacles (large trees, vegetation, buildings) along the way.

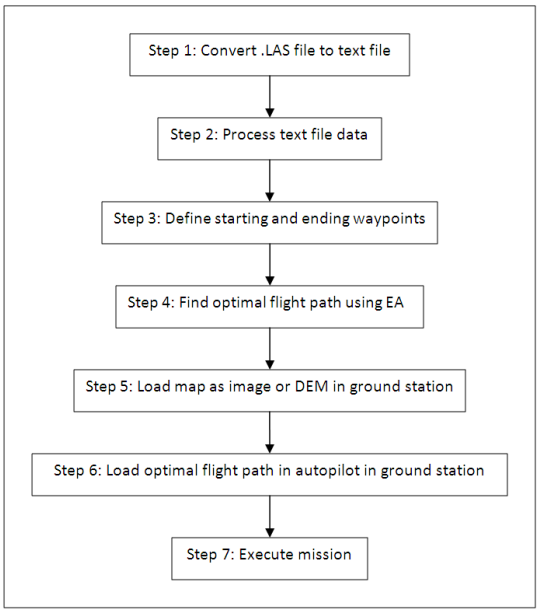


Figure 10: Flowchart of the implementation process

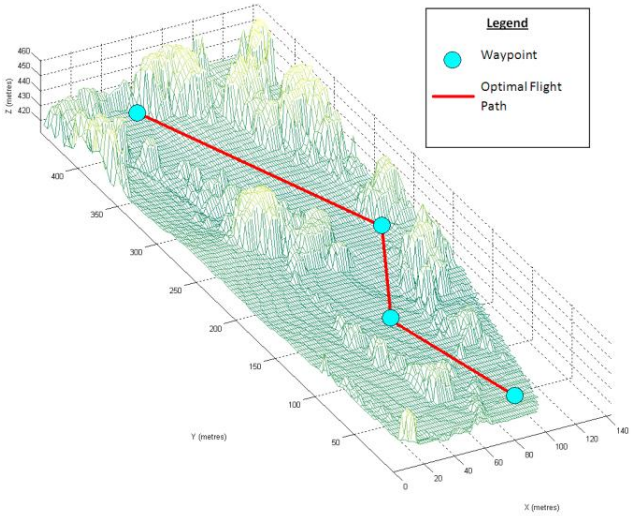


Figure 11: Three dimensional view from MATLAB optimal flight path simulation

7 Conclusion

This research has shown an enhancement in the computation speed of an EA hardware-based UAS path-planner. With this enhanced computation time, the system could be implemented and be used as an on-board path planner, re-computing flight plans in real-time. This implies the possibility of using it in a dynamic environment.

One development difficulty encountered was that the development time for the Hardware-EA using a low level hardware description language (HDL) such as VHDL is very

time consuming. Future work will explore utilizing a high level HDL such as Mitrion-C, Handel-C or Impulse C. Additionally, designing a framework or interface between the TM unit and the outside world would bring the platform one step closer to hardware application implementation.

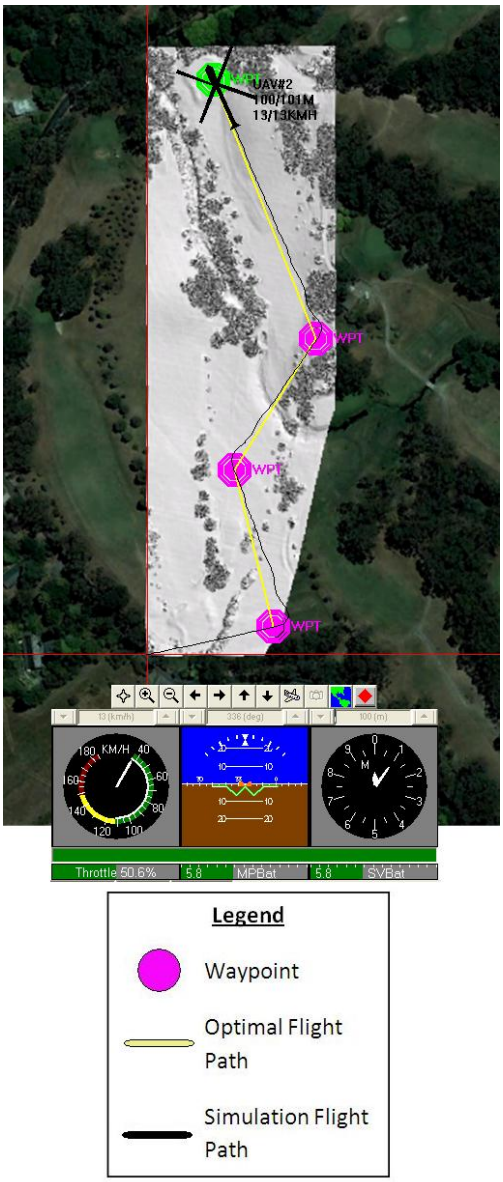


Figure 12: Top-down view from Horizon™ MicroPilot flight simulation of a flight path-solution

Acknowledgement

Computational resources and services used in this work were provided by the High Performance Computing and Research Support Group, QUT. The first three authors would like to acknowledge the Cooperative Research Centre for Spatial Information (CRCSI) for their help with the LiDAR files used in this work and the support of the CRC for National Plant Biosecurity.

References

[Allaire *et al*., 2009] F. Allaire, M. Tarbouchi, G. Labonte and G. Fusina. FPGA implementation of genetic algorithm for UAV real-time path planning. *Journal of Intelligent & Robotic Systems*, 54(1-3): 495-510, 2009.

[Aporntewan and Chongstitvatana, 2001] C. Aporntewan, and P. Chongstitvatana. A hardware implementation of the Compact Genetic Algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pp. 624-629, 2001.

[Bonissone and Subbu, 2007] S. Bonissone, and R. Subbu. Evolutionary Multiobjective Optimization on a Chip. *IEEE Workshop on Evolvable and Adaptive Hardware*, pp. 61-66, 2007.

[Cocaud, 2006] C. Cocaud. Autonomous tasks allocation and path generation of UAV's. Masters Thesis, Department of Mechanical Engineering, University of Ottawa, Ontario, 2006.

[Deb, 2001] Deb, K. Multi-objective optimization using evolutionary algorithms. Chichester ; New York, John Wiley & Sons, 2001.

[Fernando *et al*., 2010] P. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum and A. Stoica. Customizable FPGA IP Core Implementation of a General-Purpose Genetic Algorithm Engine. *IEEE Transactions on Evolutionary Computation*, 14(1): 133-149, 2010.

[Gallagher *et al*., 2004] J. Gallagher, S. Vigraham and G. Kramer. A family of compact genetic algorithms for intrinsic evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 8(2): 111-126, 2004.

[Goldberg, 1989] D. Goldberg. *Genetic algorithms in search, in: optimization and machine learning*. Boston, Kluwer Academic Publishers, 1989.

[Lavelle, 2006] S. M. Lavelle. *Planning Algorithms*. Cambridge, U.K., Cambridge University Press, 2006.

[Lee *et al*., 2008] D. S. Lee, L. F. Gonzalez, K. Srinivas and J. Periaux. Robust evolutionary algorithms for UAV/UCAV aerodynamic and RCS design optimisation. *Computers & Fluids* 37(5): 547-564, Jun 2008.

[Michalewicz, 1996] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. 3rd rev. and extended ed. Berlin; New York, Springer-Verlag, 1996.

[Pedroni, 2004] V. A. Pedroni. *Circuit design with VHDL*. Cambridge, Mass.; London, MIT Press, 2004.

[Zheng *et al*., 2003] C. W. Zheng, M. Y. Ding and C. P. Zhou. Real-time route planning for unmanned air vehicle with an evolutionary algorithm. *International Journal of Pattern Recognition and Artificial Intelligence*, 17(1): 63-81, Feb 2003.

Chapter 4

FPGA Implementation of an Evolutionary Algorithm for Autonomous Unmanned Aerial Vehicle On-Board Path Planning

In this chapter, an improved FPGA-based path planning architecture for UAV adaptation is proposed. The hardware design architecture consists of EA modules, population storage resources, and three-dimensional terrain information necessary to the path planning process, subject to constraints accounted for separately via UAV, environment, and mission profiles. Results obtained from case studies for a small UAV helicopter with environment data verify the effectiveness of the proposed FPGA-based path planner, and demonstrated convergence at rates above the typical 10 Hz update frequency of an autopilot system. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(a), 3.(a) and 4.

4.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

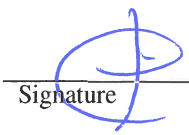
J. Kok, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Transactions on Evolutionary Computation*, vol. 17, no.2, pp. 272–281, April 2013.

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ
Name


Signature

18-6-2014
Date

Chapter 5

Computational Experiments Involving Population Size for FPGA-Based Implementation of a GA for the TSP

In this chapter, the feasibility of using an in-hardware implementation of a GA to solve the computationally expensive TSP is explored, particularly in regard to hardware resource requirements for problem and population sizes. We investigate via numerical experiments whether a small population size might prove sufficient to obtain reasonable quality solutions for the TSP, thereby permitting relatively resource efficient hardware implementation on FPGAs. Experiments were used to explore the extent to which population size can be reduced without compromising solution quality, and results show that a GA allowed to run for a large number of generations with a smaller population size can yield solutions of comparable quality to those obtained using a larger population. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(a), 2. and 3.(b).

5.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

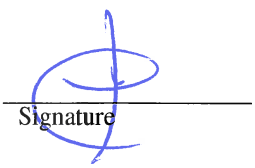
J. Kok, N. A. Kelson, L. F. Gonzalez, and T. S. Bruggemann, "Computational experiments involving population size for FPGA-based implementation of a GA for the TSP," in *Proceedings of the 4th International Conference on Computational Methods (ICCM 2012)*, 2012, pp. 1–6.

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Troy Bruggemann	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ
Name


Signature

18-6-2019
Date

ICCM2012
November 25-27, 2012, Gold Coast, Australia
www.ICCM-2012.org

Computational experiments involving population size for FPGA-based implementation of a GA for the TSP

J. Kok*[1], N. A. Kelson[2], L. F. Gonzalez[1], and T. S. Bruggemann[1]
[1]*Cooperative Research Centre for Spatial Information and Australian Research Centre for Aerospace Automation, Eagle Farm, Queensland, Australia*
[2]*HPC & Research Support Group, Queensland University of Technology, Brisbane, Queensland, Australia*
**Corresponding author: jonathan.kok@student.qut.edu.au*

Abstract

The feasibility of using an in-hardware implementation of a genetic algorithm (GA) to solve the computationally expensive travelling salesman problem (TSP) is explored, especially in regard to hardware resource requirements for problem and population sizes. We investigate via numerical experiments whether a small population size might prove sufficient to obtain reasonable quality solutions for the TSP, thereby permitting relatively resource efficient hardware implementation on field programmable gate arrays (FPGAs). Software experiments on two TSP benchmarks involving 48 and 532 cities were used to explore the extent to which population size can be reduced without compromising solution quality, and results show that a GA allowed to run for a large number of generations with a smaller population size can yield solutions of comparable quality to those obtained using a larger population. This finding is then used to investigate feasible problem sizes on a targeted Virtex-7 vx485T-2 FPGA platform via exploration of hardware resource requirements for memory and data flow operations.

Keywords: Combinatorial optimisation, field programmable gate array, genetic algorithm, hardware acceleration, travelling salesman problem.

Introduction

In the field of combinatorial optimisation, the travelling salesman problem (TSP) is a popular NP-hard and intensively investigated optimisation problem. Formally, the objective of the TSP is to find the Hamiltonian cycle with the least weight in a complete weighted graph (where vertices, edges and weights represent cities, roads and distance of that road, respectively). Applications of the TSP span across diverse fields, from more obvious applications in scheduling, planning and logistics, to more obscure ones such as DNA sequencing and manufacturing of microchips. A comprehensive review of the TSP can be found in the work by Applegate et al. (2006).

Optimisation methods for the TSP fall into two categories of exact and heuristic algorithms. Exact algorithms, such as brute force search and branch-and-cut algorithms, consider all possible ordered combination of cities and find the one solution with the least distance. The running time for such algorithms lies within a polynomial factor of $O(n!)$, where n is the number of cities. For example, a 20 cities problem would have almost 2×10^{18} possible solutions for the algorithm to search through, which can lead to a CPU time consuming process. In view of this, exact algorithms are usually practical only for small problem sizes.

Various heuristic algorithms, such as ant colony optimisation and genetic algorithm (GA), have been devised to find good solutions for larger problem sizes within a reasonable time. Instead of

searching through all possible solutions in the large search space, heuristic algorithms iteratively explore and exploit random regions to search for increasingly better solutions. However, heuristic algorithms that are executed on CPUs may still incur substantial computational run-time due to the sequential processing nature of CPUs. Therefore, hardware implementations of GA for TSP have been proposed in previous works by several authors (Graham and Nelson, 1995; Skliarova and Ferrari, 2002; Vega-Rodriguez et al., 2005; Tachibana et al., 2006; Zhou et al., 2011).

The above works are focused on introducing various implementations of GA features and are limited on the justification of parameters utilised, including population size and problem size, which hinders the extent to which their works may be built upon theoretically and empirically. They also lack in highlighting and describing the growth in resource requirements for accommodating the population size when the problem size is increased. In contrast, this work aims to investigate the relationships between (1) population size and solution quality; and (2) problem size and FPGA resource requirements. The results show that a GA with a smaller population size allowed to run for a large number of generations could obtain solutions of comparable quality to one with a larger population size. This finding allows for relatively resource efficient hardware implementation on FPGAs as a small population size is sufficient to obtain reasonable quality solutions for the TSP.

Genetic algorithm

Evolutionary computation approaches have also been used in the complex field of aerospace for path planning (Gonzalez, 2004) and aerodynamic design (Lee, 2009). GA is a population-based metaheuristic optimisation method built from the principles of biological evolution involving inherited features such as selection, crossover, and mutation (Goldberg, 1989). The fundamental procedure of a GA is illustrated in Fig. 1. The algorithm commences by initialising a parent population with random variables. During each generation, individuals from the parent population are subjected to a selection technique for reproduction. Selected individuals undergo genetic permutation which produces an offspring population. Subsequently, fitness of the offspring population is evaluated. Finally, fitter individuals of the offspring population are introduced back into the parent population. This evolutionary process is repeated until a specified termination criterion is met.

```
Genetic Algorithm

      Initialise parent population
      WHILE (Termination criteria NOT SATISFIED)
            Selection of individuals from parent population for reproduction
            Genetic permutation of individuals to produce offspring population
            Evaluation of offspring population
            Introduction of fitter offspring into parent population
      END-WHILE
```

Figure 1. Pseudo code of genetic algorithm.

Hardware implementation of genetic algorithm for the travelling salesman problem

The GA used for this study is adapted from previous work by Kirk (2007). Each individual in the population is a candidate solution represented by the vertices of a Hamiltonian path (sequence of cities) and its distance. The following experiments are conducted with the ultimate view of hardware implementation, and to this end certain key parameters are chosen as power of two, resources are limited to a given device, and speedup is achievable from parallelism and pipelining design.

Population size and solution quality

The main purpose of this experiment is to study the effects of population size on solution quality. The GA is executed on two benchmark TSPs known as TSP att48 (48 cities) and TSP att532 (532 cities), which can be found in previous work by Reinelt (1991). The population sizes of the GA for each TSP experiment is set to 2^n, $n \in \{1, ..., 8\}$. Fig. 2 and Fig. 3 show the convergence plot of the GA for TSP att48 and TSP att532, respectively. From the plots, two regions of interest (ROI) with notable characteristics are observed. ROI 1 shows that generally a GA with larger population size converges faster than one with smaller population size. On the other hand, ROI 2 shows that a GA even with a small population size when allowed to run for a large number of generations could obtain solutions comparable to one with a larger population size. This finding is noteworthy, given that hardware implementation allows for more generations to be executed within the same time period as its software counterpart. Additionally, a small population size is resource efficient for hardware implementations on FPGAs.

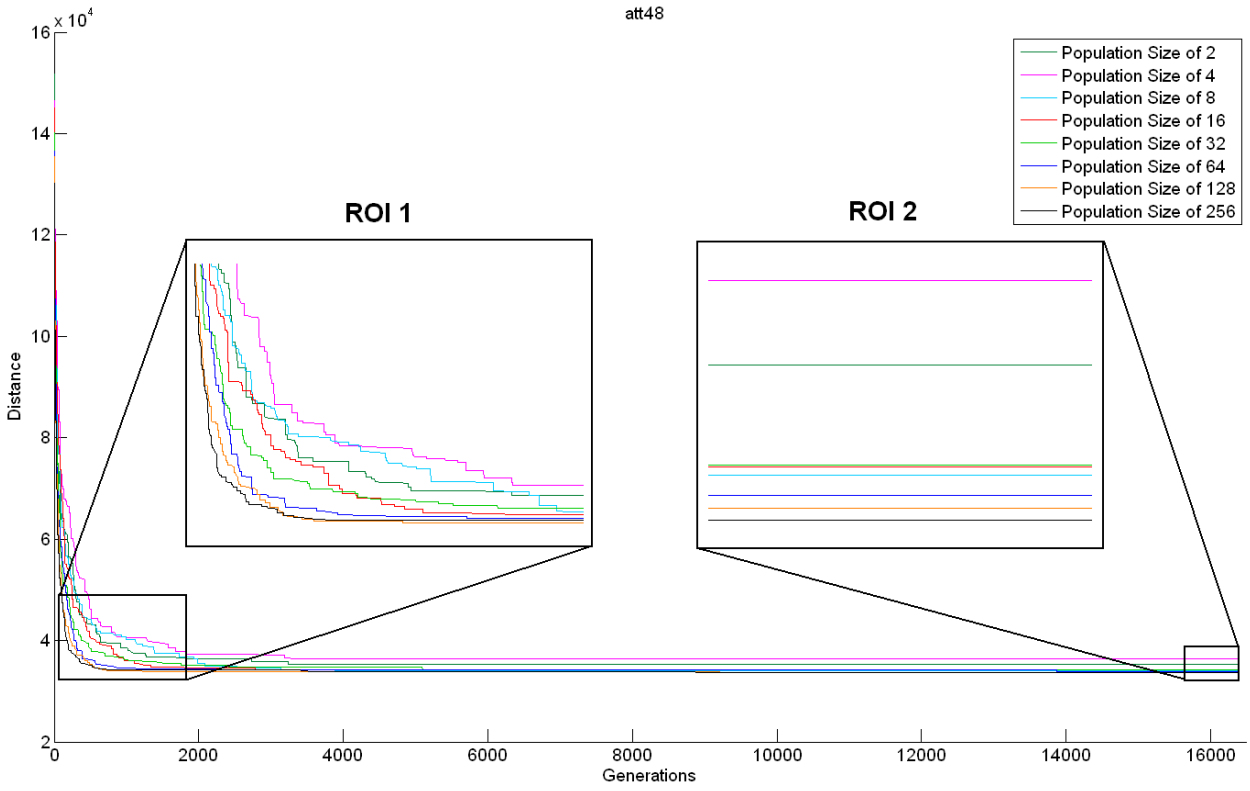


Figure 2. GA convergence result on TSP att48 across different population sizes. ROI 1 shows the early converging characteristics and ROI 2 shows the converged results after a large number of generations.

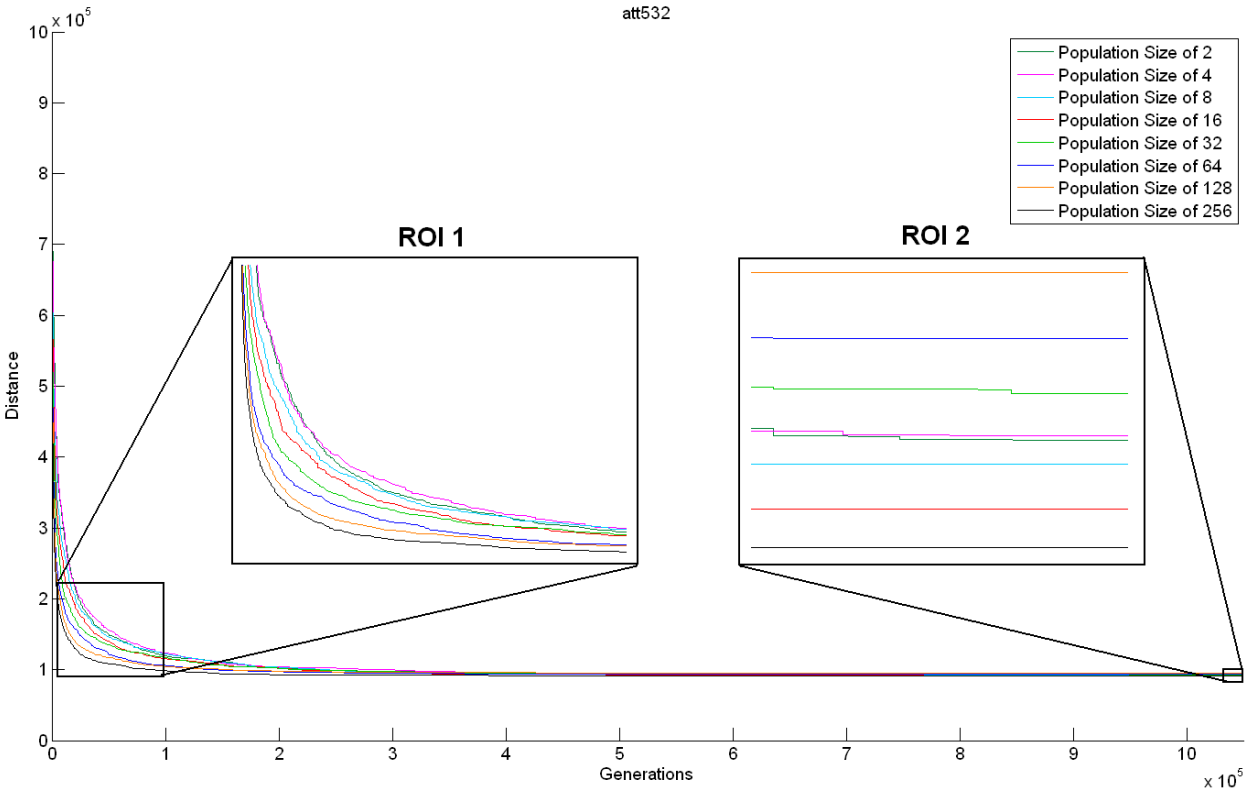


Figure 3. GA convergence result on TSP att532 across different population sizes. ROI 1 shows the early converging characteristics and ROI 2 shows the converged results after a large number of generations.

Problem size and FPGA resource requirements

The representation of individuals within the population of a GA is shown in Fig. 4. Each individual consists of a candidate solution represented by a sequence of cities and its fitness. Note that the encoding size needed for each individual is proportional to the problem size.

Individual:	C_1	C_2	... ...	C_{n-1}	C_n	Fitness

Figure 4. Representation of individuals. C_1 to C_n represents the sequence of cities for a problem size of *n* cities.

The aim of this experiment is to investigate and estimate the maximum problem size implementable for a given GA population size and FPGA architecture. Based on the above results, the population size of 2^n, $n \in \{3, ..., 6\}$ and problem size of 2^m, $m \in \{4, ..., 9\}$ was chosen. Having access to a Virtex-7 FPGA VC 707 evaluation kit, the FPGA model of Virtex-7 VX485T-2 was chosen. The hardware implementation of a GA on FPGA architecture based on previous work by Kok et al. (2012) was chosen. All internal operators were not implemented; only memory and data flow of the GA population were implemented. Note that speedup and effectiveness of implementing a GA on FPGAs have already been proven by previous authors (Graham and Nelson, 1995; Skliarova and Ferrari, 2002; Vega-Rodriguez et al., 2005; Tachibana et al., 2006; Zhou et al., 2011), and therefore

the intent of this work is not to reproduce a proven concept but rather investigate the issues related to the implementation of population-based metaheuristics on a fixed-architecture. Parameters were synthesised with Xilinx ISE 14.1. The parameters and synthesis results are tabulated in Table 1 below. From the results, a population size of above 64 will not be an ideal choice for hardware implementation on FPGAs as the maximum problem size it can address would be too small to be practical.

Table 1. Synthesis results of implementing a GA population size of 16 for different problem sizes.

Population size (number of candidate solutions in the GA)	Problem size (maximum number of cities)	Estimated resource requirements (Percentage of device utilised)
$2^3 = 8$	$2^4 = 16$	2%
$2^3 = 8$	$2^5 = 32$	6%
$2^3 = 8$	$2^6 = 64$	12%
$2^3 = 8$	$2^7 = 128$	30%
$2^3 = 8$	$2^8 = 256$	75%
$2^3 = 8$	$2^9 = 512$	164%
$2^4 = 16$	$2^4 = 16$	5%
$2^4 = 16$	$2^5 = 32$	12%
$2^4 = 16$	$2^6 = 64$	26%
$2^4 = 16$	$2^7 = 128$	61%
$2^4 = 16$	$2^8 = 256$	154%
$2^4 = 16$	$2^9 = 512$	>> 100 %
$2^5 = 32$	$2^4 = 16$	11%
$2^5 = 32$	$2^5 = 32$	24%
$2^5 = 32$	$2^6 = 64$	55%
$2^5 = 32$	$2^7 = 128$	122%
$2^5 = 32$	$2^8 = 256$	>> 100 %
$2^5 = 32$	$2^9 = 512$	>> 100 %
$2^6 = 64$	$2^4 = 16$	22%
$2^6 = 64$	$2^5 = 32$	49%
$2^6 = 64$	$2^6 = 64$	109%
$2^6 = 64$	$2^7 = 128$	>> 100 %
$2^6 = 64$	$2^8 = 256$	>> 100 %
$2^6 = 64$	$2^9 = 512$	>> 100 %

Conclusions

Implementing a population-based metaheuristics on a fixed-architecture, such as a GA on FPGA, has been proven to be a viable concept. However, limitations associated with the population size have not been addressed explicitly in literature. In this work, the relationships between (1) population size and solution quality; and (2) problem size and FPGA resource requirements are investigated. From the results and with the ultimate aim of a fixed-architecture implementation (specifically the Virtex-7 FPGA), a small population size is shown to be sufficient to obtain reasonable quality solutions for the TSP, thereby permitting relatively resource efficient hardware implementations on FPGAs. Also, population sizes of above 64 will not be practical as they can only accommodate to small problem sizes of below 16, which is not ideal in the real-world where

problem sizes are large by nature. Future work will explore avenues to address larger problem sizes of above 1,000 cities for hardware implementation on FPGAs.

Acknowledgement

The work has been supported by the Cooperative Research Centre for Spatial Information, whose activities are founded by the Australian Commonwealth's Cooperative Research Centres Programme.

References

Applegate, D. K., Bixby, R. E., Chvatal, V. and Cook, W. J. (2006), *The Travelling Salesman Problem: A Computational Study*. United Kingdom: Princeton University Press.

Goldberg, D. E. (1989), *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley Professional.

Gonzalez, L. F., Whitney, E. J., Periaux, J., Sefrioui, M. and Srinivas K. (2004), A robust evolutionary technique for inverse aerodynamic design. *Design and Control of Aerospace Systems Using Tools from Nature*, 2, pp. 24-28.

Graham, P. and Nelson, B. (1995), A hardware genetic algorithm for the travelling salesman problem on splash 2. In *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, pp. 352-361.

Kirk, J. (2007), MATLAB code for travelling salesman problem – genetic algorithm. *MATLAB Central*. [Online]. Available: www.mathworks.com/matlabcentral/fileexchange/13680.

Kok, J., Gonzalez, L., and Kelson, N. (2012), FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning. *IEEE Transactions on Evolutionary Computation*, DOI: 10.1109/TEVC.2012.2192124.

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I. and Dizdarevic, S. (1999), Genetic algorithms for the travelling salesman problem: a review of representations and operators. *Artificial Intelligence Review*, 13 (2), pp. 129-170.

Lee, D., Gonzalez, F., Periaux, J. and Srinivas, K. (2009), Evolutionary optimisation methods with uncertainty for modern multidisciplinary design in aeronautical engineering. *Notes on Numerical Fluid Mechanics*, 100, pp. 271-284.

Michalewicz, Z. (1996), *Genetic Algorithms + Data Structures*, Springer-Verlag.

Skliarova, I. and Ferrari, A. (2002), FPGA-based implementation of genetic algorithm for the travelling salesman problem and its industrial application. In *Proceedings of the 15th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Developments in Applied Artificial Intelligence*, pp. 77-87.

Tachibana, T., Murata, Y., Shibata, N., Yasumoto, K. and Ito, M. (2006), Flexible implementation of genetic algorithms on FPGAs. In *Proceedings of the 14th International Symposium on Field Programmable Gate Arrays*, pp. 236.

Vega-Rodriguez, M. A., Gutierrez-Gil, R., Avila-Roman, J. M., Sanchez-Perez, J. M. and Gomez-Pulido, J.A. (2005), Genetic algorithms using parallelism and FPGAs: the TSP as case study. In *Proceedings of the 2005 International Conference Workshops on Parallel Processing*, pp. 573- 579.

Zhou, Y., Gu, J., Dong, Y. and Han, H. (2011), Implementation of genetic algorithm for TSP based on FPGA. In *Proceedings of the 2011 Chinese Control and Decision Conference*, pp. 2226-2231.

Chapter 6

FPGA Implementation of a Micro-Genetic Algorithm for Solving the Travelling Salesman Problem

In this chapter, an FPGA-based micro-GA for solving the TSP adaptation is proposed and described. The micro-GA is a well-suited optimisation algorithm for parallel execution via hardware implementation, mainly due to the low resource requirements from its small population size and the independent operation of its individuals during the evolution process. The proposed architecture aims to solve TSPs at a faster rate than software algorithms using a micro-GA implemented entirely on an FPGA device. The architecture of the proposed design is simple and modular for modifications and extensions to suit the preference of a given application. Results obtained from experiments with benchmark TSPs verify the effectiveness of the proposed FPGA-based TSP solver. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(a), 3.(b) and 4.

6.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

J. Kok, T. S. Bruggemann, L. F. Gonzalez, and N. A. Kelson, "FPGA implementation of a micro-genetic algorithm for solving the travelling salesman problem," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2013 (Submitted).

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Troy Bruggemann	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ — Name

Signature

18-6-2014 — Date

FPGA Implementation of a Micro-Genetic Algorithm for Solving the Travelling Salesman Problem

Jonathan Kok, *Member, IEEE,* Troy Bruggemann, Felipe Gonzalez, and Neil Kelson

***Abstract*—In this paper, a hardware-based micro-genetic algorithm (micro-GA) architecture for the travelling salesman problem (TSP) adaptation is proposed and described. The micro-GA is a well-suited optimisation algorithm for parallel execution via industrial hardware electronics implementation, mainly due to the low resource requirements from its small population size and the independent operation of its individuals during the evolution process. The proposed architecture aims to solve TSPs at a faster rate than software algorithms using a micro-GA implemented entirely on a field programmable gate array (FPGA) device. The architecture of the proposed design is simple and modular for modifications and extensions to suit the preference of a given application. The design has been successfully synthesised and implemented on a Xilinx Virtex-7 FPGA device with very low device utilisation allowing opportunities for algorithm extensions. Results obtained from experiments with benchmark TSPs verify the effectiveness of the proposed FPGA-based TSP solver and demonstrated an average speedup of 70 times faster when compared to an equivalent software-based micro-GA TSP solver and 26 times faster when compared to the powerful Concorde TSP solver.**

***Index Terms*—Field programmable gate array, heuristics, micro-genetic algorithm, travelling salesman problem.**

I. Introduction

IN the field of combinatorial optimisation, the travelling salesman problem (TSP) has been intensively investigated [1]. The TSP has its use in a wide range of practical applications, such as manufacturing of microchips [2], testing of printed circuit boards [3], cold rolling scheduling [4], automation of disassembly system [5], and routing of shop floor logistics [6]. Formally stated, the objective of the TSP is to find the Hamiltonian cycle with the least weight in a complete weighted graph (where vertices, edges and weights represent cities, roads and distance of roads, respectively).

In terms of computational complexity theory, the TSP belongs to the class of NP-complete problems such that an optimal solution for even moderate size problem can be intractable to solve [7]. Hence, heuristics have been proposed for yielding good solutions within reasonable time. Heuristics such as genetic algorithms (GAs) explore and exploit random regions of the search space for increasingly better solutions instead of performing an extensive analysis on all possible solutions. A comprehensive survey of the TSP in conjunction with exact algorithms and heuristics for solving it can be found in [8], while a detailed literature review of chromosome representation and genetic operators for GAs can be found in [9].

As effective as heuristics are, they still require considerable computational run-time for software-based execution. Therefore, hardware implementations on field programmable gate arrays (FPGAs) of heuristics such as ant colony optimisation [10], particle swarm optimisation [11], neural network algorithm [12], [13], DNA algorithm [14], and GA [15], [16] have been proposed. On interest here is the GA which is potentially well suited for hardware implementation as its population-based nature and independent genetic operations allows for individuals to be handled concurrently. Moreover, it is noteworthy that the GA has also been argued to be one of the best available heuristics for solving the TSP [17].

In view of the potential advantages, a few studies have explored the application of GAs for solving the TSP problem using FPGAs [18]–[24]. Very early work [18] attempted a hardware implementation of a GA for solving the TSP on the Splash reconfigurable computing platform. However, due to the limited reconfigurable logic resources on FPGAs at that time, separate GA operations had to be distributed amongst four FPGAs for execution. Note that subsequent advances in FPGA technology now feasibly allow for an entire GA system, such as a micro-GA, to be implemented entirely on a single FPGA, as is undertaken here. Later work [19] presented and implemented a crossover operator on FPGA for solving the TSP. In [20], alternative versions of a GA for FPGA implementation were developed using Handel-C (a high level C like programming language that targets low-level hardware instantiation). However, in contrast to the present work, no hardware-based architecture was actually proposed, as the focus of that study was on development efforts aimed at improving the Handel-C encoding. In [21], a basic island GA architecture suitable for hardware implementation including management, crossover, mutation, and evaluation modules was proposed. Parallel implementation of the individual modules was not considered and instead sought via the inclusion of an immigration module to oversee periodic exchange of individuals between separate concurrently running GAs. Their approach also involved implicit handling of selection and update phases through a management module, as opposed to the present approach which explicitly considers distinct selection and

J. Kok, T. Bruggemann and F. Gonzalez are with the Cooperative Research Centre for Spatial Information and the Australian Research Centre for Aerospace Automation, Brisbane Airport, Queensland 4007 Australia (e-mail: j.kok@qut.edu.au; t.bruggemann@qut.edu.au; felipe.gonzalez@qut.edu.au).

N. Kelson is with the High Performance Computing and Research Support Group, Division of TILS, Queensland University of Technology, Brisbane, Queensland 4000 Australia (e-mail: n.kelson@qut.edu.au).

update phases to allow for greater modularity. In [22], the design and hardware implementation of a parameterised GA on an FPGA was proposed. A control module was implemented to synchronise operations rather than developing on possible parallelism aspects of the hardware implementation. Although insufficient design details were provided, two more recent proposals for a steady-state GA on FPGA [23] and a pipelining structure for GA [24] can also be mentioned.

In contrast to the previously noted GA-based studies, a framework for FPGA implementation of a micro-GA for solving the TSP is proposed in this work. While functionally similar to a traditional GA, the micro-GA acts on a smaller population size and re-initialises the population when some level of convergence is reached [25]. Micro-GAs have been shown to be as effective as a conventional GA with a large population size [26]. Also, the small population size feature of the micro-GA is beneficial for hardware implementation as it uses comparatively less hardware device resources than a conventional GA, thereby allowing for e.g. a relatively greater amount of unused FPGA logic resources to be instead employed for future extensions or updates to an existing architecture. In the present approach, the input parameters of the micro-GA include a problem-specific cost matrix instead of a fitness module as used in the previous GA-based studies mentioned above - a potentially desirable feature in the FPGA implementation as it allows the core design to be reusable for different problem types without the need to re-design and re-synthesise the cost function within the evaluation module on the FPGA. Regarding parallelism, modules are designed and implemented such that individuals within each module are processed concurrently whenever possible to reduce processing time. Genetic operators, namely the partially mapped crossover (PMX) [27] and the simple inversion mutation (SIM) [28] are proposed here along with new hardware suitable arithmetic logic structures for each that are amenable to parallel processing. Note that the SIM, when applied with the survival of the fittest scheme within the overview of the micro-GA, creates a phenomenon similar to that of the widely used 2-opt local search algorithm [29]. Therefore, the proposed SIM design and implementation allows for the coupling of the 2-opt. Finally, the overall micro-GA design architecture adopts a modular approach throughout to facilitate ease of integration with other customised function modules to suit the preference of a given application.

The novel contributions of this paper are as follows:

1) A new FPGA implementation method for a micro-GA is proposed and applied for solving the TSP, speeding up the computation process. The proposed design is modular which allows for implementation and testing of different choices for each of the genetic operators or even a repository of the micro-GA solvers for related applications.
2) New hardware suitable arithmetic logic structures for the PMX and SIM are developed and deployed. The operators are designed with parallel processing attributes. The SIM is implemented to allow for the coupling of the widely used 2-opt local search algorithm.
3) A new cost matrix input parameter approach to an FPGA-based TSP solver is proposed, allowing the solver to handle different TSP types without the need for re-designing and re-synthesising of the FPGA.

The rest of the paper is organised as follows. Section II presents the TSP problem formulation. Section III gives a description of the features necessary for a micro-GA to effectively handle the combinatorial characteristics of the TSP. Section IV describes in detail the proposed micro-GA architecture and the FPGA implementation aspects of each module. Section V reports on the simulations and experiments conducted. Finally, Section VI concludes with a brief summary and future work.

II. The Travelling Salesman Problem

The travelling salesman problem can be formulated in terms of graph theory as a complete graph $G = (V, E)$, where the set of vertices $V = \{1, ..., n\}$ represent the cities and the set of edges $e_{ij} \in E$ represents the cost between cities i and j. Given this problem statement, the goal of the TSP is to find a Hamiltonian cycle (i.e., each vertex is visited exactly once) with the least cost in the graph G. Any algorithm solving the TSP has to search through and compare all $(n-1)!$ possible solutions which can result in a computationally intensive process. For example, a 20-city TSP would have to consider almost 2×10^{18} possible solutions.

III. Micro-Genetic Algorithm for TSP

As earlier noted, the Micro-GA is a population-based metaheuristic optimisation algorithm using techniques inspired by the principles of biological evolution, such as selection, crossover, mutation, and reproduction [30]. The fundamental procedure of a micro-GA is illustrated in Fig. 1 and can be described as follows. The algorithm commences by initialising a parent population with random variables. During each generation, individuals from the parent population are subjected to a selection technique for reproduction. Selected individuals undergo genetic permutation which produces an offspring population. Subsequently, fitness of the offspring population is evaluated. Finally, fitter individuals of the offspring population are introduced back into the parent population. This evolutionary process is repeated until a specified termination criterion is met.

The following subsections describe the representation of individuals and the genetic operators used in the proposed micro-GA design for the TSP problem being considered here.

A. Representation of Individuals

An *individual* refers to a candidate TSP path solution within the micro-GA which is stored and permutated by the evolutionary process, and for which an encoding is needed to represent properties of a feasible solution. Of the various different representations such as binary, adjacency, ordinal, matrix, and path which have been proposed to solve the TSP [17], the real path representation has been used here. For this representation, cities are indexed as a list and the encoding

```
BEGIN Proposed micro-GA
  Initialise population
  WHILE Termination criteria NOT SATISFIED
    IF Convergence criteria SATISFIED
      Re-initialisation of offspring population
    END-IF
    Selection of individuals from parent population for reproduction
    Genetic permutation of individuals to produce offspring population
    Evaluation of offspring population
    Introduction of fitter offspring into parent population
  END-WHILE
  Output best individual
END Proposed micro-GA
```

Fig. 1. Pseudo code of proposed micro-genetic algorithm.

for each individual is in the form of a vector representing a sequence of cities to visit. For example, the vector [1 17 3] represents the path where cities with corresponding indexes 1, 17 and 3 are visited in the order given. Note that real rather than binary encoding has been employed here, as the latter requires special procedures and additional repair functionality for Hamiltonian cycle validation and genetic operators which have been argued to produce poor performance [31].

B. Genetic Operators

The micro-GA utilises two types of genetic operators, namely, crossover and mutation. In principle, the crossover operator takes two parent path solutions and exchanges characteristics between them, resulting in two new offspring path solutions with shared traits. In contrast, the mutation operator acts on a single parent path solution by inducing a small permutation and creating one new offspring path solution with minor variations. Note that the genetic operators are dependent on the application and the representation used. For the TSP problem, the operators must be specifically designed to e.g. maintain the path solution validity of a Hamiltonian cycle.

1) Partially-Mapped Crossover: As a suitable crossover operator for the TSP, the partially mapped crossover (PMX) operator [27] was chosen, whereby ordering and value information from parent path solutions are passed on to offspring path solutions. This is achieved by mapping a portion of one parent onto a portion of the other and exchanging the remaining value information.

As an illustration of the PMX, consider the following path solutions

parent 1: [1 2 3 4 5 6 7 8] and
parent 2: [2 4 6 8 1 3 5 7]

where initially the portion for mapping is randomly selected to be between the fourth and sixth elements of both parent solutions. This would define the bidirectional mapping $4 \leftrightarrow 8$, $5 \leftrightarrow 1$, $6 \leftrightarrow 3$. The mapping portions of the parent are then copied into the offspring solutions, resulting in

offspring 1: [$*$ $*$ $*$ $|4\ 5\ 6|$ $*$ $*$ $*$] and
offspring 2: [$*$ $*$ $*$ $|8\ 1\ 3|$ $*$ $*$ $*$].

Subsequently, offspring 1 is consecutively filled up with elements copied from parent 2 and vice versa. If an element that is to be copied over is already present in the offspring, then it is replaced according to the mapping mentioned above. In this example, the first and eighth elements of offspring 1, values 2 and 7, are copied directly from parent 2 as they are absent in offspring 1. However, the second, third and seventh elements of parent 2 are already present in offspring 1, therefore by referring to the mapping specified above, the second, third and seventh elements to be copied onto offspring 1, values 4, 6 and 5, are replaced with values 8, 3 and 1, respectively. Finally, applying the same principles to offspring 2 results in

offspring 1: [2 8 3 $|4\ 5\ 6|$ 1 7] and
offspring 2: [5 2 6 $|8\ 1\ 3|$ 7 4].

Note that the absolute ordering of some elements in both parents is passed on to the offspring solutions.

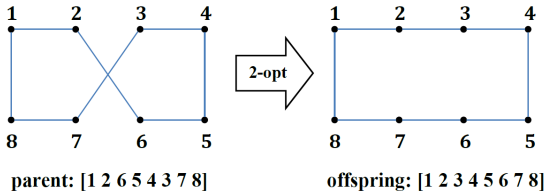


Fig. 2. Example of 2-opt operation.

2) 2-Opt: As a suitable mutation operator for the TSP, the 2-opt search algorithm [28] was chosen for implementation. The aim of this operation is to unknot two sub-tours within a path solution into a single shorter tour (see Fig 2). This is achieved by randomly selecting a portion of the path solution and reversing its order.

As an example of the 2-opt operator, consider the parent path solution

parent: [1 2 6 5 4 3 7 8],

where the randomly selection portion between elements three and six is chosen. After reversal, this results in

offspring: [1 2 4 5 6 7 8].

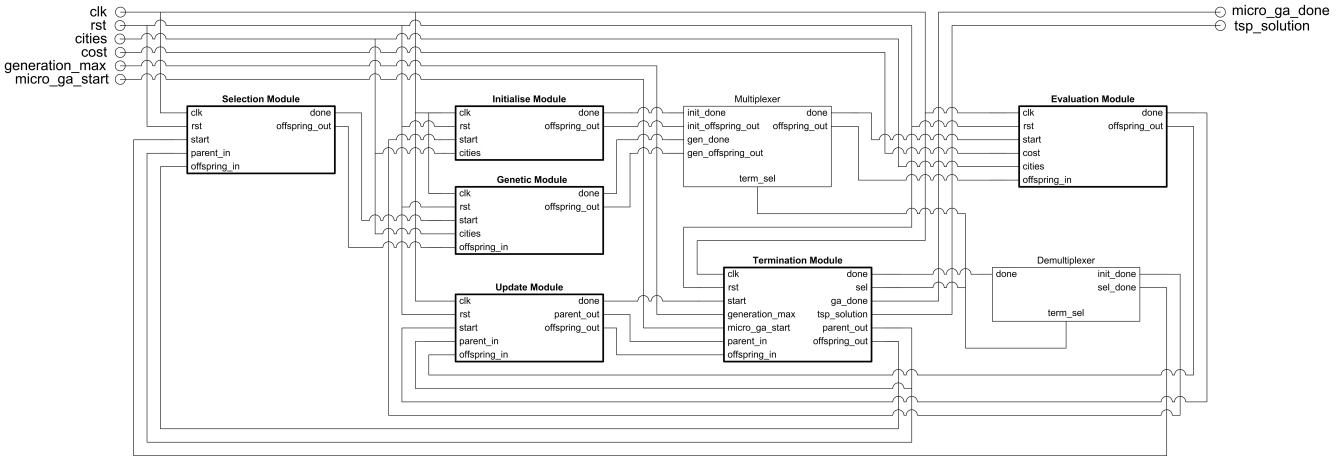


Fig. 3. Schematic of the proposed micro-GA architecture.

IV. Micro-GA Hardware Implementation Details

A. Architecture

The proposed FPGA hardware-based micro-GA architecture is depicted in Fig 3. The inputs are the clock, reset, number of cities, cost matrix of the problem, maximum number of generations, and micro-GA start signals. Upon satisfaction of the termination criteria, the system outputs a done flag and the TSP solution. In addition to these, the proposed architecture utilises six operational modules (initialise, selection, genetic, evaluation, update, and termination) along with two multiplexers used to alter the data flow for the purpose of re-initialisation. Each module is controlled by a Mealy finite-state machine [32] which manages the data and parallel execution of internal operations. All modules except the termination module have the same process flow (i.e. read input, parallelised operations, and write output), and all include memory resources that receive, transmit and store the entire micro-GA population made up of equal sized parent and offspring populations. Note that a random number generator (RNG) is used in the initialise, selection and genetic modules, and for this purpose the 32-bits Mersenne Twister RNG is implemented as recommended by Matsumoto and Nishimura [33]. The chosen RNG has a uniform distribution and a large prime period of $2^{19937} - 1$, which effectively results in long pseudo-random sequences with no repetition. The following subsections describe further each of the modules.

B. Termination Module

The *termination module*, which runs after the update module has completed, is the core module that controls the overall process flow of the micro-GA by monitoring the status of the termination and convergence criteria. The termination criterion used here terminates the algorithm when a user defined maximum generation count has been reached, and to this end the module uses an internal counter for the current generation count to compare with the maximum generation count at every generation. A terminate flag is triggered when the termination criterion is satisfied, whereby the micro-GA stops and the best individual in its current population is output. In contrast, the micro-GA does not terminate but instead re-initialises the offspring population when the convergence criterion is satisfied. Here convergence is met when both parent and offspring have the same fitness score, indicating no further improvements can be made in its current state. When convergence occurs, a re-initialise flag is triggered which is connected to the selector input of the multiplexers, causing the algorithm to proceed with the initialise module instead of the usual continuation with the selection module. The overall process flow of the termination module is illustrated in Fig 4.

C. Initialise Module

The *initialise module* only runs when triggered after the termination module has completed. It re-initialises all individuals of the offspring population when convergence is satisfied. The only inputs are the *start* flag and number of *cities*, which when triggered cause the module to generate and output a new random sequence of cities for each individual in the offspring population.

D. Selection Module

The *selection module* runs after the termination module has completed with re-initialise flag not triggered. The module implements the effective tournament selection technique [34] in which pairs of individuals throughout the entire population are randomly chosen for their respective fitness to be compared against. The random numbers used for selecting the individuals are generated by an internal RNG, as noted earlier. Subsequently, the better individuals (the ones with smaller cost association) are copied and stored onto the output offspring population. The selection operation thereby promotes reproduction from fitter individuals through which useful information is preserved and exploited.

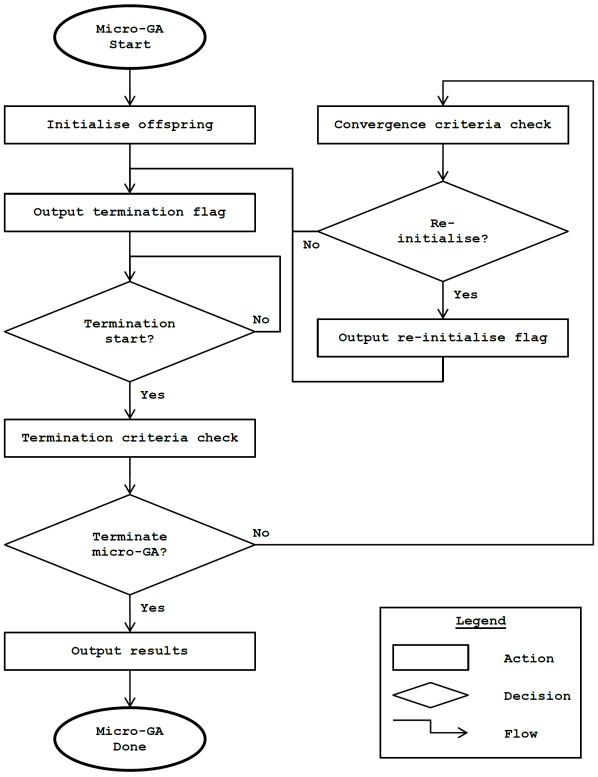


Fig. 4. Process flow of the termination module controlling the micro-GA.

```
BEGIN PMX
Step 1:
  a = RNG(cities)
  b = RNG(cities)
Step 2:
  IF (b < a) THEN
    a = b
    b = a
  END-IF
Step 3:
  c = a
Step 4:
  FOR n IN 1 TO cities LOOP
    IF ((n < a) OR (b < n)) THEN
      solution_1(n) = solution_1_old(c)
      solution_2(n) = solution_2_old(c)
    ELSE
      solution_1(n) = solution_2_old(c)
      solution_2(n) = solution_1_old(c)
    END-IF
  END-FOR
Step 5:
  FOR n IN 1 TO cities LOOP
    IF ((n < a) OR (b < n)) AND
      (solution_1(n) = solution_1(c)) AND
      (n <= cities) THEN
        solution_1(n) = solution_1_old(c)
    END-IF
    IF ((n < a) OR (b < n)) AND
      (solution_2(n) = solution_2(c)) AND
      (n <= cities) THEN
        solution_2(n) = solution_2_old(c)
    END-IF
  END-FOR
Step 6:
  IF (c < b) THEN
    c = c + 1
    GO TO Step 5
  END-IF
END PMX
```

Fig. 5. Steps for the PMX operation.

E. Genetic Module

The *genetic module* outputs permuted offspring individuals to discover possible better solutions. It runs after the selection module has completed. Half of the input offspring are designated for PMX and the other half for 2-opt. As mentioned above, the objective of the PMX is to share and exchange useful sequence of cities from two individuals, whereas the objective of the 2-opt is to unknot an individuals path solution that crosses over itself. To the authors' knowledge, synthesisable hardware descriptions of these arithmetic logic structures have not been previously reported, and therefore new proposals for hardware implementation of the PMX and 2-opt have been developed for this work, as detailed below.

1) Partially-Mapped Crossover: The algorithmic description of the PMX operation suitable for FPGA hardware based implementation is illustrated in Fig 5. The overall PMX process is controlled by a Mealy finite-state machine where each step representing each state is executed in a single clock cycle, as follows. *Steps 1 & 2* randomly select and order two numbers between 1 and the maximum number of *cities* to be used as the portion for mapping. *Step 3* initialises a counter to keep track of mapped states, *Step 4* copies the portions for mapping across the two solutions, and *Step 5* checks and maps the appropriate portions. Finally, *Step 6* ensures that all mapping within the selected portion is executed. Effectively, the *for loop* in hardware is executed concurrently, hence a minimum of six clock cycles up to a maximum of six plus the number of cities clock cycles are required to complete this operation.

2) SIM: The steps for the SIM operation are shown in Fig 6. The overall SIM process is controlled by a Mealy finite-state machine akin to that of the PMX operation. The steps carried out by the SIM are as follows. *Steps 1 & 2* randomly select and order two numbers between 1 and the maximum number of *cities* to be used as the portion for swapping. *Step 3* swaps the positions of cities within the portions for swapping. The total clock cycles required to complete this operation is always four. Functioning in conjunction with the update module, which ensures the survival of the fittest individual, the SIM forms the basis for a 2-opt local search process within the micro-GA.

F. Evaluation Module

The *evaluation module* runs either after the initialise module or the genetic module is completed, as determined by the selector input triggered by the convergence criteria. This module uses the cost matrix input to the TSP, which stores the cost associated with every possible pair of cities (i.e. for a problem with N cities, the cost matrix is of size NxN). The module calculates and stores the fitness of the genetically permutated offspring population by accumulating the total cost associated with a given solution. The steps performed by the evaluation module are illustrated in Fig 7. Note that the total

```
BEGIN SIM
Step 1:
  a = RNG(cities)
  b = RNG(cities)
Step 2:
  IF (b < a) THEN
    a = b
    b = a
  END-IF
Step 3:
  FOR n IN 1 TO cities LOOP
    IF ((a <= n) AND (n <= b)) THEN
      solution(n) = solution_old(cities-n)
    END-IF
  END-FOR
END SIM
```

Fig. 6. Steps for the SIM operation.

```
BEGIN Evaluation
Step 1:
  fitness = 0
  temp = 0
  a = 1
Step 2:
  temp = cost(solution(a),solution(a+1))
Step 3:
  fitness = fitness + temp
Step 4:
  IF (a < cities-1) THEN
    a = a + 1
    GO TO Step 2
  END-IF
Step 5:
  temp = cost(solution(1),solution(cities))
Step 6:
  fitness = fitness + temp
END Evaluation
```

Fig. 7. Steps for the evaluation operation.

clock cycles required to complete the evaluation is always three times the number of cities. Handling the evaluation process via a cost matrix input parameter instead of the typical problem-related cost function, allows the proposed FPGA-based micro-GA architecture to be robust for solving different types of TSPs without *a priori* knowledge and the need to update a suitable cost function.

G. Update Module

The *update module* runs after the evaluation module is completed to integrate the genetically altered offspring into the parent population and to promote elitism. The parent and offspring populations are concatenated into a single population which is then sorted according to their fitness levels for later use to ensure that fitter individuals always remain in the parent population at the start of subsequent generation cycles. The sorting algorithm implemented is the bubble sort which comes with the advantages of simple coding and algorithmic structure with only one auxiliary memory space requirement, and for a population size n with a deterministic computational complexity of $O(n^2)$ [35]. As the population size of the micro-GA is small, this computational complexity does not significantly impact on the overall evolutionary process.

V. Results

A. Implementation Details

The performance of the proposed system was compared against both its software counterpart and the Concorde TSP solver [36] which has been argued to be one of the best and fastest TSP solvers currently available [37]. The software counterpart of the micro-GA TSP solver was implemented on an Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33GHz with 3.49 GB of RAM. The three main parameters to setup before synthesising the design are the population size of the micro-GA, the maximum possible number of generations, and the maximum number of cities for the synthesised implementation to consider, which are set to $2^4 = 16$, $2^{21} = 2,097,152$ and $2^8 = 256$, respectively. Both the software and hardware micro-GA implementations used the same aforementioned algorithm parameters.

The overall design was synthesised by the Xilinx ISE 14.3 with the Xilinx Virtex-7 XC7VX485T FPGA as the target device. The design goal was set to "balanced, which implies that no optimisation for speed or utilisation of FPGA resources was considered. Once the design was synthesised successfully, it was then compiled and built for implementation. This process consists of translating, mapping, placing and routing of the signals. For the design implementation process, no partition was specified and the design was translated and mapped successfully. All signals were placed and routed successfully as well, and all timing constraints were met. The breakdown of the device utilisation is as follows. 31855 slice registers (5% of available), 21353 slice LUTs (7% of available), 3145 LUT-FF pairs (3% of available), and 13 block RAM (1% of available) of the total resources were used. The overall FPGA design had a maximum operating frequency of 169.436 MHz. The design has low resource footprint allowing further opportunities for algorithm extensions. A top level UART module was implemented as a wrapper (see e.g., Chu [38]) for the proposed micro-GA hardware implementation to communicate between a PC and the FPGA device. Note that the proposed methodology should be suitable for implementation on any other FPGA target devices.

B. Experiments

Benchmark TSP solutions from the TSPLIB [39] were used to evaluate the effectiveness of the proposed FPGA-based micro-GA TSP solver. The results of both software and hardware micro-GA implementations for each of the experiments were averaged over 100 runs. The results include the number of generations and the run-time it took to solve the given TSP. Note that the TSPs are always solved to the optimum solution by both the hardware and software micro-GA implementations. Additionally, the TSP solutions from the Concorde TSP solver are included.

C. Results

Table I shows the results for the experiments and the speedup the FPGA-based implementation had over both Concorde and the software-based micro-GA TSP solver. The results of the

TABLE I
RESULTS OF THE PROPOSED FPGA-BASED MICRO-GA IMPLEMENTATION.

	Concorde TSP solver	micro-GA for TSP					
		Software implementation		FPGA-based implementation			
Benchmark TSP	Run-time (ms)	Generations to solve	Run-time (ms)	Generations to solve	Run-time (ms)	Speedup over Concorde	Speedup over software
burma14	20	297	193	108	1.2	16.7	160.8
ulysses16	120	656	357	289	4.1	29.3	87.1
ulysses22	290	863	442	518	11.8	24.6	37.4
ch130	650	1226	804	1036	20.13	32.0	39.6
ch150	910	1789	1126	1354	29.5	30.8	28.5

Concorde TSP solver were taken from [36]. The software implementation of the micro-GA performed computationally slower than the Concorde TSP solver. The proposed FPGA-based implementation of the micro-GA was on average 70 times faster than its software counterpart and 26 times faster than the Concorde TSP solver. It can be seen that the computational run-times for all the TSP solving methods were relative to the given problem difficulty. The effectiveness of the FPGA-based implementation remained consistent across the different TSPs.

VI. CONCLUSIONS

In this paper, a robust FPGA-based micro-GA for solving TSPs is proposed and described. The micro-GA is a well-suited optimisation algorithm for hardware implementation, mainly due to its low-resource requirement to perform effectively. The architecture of the proposed design is simple and modular for modifications and extensions. It has been successfully implemented and evaluated using benchmark TSPs without *a priori* knowledge. The proposed FPGA-based implementation performed on average 70 times faster when compared to an equivalent software-based micro-GA TSP solver and 26 times faster when compared to the powerful Concorde TSP solver. While this work is focused on TSPs, the proposed approach has the potential to be more widely applicable in other optimisation applications requiring efficient and effective optimisation techniques. Future work will involve the investigation of partitioning decisions (e.g. vehicle routing problem, bin-packing) for large-scale TSP to be split into manageable sub-problems in which a single or multiple micro-GA, whose input is controlled by a high-level partitioning decision maker, is able to solve large-scale TSPs for implementation in more complex practical applications.

ACKNOWLEDGMENTS

The work has been supported by the Cooperative Research Centre for Spatial Information with funding provided by the Australian Commonwealth Cooperative Research Centres Programme.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: A computational study*, ser. Princeton Series in Applied Mathematics. Princeton University Press, 2011.

[2] R. Kumar and Z. Luo, "Optimizing the operation sequence of a chip placement machine using TSP model," *IEEE Transactions on Electronics Packaging Manufacturing*, vol. 26, no. 1, pp. 14–21, 2003.

[3] A. Gonzalez-Rodriguez and A. Gonzalez-Rodriguez, "Collision-free motion planning and scheduling," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 3, pp. 657–665, 2011.

[4] J. Zhao, Q. Liu, W. Wang, Z. Wei, and P. Shi, "A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling," *Information Sciences*, vol. 181, no. 7, pp. 1212–1223, April 2011.

[5] I. Eguia, S. Lozano, J. Racero, and F. Guerrero, "A methodological approach for designing and sequencing product families in reconfigurable disassembly systems," *Journal of Industrial Engineering and Management*, vol. 4, no. 3, pp. 418–435, 2011.

[6] E. K. Xidias, A. C. Nearchou, and N. A. Aspragathos, "Integrating path planning, routing, and scheduling for logistics operations in manufacturing facilities," *Cybernetics and Systems*, vol. 43, no. 3, pp. 143–162, 2012.

[7] C. H. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977.

[8] G. Laporte, "A concise guide to the traveling salesman problem," *Journal of the Operational Research Society*, vol. 61, pp. 35–40, January 2010.

[9] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, April 1999.

[10] C.-F. Juang, C.-M. Lu, C. Lo, and C.-Y. Wang, "Ant colony optimization algorithm for fuzzy controller design and its FPGA implementation," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 3, pp. 1453–1462, March 2008.

[11] G. S. Tewolde, D. M. Hanna, and R. E. Haskell, "A modular and efficient hardware architecture for particle swarm optimization algorithm," *Microprocessors and Microsystems*, vol. 36, no. 4, pp. 289–302, 2012.

[12] D. Zhang, "A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 2, pp. 551–561, 2008.

[13] M. Valtierra-Rodriguez, R. A. Osornio-Rios, A. Garcia-Perez, and R. de Jesus Romero-Troncoso, "FPGA-based neural network harmonic estimation for continuous monitoring of the power line in industrial applications," *Electric Power Systems Research*, vol. 98, pp. 51–57, 2013.

[14] C.-C. Tsai, "FPGA-based parallel DNA algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 1016–1026, 2011.

[15] P. R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, and A. Stoica, "Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 1, pp. 133–149, 2010.

[16] J. Kok, L. Gonzalez, and N. Kelson, "FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 2, pp. 272–281, 2013.

[17] N. Kumar, "A study of genetic algorithm to solve the travelling salesman problem," *Journal of Global Research in Computer Science*, vol. 3, no. 3, pp. 33–37, 2012.

[18] P. Graham and B. E. Nelson, "A hardware genetic algorithm for the travelling salesman problem on SPLASH 2," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '95. London, UK, UK: Springer-Verlag, 1995, pp. 352–361. [Online]. Available: http://dl.acm.org/citation.cfm?id=647922.760933

[19] I. Skliarova and A. B. Ferrari, "FPGA-based implementation of genetic algorithm for the traveling salesman problem and its industrial application," in *Proceedings of the 15th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Developments in Applied Artificial Intelligence*, ser. IEA/AIE '02. London, UK, UK: Springer-Verlag, 2002, pp. 77–87. [Online]. Available: http://dl.acm.org/citation.cfm?id=646864.708091

[20] M. A. Vega-Rodriguez, R. Gutierrez-Gil, J. M. Avila-Roman, J. M. Sanchez-Perez, and J. A. Gomez-Pulido, "Genetic algorithms using parallelism and FPGAs: The TSP as case study," in *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, ser. ICPPW '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 573–579. [Online]. Available: http://dx.doi.org/10.1109/ICPPW.2005.36

[21] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "Proposal of flexible implementation of genetic algorithms on FPGAs," *Systems and Computers in Japan*, vol. 38, no. 13, pp. 28–38, 2007.

[22] K. M. Deliparaschos, G. C. Doyamis, and S. G. Tzafestas, "A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation," *International Journal of Electronics*, vol. 95, no. 11, pp. 1149–1166, 2008.

[23] P. V. Santos and J. C. Alves, "FPGA based engines for genetic and memetic algorithms," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications (FPL)*, 2010, pp. 251–254.

[24] Y. cong Zhou, J. hua Gu, Y.-F. Dong, and H. ping Han, "Implementation of genetic algorithm for TSP based on FPGA," in *Proceedings of the 2011 Chinese Control and Decision Conference (CCDC)*, May 2011, pp. 2226–2231.

[25] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization," *SPIE: Intelligent Control and Adaptive Systems*, vol. 1196, pp. 289–296, 1989.

[26] G. Abu-Lebdeh and R. F. Benekohal, "Convergence variability and population sizing in micro-genetic algorithms," *Computer-Aided Civil and Infrastructure Engineering*, vol. 14, no. 5, pp. 321–334, 1999.

[27] D. E. Goldberg and R. Lingle, "Alleles, loci and the TSP," in *Proceedings of the First International Conference on Genetic Algorithms*, 1985, pp. 154–159.

[28] J. H. Holland, *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[29] G. A. Croes, "A method for solving traveling salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

[30] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Boston, MA, USA: Addison-Wesley Professional, 1989.

[31] O. Abdoun, J. Abouchabaka, and C. Tajani, "Analyzing the performance of mutation operators to solve the travelling salesman problem," *International Journal of Emerging Sciences*, vol. 2, no. 1, pp. 61–77, 2012.

[32] Z. Navabi, *VHDL: Analysis and modeling of digital systems*, ser. Electrical & electronic technology series. McGraw-Hill, 1998.

[33] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, January 1998.

[34] L. Miller, B. L. Miller, and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, pp. 193–212, 1995.

[35] O. Astrachan, "Bubble sort: an archaeological algorithmic analysis," in *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '03. New York, NY, USA: ACM, 2003, pp. 1–5.

[36] D. Applegate, R. Bixby, V. Chvatal, and W. Cook, "Concorde TSP Solver," Online, December 2003, http://www.tsp.gatech.edu/concorde/index.html.

[37] M. Hahsler and K. Hornik, "TSP – infrastructure for the traveling salesperson problem," *Journal of Statistical Software*, vol. 23, no. 2, pp. 1–21, 2007.

[38] P. P. Chu, *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. Wiley-Interscience, 2011, ch. UART.

[39] G. Reinelt, "TSPLIB - A traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

Jonathan Kok is a PhD candidate affiliated with the Australian Research Centre for Aerospace Automation (ARCAA) and the School of Engineering Systems at Queensland University of Technology (QUT). His research focuses on optimisation problems and algorithms in aerospace, specifically involving evolutionary algorithms and FPGAs.

Troy Bruggemann is a Postdoctoral Research Fellow at the Australian Research Centre for Aerospace Automation (ARCAA), Queensland University of Technology (QUT). He holds a Bachelor and Masters degree in Aerospace Avionics Engineering and completed his PhD in high integrity satellite navigation at QUT in 2009. His current research interests include metaheuristics for combinatorial optimization and real-world routing problems in aerospace.

Luis Felipe Gonzalez is a Lecturer at the Queensland University of Technology (QUT) and the Australian Research Centre for Aerospace Automation (ARCAA). He has a degree in Mechanical Engineering. He completed his PhD on Multidisciplinary Design Optimization methods for UAV systems at The University of Sydney in 2005. Felipe joined ARCAA in 2006 and has directed his attention to enabling technologies for flight control and optimization of civilian UAVs systems. He has developed 6 operational UAVs and has written 18 journal papers and more than 35 peer reviewed papers in the topic of evolutionary algorithms and optimization.

Neil Kelson is Researcher Services Manager, HPC & Research Support Group, Queensland University of Technology (QUT). He holds degrees in Mathematics, Education and Information Technology and completed his PhD at QUT in 2000. Neils technology and research interests are focussed on the effective utilisation of advanced computing tools and technologies for research, especially in the areas of fluid mechanics, code optimisation and parallelisation, and the use of FPGAs for high performance embedded and supercomputing applications.

Chapter 7

An FPGA-Based Approach to Multi-Objective Evolutionary Algorithm for Multi-Disciplinary Design Optimisation

In this chapter, the FPGA-based multi-objective EA approach for multi-objective optimisation problems is investigated. The NSGA-II has been well studied and established for large and complex problems, such as those inherited in multi-objective optimisation problems. The NSGA-II is implemented on an FPGA device. The performance of the FPGA-based NSGA-II on different types of Pareto front geometry, such as convex, concave and discontinuous, is investigated to verify the effectiveness of the proposed design. Results show that the NSGA-II on FPGA is three orders of magnitude faster than its software version. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(b), 3.(c). and 4.

7.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

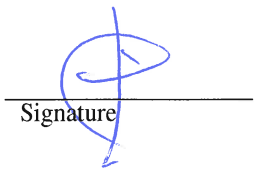
J. Kok, L. Gonzalez, N. Kelson, and J. Periaux, "An FPGA-based approach to multi-objective evolutionary algorithm for multi-disciplinary design optimisation," in *Proceedings of the 2011 Evolutionary and Deterministic Methods for Design, Optimization and Control (Eurogen 2011)*, 2011.

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓
Jacques Periaux	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ — Name



Signature

18-6-2014 — Date

EVOLUTIONARY AND DETERMINISTIC METHODS FOR DESIGN, OPTIMIZATION AND CONTROL
C. Poloni, D. Quagliarella, J. Périaux, N. Gauger and K. Giannakoglou (Eds.)

AN FPGA-BASED APPROACH TO MULTI-OBJECTIVE EVOLUTIONARY ALGORITHM FOR MULTI-DISCIPLINARY DESIGN OPTIMISATION

Jonathan Kok, Felipe Gonzalez
Australian Research Centre for Aerospace Automation (ARCAA)
Queensland University of Technology (QUT)
Eagle Farm, 4009 Queensland, Australia
Email: jonathan.kok@student.qut.edu.au; felipe.gonzalez@qut.edu.au
Web page: www.arcaa.aero

Neil Kelson
High Performance Computing and Research Support Group, Division of TILS
Queensland University of Technology (QUT)
Brisbane, 4000 Queensland, Australia
Email: n.kelson@qut.edu.au
Web page: www.itservices.qut.edu.au/hpc

Jacques Periaux*
International Center for Numerical Methods in Engineering (CIMNE)/UPC
Universidad Politecnica de Cataluna
Campus Norte UPC, 08034 Barcelona, Spain
Email: jperiaux@gmail.com
Web page: www.cimne.upc.edu

Abstract. This paper investigates the field programmable gate array (FPGA) approach for multi-objective and multi-disciplinary design optimisation (MDO) problems. One class of optimisation method that has been well-studied and established for large and complex problems, such as those inherited in MDO, is multi-objective evolutionary algorithms (MOEAs). The MOEA, nondominated sorting genetic algorithm II (NSGA-II), is hardware implemented on an FPGA chip. The NSGA-II on FPGA application to multi-objective test problem suites has verified the designed implementation effectiveness. Results show that NSGA-II on FPGA is three orders of magnitude better than the PC based counterpart.

Key words: Field programmable gate array (FPGA), multi-disciplinary design optimisation (MDO), multi-objective evolutionary algorithm (MOEA)

1 INTRODUCTION

Multi-disciplinary design optimisation (MDO) has been and is actively applied for solving design problems in aerospace, mechanical and electrical engineering. The aim of MDO is to generate superior designs by the simultaneous exploitation of incorporated interactions between disciplines[1]. However, the inclusion of interacting subsystems increases the problem complexity and resource requirements, where the search space is larger and the associated objective functions are computationally and memory intensive. Therefore, robust and efficient optimisation methods that are also practical in terms of computational run-time are indispensable in the field of MDO.

One class of optimisation method that has been well-studied and established for large and complex problems, such as those inherited in MDO, is Multi-Objective Evolutionary Algorithms (MOEAs)[2,3,4]. MOEAs belong to a class of generic population-based metaheuristic optimisation methods built from the principles of biological evolution. MOEA simultaneously optimises a set of candidate design solutions through genetic operations that explore and exploit interesting regions of the search space without *a priori* knowledge, thus offering exceptional search adaptability for finding Pareto fronts on large and complex unknown problem domains. MOEAs have been suggested primarily because of their ability to emphasise the search towards the true Pareto optimal region and obtain a set of Pareto optimal design solutions in one simulation run. The nature of MOEAs being population-based metaheuristics makes them well suited for solving MDO problems[2], as the fundamental evolutionary process deals simultaneously with a population of candidate design solutions which are iteratively improved after each generation. Hence, constantly maintaining and producing a population of optimised design solutions that relates to a particular Pareto front. MOEA techniques differ in three implementation details — namely, fitness assignment, diversity preservation, and elitism.

One approach to speed up the runtime of MOEAs and MDO methods is to implement its behaviour on an FPGA device, where true parallel execution and hardware dedication is possible. An FPGA device is made up a finite number of programmable logic components to be configured for performing complex combinational functions. FPGA technology benefits from faster response times and customised functionality to accurately meet application requirements, which is contributed by the capability of controlling the design from the hardware level. With this, MDO methods that take several hours to run could be executed in factors of seconds, impacting significantly on the development cycle and cost of a project. Furthermore, the nature of MOEAs being population-based in which individuals optimises independently, makes them well suited for the adoption of true hardware parallelism on FPGA. Theory on FPGA programming and features can be found in Chu[5].

This paper describes the extension of previous work by Kok *et al.*[6] on coupling FPGA to MDO search algorithms. Our proposed hardware design adopts main features from the popular nondominated sorting genetic algorithm II (NSGA-II)[7], such as crowding distance assignment[8] and domination-based Pareto front ranking[9].

The rest of the paper is organised as follows: Section 2 presents the methodology for the NSGA-II and its FPGA mapping aspects. Section 3 shows validation and comparison of the NSGA-II and NSGA-II on FPGA by solving multi-objective mathematical test. Finally, Section 4 presents the conclusions.

2 METHODOLOGY

In multi-objective optimisation, all relevant disciplines are associated with one or several optimisation objectives, there is no single design solution that is uniquely optimum as compared to every other possible design solution with respect to all objectives, given that its optimality is subjected to the compromise arising from other conflicting objectives. Thereby, the goal of MOEA is to obtain a set of design solutions in which no other solutions are superior to those in its set when all objectives are considered. This non-dominated set of solutions, also known as the Pareto front, provides decision makers with the baseline for making an educated compromised choice from amongst the many.

In the following subsection, the evolutionary optimisation methods, the NSGA-II and NSGA-II implemented on FPGA are presented.

2.1 NSGA-II

In the instance of the NSGA-II, it incorporates fast nondominated sorting, crowding distance assignment, and elitism selection process[7].

The NSGA-II algorithm (see Figure 1) is a well known algorithm[7] but its description is repeated here as to provide background for the NSGA-II on FPGA. It starts by initialising the parent population, P_0, randomly, upon which each of the M objectives value, $V_m, m \in \{1, \ldots, M\}$, is evaluated as accordingly. The population is made up of N individuals that are represented by a vector consisting of a candidate solution, $\mathbf{x}$, its objective values, $\{V_1, \ldots, V_M\}$, the overall fitness value, F, and neighbourhood diversity value, D. Beginning of each generation, t, an offspring population, Q_t, which has the same chromosome structure as P_t, is selected for the evolutionary process with the selection pressure focusing on the elites which have better F and D values. Q_t then undergoes genetic operations involving simulated binary crossover (SBX) and polynomial mutation. The permutated Q_t is then evaluated for each of the M objectives value, $V_m, m \in \{1, \ldots, M\}$. Next, the concatenated $R_t = P_t \bigcup Q_t$ is fast nondominated sorted, where F of each chromosome is assigned a rank value according to the nondominated front it lies on with respect to every other chromosome in R_t. A generation cycle is completed with the assignment of D for each individual in R_t, which is a diversity value according to the crowding distance it constitutes with respect to the adjacent chromosomes on its nondominated front rank. The NSGA-II algorithm executes iteratively until a specified stopping criteria such as maximum number of generations, t_{max}, has been met.

```
NSGA-II()
Initialise population()
WHILE (Termination Criteria NOT MET)
      Elitism selection technique()
      Genetic operations()
      Objectives evaluation()
      Fast nondominated sorting()
      Crowding distance assignment()
END
```

Figure 1: Pseudocode for NSGA-II.

2.2 NSGA-II on FPGA

The theoretical foundations of the evolutionary algorithms rely on a binary coded representation, where genetic operators produce the best outcome due to the binary-chained nature by which biological evolution is handled[10]. This binary coded aspect is complimentary when mapping MOEA on FPGA, where hardware circuits operate on a binary logic level. Using modern FPGA design softwares, such as Xilinx ISE design suite, circuits can be easily designed and implemented for rapid prototyping on hardware, thereby avoiding the long fabrication processing of application specific

integrated circuit (ASIC) design. After configuring a predefined circuit on an FPGA, system developers are allowed to make design changes or functional enhancements as necessary without requiring the time and cost involved in ASIC redesign, hence offering viable long-term maintainability. In operational mode, the parallelism and pipelining design capabilities of FPGAs contribute to the significant speedup over instruction stream processors.

2.2.1 Overview

The proposed algorithmic architecture of the NSGA-II on FPGA, which exploits parallelism on an iteration level, is depicted below in Figure 2. The algorithm incorporates the key features, fixed-point representation, random number generator, crossover, mutation, Pareto front ranking, crowding distance assignment, selection, and evaluation, which are needed for securing diverse Pareto-optimal fronts. The population, consisting of candidate design solutions, is stored on the dedicated block RAM onboard the FPGA. Tournament selection is randomly carried out across the population, determining better candidate design solutions to be genetically altered, which produces the preceding offspring population. After the offspring have been crossovered, mutated and evaluated, they are concatenated with the parent population to undergo Pareto front ranking and crowding distance assignment. The higher ranking and wider spread solutions are updated back into the population block RAM. It should be noted that the data flow arrow in Figure 2 denotes parallel processing.

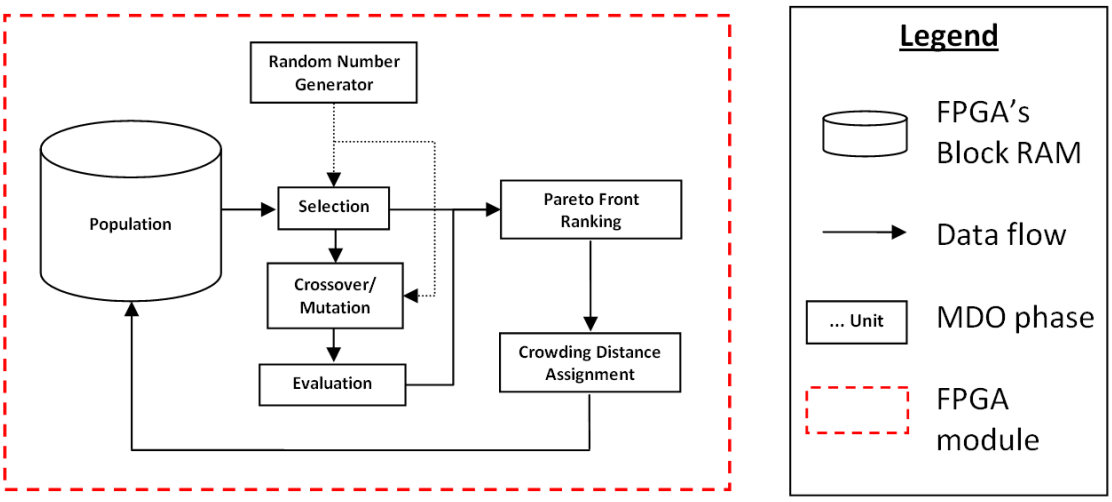


Figure 2: Architecture of the FPGA-based NSGA-II algorithm.

2.2.2 Representation

The representation of candidate design solutions can be coded in real-coded binary, floating-point or fixed-point numbers. Michalewicz[10] experimented and concluded that floating-point and fixed-point representation is faster, more consistent and higher in precision than real-coded representation. Since the complexity of floating-point arithmetic consumes a larger logic footprint and is not as efficient

as fixed-point arithmetic, representation of the candidate parameters is therefore encoded in fixed-point format.

2.2.3 Random number generator

Randomness and periodicity are two main factors to consider when implementing a logic level random number generator (RNG) for an FPGA. Matsumoto and Nishimura[11] proposed a pseudo RNG called Mersenne Twister, which was argued to be as fast and random as the standard ANSI-C "rand()". The Mersenne Twister is essentially a uniformly distributed pseudo RNG based on a matrix linear recurrence over a large finite binary field. Another advantage of FPGA implementation of a Mersenne Twister is its low resource consumption and its ability to generate new random sequences at every clock cycle.

2.2.4 Population module

The Population module, which is implemented as a block RAM, contains the population matrix comprising n chromosomes of candidate solution, $\mathbf{x}$, objective values, $\mathbf{V}$, fitness value, F, and diversity value, D, (see Figure 3). The Population module is responsible for broadcasting selected chromosomes for the evolutionary process. At the end of each generation, it receives the evaluated offspring and updates them into the Population module.

Chromosome index	Candidate solution			Objective values			Fitness value	Diversity value
1	$x_{1,1}$	...	$x_{1,N}$	$V_{1,N+1}$	...	$V_{1,N+M}$	$F_{1,N+M+1}$	$D_{1,N+M+2}$
2	$x_{2,1}$	...	$x_{2,N}$	$V_{2,N+1}$	...	$V_{2,N+M}$	$F_{2,N+M+1}$	$D_{2,N+M+2}$
⋮	⋮		⋮	⋮		⋮	⋮	⋮
n	$x_{n,1}$	...	$x_{n,N}$	$V_{n,N+1}$	...	$V_{n,N+M}$	$F_{n,N+M+1}$	$D_{n,N+M+2}$

Figure 3: Design of Population module.

2.2.5 Crossover/mutation

An SBX operation and a polynomial mutation proposed by Deb and Agarwal[12] are implemented. The intention of a crossover operation is to exchange useful information between two candidate solutions, whereas a mutation operation is aimed to slightly alter a candidate solution. Thus, crossover and mutation can be seen as exploitation and exploration respectively. A balance between them is applied to guide a search algorithm.

2.2.6 Evaluation

The evaluation module consists of the analysis functions to be optimised for the specific MDO application.

2.2.7 Pareto front ranking

Pareto front ranking is based on the non-dominance feature of a candidate solution[9]. Design solution A is said to dominate design solution B if all of design solution A's

fitness is better than design solution B's, else design solution B is non-dominated. Non-dominated design solutions are allocated higher rank than dominated ones, hence ensuring the preservation of Pareto optimal design solutions.

2.2.8 Crowding distance assignment

A technique proposed by Deb[9] known as crowding distance assignment is implemented to maintain the diversity of the Pareto fronts. The advantage of this technique lies in the nature by which it operates, whereby it does not require any performance dependent parameter.

2.2.9 Selection

Tournament selection based on the Pareto front rank and crowding distance is used as a competition winning criteria for the next generation of offspring. Higher ranking solutions wins over lower ranking solutions. If two solutions are of the same rank, the solution with higher crowding distance wins. This method ensures the survival of the fittest.

2.3 NSGA-II on FPGA Implementation

The NSGA-II features described above are translated into very-high-speed integrated circuits hardware description language (VHDL), which is a hardware description language used to describe logic circuitry for FPGAs.

3 EXPERIMENTS AND RESULTS

3.1 Test Problems

The general formulation of a multi-objective optimisation problem can be represented in the following form:

$$\left.\begin{array}{lll} \text{Minimise/Maximise} & f_m(\mathbf{x}), & m = 1, 2, \ldots, M; \\ \text{subjected to} & g_j(\mathbf{x}), & j = 1, 2, \ldots, J; \\ & x_n^{(L)} \leq x_n \leq x_n^{(U)}, & n = 1, 2, \ldots, N. \end{array}\right\} \quad (1)$$

where $f_m, m \in \{1, \ldots, M\}$, are the objective functions to be minimised or maximised, $\mathbf{x} = (x_n, \ldots, x_N), n \in \{1, \ldots, N\}$, is the "optimisation vector" of N design variables that are individually restricted by lower $x_n^{(L)}$ and upper $x_n^{(U)}$ bounds, and $g_j, j \in \{1, \ldots, J\}$, are constraint functions.

Four multi-objective mathematical test problems are used to validate the performance of NSGA-II on FPGA. The first two test problems are SCH[13] and FON[14] where the true Pareto fronts are convex and concave, respectively. The subsequent two test problems are POL[15] and KUR[16] where both true Pareto fronts are difficult, discontinuous and concave. These test problems are described in Table 1. Note that concave problems pose difficulties for classical weighted sum approaches[9].

3.2 Algorithm Parameters

The software version of the NSGA-II is implemented on an Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33GHz, 3.49 GB of RAM, whereas the proposed NSGA-II on FPGA was implemented and simulated on a Xilinx Virtex 4 (xc4vlx200-11ff1513).

The parameters used for both algorithms and all the test problems are set according to Deb[9] recommendations:

- population size = 100,
- number of generations = 250,
- crossover rate = 90%,
- mutation rate = 10%.

Test problem	Objective functions	Pareto optimal front geometry
SCH	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	Convex
FON	$f_1(x) = 1 - \exp(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}}))$ $f_2(x) = 1 - \exp(-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}}))$	Concave
POL	$f_1(x) = [1 + (A_1 - B_1)^2 + (A_2 - B_2)^2]$ $f_2(x) = [(x_1+3)^2 + (x_2+1)^2]$ $A_1 = 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2$ $A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$ $B_1 = 0.5\sin x_1 - 2\cos x_1 + \sin x_2 - 1.5\cos x_2$ $B_2 = 1.5\sin x_1 - \cos x_1 + 2\sin x_2 - 1.5\cos x_2$	Discontinuous, concave
KUR	$f_1(x) = \sum_{i}^{2}(-10\exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}))$ $f_2(x) = \sum_{i=1}^{3}\lvert x_i\rvert^{0.8} + 5\sin x_i^3$	Discontinuous, concave

Table 1: Test problems used in this study.

3.3 Experimental Results

Five simulation runs are experimented for each SCH, FON, POL, and KUR test problems to test the effectiveness of NSGA-II on FPGA. Figure 4 and Figure 5 show one of the results comparing the Pareto front obtained by the NSGA-II and NSGA-II on FPGA for the test problems. It can be seen that good solution quality was achieved for each test problem. The computational runtime results are shown in Table 2.

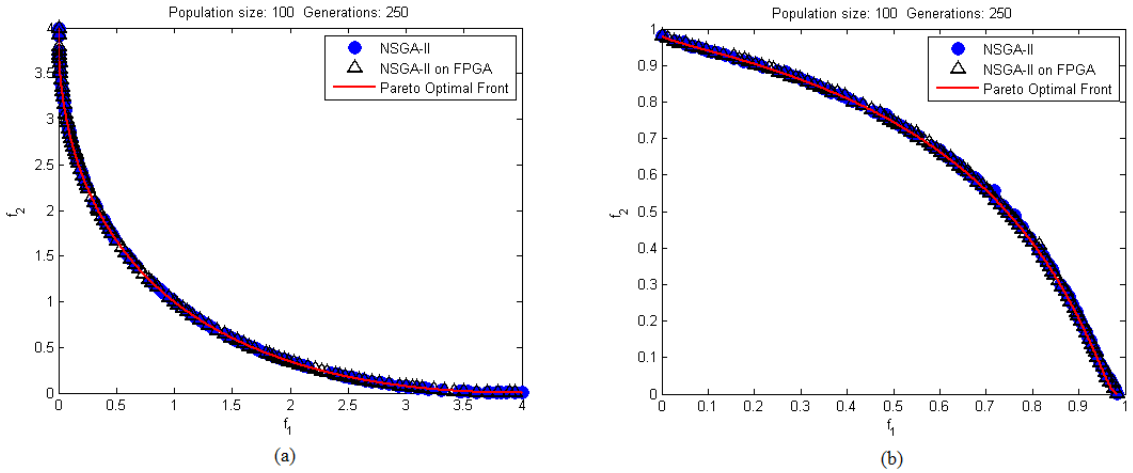


Figure 4: Comparison of Pareto front obtained by the NSGA-II and NSGA-II on FPGA for (a) SCH and (b) FON.

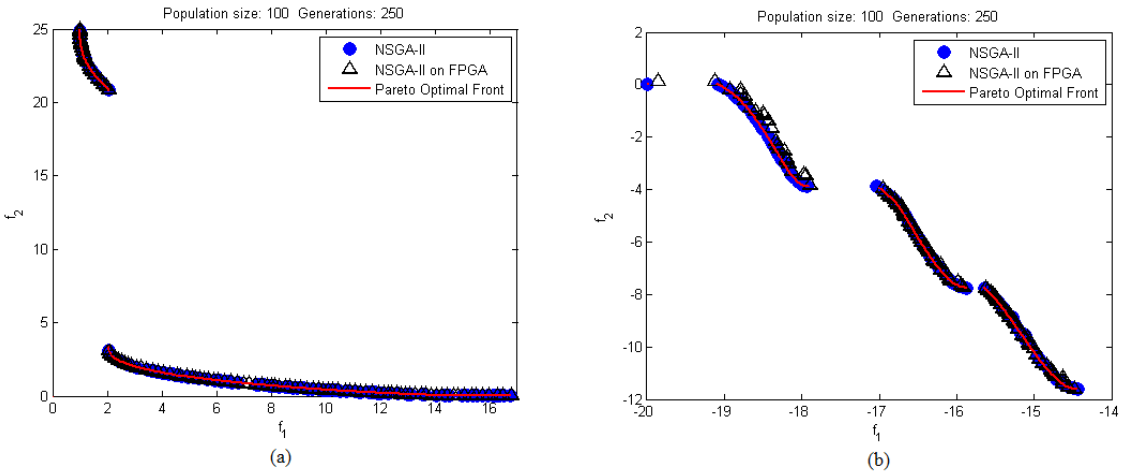


Figure 5: Comparison of Pareto front obtained by the NSGA-II and NSGA-II on FPGA for (a) POL and (b) KUR.

Test problem	Computational run-time (ms)		Speed improvement
	NSGA-II	NSGA-II on FPGA	
SCH	6 902.4	6.25	× 1 104
FON	7 981.2	6.42	× 1 243
POL	4 341.0	7.14	× 608
KUR	9 535.8	6.70	× 1 423

Table 2: Computational run-time results averaged over five simulation runs.

Size and execution speed analysis of the designed NSGA-II on FPGA is as follows. There is a total 178 000 configurable logic blocks utilised on the FPGA, which is equivalent to 5% of the overall resources available on a Xilinx Virtex 4 FPGA chip. The maximum working frequency is 32 MHz. The selection module takes two clock cycles to compare two random candidate design solutions. The crossover and mutation modules each takes one clock cycle to perform genetic operation on each candidate design solution. The Pareto front ranking and crowding distance assignment modules each takes 100 clock cycles to compare the entire population. Since the objective is to verify the NSGA-II on FPGA effectiveness, the evaluation module is implemented as lookup tables, in which takes one clock cycle to assign the appropriate objective values.

4 CONCLUSION

The hardware implementation of the NSGA-II algorithm for MDO problems is proposed in this paper. The NSGA-II is effective without *a priori* problem knowledge and capable of simultaneously optimising the design problem in a single simulation run. The proposed NSGA-II on FPGA performance is demonstrated from four examples and comparisons. The simulation results indicate that the solutions obtained from the NSGA-II on FPGA are comparable to its software counterpart. The hardware NSGA-II implementation achieved a speed up of approximately 1 300 times over the software implementation, which is attractive for practical multi-objective and MDO applications.

ACKNOWLEDGEMENTS

Computational resources and services used in this work were provided by the Australian Research Centre for Aerospace Automation (ARCAA) and the High Performance Computing and Research Group of the Queensland University of Technology (QUT), Brisbane, Australia.

REFERENCES

[1] Sobieszczanski-Sobieski, J. and Haftka, R. T. Multi-disciplinary aerospace design optimization: Survey of recent developments. *Structural and Multidisciplinary Optimization* **14**, 1–23 (1997).

[2] Coello, C. A. C., Lamont, G. B., and Veldhuizen, D. A. V. *Evolutionary algorithms for solving multi-objective problems*. Genetic and Evolutionary Computation Series. Springer, (2007).

[3] Lee, D. S., Gonzalez, L. F., Periaux, J., and Srinivas, K. Robust design optimisation using multi-objective evolutionary algorithms. *Computers and Fluids* **37**(5), 565–583 June (2008).

[4] Zitzler, E. Two decades of evolutionary multi-criterion optimization: A glance back and a look ahead. In *IEEE Symposium on Computational Intelligence in Multicriteria Decision Making*, 318 (, Honolulu, Hawaii, 2007).

[5] Chu, P. P. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. Wiley-Interscience, (2008).

[6] Kok, J., Gonzalez, F., Kelson, N., and Gurnett, T. A hardware-based multidisciplinary design optimisation method for aeronautical application. In IV International Conference on Computational Methods for Coupled Problems in Science and Engineering, M. Papadrakakis, E. O. and Schrefler, B., editors, (2011).

[7] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* **6**(2), 182–197 April (2002).

[8] DeJong, K. A. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, (1975).

[9] Deb, K. *Multi-objective optimization using evolutionary algorithms*. Wiley-Interscience series in systems and optimization. John Wiley and Sons, 1st edition, (2001).

[10] Michalewicz, Z. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, 3rd rev. and extended edition, (1996).

[11] Matsumoto, M. and Nishimura, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation* **8**(1), 3–30 January (1998).

[12] Deb, K. and Agrawal, R. B. Simulated binary crossover for continuous search space. *Complex Systems* **9**(2), 115–148 (1995).

[13] Schaffer, J. D. *Some experiments in machine learning using vector evaluated genetic algorithms (artificial intelligence, optimization, adaptation, pattern recognition).* PhD thesis, Vanderbilt University, Nashville, TN, USA, (1984).

[14] Fonseca, C. M. and Fleming, P. J. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation* **3**(1), 1–16 (1995).

[15] Poloni, C., Giurgevich, A., Onesti, L., and Pediroda, V. Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics. *Computer Methods in Applied Mechanics and Engineering* **186**(2-4), 403–420 June (2000).

[16] Kursawe, F. A variant of evolution strategies for vector optimization. In Parallel Problem Solving from Nature, Schwefel, H.-P. and Manner, R., editors, volume 496 of *Lecture Notes in Computer Science*, 193–197. Springer Berlin / Heidelberg (1991).

Chapter 8

Multi-Objective Evolutionary Algorithm using FPGA-Based Pipelining and Parallel Architecture: Design, Test, and Analysis

In this chapter, an improved FPGA-based multi-objective EA which utilises instruction-level parallelism and ring dataflow pipelining architecture for generating a very high throughput system is proposed. The FPGA-based design of the NSGA-II is described in relation to both the overall pipeline ring topology and the processing elements architecture. The proposed methodology is evaluated using test problems with known solutions. Analysis of results demonstrate that very high performance can be achieved with the hardware parallelising of processing elements in an MOEA over a pipeline ring architecture. With respect to the aforementioned research objectives as outlined in Section 1.3, this research paper addresses objectives 1.(b), 3.(c). and 4.

8.1 Statement of Contribution of Co-Authors

The authors listed below have certified that:

1. they meet the criteria for authorship in that they have participated in the conception, execution, or interpretation, of at least that part of the publication in their field of expertise;
2. they take public responsibility for their part of the publication, except for the responsible author who accepts overall responsibility for the publication;
3. there are no other authors of the publication according to these criteria;
4. potential conflicts of interest have been disclosed to (a) granting bodies, (b) the editor or publisher of journals or other publications, and (c) the head of the responsible academic unit, and
5. they agree to the use of the publication in the student thesis and its publication on the QUT ePrints database consistent with any limitations set by publisher requirements.

In the case of this chapter:

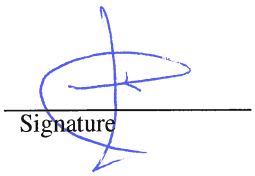
J. Kok, L. F. Gonzalez, and N. A. Kelson, "Multi-objective evolutionary algorithm using FPGA-based pipelining and parallel architecture: design, test, and analysis," *IEEE Transactions on Cybernetics*, 2013 (Submitted).

Contributor	Area of contribution (see Appendix A)			
	(a)(i)	(a)(ii)	(b)(i)	(b)(ii)
Jonathan Kok	✓	✓	✓	✓
Felipe Gonzalez	✓	✓	✓	✓
Neil Kelson	✓	✓	✓	✓

Principal supervisor confirmation:

I have sighted email or other correspondence from all Co-authors confirming their certifying authorship.

FELIPE GONZALEZ
Name


Signature

18-6-2014
Date

Multi-Objective Evolutionary Algorithm using FPGA-Based Pipelining and Parallel Architecture: Design, Test, and Analysis

Jonathan Kok, *Member, IEEE,* Felipe Gonzalez, and Neil Kelson

***Abstract*—In this paper, a hardware implementation of a multi-objective evolutionary algorithm (MOEA) on field programmable gate array (FPGA) for multi-objective optimisation is proposed. MOEAs are generic population-based metaheuristics that mimic the behaviour of biological evolution. The MOEAs population consists of candidate solutions that are rendered independently within the evolution process allowing for the exploitation of instruction-level parallelism and ring dataflow pipelining. The underlying binary nature of the evolutionary stages is complementary for the hardware implementation of fast primitive bitwise operations. These two main advantageous features of MOEAs make them well suited for FPGA-based implementations. The FPGA-based design of the NSGA-II, a widely used MOEA, is described in relation to both the overall pipeline ring topology and the processing elements architecture. The proposed methodology is tested using known test problems. Analysis of results demonstrate that very high performance can be achieved with the hardware parallelising of processing elements in an MOEA over a pipeline ring architecture.**

***Index Terms*—Evolutionary computation, field programmable gate array, multi-objective evolutionary algorithms, parallelism, pipeline.**

I. Introduction

REAL-WORLD multi-objective optimisation problems are often NP-hard, complex, and time consuming [1]. One optimisation method to address these types of optimisation problems is through the use of population-based metaheuristics, such as evolutionary algorithms (EAs) [2], [3]. In principle, EAs simulates the behaviour dynamics of a biological evolution process, by which guides the quality of solutions towards a level of optimality [4]. During the evolution process, a population of candidate solutions is iteratively evolved from generation to generation by the interactions of two main mechanisms: selection and reproduction. The quality of each candidate solution is associated with a relative fitness value assigned by an evaluation function. Based on these fitness values, the selection operator (e.g., tournament, roulette wheel, ranking, truncation [5]) is instituted to pressure the evolution process towards domination. The reproduction operator (e.g., mutation, crossover [6]) probabilistically applies variation on selected candidate solutions, by which heuristically steers the search towards solutions of diverse quality. The generic nature of EAs allows them to adapt and perform well across a diverse range of optimisation problems, including aerospace engineering [7], civil engineering [8], economics [9], robotics [10], and structural design [11].

Since the 1950s, metaheuristics were introduced to address optimisation problems involving very large search spaces, in which exact algorithms might not be feasible due to the required computational expenses. However, modern problems have become increasingly complex such that even applying metaheuristics for these problems nowadays can be computationally intensive. The main computational overhead is induced by the sequential computation of candidate solutions recurring throughout the evolution process. The computational burden is especially evident when candidate solution involves long data width and/or large population size [12]. Fortunately, executions of evolutionary operations on candidate solutions are independent of each other, which makes the computational process on candidate solutions of the population essentially an "embarrassingly parallel" task [13].

Concepts and technologies adopted from the field of parallel computation, such as reconfigurable computing with field programmable gate arrays (FPGAs), have been proposed to enhance the performance of EAs [14]–[25]. Most of these works used FPGAs to gain computational speedup by processing operations in parallel and are scoped for single objective EAs [14]–[22]. More interesting works scoped towards multi-objective approaches have been studied in only two other works [23]–[25]. Tachibana et al. [23] proposed a multi-objective evolutionary algorithm (MOEA) on FPGA based on a modified selection and overlap rejection components into a minimal generation gap GA model [26]. Their experiments compared a simulation of the FPGA implementation with its software counterpart, which resulted in a significant speed improvement from 43 seconds to 10 milliseconds. Bonissone and Subbu [24] proposed an implementation of a simple MOEA on an FPGA. They used a dominance filter to isolate an archive of non-dominated solutions, in which a comparison of each solution against every other solution is executed concurrently. Their experiment compared the FPGA design with its software counterpart, which showed significant speedup of 2 orders of magnitude. Note that it is important not to confuse the methodological use of FPGAs to develop efficient EAs with the field of evolvable hardware, in which EAs are used for evolving a combinational circuit on an FPGA to efficiently configure specialised architectures [27].

J. Kok and F. Gonzalez are with the Australian Research Centre for Aerospace Automation (ARCAA), Queensland University of Technology, Brisbane 4000 Australia (e-mail: j.kok@qut.edu.au; felipe.gonzalez@qut.edu.au).

N. Kelson is with the High Performance Computing and Research Support Group, Division of TILS, Queensland University of Technology, Brisbane 4000 Australia (e-mail: n.kelson@qut.edu.au).

In this paper, the FPGA implementation of a widely used MOEA known as non-dominated sorting genetic algorithm II (NSGA-II) is designed, tested, and analysed. The MOEAs population consists of candidate solutions that are rendered independently within the evolutionary stages allowing for the exploitation of instruction-level parallelism and ring dataflow pipelining. The underlying binary nature of the evolutionary stages is complementary for the hardware implementation of fast primitive bitwise operations. These two main advantageous features of MOEAs make them well suited for FPGA-based implementations. The modularity of the proposed design allows the system to be extended or updated with other MOEAs and applications. The goals of this work are threefold. Firstly, to propose and describe an FPGA-based architecture design of an MOEA implementation that exploits parallelism and pipelining for achieving significant speedups. Secondly, to test the FPGA-based NSGA-II prototype on test problems with known solutions for demonstrating the feasibility of an FPGA-based MOEA implementation. Thirdly, to analyse the effectiveness for achieving very high performance with the hardware parallelising of processing elements over a pipeline ring architecture.

This paper is organised as follows. Section II provides the basic concepts of multi-objective optimisation. Section III describes the working principles of an MOEA. Section IV gives a brief introduction to FPGAs and describes the advantages of mapping an MOEA to FPGA. Section V describes the pipelining and parallel design architecture of the proposed FPGA-based NSGA-II. Section VI presents the performance analysis of the FPGA-based design against test problems. Finally, Section VII outlines the conclusions of this work.

II. Multi-objective Optimisation

In this section, multi-objective optimisation is introduced using an example problem. Next, the mathematical formulation for multi-objective optimisation problems is presented.

A. Description

Most real-world optimisation problems inevitably involve the trade-offs between two or more conflicting objectives and are subject to given constraints. The process of solving problems with multiple objectives is known as multi-objective optimisation. An example of a real-world multi-objective optimisation problem is that of aircraft design, where encompassing the entire vehicle as one system observes the strong coupling between disciplines. For this example, the design space includes variables essential to other disciplines such as aerodynamics for airfoil and wing shape, structural analysis for aircraft performance, propulsion for engine thermodynamics, and economics for cost practicability. Considering the involvement of conflicting objectives, such as performance and cost, means that there will not be a candidate solution that is optimal in all objective aspects [2]. For this reason, the aim of multi-objective optimisation is to provide enough information for the decision maker to select prudently from the choices of possible candidate solutions with inherited trade-offs.

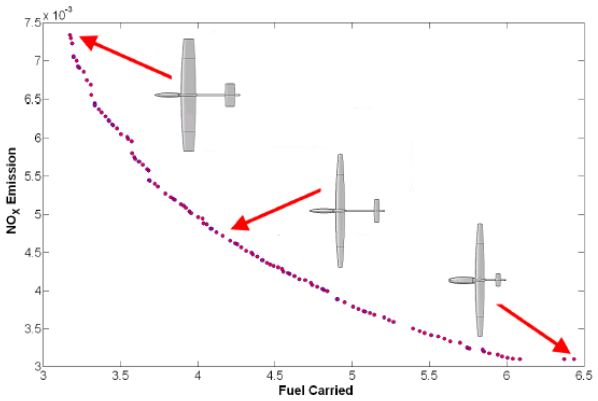


Fig. 1. An example of the Pareto front solutions for an unmanned aircraft design problem.

The concept of dominance is often used as a comparison metric between two possible candidate solutions. That is, candidate solution A is said to dominate candidate solution B if all of candidate solution A's objectives are better than candidate solution B's, else candidate solution B is non-dominated. Applying the concept of dominance across a set of candidate solutions will distinguish the non-dominated set of candidate solutions which resides in the decision space. This non-dominates set when corresponded into the objective space is known as the Pareto front. Figure 1 illustrates the Pareto front obtained after a hypothetical multi-objective optimisation process for an unmanned aircraft design problem with two conflicting objectives. It shows two extreme optimal solutions and one of the possible trade-offs in between. Note that, any preference made from the Pareto front are part of a non-dominated set of solutions, hence optimality is not compromised from the decision making.

B. Problem Definition

A mathematical formulation of the multi-objective optimisation problem can be represented in the following form:

$$\left.\begin{array}{lll} \text{Minimise} & f_m(\mathbf{x}), & m = 1, 2, \ldots, M; \\ \text{subject to} & g_j(\mathbf{x}) \geq 0, & j = 1, 2, \ldots, J; \\ & h_k(\mathbf{x}) = 0, & k = 1, 2, \ldots, K; \\ & x_n^{(L)} \leq x_n \leq x_n^{(U)}, & n = 1, 2, \ldots, N. \end{array}\right\} \quad (1)$$

where $f_m, m \in \{1, \ldots, M\}$, are the respective objective functions, $\mathbf{x} = (x_n, \ldots, x_N)$, $n \in \{1, \ldots, N\}$, denotes a vector of N design variables that are individually constrained by lower $x_n^{(L)}$ and upper $x_n^{(U)}$ bounds, $g_j, j \in \{1, \ldots, J\}$, are inequality constraints, and $h_k, k \in \{1, \ldots, K\}$, are equality constraints. Objective functions with maximisation properties can be converted to a minimisation problem by multiplying the objective values by -1. Optimisation methods ranging from gradient-based algorithms (e.g., Newton's method [28], sequential quadratic programming [29], steepest descent [30]) to population-based metaheuristics (e.g., ant colony optimisation [31], evolutionary algorithms [32], particle swarm optimi-

sation [33]), have been proposed for solving multi-objective optimisation problems.

III. Multi-objective Evolutionary Algorithm

In this section, a class of population-based metaheuristic known as multi-objective evolutionary algorithm (MOEA) is introduced.

A. Description

MOEA is a population-based metaheuristic inspired by biological evolution, particularly through the mechanisms of selection pressure and reproduction operation. MOEAs were proposed for handling multi-objective optimisation problems effectively [2]. The working principle of an MOEA is described as follows. A *population* refers to a set of candidate solutions and *offspring* population is the result of each evolutionary *generation* of the preceding *parent* population. The MOEA begins by randomly initialising the first parent population. Subsequently, the population is exposed to a selection pressure, such as tournament selection, roulette wheel selection, ranking selection, or truncation selection [5], through which fitter solutions in a population have better chance of survival as the evolution process progresses.

An offspring population is reproduced from genetically altering selected candidate solutions of the parent population through the use of crossover operators and mutation operators. The crossover operation exchanges portions of information to create possibly better solutions from shared traits, whereas the mutation operation randomly alters portions of a candidate solution to stimulate genetic diversity [6]. The offspring population is updated into the parent population using a replacement scheme (such as generational replacement where offspring population overwrites all of parent population, environmental replacement where worst solutions are deleted incrementally until population reaches a predefined minimum size, and elitist replacement where best parent solutions are preserved [34]). This generational evolution process is repeated until termination criteria are met. Commonly used criteria include, maximum generation, desired solution, maximum computational run-time, convergence plateau, or any combinations of the above.

B. Fundamental Features

Three fundamental search features are required in MOEAs to efficiently guide the entire population towards a uniformly diverse set of non-dominated solutions: fitness assignment, diversity preservation, and elitism.

1) Fitness Assignment: The objective of the fitness assignment is to translate the vector of values associated with each evaluated objective functions for a given candidate solution into a single qualitative measure. This measure is used to promote convergence of the current Pareto front towards the optimal Pareto front. Pareto-based schemes [35] use the concept of dominance to guide the optimisation process, while indicator-based schemes [36] rely on a specified quality measure for performance.

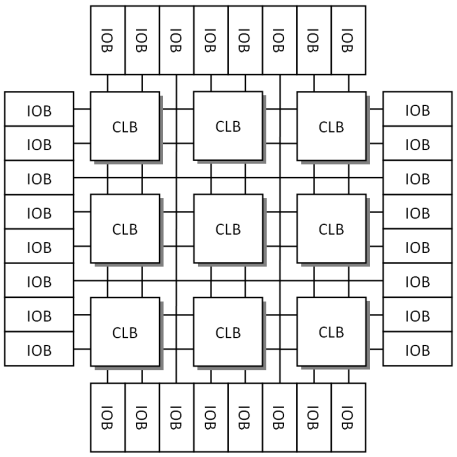


Fig. 2. FPGA block structure.

2) Diversity Preservation: The objective of the diversity preservation is to maintain and diversify the population of solutions. Sharing schemes [37] strongly rely on *a priori* knowledge to specify a threshold parameter for measuring a distance metric between two solutions. Contrary to that, crowding schemes [38] quantifies the crowding metric of a set of candidate solutions using adjacent neighbouring information, hence are operating independent of any external parameter specification.

3) Elitism: The objective of elitism is to prevent the lost of beneficial solutions during the evolutionary process. This concept is achieved by maintaining an archive of non-dominated or preferred solutions, which were discovered earlier, for introduction back into the evolutionary process.

From this set of search features, different MOEAs are designed and implemented by the transformation of relevant features. For instance, NSGA-II incorporates non-dominance sorting, crowding distance assignment and tournament selection process [39]; and SPEA2 employs fine-grained non-dominance fitness assignment, nearest neighbour density estimation, and archive truncation method [35]. The MOEA used in this work is based on the NSGA-II.

IV. Field Programmable Gate Array

In this section, an overview of field programmable gate array technology (FPGA) is presented. Next, the advantages of coupling an MOEA to an FPGA are discussed.

A. Description

FPGAs are semiconductor devices designed to be reconfigurable for desired functionality even after shipping, unlike application-specific integrated circuits (ASICs) which are custom manufactured and fixed for a designated task. An FPGA architecture is commonly consisted of an array of configurable logic blocks (CLBs) which are connected via programmable interconnects and routed to input/output blocks (IOBs). This basic block structure is depicted in Figure 2. The CLBs are the driving components in an FPGA. Generally, every CLB

consists of look-up tables (LUTs) and flip-flops which can be configured by a switch matrix to perform combinatorial logic, shift registers or random-access memory (RAM) operation. Flexible interconnect routing routes signals between CLBs to form complex combinatorial circuit design for computational functions. These programmable interconnect also routes the CLBs to and from the IOBs. The IOBs provide the interface bridge between the internal system and input/output standards. It is important to note that lower level details and features may vary between manufacturers as well as individual FPGA families.

An overview of the contrasts between FPGA and ASIC technologies are as follows. FPGA benefits from the low non-recurring engineering costs as compared to that of an ASIC. In manufacturing for large quantities, ASIC is typically lower in per unit cost. The functionality of an FPGA can be reprogrammed in the field after development, whereas an ASIC is permanently restricted to the specifically manufactured use. The ability to reconfigure an FPGA allows rapid prototyping and faster time-to-market, whereas the development of an ASIC is tightly bounded by time-consuming floorplanning, place and route, mask, and re-spin stages of the project to meet strict design specifications. These and the recent development of higher logic density FPGAs, such as the Virtex-7 FPGA families which host up to two million logic cells [40], makes FPGAs the compelling system-on-chip platform for mission critical applications with reconfigurable requirements. FPGAs have been used in a wide range of applications, including medical imaging [41], robotics [42], computer vision [43], speech recognition [44], nuclear science [45], cryptography [46], and high performance computing applications (e.g., Fast Fourier transform [47], convolution [48]).

B. Advantages of mapping MOEA to FPGA

The main advantage of implementing an MOEA on FPGAs is the potential extents for which parallelism can be exploited. Parallelism can be applied on an *algorithm* level, whereby multiple MOEAs are processing independently or cooperatively; an *instruction* level, which handles the solutions of population concurrently within an operation; and/or *function* level, which synchronises the execution of functions and data. The entire population can naturally be dealt with concurrently due to the independent structure of individual candidate solutions [49]. Although the advantages of parallelism can be seen on many levels of the algorithm, the overall generational cycle from each evolution process to another remains sequentially dependent. Nevertheless, the throughput of the system can be greatly increased by pipelining the generational cycles via pipeline ring dataflow. Another advantage of an FPGA-based optimisation algorithm is that the results can be continuously streamed out allowing for the real-time analysis of current best solutions as the algorithm runs indefinitely.

V. Implementation of NSGA-II MOEA on FPGA

In this section, a widely used MOEA known as nondominated sorting genetic algorithm II (NSGA-II) [39] is described. Subsequently, the overview of a hardware implementation of the NSGA-II along with elaborated details of the main modules are presented. This work is focused on the NSGA-II MOEA, however the proposed methodology can be extended and applied to other MOEAs as well.

```
NSGA-II
Initialisation
While (Termination criteria not met)
    Selection
    Reproduction
    Evaluation
    Non-dominated Sorting
End Loop
```

Fig. 3. Pseudocode for NSGA-II.

Fitness Value	Diversity Value	Objective Values			Candidate Solution		
F	D	V_1	...	V_M	x_1	...	x_N

Fig. 4. Data in each chromosome.

A. Description of NSGA-II

The pseudocode for the NSGA-II is illustrated in Figure 3. The fundamental features of the NSGA-II include the incorporation of a fast non-dominated sorting, crowding distance assignment, and tournament selection process [39]. The NSGA-II optimisation algorithm comprise of a population of chromosomes that undergo a series of evolution process. Each chromosome contains information on the fitness, diversity, objectives, and candidate solution values (see Figure 4). The algorithm commences by initialising the parent population, P_0, with random candidate solutions and evaluating each of their respective M objective values, $V_m, m \in \{1, \ldots, M\}$. The population is made up of S chromosomes that are each represented by a vector consisting of its overall fitness value, F, neighbourhood diversity value, D, objective values, $\{V_1, \ldots, V_M\}$, and the candidate solution, $\mathbf{x}$, as depicted in Figure 4. The chromosomes are subjected to a cyclical evolution process where each iteration is known a *generation*. In the beginning of each generation, t, an offspring population, Q_t, which has the same chromosome structure as P_t, is selected for the evolution process. The selection pressure is focused on the elites which have better F and D values. Q_t then undergoes reproduction involving genetic operations such as simulated binary crossover (SBX) and polynomial mutation. The permutated Q_t is then evaluated for each of the M objective values, $V_m, m \in \{1, \ldots, M\}$. Using the evaluated objective values, the concatenated $R_t = P_t \bigcup Q_t$ is fast non-dominated sorted, where F of each chromosome is assigned a rank value according to the non-dominated front it lies on with respect to every other chromosome in R_t. Subsequently, the diversity value, D, is calculated according to the crowding

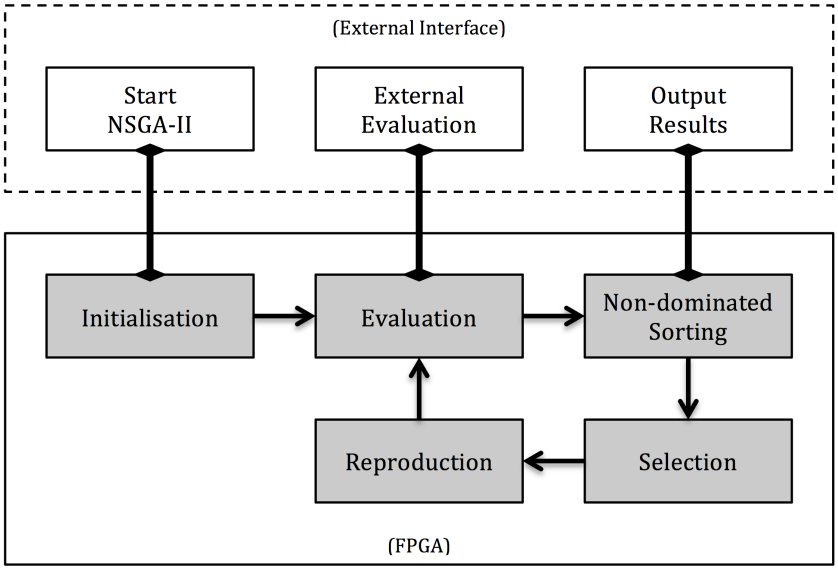


Fig. 5. Proposed hardware design architecture of NSGA-II on FPGA and the respective modules within.

Instruction Number	Pipeline Stage														
1	I	E	N	S	R										
2		I	E	N	S	R									
3			I	E	N	S	R								
4				I	E	N	S	R							
5						E	N	S	R						
6							E	N	S	R					
7								E	N	S	R				
8									E	N	S	R			
9										E	N	S	R		
10											E	N	S	R	
11												E	N	S	R
...	...														
Generation Number	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12

Fig. 6. Ring dataflow pipeline scheduling, where I is the *Initialisation* module, E is the *Evaluation* module, N is the *Non-dominated Sorting* module, S is the *Selection* module, and R is the *Reproduction* module.

distance the candidate solution constitutes with respect to the adjacent chromosomes on the same non-dominated front rank. A generation cycle is completed with the sorting of the population ordered by fitness and diversity values. The NSGA-II algorithm continues iteratively until a specified stopping criteria such as maximum number of generations, t_{max}, has been met.

B. NSGA-II on FPGA Design Overview

The proposed hardware design architecture for NSGA-II on FPGA is depicted in Figure 5. The FPGA-based NSGA-II is designed to function as an IP core implementation for different optimisation problems without the need to re-synthesis the entire design. Only the *Initialisation*, *Selection*, and *Reproduction* modules consists of internal random number generator (RNG) units. Each module is controlled by a Mealy finite-state machine [50] which manages the data and parallel execution of internal operations. Each module also consists of request-acknowledge handshake protocols for start and done flags by which ensures that data is sent and received completely before starting their respective operations. The request-acknowledge handshake protocol also allows for a controlled pipeline of heterogeneous modules, specifically a ring dataflow pipelining of the *Evaluation*, *Non-dominated Sorting*, *Selection*, and *Reproduction* modules, as illustrated in Figure 6. Within each of the modules, instruction-level parallelism is applied across the population of chromosomes. The basic operation of every module is to receive a population of chromosome from the previous module, process them and send them to the next module. Note that the *External Interface* can be an implementation that is in a different partition of the FPGA or on a separate processing device with the appropriate communication protocols. Each of the modules is described in the following subsections.

C. External Interface

The *External Interface* functions primarily as a bridge for sending and receiving data directly to the FPGA-based NSGA-II. The *Start NSGA-II* module sends a start signal to the *Initialise* module when the overall system is configured and ready for execution. The *External Evaluation* module consists of the objective functions used for evaluating the objective values, V_m, of candidate solutions generated by the evolution process. The number of internal evaluation units is equal to the number of chromosomes in the population and they operate concurrently. The *Output Results* module periodically receives the current population of chromosomes each time the *Non-dominated Sorting* module has completed its operation. As the population is being received, an internal counter is incremented to keep track of the number of generations that has taken place. The outputs of current population and

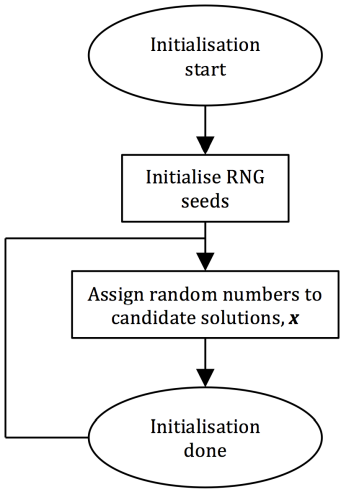


Fig. 7. Dataflow diagram of the state machine in the Initialisation module.

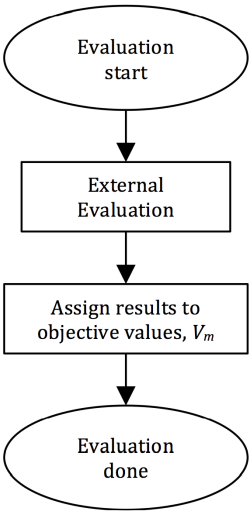


Fig. 8. Dataflow diagram of the state machine in the Evaluation module.

generation counts are being continuously streamed in allowing for the real-time analysis of solutions as the algorithm runs indefinitely. Note that the *External Interface* can be implemented on a different partition of the FPGA or on a separate processing device, such as a host computer that is connected to a network of parallel computers where the *External Evaluation* is distributed for concurrent computing.

D. Initialisation Module

The aim of the *Initialisation* module is to generate a population of candidate solutions with random values. The dataflow of the state machine in the *Initialisation* module is shown in Figure 7. After receiving a start signal from the *Start NSGA-II* module, the RNGs are initialised with different seeds. The RNG will be accessed concurrently, which means that if all RNGs are initialised with the same seed, then all RNGs will be producing the same random number output at a given interval, which is not a desired attribute hence different seeds are used to initialise the RNGs. The population of candidate solutions, **x**, are concurrently assigned with random values generated by the RNGs. Subsequently, the new population and a request done signal is sent to the *Evaluation* module. Upon receiving the acknowledgement signal from the *Evaluation* module, the *Initialisation* module restarts to assign another cluster of random values for the next set of candidate solutions to be sent to the *Evaluation* module. The *Initialisation* module continues to generate new random populations until the first sent population has completed a generation cycle, hence producing a constant throughput system.

E. Evaluation

The *Evaluation* module does not perform any computation internally, rather it channels the population of candidate solutions to be evaluated via the *External Evaluation* module within the *External Interface*. The dataflow of the state machine in the*Evaluation* module is shown in Figure 8. Once the *External Evaluation* module has completed evaluating the population of candidate solutions with respect to all objective functions, the returned results are concurrently assigned to the objective values, V_m, of each chromosome.

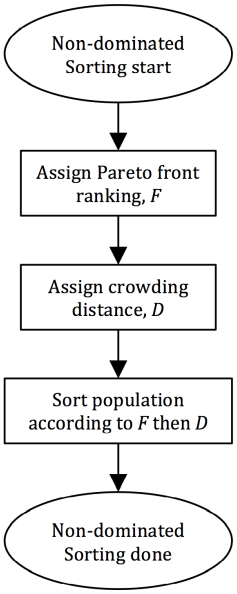


Fig. 9. Dataflow diagram of the state machine in the Non-dominated Sorting module.

F. Non-dominated Sorting

The *Non-dominated Sorting* module contains the core fundamental features of the NSGA-II, namely, the assignment of the Pareto front ranking and crowding distance metric. The dataflow of the state machine in the *Non-dominated Sorting* module is shown in Figure 9. The new population that is being received is combined with the previous population, which is

Crowding Distance Assignment	
Step 1:	Sort population according to fitness values of Objective 1 using Bubble Sort operation
Step 2:	Calculate difference between adjacent Objective 1 fitness values for each chromosome using subtraction operation
Step 3:	Sort population according to fitness values of Objective 2 using Bubble Sort operation
Step 4:	Calculate difference between adjacent Objective 2 fitness values for each chromosome using subtraction operation
Step 5:	Calculate total crowding distance of the differences for each chromosome using addition operation

Fig. 12. Steps for calculating the crowding distance.

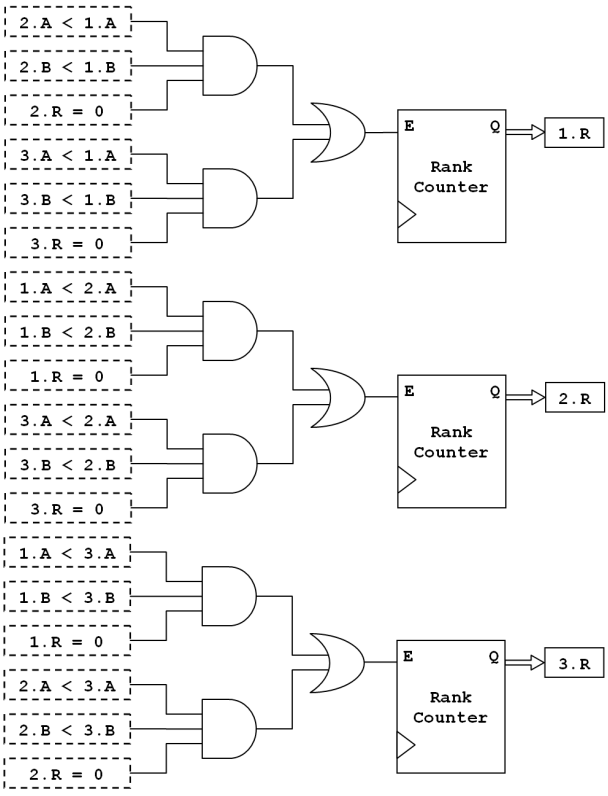


Fig. 10. Example illustrating the combinational logic of a non-dominance front ranking circuit design.

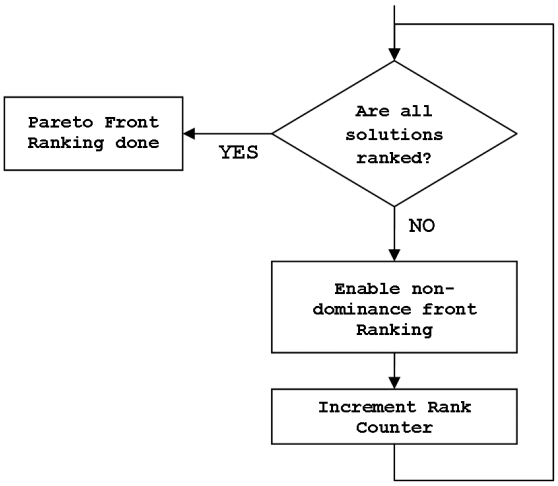


Fig. 11. Flowchart illustrating the Pareto front ranking sequence.

stored internally, creating a temporary population of double the original size. The Pareto front ranking is based on the non-dominance feature of a candidate solution with respect to every other solutions [2]. Recapping on the concept of dominance from Section II-A – candidate solution A is said to dominate candidate solution B if all of candidate solution A's fitness is strictly better than candidate solution B's, else candidate solution B is non-dominated. Non-dominated candidate solutions are allocated higher rank than dominated ones, hence ensuring the survival of non-dominated candidate solutions. The implementation of this module consists of a mix of combinational logic (see Figure 10) and sequential logic (see Figure 11) circuit designs. The figures illustrate the gate level implementation of three candidate solutions (1, 2, 3) with two objective values (A, B). At every clock cycle, the non-dominated solutions are assigned a rank (R) from the rank counter which increments every clock cycle until every solution is ranked.

The crowding distance assignment is implemented using sequential logic representing the steps depicted in Figure 12. Gate level implementations of subtraction and addition operations are used to calculate the crowding distance of each chromosome. The aim of this operation is to assign each chromosome a crowding distance metric relative to adjacent solutions [2]. A bubble sort sorting algorithm is implemented to sort the population according to fitness value, F, then diversity value, D. Elitism is achieved by discarding the lower half of the sorted population, which is the worst of the sorted population. The sorting ensures that the candidate solutions do not worsen but either remain the same or improve. The resulting output population is hence reduced back to the

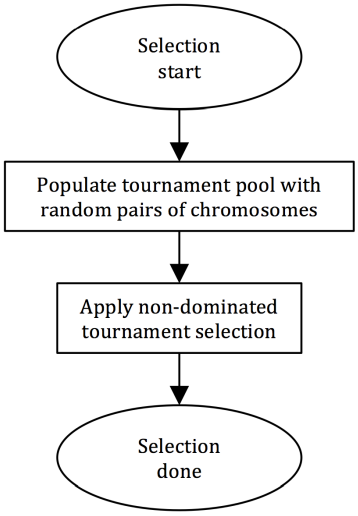


Fig. 13. Dataflow diagram of the state machine in the Selection module.

original size and is additionally stored internally to be used when the module is restarted.

G. Selection

The aim of the *Selection* module is to apply selection pressure in the pooling of a new population. The dataflow of the state machine in the *Selection* module is shown in Figure 13. Tournament selection based on the Pareto front ranking and crowding distance is implemented as a competition winning criteria for the next generation of solutions. Higher ranking solutions wins over lower ranking solutions. If two solutions are of the same rank, the solution with higher crowding distance wins. This method is implemented using comparison operators.

H. Reproduction

The *Reproduction* module uses genetic crossover and mutation operators concurrently to permute the candidate solutions of the population. The dataflow of the state machine in the *Reproduction* module is shown in Figure 14.The crossover operator implements a single-point binary crossover technique (see Figure 15) and the mutation operator implements XOR operations across randomly selected bits (see Figure 16). The intention of a crossover operation is to exchange useful information between two candidate solutions, whereas a mutation operation is aimed to randomly alter the candidate solutions. Thus, crossover and mutation can be seen as exploitation and exploration, respectively. A balance between them is necessary to guide a search algorithm effectively.

VI. Experimental Test and Analysis

A. Test Environment

Test and analysis were performed on multi-objective mathematical test problems with known solutions to verify the effectiveness of the proposed FPGA-based NSGA-II system.

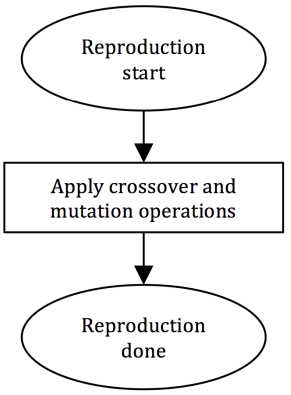


Fig. 14. Dataflow diagram of the state machine in the Reproduction module.

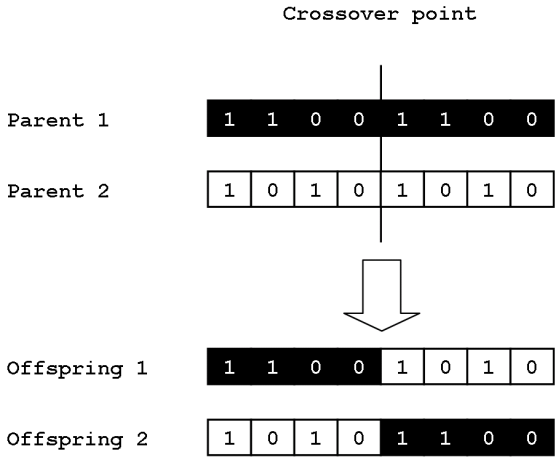


Fig. 15. Example illustrating the bitwise operation of the crossover process.

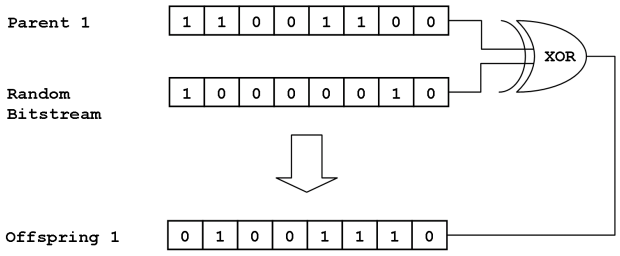


Fig. 16. Example illustrating the bitwise operation of the mutation process.

The first two test problems are SCH [51] and FON [52] where the true Pareto fronts are convex and concave, respectively. The subsequent two test problems are POL [53] and KUR [54] where both true Pareto fronts are difficult, discontinuous and concave. All test problems are described in Table I.

The parameters of the FPGA-based NSGA-II used for the tests are as follows. The population size within each module was implemented with the size of 32. In the *Reproduction* module, half of the population is crossovered and the remaining half is mutated. The population solutions were captured at 500 generations. The same parameters were used for the software version of the NSGA-II.

TABLE I
TEST PROBLEMS USED IN THIS STUDY

Test problem	Objective functions	Pareto front geometry
SCH	$f_1(x) = x^2$ $f_2(x) = (x-2)^2$	Convex
FON	$f_1(x) = 1 - \exp(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}}))$ $f_2(x) = 1 - \exp(-\sum_{i=1}^{3}(x_i + \frac{1}{\sqrt{3}}))$	Concave
POL	$f_1(x) = [1 + (A_1 - B_1)^2 + (A_2 - B_2)^2]$ $f_2(x) = [(x_1+3)^2 + (x_2+1)^2]$ $A_1 = 0.5\sin 1 - 2\cos 1 + \sin 2 - 1.5\cos 2$ $A_2 = 1.5\sin 1 - \cos 1 + 2\sin 2 - 0.5\cos 2$ $B_1 = 0.5\sin x_1 - 2\cos x_1 + \sin x_2 - 1.5\cos x_2$ $B_2 = 1.5\sin x_1 - \cos x_1 + 2\sin x_2 - 1.5\cos x_2$	Discontinuous, concave
KUR	$f_1(x) = \sum_{i}^{2}(-10\exp(-0.2\sqrt{x_i^2 + x_{i+1}^2}))$ $f_2(x) = \sum_{i=1}^{3}\lvert x_i\rvert^{0.8} + 5\sin x_i^3$	Discontinuous, concave

TABLE II
COMPUTATIONAL RUN-TIME RESULTS AVERAGED OVER TEN SIMULATION RUNS

Test problem	Computational run-time (ms)		Speed improvement
	PC-based NSGA-II	FPGA-based NSGA-II	
SCH	13,204.4	6.3	× 2,096
FON	14,812.2	6.4	× 2,002
POL	8,143.0	7.1	× 1,147
KUR	18,125.5	6.7	× 2,705

For the experiment environment involving the FPGA-based design, a Simulink model via Xilinx System Generator 2013.2 was used to couple the *External Interface* and the proposed FPGA design. In the Simulink model, HDL code of the FPGA-based NSGA-II design was imported using the Xilinx "blackbox block. The evaluation functions were implemented in Vivado HLS 2013.2, which is a high-level synthesis tool for translating C-based designs into digital hardware, and imported into the Simulink model using the Xilinx Vivado HLS block.

The proposed NSGA-II on FPGA design was synthesised using Xilinx Vivado 2013.2 with the target device of a Xilinx Virtex-7 FPGA VC707 Evaluation Kit (XC7VX485T-2FFG1761C), and the design goal was set to "balanced". The software version of the NSGA-II is implemented on an Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33GHz with 3.49 GB of RAM.

B. Analysis

Ten simulation runs were conducted for each of the test problems to analyse the effectiveness and performance of the proposed FPGA-based NSGA-II when compared to a PC-based software implementation. Solution qualities equivalent to those of the software version were attained by the FPGA-based implementation for each of the test problems. This is an expected result as the behaviour of both implementations are similar when analysed from the algorithms functional perspective. The computational run-time results are tabulated in Table II. It can be seen that very high performance can be achieved with pipelining and parallelism from the FPGA-based design.

Analyses of the resource utilisation and execution speed of the FPGA-based NSGA-II design are as follows. There were a total of $302,229$ slice logics utilised on the FPGA, which is equivalent to 27% of the overall resources available on the Xilinx Virtex-7 FPGA device. The maximum operating frequency was 100 MHz. The *Initialisation* module took one clock cycle to concurrently assign the initial population with random values. The *Selection* module took two clock cycles to concurrently compare and select from two random candidate solutions. The *Reproduction* modules took one clock cycle to concurrently perform the genetic crossover and mutation operations on each candidate solution. The *Non-dominated Sorting* module took an average of 150 clock cycles for computing the fitness and diversity values. The *Evaluation* module, which was generated by Vivado HLS, took two, four, seven, and eight clock cycles for concurrently evaluating the SCH, FON, POL, and KUR, respectively. The number of clock cycles to evaluate a function is dependent on the steps within the computation tree. Hence, good problem design and coding skill can greatly reduce the total required clock cycles. The average speed up achieved from the FPGA-based design was 1,987 times faster than its software counterpart.

VII. CONCLUSION

In this paper, a hardware design of an NSGA-II MOEA on FPGA for multi-objective optimisation problems is proposed. Parallelism was naturally exploitable when dealing with the

population-based metaheuristics as each of the individual candidate solutions is processed independently. The cyclic evolution process allows for the implementation of a ring dataflow pipeline, which highly increases the throughput of the system. The design architecture, communication and execution flow are described in detail. The FPGA-based NSGA-II computational run-time performance was tested and analysed from mathematical test problems with known solutions. Results of simulations demonstrate the ability of the FPGA-based optimisation algorithm to generate solutions an average of 1,987 times faster than its software version. The overall design consumed only 27% of the available resources on the Virtex-7 FPGA device. While present work is focused on the NSGA-II MOEA, the methodology has the potential to be widely applicable in other MOEAs that also might benefit from pipelining and parallelism architecture. Accelerated performance is beneficial for mission critical systems such as in stock market trading, navigation systems and real-time surveillance systems.

Future work will involve the development of an open source EA-based VHDL library to host a variety of EA module designs, so that algorithm implementation can be applied flexibly and suitability for different applications. Another aspect of future work will be the examination of the reconfiguration capability of FPGA technology, which allows for different module designs to be downloaded and implemented onto the FPGA whilst an existing system is in operation.

ACKNOWLEDGMENT

Computational resources and services used in this work were provided by the High Performance Computing and Research Support Group, Division of TILS, Queensland University of Technology, Brisbane, Australia.

REFERENCES

[1] K. Fleszar, C. Glaber, F. Lipp, C. Reitwiebner, and M. Witek, "The complexity of solving multiobjective optimization problems and its relation to multivalued functions," *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 18, pp. 1–53, 2011.

[2] K. Deb, *Multi-objective optimization using evolutionary algorithms*, 1st ed., ser. Wiley-Interscience series in systems and optimization. John Wiley and Sons, 2001.

[3] C. A. Coello Coello and G. B. Lamont, *Applications of multi-objective evolutionary algorithms*, ser. Advances in natural computation. World Scientific, 2004.

[4] T. Baeck, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*. Taylor and Francis, 1997.

[5] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, December 1996.

[6] W. M. Spears, *Evolutionary algorithms: The role of mutation and recombination*, ser. Natural Computing Series. Springer, 2000.

[7] L. F. Gonzalez, J. Periaux, K. Srinivas, and E. J. Whitney, "Evolutionary optimization tools for multi objective design in aerospace engineering: From theory to MDO applications," in *Evolutionary Algorithms And Intelligent Tools In Engineering Optimization*, W. Annicchiarico, J. Periaux, M. Cerrolaza, and G. Winter, Eds. UK: WIT Press, 2004.

[8] Z. K. Awad, T. Aravinthan, Y. Zhuge, and F. Gonzalez, "A review of optimization techniques used in the design of fibre composite structures for civil engineering applications," *Materials and Design*, vol. 33, pp. 534–544, January 2012.

[9] J. Branke, B. Scheckenbach, M. Stein, K. Deb, and H. Schmeck, "Portfolio optimization with an envelope-based multi-objective evolutionary algorithm," *European Journal of Operational Research*, vol. 199, no. 3, pp. 684–693, December 2009.

[10] Z. Li, B. Xu, L. Yang, J. Chen, and K. Li, "Quantum evolutionary algorithm for multi-robot coalition formation," in *Proceedings of the First ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, ser. GEC '09. New York, NY, USA: ACM, 2009, pp. 295–302.

[11] D.-S. Lee, L. Gonzalez, J. Periaux, and G. Bugeda, "Double-shock control bump design optimization using hybridized evolutionary algorithms," *Journal Of Aerospace Engineering - Part G*, vol. 225, pp. 1–181 175–1192, March 2011.

[12] P. J. Fleming and R. C. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control Engineering Practice*, vol. 10, no. 11, pp. 1223–1241, 2002.

[13] E. Cantu-Paz and D. E. Goldberg, "Efficient parallel genetic algorithms: Theory and practice," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 221–238, June 2000.

[14] S. D. Scott, A. Samal, and S. Seth, "HGA: A hardware-based genetic algorithm," in *Proceedings of the FPGA'95 ACM third international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM, 1995, pp. 53–59.

[15] M. So and A. Wu, "FPGA implementation of four-step genetic search algorithm," in *Proceedings of the 6th IEEE International Conference on Electronics, Circuits and Systems*, 1999.

[16] C. Aporntewan and P. Chongstitvatana, "A hardware implementation of the compact genetic algorithm," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, Seoul, South Korea, May 2001, pp. 624–629.

[17] B. Shackleford, E. Okushi, M. Yasuda, H. Koizumi, K. Seo, T. Iwamoto, and H. Yasuura, "High-performance hardware design and implementation of genetic algorithms," *Hardware Implementation of Intelligent Systems*, vol. -, pp. 53–87, 2001.

[18] M. Hamid and S. Marshall, "FPGA realisation of the genetic algorithm for the design of grey-scale soft morphological filters," in *Proceedings of the International Conference on Visual Information Engineering*, 2003.

[19] H. E. Mostafa, A. I. Khadragi, and Y. Y. Hanafi, "Hardware implementation of genetic algorithm on FPGA," in *Proceedings of the Twenty-First National Radio Science Conference*, March 2004, pp. 1–9.

[20] F. C. J. Allaire, M. Tarbouchi, G. Labonte, and G. Fusina, "FPGA implementation of genetic algorithm for UAV real-time path planning," *Journal of Intelligent and Robotic Systems*, vol. 54, no. 1-3, pp. 495–510, March 2009.

[21] P. R. Fernando, S. Katkoori, D. Keymeulen, R. Zebulum, and A. Stoica, "Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 1, pp. 133–149, 2010.

[22] J. Kok, L. F. Gonzalez, R. Walker, N. Kelson, and T. Gurnett, "A synthesizable hardware evolutionary algorithm design for unmanned aerial system real-time path planning," in *Proceedings of the 2010 Australasian Conference on Robotics and Automation*, Brisbane, Australia, December 2010.

[23] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "A hardware implementation method of multi-objective genetic algorithms," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, 2006, pp. 3153–3160.

[24] S. Bonissone and R. Subbu, "Evolutionary multiobjective optimization on a chip," in *Proceedings of the 2007 IEEE Workshop on Evolvable and Adaptive Hardware*, 2007, pp. 61–66.

[25] J. Kok, F. Gonzalez, N. Kelson, and J. Periaux, "An FPGA-based approach to multi-objective evolutionary algorithm for multi-disciplinary design optimisation," in *Proceedings of the Evolutionary and Deterministic Methods for Design, Optimization and Control (Eurogen 2011)*, C. Poloni, D. Quagliarella, J. Periaux, N. Gauger, and K. Giannakoglou, Eds., Capua, Italy, 2011.

[26] H. Satoh, I. Ono, and S. Kobayashi, "Minimal generation gap model for GAs considering both exploration and exploitation," in *Proceedings of the 1996 IZZUKA*, 1996.

[27] G. W. Greenwood and A. M. Tyrrell, *Introduction to evolvable hardware: a practical guide for designing self-adaptive systems*, ser. IEEE Series on Computational Intelligence. John Wiley and Sons, 2006.

[28] T. J. Ypma, "Historical development of the Newton-Raphson method," *SIAM Review*, vol. 37, no. 4, pp. 531–551, 1995.

[29] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[30] H. B. Curry, "The method of steepest descent for non-linear minimization problems," *Quarterly of Applied Mathematics*, vol. 2, pp. 258–261, 1944.

[31] M. Dorigo and T. Stutzle, *Ant colony optimization*. MIT Press, 2004.

[32] J. Periaux, D. Lee, L. Gonzalez, and K. Srinivas, "Fast reconstruction of aerodynamic shapes using evolutionary algorithms and virtual nash

strategies in a CFD design environment," *Journal of Computational and Applied Mathematics*, vol. 232, no. 1, pp. 61–71, 2009.
[33] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.
[34] A. Liefooghe, L. Jourdan, T. Legrand, J. Humeau, and E.-G. Talbi, "ParadisEO-MOEO: A software framework for evolutionary multi-objective optimization," in *Advances in Multi-Objective Nature Inspired Computing*, ser. Studies in Computational Intelligence, C. Coello Coello, C. Dhaenens, and L. Jourdan, Eds. Springer Berlin / Heidelberg, 2010, vol. 272, pp. 87–117.
[35] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, CH-8092 Zurich, Switzerland, Tech. Rep. 103, May 2007.
[36] E. Zitzler and S. Kunzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 832–842.
[37] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms and Their Application*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 41–49.
[38] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, Ann Arbor, MI, USA, 1975.
[39] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.
[40] M. Santarini, "Xilinx redefines state of the art with new 7 series FPGAs," *Xcell Journal*, vol. Third Quarter, no. 72, pp. 6–11, 2010.
[41] M. Leeser, S. Coric, E. Miller, H. Yu, and M. Trepanier, "Parallel-beam backprojection: An FPGA implementation optimized for medical imaging," *The Journal of VLSI Signal Processing*, vol. 39, pp. 295–311, 2005.
[42] S. Herbrechtsmeier, U. Witkowski, and U. Ruckert, "BeBot: a modular mobile miniature mobot platform supporting hardware reconfiguration and multi-standard communication," in *Progress in Robotics*, ser. Communications in Computer and Information Science, J.-H. Kim, S. S. Ge, P. Vadakkepat, N. Jesse, A. Al Manum, S. Puthusserypady K, U. Rckert, J. Sitte, U. Witkowski, R. Nakatsu, T. Braunl, J. Baltes, J. Anderson, C.-C. Wong, I. Verner, and D. Ahlgren, Eds. Springer Berlin Heidelberg, 2009, vol. 44, pp. 346–356.
[43] M. Arias-Estrada and C. Torres-Huitzil, "Real-time field programmable gate array architecture for computer vision," *Journal Electronic Imaging*, vol. 10, no. 1, pp. 289–296, January 2001.
[44] B. Schrauwen, M. DaHaene, D. Verstraeten, and J. V. Campenhout, "Compact hardware liquid state machines on FPGA for real-time speech recognition," *Neural Networks*, vol. 21, no. 2-3, pp. 511–523, 2008.
[45] N. G. Johnson-Williams, R. S. Miyaoka, X. Li, T. K. Lewellen, and S. Hauck, "Design of a real time FPGA-based three dimensional positioning algorithm," *IEEE Transactions on Nuclear Science*, vol. 58, no. 1, pp. 26–33, February 2011.
[46] W. N. Chelton and M. Benaissa, "Fast elliptic curve cryptography on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, feb. 2008.
[47] K. S. Hemmert and K. D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," in *Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Los Alamitos, CA, USA, April 2005, pp. 171–180.
[48] E. Jamro and K. Wiatr, "Convolution operation implemented in FPGA structures for real-time image processing," in *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis*, 2001, pp. 417–422.
[49] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proceedings of the Eighth international joint conference on Artificial intelligence - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1983, pp. 422–425.
[50] Z. Navabi, *VHDL: Analysis and modeling of digital systems*, ser. Electrical & electronic technology series. McGraw-Hill, 1998.
[51] J. D. Schaffer, "Some experiments in machine learning using vector evaluated genetic algorithms (artificial intelligence, optimization, adaptation, pattern recognition)," Ph.D. dissertation, Vanderbilt University, Nashville, TN, USA, 1984.
[52] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation*, vol. 3, no. 1, pp. 1–16, 1995.
[53] C. Poloni, A. Giurgevich, L. Onesti, and V. Pediroda, "Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, no. 2-4, pp. 403–420, June 2000.
[54] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Parallel Problem Solving from Nature*, ser. Lecture Notes in Computer Science, H.-P. Schwefel and R. Manner, Eds. Springer Berlin / Heidelberg, 1991, vol. 496, pp. 193–197.

Jonathan Kok is a PhD candidate affiliated with the Australian Research Centre for Aerospace Automation (ARCAA) and the School of Engineering Systems at Queensland University of Technology (QUT). His research focuses on optimisation problems and algorithms in aerospace, specifically involving evolutionary algorithms and FPGAs.

Luis Felipe Gonzalez is a Senior Lecturer at the Queensland University of Technology (QUT) and the Australian Research Centre for Aerospace Automation (ARCAA). He has a degree in Mechanical Engineering. He completed his PhD on Multidisciplinary Design Optimisation methods for UAV systems at The University of Sydney in 2005. Felipe joined ARCAA in 2006 and has directed his attention to enabling technologies for optimising autonomy, path planning, and sensor system integration for UAVs conducting environmental tasks. He has developed 10 operational UAVs and has written 18 journal papers and more than 40 peer reviewed papers in the topic of evolutionary algorithms and optimisation.

Neil Kelson is User Services Manager of QUT's HPC & Research Support Group. This role involves extensive consultation/collaboration with researchers and research groups regarding effective utilisation of advanced computing tools and technologies. He has a PhD in computational science and also holds degrees in Mathematics, Education and Information Technology. Neil's technology and research interests are focussed on engineering computational fluid dynamics, code optimisation and parallelisation, and the use of FPGAs for high performance embedded and supercomputing applications.

This page intentionally left blank.

Chapter 9

Conclusions

EAs are effective evolutionary computation techniques but can be computationally intensive to implement for complex problems. In recent years, advancements in FPGA technology have enabled the manufacturing of high-density FPGAs allowing its use for hardware accelerating EAs. The population-based characteristic of EAs allows potential levels of parallelism that FPGA technology can exploit for concurrent processing. However, the mapping of complex software algorithms, such as EAs, to hardware is not straightforward [21].

In this thesis, design methodologies and architectures of hardware-based EAs for solving aerospace optimisation problems on FPGAs have been proposed. The aim of this thesis is to provide knowledge that contributes to the theory and application of effective FPGA-based EA architectures. Parallelism was exploited by rendering individuals of the population concurrently within each of the biological evolutionary processes. The robustness and effectiveness was demonstrated through evaluation across several practical aerospace optimisation applications consisting of different problem characteristics.

This thesis also highlights investigation and development of FPGA-based EAs that encompasses both major fields of optimisation: single-objective and multi-objective optimisations. Solving multi-objective optimisation problems using EAs requires the addition of fitness assignment and diversity preservation features onto single-objective EAs for the retention of Pareto optimal solutions. Therefore, the system architectures of the proposed FPGA-based multi-objective EAs explicitly include fitness assignment and diversity preservation modules to enable the convergence and storage of Pareto optimal solutions in the population memory.

Furthermore, it is important to highlight that robustness of FPGA-based EAs has been evaluated alongside of the path planning problem, TSP, and multi-objective test functions, that are but a few of many possible aerospace optimisation applications. It is anticipated that our proposed FPGA-based path planner and TSP solver may be integrated to on-board UAVs or aircraft flight systems for civilian applications, such as infrastructure aerial surveys, while adhering to the size, weight and power constraints of aircraft requirements.

The results from the initial FPGA-based EA for path planning showed speed improvement of 52,000 times its software version (see Chapter 3). This architecture was further developed with architecture managed by a control unit in which demonstrated performance meeting the 10Hz update frequency of a typical autopilot system (see Chapter 4). The investigation on hardware implications with regards to population size and solution quality highlighted that a small population size was shown to be sufficient for obtaining quality solutions for the TSP, thereby a large population EA implementation is not necessary as it is an inefficient use of FPGA resources (see Chapter 5). With this findings, a small population-based EA known as micro-GA was designed for solving the TSP in which performed on average 70 times faster when compared

to an equivalent software version and 26 times faster when compared to the powerful Concorde TSP solver (see Chapter 6). The results from the preliminary FPGA-based NSGA-II multi-objective EA achieved speedup of approximately 1,300 times over its software version (see Chapter 7). This architecture was improved to allow for pipelining and a robust external data interface in which demonstrated performance of 1,987 times faster than its software version (see Chapter 8). The hardware implementation of a sorting algorithm used in the chapter 8 is based on the bubble sort algorithm [130, p. 69]. The list of FPGA devices and CPU specifications used in each chapter of this thesis is shown in Table 9.1. A summary of the FPGA designs used is shown in Table 9.2.

Table 9.1: Summary of FPGA devices and CPU used in each chapter.

Chapter	**FPGA device**	**CPU specifications**
3	Xilinx Virtex 4 XC4VLX200	Intel(R) Core(TM)2 Duo Core CPU @ 2.66 GHz
4	Xilinx Virtex-4 XC4VLX200	N/A
5	Xilinx Virtex-7 XC7VX485T	N/A
6	Xilinx Virtex-7 XC7VX485T	Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33 GHz
7	Xilinx Virtex-4 XC4VLX200	Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33 GHz
8	Xilinx Virtex-7 XC7VX485T	Intel(R) Core(TM)2 Duo CPU E8600 @ 3.33 GHz

There is no correlation between device utilisation and speed improvement. Each type of optimisation application requires a substantially different FPGA-based design solution. The amount of device utilisation is dependent on the goal of the FPGA-based design. For instance, the multi-objective EA in chapter 8 incorporates pipelining that is not considered in chapter 7. Hence, the FPGA-based design in chapter 8 had a higher device utilisation footprint as well as a better speed improvement over the design from

Table 9.2: Summary of FPGA designs used in each chapter.

Chapter	Application	Device utilisation	Speed improvement
3	Path planning	20%	52,000
4	Path planning	32%	N/A
5	TSP	N/A	N/A
6	TSP	7%	70.66 (averaged)
7	Multi-objective functions	5%	1,094.5 (averaged)
8	Multi-objective functions	27%	1,987.5 (averaged)

chapter 7.

The proposed design methodologies can be considered advancements in addressing physical and processing constraints for UAV or highly automated aircraft systems. The proposed FPGA-based EAs can be used to deliver instantaneous decision making capability ready for deployment onto UAV platform. Additionally, the knowledge discovered in this research provides a greater degree of confidence concerning the effectiveness of developing and implementing EAs on FPGA hardware devices.

9.1 Summary of Key Contributions

In this thesis, the proposed FPGA-based EAs offer technological advancement capabilities that can play a significant role in UAVs and highly automated systems. The key contributions of this research are:

- Proposed novel FPGA-based EA design methodologies and architectures for both single-objective and multi-objective optimisation allowing for adaptation in applications where the coupling of EA with FPGA is beneficial;
- Verified the efficiency of an EA with small population size, which supports the development of resource efficient FPGA-based EA architectures;

- Demonstrated the versatility of its potential in aerospace applications through evaluation on path planning problem, TSP, and multi-objective optimisation test function; and

- Established performance bounds for speedup achieved by FPGA-based EAs over software versions showing significant reductions in computational run-time.

9.2 Future Work

Recommendations for future research include:

- Investigation of fault-tolerant system approaches relative to the FPGA implementation of specific EAs, such as diversification, replication, and redundancy.

- Exploration of different techniques for handling large-scale optimisation problem sizes.

- Further study into hardware implications and potential reengineering of multi-objective EA features for generating high-resolution Pareto fronts effectively.

- Development of a dynamic reconfigurable FPGA-based system architecture to allow for an entire system needed for a UAV application to be managed on a unified FPGA-based architecture. For example, the civilian application of a UAV for automated weed control will require a suitable architecture for the integration and management of image/data processing algorithm, force-landed procedure, path planning algorithm, tracking control system, vision navigation, and mission monitoring protocols.

This page intentionally left blank.

References

[1] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*, ser. Natural Computing Series. Springer, 2003.

[2] H. M. Omar, "Designing integrated fuzzy guidance law for aerodynamic homing missiles using genetic algorithms," *Transactions of the Japan Society for Aeronautical and Space Sciences*, vol. 53, no. 180, pp. 99–104, 2010.

[3] C. S. Pan, Y. Zhang, L. Yang, and S. Qu, "The multi-target fire distribution strategy research of the anti-air fire based on the genetic algorithm," *International Journal of Innovative Computing Information and Control*, vol. 8, no. 4, pp. 2803–2810, 2012.

[4] H. Liu, X. B. Hu, S. N. Yang, K. Zhang, and E. Di Paolo, "Application of complex network theory and genetic algorithm in airline route networks," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2214, pp. 50–58, 2011.

[5] M. Soolaki, I. Mahdavi, N. Mahdavi-Amiri, R. Hassanzadeh, and A. Aghajani, "A new linear programming approach and genetic algorithm for solving airline boarding problem," *Applied Mathematical Modelling*, vol. 36, no. 9, pp. 4060–4072, 2012.

[6] A. A. El-Mahallawy, H. A. Yousef, M. I. El-Singaby, A. A. Madkour, and A. M. Youssef, "Robust flight control system design using H-infinity loop-shaping and recessive trait crossover genetic algorithm," *Expert Systems with Applications*, vol. 38, no. 1, pp. 169–174, 2011.

[7] F. Ucan and D. T. Altlar, "Using genetic algorithms for navigation planning in dynamic environments," *Applied Computational Intelligence and Soft Computing*, vol. 2012, pp. 1–16, 2012.

[8] F. Wang and M. Chen, "Compound helicopter optimum design based on genetic algorithm," *Advanced Materials Research*, vol. 591–593, pp. 92–95, 2012.

[9] Y. V. Pehlivanoglu and B. Yagiz, "Aerodynamic design prediction using surrogate-based modeling in genetic algorithm architecture," *Aerospace Science and Technology*, vol. 23, no. 1, pp. 479–491, 2012.

[10] M. Darrah, E. Fuller, T. Munasinghe, K. Duling, M. Gautam, and M. Wathen, "Using genetic algorithms for tasking teams of raven UAVs," *Journal of Intelligent and Robotic Systems*, vol. 70, no. 1–4, pp. 361–371, 2013.

[11] J. Holland, *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[12] ——, "Genetic algorithms and adaptation," in *Adaptive Control of Ill-Defined Systems*, ser. NATO Conference Series, O. Selfridge, E. Rissland, and M. Arbib, Eds. Springer US, 1984, vol. 16, pp. 317–333.

[13] ——, "Studying complex adaptive systems," *Journal of Systems Science and Complexity*, vol. 19, no. 1, pp. 1–8, 2006.

[14] A. Yang, Y. Shan, and L. T. Bui, *Success in evolutionary computation*, ser. Studies in Computational Intelligence. Springer, 2008.

[15] N. Nedjah, E. Alba, and L. De Macedo Mourelle, *Parallel evolutionary computations*, ser. Studies in Computational Intelligence. Springer, 2006.

[16] E. Alba, *Parallel metaheuristics: A new class of algorithms*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2005.

[17] P. P. Chu, *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. Wiley-Interscience, 2008.

[18] S. Trimberger, "A reprogrammable gate array and applications," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 1030–1041, 1993.

[19] J. Young and A. R. Price, "FPGA based UAV flight controller," in *Proceedings of the 11th Australian International Aerospace Congress*, Melbourne, Australia, 2005, pp. 1–5.

[20] K. Deb, *Multi-objective optimization using evolutionary algorithms*, 1st ed., ser. Wiley-Interscience series in systems and optimization. John Wiley and Sons, 2001.

[21] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Wiley, 2011.

[22] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, ser. A Wiley-Interscience publication. Wiley, 2000.

[23] Teal Group Corporation, "Teal Group predicts worldwide UAV market will total $89 billion in its 2013 UAV market profile and forecast," Press Release, 2013.

[24] D. Lohpetch and D. Corne, "Multiobjective algorithms for financial trading: Multiobjective out-trades single-objective," in *Proceedings of the 2011 IEEE Congress on Evolutionary Computation*, 2011, pp. 192–199.

[25] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[26] H. P. Williams, *Model building in mathematical programming*. Chichester, New York: Wiley, 1999.

[27] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*. Mineola, New York: Dover Publications, 1998.

[28] D. L. Applegate, *The Traveling Salesman Problem: A Computational Study*, ser. Princeton Series in Applied Mathematics. Princeton University Press, 2006.

[29] K. Apt, *Principles of constraint programming*. Cambridge University Press, 2003.

[30] M. C. Fu, "Optimization for simulation: Theory vs. practice," *INFORMS Journal on Computing*, vol. 14, no. 3, pp. 192–215, 2002.

[31] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[32] H. H. Goldstine, F. J. Murray, and J. von Neumann, "The Jacobi method for real symmetric matrices," *Journal of the ACM*, vol. 6, pp. 59–96, 1959.

[33] R. Fletcher, "Conjugate gradient methods for indefinite systems," in *Numerical Analysis*, ser. Lecture Notes in Mathematics, G. Watson, Ed. Springer Berlin / Heidelberg, 1976, vol. 506, pp. 73–89.

[34] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal of Scientific Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[35] E. Aarts and J. K. Lenstra, *Local search in combinatorial optimization*. New York: John Wiley and Sons, 1997.

[36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[37] F. Glover, "Tabu search: Part 1," *ORSA Journal On Computing*, vol. 1, no. 3, pp. 190–206, 1989.

[38] T. Baeck, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*. Taylor and Francis, 1997.

[39] M. Dorigo and T. Stutzle, *Ant colony optimization*. MIT Press, 2004.

[40] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[41] K. S. Hemmert and K. D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," in *Proceedings of the Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Los Alamitos, CA, 2005, pp. 171–180.

[42] K. K. Anumandla, R. Peesapati, S. L. Sabat, and S. K. Udgata, "SoC based floating point implementation of differential evolution algorithm using FPGA," *Design Automation for Embedded Systems*, pp. 1–20, 2013.

[43] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, and A. Dollas, "HC-CART: A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 4, pp. 47:1–47:25, 2013.

[44] S.-T. Pan and M.-L. Lan, "An efficient hybrid learning algorithm for neural network–based speech recognition systems on FPGA chip," *Neural Computing and Applications*, pp. 1–7, 2013.

[45] S. Cruz, D. Munoz, M. Conde, C. Llanos, and G. Borges, "FPGA implementation of a sequential extended kalman filter algorithm applied to mobile robotics localization problem," in *Proceedings of the 2013 IEEE Fourth Latin American Symposium on Circuits and Systems*, 2013, pp. 1–4.

[46] X.-d. Liu, "Research on the multi-target tracking algorithm based on double FPGA," in *Proceedings of the 19th International Conference on Industrial Engineering and Engineering Management*, E. Qi, J. Shen, and R. Dou, Eds. Springer Berlin Heidelberg, 2013, pp. 471–477.

[47] S.-A. Li, M.-H. Yang, C.-W. Weng, Y.-H. Chen, C.-H. Lo, and C.-C. Wong, "Ant colony optimization algorithm design and its FPGA implementation," in *Proceedings of the 2012 International Symposium on Intelligent Signal Processing and Communications Systems*, 2012, pp. 262–265.

[48] K. Rahimunnisa, P. Karthigaikumar, S. Rasheed, J. Jayakumar, and S. SureshKumar, "FPGA implementation of AES algorithm for high throughput using folded parallel architecture," *Security and Communication Networks*, 2012.

[49] G. Mingas, E. Tsardoulias, and L. Petrou, "An FPGA implementation of the SMG-SLAM algorithm," *Microprocessors and Microsystems*, vol. 36, no. 3, pp. 190–204, 2012.

[50] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA Implementation(s) of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 2, pp. 374–388, 2012.

[51] C. Gonzalez, J. Resano, A. Plaza, and D. Mozos, "FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 1, pp. 248–261, 2012.

[52] C. Le Lann, D. Boland, and G. Constantinides, "The Krawczyk algorithm: Rigorous bounds for linear equation solution on an FPGA," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds. Springer Berlin / Heidelberg, 2011, vol. 6578, pp. 287–295.

[53] N. G. Johnson-Williams, R. S. Miyaoka, X. Li, T. K. Lewellen, and S. Hauck, "Design of a real time FPGA-based three dimensional positioning algorithm," *IEEE Transactions on Nuclear Science*, vol. 58, no. 1, pp. 26–33, 2011.

[54] B. Abhishek, D. T. Sankar, N. Islam, and S. S. Kumar, "FPGA based design of robust spatial domain image watermarking algorithm," in *Power Electronics and Instrumentation Engineering*, ser. Communications in Computer and Information Science, V. V. Das, J. Stephen, and N. Thankachan, Eds. Springer Berlin Heidelberg, 2011, vol. 102, pp. 91–95.

[55] S. Dikmese, A. Kavak, K. Kucuk, S. Sahin, and A. Tangel, "FPGA based implementation and comparison of beamformers for CDMA2000," *Wireless Personal Communications*, vol. 57, pp. 233–253, 2011.

[56] Z. Ding, W. Shu, and M.-Y. Wu, "FPGA based parallel transitive closure algorithm," in *Proceedings of the 2011 ACM Symposium on Applied Computing*, ser. SAC '11. New York: ACM, 2011, pp. 393–394.

[57] X. Zhang, "The FPGA implementation of modified Goertzel algorithm for DTMF signal detection," in *Proceedings of the 2010 International Conference on Electrical and Control Engineering*, 2010, pp. 4811–4815.

[58] Y. Wang, Q. Zhao, L. Jiang, and Y. Shao, "Ultra high throughput implementations for MD5 hash algorithm on FPGA," in *High Performance Computing and Applications*, ser. Lecture Notes in Computer Science, W. Zhang, Z. Chen, C. Douglas, and W. Tong, Eds. Springer Berlin / Heidelberg, 2010, vol. 5938, pp. 433–441.

[59] A. Dinu, M. Cirstea, and S. Cirstea, "Direct neural-network hardware-implementation algorithm," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 5, pp. 1845–1848, 2010.

[60] A. Annovi and M. Beretta, "A fast general-purpose clustering algorithm based on FPGAs for high-throughput data processing," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 617, no. 1–3, pp. 254–257, 2010.

[61] A. Jain, P. Gambhir, P. Jindal, M. Balakrishnan, and K. Paul, "FPGA accelerator for protein structure prediction algorithm," in *Proceedings of the 5th Southern Conference on Programmable Logic*, 2009, pp. 123–128.

[62] M. A. Ibarra-Manzano, D. L. Almanza-Ojeda, M. Devy, J. L. Boizard, and J. Y. Fourniols, "Stereo vision algorithm implementation in FPGA using census transform for effective resource optimization," in *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, 2009, pp. 799–805.

[63] C. Claus, R. Huitl, J. Rausch, and W. Stechele, "Optimizing the SUSAN corner detection algorithm for a high speed FPGA implementation," in *Proceedings of the 2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 138–145.

[64] R. Maes, P. Tuyls, and I. Verbauwhede, "Low-overhead implementation of a soft decision helper data algorithm for SRAM PUFs," in *Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, C. Clavier and K. Gaj, Eds. Springer Berlin / Heidelberg, 2009, vol. 5747, pp. 332–347.

[65] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "FPGA-based face detection system using Haar classifiers," in *Proceedings of the 2009 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2009, pp. 103–112.

[66] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Proceedings of the Foundations of Genetic Algorithms*, 1991, pp. 69–93.

[67] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.

[68] W. M. Spears, *Evolutionary algorithms: The role of mutation and recombination*, ser. Natural Computing Series. Springer, 2000.

[69] A. Liefooghe, L. Jourdan, T. Legrand, J. Humeau, and E.-G. Talbi, "ParadisEO-MOEO: A software framework for evolutionary multi-objective optimization," in *Advances in Multi-Objective Nature Inspired Computing*, ser. Studies in Computational Intelligence, C. Coello Coello, C. Dhaenens, and L. Jourdan, Eds. Springer Berlin / Heidelberg, 2010, vol. 272, pp. 87–117.

[70] L. Sekanina, "Evolvable hardware: From applications to implications for the theory of computation," in *Unconventional Computation*, ser. Lecture Notes in Computer Science, C. Calude, J. Costa, N. Dershowitz, E. Freire, and G. Rozenberg, Eds. Springer Berlin / Heidelberg, 2009, vol. 5715, pp. 24–36.

[71] S. D. Scott, A. Samal, and S. Seth, "HGA: A hardware-based genetic algorithm," in *Proceedings of the 1995 ACM/SIGDA Third International Symposium on Field-Programmable Gate Arrays*, New York, NY, 1995, pp. 53–59.

[72] G. Tufte and P. Haddow, "Prototyping a GA pipeline for complete hardware evolution," in *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, A. Stoica, D. Keymeulen, and J. Lohn, Eds. IEEE Computer Society, 1999.

[73] O. Kitaura, H. Asada, M. Matsuzaki, T. Kawai, H. Ando, and T. Shimada, "A custom computing machine for genetic algorithms without pipeline stalls," in *Proceedings of the 1999 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, 1999, pp. 577–584.

[74] G. Harik, F. Lobo, and D. Goldberg, "The compact genetic algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 287–297, 1999.

[75] M. So and A. Wu, "FPGA implementation of four-step genetic search algorithm," in *Proceedings of the 6th IEEE International Conference on Electronics, Circuits and Systems*, vol. 2, 1999, pp. 1143–1146.

[76] S. Perkins, R. Porter, and N. Harvey, "Everything on the chip: A hardware-based self-contained spatially-structured genetic algorithm for signal processing," in *Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware*, J. Miller, A. Thompson, P. Thomson, and T. Fogarty, Eds., 2000.

[77] P. Martin, "A hardware implementation of a genetic programming system using FPGAs and Handel-C," *Genetic Programming and Evolvable Machines*, vol. 2, no. 4, pp. 317–343, 2001.

[78] C. Aporntewan and P. Chongstitvatana, "A hardware implementation of the compact genetic algorithm," in *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 1, Seoul, South Korea, 2001, pp. 624–629.

[79] B. Shackleford, E. Okushi, M. Yasuda, H. Koizumi, K. Seo, T. Iwamoto, and H. Yasuura, "High-performance hardware design and implementation of genetic algorithms," *Hardware Implementation of Intelligent Systems*, pp. 53–87, 2001.

[80] M. Hamid and S. Marshall, "FPGA realisation of the genetic algorithm for the design of grey-scale soft morphological filters," in *Proceedings of the International Conference on Visual Information Engineering*, 2003.

[81] G. R. Kramer, J. C. Gallagher, and M. Raymer, "On the relative efficiencies of *cGA variants for intrinsic evolvable hardware: Population, mutation, and random immigrants," in *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*, 2004, pp. 225–230.

[82] K. Glette and J. Torresen, "A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device," in *Proceedings of the Sixth International Conference on Evolvable Hardware*, 2005, pp. 66–75.

[83] I. A. McManus, "A multidisciplinary approach to highly autonomous UAV mission planning and piloting for civilian airspace," Ph.D. dissertation, Faculty of Built Environment and Engineering, Queensland University of Technology, Brisbane, Queensland, Australia, 2005.

[84] B. Girau and A. Boumaza, "Embedded harmonic control for dynamic trajectory planning on FPGA," in *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, 2007, pp. 244–249.

[85] L. Vacariu, F. Roman, M. Timar, T. Stanciu, R. Banabic, and O. Cret, "Mobile robot path-planning implementation in software and hardware," in *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*, 2007, pp. 140–145.

[86] N. Sudha and A. R. Mohan, "Design of a hardware accelerator for path planning on the euclidean distance transform," *Journal of Systems Architecture*, vol. 54, pp. 253–264, 2008.

[87] T. K. Priya, P. R. Kumar, and K. Sridharan, "A hardware-efficient scheme and FPGA realization for computation of single pair shortest path for a mobile automaton," *Microprocessors and Microsystems*, vol. 30, pp. 413–424, 2006.

[88] O. Hachour, "A genetic_FPGA algorithm path planning of an autonomous mobile robot," in *Proceedings of the 10th WSEAS International Conference on Mathematical Methods, Computational Techniques and Intelligent Systems*, 2008, pp. 66–71.

[89] F. C. J. Allaire, M. Tarbouchi, G. Labonte, and G. Fusina, "FPGA implementation of genetic algorithm for UAV real-time path planning," *Journal of Intelligent and Robotic Systems*, vol. 54, no. 1–3, pp. 495–510, 2009.

[90] H.-C. Huang, C.-C. Tsai, and S.-C. Lin, "SoPC-based parallel elite genetic algorithm for global path planning of an autonomous omnidirectional mobile robot," in *Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 1959–1964.

[91] M. Schmidt and D. Fey, "An optimized FPGA implementation for a parallel path planning algorithm based on marching pixels," in *Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs*, 2010, pp. 442–447.

[92] S. Akishita, S. Kawamura, and K. Hayashi, "Laplace potential for moving obstacle avoidance and approach of a mobile robot," in *Proceedings of the Japan-*

USA Symposium on Flexible Automation, A Pacific Rim Conference, 1990, pp. 139–142.

[93] J. O. Kim and P. K. Khosla, "Real-time obstacle avoidance using harmonic potential functions," *IEEE Transactions on Robotics and Automation*, vol. 8, pp. 338–349, 1992.

[94] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to algorithms*. New York: MIT Press, 1990.

[95] N. Sudha, S. Nandi, and K. Sridharan, "Cellular architecture for euclidean distance transformation," *Computers and Digital Techniques*, vol. 147, pp. 335–342, 2000.

[96] S. Morgan, "A comparison of simplex method algorithms," Master's thesis, University of Florida, Gainesville, FL, USA, 1997.

[97] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, 1979.

[98] C. Cocaud, "Autonomous tasks allocation and path generation of UAV's," Master's thesis, Department of Mechanical Engineering, University of Ottawa, Ontario, Canada, 2006.

[99] I. Al-Taharwa, A. Sheta, and M. Al-Weshah, "A mobile robot path planning using genetic algorithm in static environment," *Journal of Computer Science*, vol. 4, pp. 341–344, 2008.

[100] D. Fey and D. Schmidt, "Marching pixels: a new organic computing principle for high speed CMOS camera chip," in *Proceedings of the ACM International Conference on Computing Frontiers*, 2005, pp. 1–9.

[101] J. Barraquand, B. Langlois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 224–241, 1992.

[102] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: A computational study*, ser. Princeton Series in Applied Mathematics. Princeton University Press, 2011.

[103] R. Kumar and Z. Luo, "Optimizing the operation sequence of a chip placement machine using TSP model," *IEEE Transactions on Electronics Packaging Manufacturing*, vol. 26, no. 1, pp. 14–21, 2003.

[104] J. Zhao, Q. Liu, W. Wang, Z. Wei, and P. Shi, "A parallel immune algorithm for traveling salesman problem and its application on cold rolling scheduling," *Information Sciences*, vol. 181, no. 7, pp. 1212–1223, 2011.

[105] I. Eguia, S. Lozano, J. Racero, and F. Guerrero, "A methodological approach for designing and sequencing product families in reconfigurable disassembly systems," *Journal of Industrial Engineering and Management*, vol. 4, no. 3, pp. 418–435, 2011.

[106] E. K. Xidias, A. C. Nearchou, and N. A. Aspragathos, "Integrating path planning, routing, and scheduling for logistics operations in manufacturing facilities," *Cybernetics and Systems*, vol. 43, no. 3, pp. 143–162, 2012.

[107] R. Matai, S. Singh, and M. Lal, *Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches*. InTech, Dec. 2010, ch. 1. [Online]. Available: http://dx.doi.org/10.5772/12909

[108] C. H. Papadimitriou, "The Euclidean travelling salesman problem is NP-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237–244, 1977.

[109] G. Laporte, "A concise guide to the traveling salesman problem," *Journal of the Operational Research Society*, vol. 61, pp. 35–40, 2010.

[110] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[111] N. Kumar, "A study of genetic algorithm to solve the travelling salesman problem," *Journal of Global Research in Computer Science*, vol. 3, no. 3, pp. 33–37, 2012.

[112] P. Graham and B. E. Nelson, "A hardware genetic algorithm for the travelling salesman problem on SPLASH 2," in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*, 1995, pp. 352–361.

[113] I. Skliarova and A. B. Ferrari, "FPGA-based implementation of genetic algorithm for the traveling salesman problem and its industrial application," in *Proceedings of the 15th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Developments in Applied Artificial Intelligence*, 2002, pp. 77–87.

[114] M. A. Vega-Rodriguez, R. Gutierrez-Gil, J. M. Avila-Roman, J. M. Sanchez-Perez, and J. A. Gomez-Pulido, "Genetic algorithms using parallelism and FPGAs: The TSP as case study," in *Proceedings of the 2005 International Conference on Parallel Processing Workshops*, 2005, pp. 573–579.

[115] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "Proposal of flexible implementation of genetic algorithms on FPGAs," *Systems and Computers in Japan*, vol. 38, no. 13, pp. 28–38, 2007.

[116] K. M. Deliparaschos, G. C. Doyamis, and S. G. Tzafestas, "A parameterised genetic algorithm IP core: FPGA design, implementation and performance evaluation," *International Journal of Electronics*, vol. 95, no. 11, pp. 1149–1166, 2008.

[117] P. V. Santos and J. C. Alves, "FPGA based engines for genetic and memetic algorithms," in *Proceedings of the 2010 International Conference on Field Programmable Logic and Applications*, 2010, pp. 251–254.

[118] Y. cong Zhou, J. hua Gu, Y.-F. Dong, and H. ping Han, "Implementation of genetic algorithm for TSP based on FPGA," in *Proceedings of the 2011 Chinese Control and Decision Conference*, 2011, pp. 2226–2231.

[119] S. Helbig and D. Pateva, "On several concepts for epsilon-efficiency," *Operations Research Spectrum*, vol. 16, no. 3, pp. 179–186, 1994.

[120] G. V. Reklaitis, A. Ravindran, and K. M. Ragsdell, *Engineering optimization: Methods and applications*. New York: Wiley, 1983.

[121] E. Zitzler and S. Kunzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 832–842.

[122] D. E. Goldberg and J. Richardson, "Genetic algorithms with sharing for multimodal function optimization," in *Proceedings of the Second International Conference on Genetic Algorithms and Their Application*, 1987, pp. 41–49.

[123] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, College of Engineering, University of Michigan, Ann Arbor, MI, USA, 1975.

[124] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," Swiss Federal Institute of Technology, Tech. Rep. 103, 2007.

[125] S. Tiwari, G. Fadel, and K. Deb, "AMGA2: Improving the performance of the archive-based micro-genetic algorithm for multi-objective optimization," *Engineering Optimization*, vol. 43, no. 4, pp. 377–401, 2011.

[126] A. Arias-Montano, C. A. Coello Coello, and E. Mezura-Montes, "Multiobjective evolutionary algorithms in aeronautical and aerospace engineering," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 5, pp. 662–694, Oct 2012.

[127] T. Tachibana, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito, "A hardware implementation method of multi-objective genetic algorithms," in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, 2006, pp. 3153–3160.

[128] S. Bonissone and R. Subbu, "Evolutionary multiobjective optimization on a chip," in *Proceedings of the 2007 IEEE Workshop on Evolvable and Adaptive Hardware*, 2007, pp. 61–66.

[129] H. Satoh, I. Ono, and S. Kobayashi, "Minimal generation gap model for GAs considering both exploration and exploitation," in *Proceedings of the IZZUKA '96 International Conference on Soft Computing*, 1996, pp. 494–497.

[130] A. A. Jerraya, *Behavioral Synthesis and Component Reuse with VHDL.* Springer US, 1997.

This page intentionally left blank.

Appendix B

Definition of Authorship and Contribution to Publication

In accordance with Section 2.6.8 of the QUT Manual of Policies and Procedures (MOPP 2.6.8) [1] which cites the Australian Code for the Responsible Conduct of Research, authorship is defined as being based on substantial contributions in a combination of:

(a) (i) conception and design, **or**

(ii) analysis and interpretation of research data; **and**

(b) (iii) drafting significant parts of a work, **or**

(iv) critically revising it so as to contribute to the interpretation.

[1] http://www.mopp.qut.edu.au/D/D_02_06.jsp