

Development of Evolutionary Algorithms for Intelligent Real-Time Control Systems using Field-Programmable Technology

**Department of Informatics
School of Information and Communication Technologies
University of Piraeus**

PhD Thesis

ABDOALNASIR ALMABROK



Piraeus, June 2026

Development of Evolutionary Algorithms for Intelligent Real-Time Control Systems using Field-Programmable Technology

ABDOALNASIR ALMABROK

PhD Thesis
was prepared during the Programme of Doctoral Studies
of the Department of Informatics
of the School of Information and Communication Technologies
of the University of Piraeus
for the degree of Doctor of Philosophy

3-Member Supervising Committee

Mihalis Psarakis
Professor
Univ. of Piraeus
(Supervisor)

Anastasios Dounis
Professor
Univ. of West Attica
(Member)

Christos Douligeris
Professor
Univ. of Piraeus
(Member)

The PhD Thesis was presented before the 7-Member
Examination Committee and approved on
29/06/2026

7-Member Examination Committee

Mihalis Psarakis
Professor
Univ. of Piraeus

Anastasios Dounis
Professor
Univ. of West Attica

Christos Douligeris
Professor
Univ. of Piraeus

Panteleimon Asvestas
Professor
Univ. of West Attica

Panagiotis Kotzanikolaou
Professor
Univ. of Piraeus

Aggelos Pikrakis,
Associate Professor
Univ. of Piraeus

Athanasios Papadimitriou
Assistant Professor
Univ. of Peloponnese

ACKNOWLEDGEMENTS

My deepest gratitude goes, first and foremost, to Professor *Mihalis* and Professor *Dounis*, my supervisors, for the constant encouragement and guidance. Without the consistent and illuminating instruction, this thesis could not have reached its present form.

My thanks would go to my beloved parents *Mother, and Father*, who used to ask me every day if I got my Ph.D.

A great wife *Naima* for their loving consideration and great confidence in me all through these years.

My kisses to my children (*Noran, Nezar, Nariman*).

© 2026 Abdoalnasir Almabrok

All rights reserved. This Doctoral Dissertation is submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy. No part of the material protected by this Copyright notice may be reproduced or utilized from or by any means, electronic or mechanical, including prototyping, recording, or by any information storage and retrieval system, without the prior permission of the authors.

Requests can be sent at:

- ***nass_90210@yahoo.com***
- ***nassnaan@unipi.gr***

LIST OF PUBLICATIONS

- [P1] **Abdoalnasir Almabrok**, M. Psarakis, A. Dounis. An efficient FPGA implementation of the big bang-big crunch optimization algorithm. in *Applied Reconfigurable Computing. Architectures, Tools, and Applications: 14th International Symposium, ARC 2018*, Santorini, Greece, May 2-4, 2018, Proceedings 14. 2018. Springer.
Doi.org/10.1007/978-3-319-78890-6_14
- [P2] **Abdoalnasir Almabrok**, M. Psarakis, and A. Dounis, "Fast tuning of the PID controller in an HVAC system using the big bang-big crunch algorithm and FPGA technology". *Algorithms*, 2018. 11(10): p. 146.
Doi. org/10.3390/a11100146
- [P3] M. Psarakis, A. Dounis, **Abdoalnasir Almabrok**, S. Stavrinidis, G. Gkekas. "An FPGA-based accelerated optimization algorithm for real-time applications". *Journal of Signal Processing Systems*. 2020 Oct;92(10):1155-76.
Doi.org/10.1007/s11265-020-01522-5
- [P4] **Abdoalnasir Almabrok**, M. Psarakis, A. Dounis. "Implementing a Hardware-Based Big-Bang Big-Crunch optimized controller with Fractional-Order for a Heating, Ventilation, and air conditioning system". *Energy and Buildings*, Volume 299, 2023, 113613, ISSN 0378-7788.
Doi.org/10.1016/j.enbuild.2023.113613.
- [P5] **Abdoalnasir Almabrok**, M. Psarakis and A. Dounis, "Considerations on LAMDA Controller With BB-BC Tuning Toward Hardware Implementation" in *IEEE Transactions on Industrial Informatics*, vol. 20, no. 11, pp. 13113-13123, Nov. 2024,
Doi.10.1109/TII.2024.3431089.

ABSTRACT

Ensuring optimal performance of control units, such as Proportional–Integral–Derivative (PID) controllers, is essential for maintaining efficiency, stability, and fast response time in industrial automation systems, such as Heating, Ventilation, and Air Conditioning (HVAC) systems. However, conventional controllers often struggle with the complex, non-linear dynamics and disturbances present in real-time industrial applications. These challenges have created a growing need for advanced, robust, and adaptive optimization techniques. This PhD thesis focuses on a robust optimization algorithm, the Big Bang Big Crunch (BB-BC), as a fast, reliable solution for auto-tuning control parameters in practical applications.

Motivated by the increasing demand for high-speed, energy-efficient, and real-time control in industrial systems, this work explores hardware-accelerated implementations of optimization algorithms, focusing on Field-Programmable Gate Arrays (FPGAs). Although the BB-BC algorithm is well-suited for real-time applications due to its rapid convergence and low computational complexity, software-based implementations running on conventional processors often fail to meet strict timing constraints. To overcome this limitation, the thesis proposes a novel FPGA-based architecture for the BB-BC algorithm. Field programmable technology offers a realistic solution to improve response time and save power consumption, and presents an efficient way to implement complex algorithms, improving real-time response in control systems. In other words, the field programmable technology provides a flexible and parallel computing environment for executing optimization algorithms in real-time industrial applications. In this thesis, we show that the BB-BC algorithm is well-suited for FPGA-based acceleration; we propose hardware performance mechanisms that overcome the design limitations observed in classical genetic evolutionary algorithms, which often hinder the performance of their hardware-based accelerators and exhibit limitations, particularly in terms of scalability and response time. We present an FPGA-based accelerator for the BB-BC algorithm, which incorporates a fully pipelined design of both BB-BC phases, Big Bang Phase (BBP), and Big Crunch Phase (BCP). A robust methodological framework for this FPGA-based accelerator has been developed and validated on a typical optimization problem using standard mathematical benchmarks. The experimental results demonstrate significant speedup over software counterparts.

Despite the growing interest in robust, real-time optimization for industrial control systems typically characterized by disturbances, nonlinearities, and time delays, existing literature offers few solutions that combine fast-convergent algorithms with hardware acceleration. This thesis addresses this gap by studying the integration of BB-BC into multiple control strategies, including PID, Fractional-Order PID (FOPID), and the Learning Algorithm for Multivariate Data Analysis (LAMDA), and by evaluating their FPGA-based implementations.

In particular, the thesis examines the use of BB-BC to tune PID and FOPID controllers, followed by hardware deployment on FPGA platforms. The approach is validated on an HVAC system, using various modeling strategies such as Second-Order Plus Time Delay (SOPTD) models. Additionally, BB-BC-powered PID, FOPID, and LAMDA controllers are designed and analyzed for robustness under different disturbance profiles, enabling comparative evaluation across diverse industrial scenarios. MATLAB/Simulink simulations of the combined optimization–control framework further highlight the superior timing behavior, scalability, and energy efficiency of the FPGA-accelerated BB-BC solutions compared to traditional techniques. The key objectives of this PhD thesis are to:

- Develop a novel FPGA-based implementation of the BB-BC optimization algorithm and demonstrate its performance through representative case studies.
- Integrate the BB-BC algorithm into PID and FOPID controllers for parameter optimization, and validate the proposed approach on HVAC systems.
- Extend the BB-BC optimization approach to LAMDA controllers to explore its applicability in more complex industrial environments.
- Enhance real-time performance and energy efficiency through hardware-accelerated optimization using FPGA technology.

Keywords: *Big Bang-Big Crunch (BB-BC) algorithm; Evolutionary algorithms; Field Programmable Gate Array (FPGA); Hardware Description Language (HDL); PID controller; Fractional Order PID (FOPID) controller; Learning Algorithm Multivariate Data Analysis (LAMDA).*

ΠΕΡΙΛΗΨΗ

Η διασφάλιση της βέλτιστης απόδοσης των μονάδων ελέγχου, όπως οι αναλογικοί-ολοκληρωτικοί-παραγώγιμοι ελεγκτές “Proportional-Integral-Derivative” (PID), είναι απαραίτητη για τη διατήρηση της αποδοτικότητας, της σταθερότητας και του γρήγορου χρόνου απόκρισης σε βιομηχανικά συστήματα αυτοματισμού, όπως τα συστήματα θέρμανσης, εξαερισμού και κλιματισμού “Heating, Ventilation, and Air Conditioning” (HVAC). Ωστόσο, οι συμβατικοί ελεγκτές συχνά αντιμετωπίζουν τις πολύπλοκες, μη-γραμμικές δυναμικές και τις διαταραχές που υπάρχουν σε βιομηχανικές εφαρμογές πραγματικού χρόνου. Αυτές οι προκλήσεις έχουν δημιουργήσει μια αυξανόμενη ανάγκη για προηγμένες, ισχυρές και προσαρμοστικές τεχνικές βελτιστοποίησης. Αυτή η διδακτορική διατριβή εστιάζει σε έναν ισχυρό αλγόριθμο βελτιστοποίησης, τον “*Big Bang-Big Crunch*” (BB-BC), ως μια γρήγορη και αξιόπιστη λύση για την αυτόματη ρύθμιση παραμέτρων ελέγχου σε πρακτικές εφαρμογές.

Με κίνητρο την αυξανόμενη ζήτηση για έλεγχο υψηλής ταχύτητας, ενεργειακά αποδοτικό και σε πραγματικό χρόνο σε βιομηχανικά συστήματα, η παρούσα εργασία διερευνά υλοποιήσεις αλγορίθμων βελτιστοποίησης με επιτάχυνση υλικού, εστιάζοντας σε προγραμματιζόμενες στο πεδίο συστοιχίες πυλών “Field-Programmable Gate Arrays” (FPGA). Παρόλο που ο αλγόριθμος BB-BC είναι κατάλληλος για εφαρμογές πραγματικού χρόνου λόγω της ταχείας σύγκλισής του και της χαμηλής υπολογιστικής πολυπλοκότητάς του, οι υλοποιήσεις που βασίζονται σε λογισμικό και εκτελούνται σε συμβατικούς επεξεργαστές συχνά δεν πληρούν αυστηρούς χρονικούς περιορισμούς. Για να ξεπεραστεί αυτός ο περιορισμός, η διατριβή προτείνει μια νέα αρχιτεκτονική βασισμένη σε FPGA για τον αλγόριθμο BB-BC. Η τεχνολογία προγραμματισμού πεδίου προσφέρει μια ρεαλιστική λύση για τη βελτίωση του χρόνου απόκρισης και την εξοικονόμηση κατανάλωσης ενέργειας, και παρουσιάζει έναν αποτελεσματικό τρόπο υλοποίησης σύνθετων αλγορίθμων, βελτιώνοντας την απόκριση σε πραγματικό χρόνο σε συστήματα ελέγχου. Με άλλα λόγια, η τεχνολογία προγραμματισμού πεδίου παρέχει ένα ευέλικτο και παράλληλο υπολογιστικό περιβάλλον για την εκτέλεση αλγορίθμων βελτιστοποίησης σε βιομηχανικές εφαρμογές πραγματικού χρόνου. Σε αυτή τη διατριβή, δείχνουμε ότι ο αλγόριθμος BB-BC είναι

κατάλληλος για επιτάχυνση με χρήση τεχνολογίας FPGA. Προτείνουμε μηχανισμούς απόδοσης υλικού που ξεπερνούν τους περιορισμούς σχεδιασμού που παρατηρούνται στους κλασικούς γενετικούς εξελικτικούς αλγόριθμους, οι οποίοι συχνά εμποδίζουν την απόδοση των επιταχυντών που βασίζονται σε υλικό και παρουσιάζουν περιορισμούς, ιδιαίτερα όσον αφορά την επεκτασιμότητα και τον χρόνο απόκρισης. Παρουσιάζουμε έναν επιταχυντή βασισμένο σε FPGA για τον αλγόριθμο BB-BC, ο οποίος ενσωματώνει τη σχεδίαση και των δύο φάσεων του αλγορίθμου, της Φάσης Μεγάλης Έκρηξης “*Big Bang Phase*” (BBP) και της Φάσης Μεγάλης Κρούσης “*Big Crunch Phase*” (BCP), με μηχανισμό διοχέτευσης (pipelined design). Ένα ισχυρό μεθοδολογικό πλαίσιο για αυτόν τον FPGA επιταχυντή έχει αναπτυχθεί και επικυρωθεί σε ένα τυπικό πρόβλημα βελτιστοποίησης χρησιμοποιώντας τυπικά μαθηματικά benchmarks. Τα πειραματικά αποτελέσματα καταδεικνύουν σημαντική επιτάχυνση σε σύγκριση με αντίστοιχες υλοποιήσεις λογισμικού.

Παρά το αυξανόμενο ενδιαφέρον για εύρωστη βελτιστοποίηση σε πραγματικό χρόνο για βιομηχανικά συστήματα ελέγχου - που συνήθως χαρακτηρίζονται από διαταραχές, μη γραμμικότητες και χρονικές καθυστερήσεις - η υπάρχουσα βιβλιογραφία προσφέρει λίγες λύσεις που συνδυάζουν αλγόριθμους ταχείας σύγκλισης με επιτάχυνση υλικού. Αυτή η διατριβή αντιμετωπίζει αυτό το κενό μελετώντας την ενσωμάτωση του BB-BC σε πολλαπλές στρατηγικές ελέγχου, συμπεριλαμβανομένων των PID, Fractional-Order PID (FOPID) και του Αλγορίθμου Μάθησης για Ανάλυση Πολυμεταβλητών Δεδομένων “*Learning Algorithm for Multivariate Data Analysis*” (LAMDA), και αξιολογώντας τις υλοποιήσεις τους με χρήση FPGA.

Συγκεκριμένα, η διατριβή εξετάζει τη χρήση του BB-BC για την ρύθμιση των ελεγκτών PID και FOPID, ενώ προτείνει και την υλοποίησή τους με χρήση υλικού σε πλατφόρμες FPGA. Η προσέγγιση επικυρώνεται σε ένα σύστημα HVAC, χρησιμοποιώντας διάφορες στρατηγικές μοντελοποίησης, όπως μοντέλα Δεύτερης Τάξης Συν Χρονικής Καθυστερήσης “*Second-Order Plus Time Delay*” (SOPTD). Επιπλέον, οι ελεγκτές PID, FOPID και LAMDA που συντονίζονται με χρήση του BB-BC, σχεδιάζονται και αναλύονται για την ανθεκτικότητά τους υπό διαφορετικά προφίλ διαταραχών, επιτρέποντας τη συγκριτική αξιολόγηση σε διαφορετικά βιομηχανικά σενάρια. Προσομοιώσεις *MATLAB/Simulink* του συνδυασμένου πλαισίου βελτιστοποίησης-ελέγχου υπογραμμίζουν περαιτέρω την ανώτερη συμπεριφορά

χρονισμού, την επεκτασιμότητα και την ενεργειακή απόδοση των λύσεων BB-BC με επιτάχυνση FPGA σε σύγκριση με τις παραδοσιακές τεχνικές.

Οι βασικοί στόχοι αυτής της διδακτορικής διατριβής είναι:

- Η ανάπτυξη μιας νέας υλοποίησης του αλγορίθμου βελτιστοποίησης BB-BC με χρήση FPGA και η επίδειξη της απόδοσής της μέσω αντιπροσωπευτικών περιπτώσεων μελέτης.
- Η ενσωμάτωση του αλγορίθμου BB-BC σε ελεγκτές PID και FOPID για βελτιστοποίηση παραμέτρων και η επικύρωση της προτεινόμενης προσέγγισης σε συστήματα HVAC.
- Η επέκταση της προσέγγισης βελτιστοποίησης BB-BC σε ελεγκτές LAMDA για να διερευνηθεί η εφαρμογή της σε πιο σύνθετα βιομηχανικά περιβάλλοντα.
- Η βελτίωση της απόδοσης πραγματικού χρόνου και της ενεργειακής απόδοσης μέσω βελτιστοποίησης με επιτάχυνση υλικού χρησιμοποιώντας τεχνολογία FPGA.

Λέξεις κλειδιά: *Big Bang-Big Crunch (BB-BC) αλγόριθμος; Εξελικτικοί Αλγόριθμοι; Προγραμματιζόμενες στο Πεδίο Συστοιχίες Πυλών (FPGAs); Γλώσσα Περιγραφής Υλικού; Ελεγκτής PID; Ελεγκτής PID Κλασματικής Τάξης (FOPID); Αλγόριθμος Μάθησης Πολυμεταβλητής Ανάλυσης Δεδομένων (LAMDA).*

TABLE OF CONTENTS

1	Introduction.....	2
1.1	Evolutionary algorithms.....	2
1.2	Cutting-Edge in EA.....	3
1.3	Evolutionary Algorithms: Advances and Applications	3
1.4	Key factors of EA.....	5
1.5	Motivation and objectives of the thesis.....	6
1.6	Contributions of the thesis.....	8
1.7	Organization of the thesis	10
2	Evolutionary Algorithms and Optimization Controllers	12
2.1	Introduction	12
2.2	Evolutionary Algorithms	13
2.3	Big Bang – Big Crunch Algorithm	15
2.4	Optimization Controllers.....	18
2.5	PID Control Algorithm	18
2.6	Fractional Order $PI\lambda D\mu$ (FOPID) Controller	20
2.6.1	GL-based FOPID.....	22
2.6.2	Discretized (D) FOPID Controller	23
2.6.3	Advantages of the FOPID Controller over the PID Controller	24
2.7	LAMDA-Based Controller.....	25
2.7.1	OWA Operator.....	27
2.8	Integration of EAs into FPGAs	29
2.9	HVAC System Integration with Control Strategies	30
2.9.1	HVAC Systems using PID, FOPID, and LAMDA Controllers	32
3	FPGA-based Acceleration of the BB-BC Optimization Algorithm	35
3.1	Introduction	35

3.2	Related work.....	38
3.3	Proposed FPGA architecture	40
3.3.1	Pipelined BB-BC engine.....	40
3.3.2	High-speed reduction circuits for the CoM calculation.....	43
3.3.3	Parallel FPGA BB-BC architecture	44
3.3.4	Fitness function approximation method	46
3.3.5	Application of the ANFIS method	49
3.4	Evaluation of the FPGA BB-BC architecture	53
3.4.1	Comparison with GA for different fitness functions.....	53
3.4.2	Evaluation of FPGA area and frequency.....	55
3.4.3	Comparison with an HLS-based accelerator	59
3.4.4	Design optimization considerations.....	62
3.4.5	Comparison with s/w and GPU-based approaches	62
3.4.6	Case study: image search problem.....	64
3.5	Summary	65
4	Efficient Implementations of PID Controllers for HVAC Systems.....	68
4.1	Introduction	68
4.2	Optimizing the PID controller of an HVAC with the BB-BC algorithm	73
4.2.1	Tuning the PID controller parameters using the BB-BC algorithm.....	74
4.2.2	Implementing the PID Controller using FPGA.....	77
4.2.3	Implementing the BB-BC algorithm using FPGA	77
4.2.4	FPGA-In-The-Loop (FIL)	80
4.2.5	HVAC system architecture.....	81
4.2.6	Experimental evaluation of the proposed PID-HVAC system.....	83
4.3	Optimizing the FOPID controller of an HVAC with the BB-BC algorithm.....	90
4.3.1	Regulating an HVAC system using the FOPID controller	91
4.3.2	Tuning the FOPID controller parameters using the BB-BC algorithm.....	92
4.3.3	Implementing the FOPID-BB-BC controller using FPGA	98
4.3.4	Experimental evaluation of the FOPID-HVAC system	103
4.4	Summary	104

5	An FPGA-based LAMDA controller with BB-BC tuning for HVAC systems.....	107
5.1	Introduction	107
5.2	Tuning the LAMDA Controller of a MIMO HVAC System using the BB-BC Algorithm.....	110
5.2.1	Integration of the LAMDA Controller into HVAC.....	111
5.3	Evaluation of the BB-BC-LAMDA Control Unit.....	114
5.4	FPGA-based Implementation of the System.....	116
5.4.1	FPGA-based BB-BC engine	116
5.4.2	LAMDA Controller Architecture.....	117
5.4.3	BB-BC-LAMDA Controller Design.....	119
5.4.4	Simulation Interpretation and Results.....	122
5.4.5	Limitations and considerations.....	123
5.5	Summary	123
6	Conclusions and Outlook.....	125
6.1	Summary of the Work Done	125
6.2	Summary of the Contributions	126
6.3	Future Work.....	127
7	References.....	130

LIST OF FIGURES

Figure 1. Classification of EAs.....	3
Figure 2. The utilization of the BB-BC algorithm in recent years	7
Figure 3. The EA process as a loop.....	14
Figure 4. BB-BC optimization algorithm flow chart.	16
Figure 5. Block diagram of a typical PID control structure	19
Figure 6. Block diagram of a typical FOPID controller	22
Figure 7. Block diagram of a GL-based FOPID controller.....	22
Figure 8. Block diagram of a discretized (D)-FOPID controller.....	23
Figure 9. General scheme of the LAMDA algorithm	29
Figure 10. Block diagram of the LAMDA controller	29
Figure 11. Block diagram of a typical HVAC system	32
Figure 12. Block diagram of a conventional HVAC system and PID control structure.....	33
Figure 13. Block diagram of the BB-BC engine	41
Figure 14. Pipelined execution of the BB-BC engine.....	41
Figure 15. Parallel BB-BC architecture: a) CalcCoM module, (b) 2-cand-add module, (c) CoM-reduction module, (d) Four parallel BB-BC engines	45
Figure 16. A two-input first-order Takagi–Sugeno fuzzy reasoning	47
Figure 17. An equivalent ANFIS structure to a first-order Takagi–Sugeno fuzzy	47
Figure 18. ANFIS model structure with two inputs and one output for	50
Figure 19. T-S membership functions plot for inputs x_1, x_2	50
Figure 20. Sum of square error versus the number of epochs for training dataset	51
Figure 21. Surface graph showing approximation of the Ackley function	52
Figure 22. Performance comparison between BB-BC and GA (Convergence speed)	55
Figure 23. Comparison between single and dual engines for the Rosenbrock function: a) execution time (μs), b) execution time $\times$ area.....	58
Figure 24. Comparison between HDL operators and CoreGen IP cores for the Sphere function: a) execution time (μs), b) execution time $\times$ area	59
Figure 25. Convergence and time response in seconds of the objective functions ($ITAE, ITSE, IAE, ISE$) for different numbers of iterations: a) 100, b) 200, c) 500 and d) 1000 iterations.....	76
Figure 26. Block diagram of a typical PID controller FPGA architecture.....	77
Figure 27. Top-level diagram of the PID-BB-BC FPGA architecture	79

Figure 28. The FIL acceleration scheme	80
Figure 29. Proposed system architecture	81
Figure 30. Block diagram of our MATLAB/Simulink model	83
Figure 31. Time responses in seconds of four optimization methods (BB-BC, BB-BC-FPGA, PSO, GA) for different numbers of iterations (a, b, c, and d for 100, 200, 500, and 1000 iterations, respectively).....	85
Figure 32. The (a) system output, (b) control signal, (c) error signal, and.....	87
Figure 33. Objective function convergence for all optimization methods and for different numbers of iterations (a, b, c, and d for 100, 200, 500, and 1000 iterations, respectively) .	88
Figure 34. Objective function convergence for all four optimization methods.....	90
Figure 35. System architecture: FOPID controller, BB-BC & HVAC	93
Figure 36. Time responses (<i>a, b, c, d, e, and f</i>) of the FOPID controller using the six HVAC cases for the six performance indexes (<i>J1, J2, J3, J4, J5, and J6</i>).....	96
Figure 37. The effectiveness of the FOPID-BB-BC scheme against a disturbance signal.....	96
Figure 38. D-FOPID controller parameters (K_p, K_i, K_d, λ , and μ), error $e(k)$, and output $u(k)$	99
Figure 39. Discretized (D)-FOPID controller structure	100
Figure 40. GL-FOPID controller design structure	101
Figure 41. GL-FOPID & BB-BC FPGA design structure and HVAC system	103
Figure 42. Control and error signals for different numbers of BB-BC iterations	104
Figure 43. Block diagram of the LAMDA / BB-BC scheme for the HVAC system.....	112
Figure 44. Output signals of (a) temperature and (b) humidity	113
Figure 45. The step response curves of (a) temperature and (b) humidity.....	116
Figure 46. Block diagram of the LAMDA controller architecture	119
Figure 47. Design of the BB-BC-LAMDA for the MIMO HVAC system.....	120
Figure 48. FSM of the BB-BC-LAMDA controller	122

LIST OF TABLES

Table 1. Equivalence of parameters for PID controllers	20
Table 2. FOPID vs. PID feature comparison.....	24
Table 3. FPGA BB-BC architecture parameters.....	43
Table 4. BB-BC engine latency and arithmetic operations	43
Table 5. Parallel BB-BC scheme latency and operations	46
Table 6. The Sugeno-type if-then rules	51
Table 7. ANFIS parameter counts for approximating the Ackley function	51
Table 8. Performance comparison between BB-BC and GA.....	54
Table 9. Reconfigurable resources and maximum operating frequency of the FPGA BB-BC	56
Table 10. Reconfigurable resources and maximum operating frequency of the FPGA BB-BC architecture for the three fitness functions (Rosenbrock, Simple, and Sphere)	56
Table 11. Area comparison between the HLS-based and the proposed	61
Table 12. Performance comparison between the HLS-based and the proposed.....	61
Table 13. Synthesis report for the original Ackley and the ANFIS-based Ackley functions	62
Table 14. Execution time of the C-program, CUDA version and FPGA approach of the BB-BC algorithm for the Rosenbrock function for $M=500$ (in ms)	63
Table 15. Execution time of the GA s/w and BB-BC s/w in an embedded platform and the FPGA BB-BC approach for the Rosenbrock function for $N=256$ (in ms) Error! Bookmark not defined.	
Table 16. Execution time of the C-program, CUDA version and FPGA approach of the BB-BC	65
Table 17. Comparison of different errors, <i>ITAE</i> , <i>ITSE</i> , <i>IAE</i> , <i>ISE</i> as performance index.....	75
Table 18. FPGA resource utilization summary of the PID-BBBC control unit	79
Table 19. Transient performance comparison.....	86
Table 20. Objective function convergence details (iterations and execution time)	89
Table 21. Computation time in (s) for GA, BB-BC (in sw)and BB-BC-FPGA (proposed approach)	90
Table 22. Control parameters and time response for the six HVAC use cases	94
Table 23. Time response performance for the BB-BC, GA, $H\infty$ algorithms	97
Table 24. FOPID control parameters for the BB-BC, GA, PSO algorithms	97

Table 25. Transient performance comparison.....	98
Table 26. Specifications of MPC	98
Table 27. GL-FOPID performance vs. window size (L)	102
Table 28. FPGA resource utilization of the FOPID-BB-BC schemes on a Zynq UltraScale+ device.....	103
Table 29 Performance of the FPGA implementation of the GL-FOPID-BB-BC controller ..	104
Table 30. Numerical values of the system parameters	111
Table 31. Numerical values of the parameters at the operating equilibrium point	111
Table 32. Membership functions and min & max outputs of LAMDA & HVAC system	112
Table 33. BB-BC algorithm parameters for LAMDA controller optimization	114
Table 34. Time response for temperature with different iteration numbers	114
Table 35. Time response for humidity with different iteration numbers	114
Table 36. Time response: original LAMDA vs. optimization algorithms (GA, PSO, and BB-BC)	115
Table 37. Number of iterations, MFs and constant gains of the LAMDA controller	115
Table 38. Hardware BB-BC-LAMDA architecture	117
Table 39. BB-BC-LAMDA latency and arithmetic operations	121
Table 40. FPGA resources utilization for the BB-BC-LAMDA design.....	122

ACRONYMS

ACO	Ant Colony Optimization
ANFIS	Adaptive Neuro Fuzzy Inference System
BB	Big Bang
BBP	Big Bang Phase
BC	Big Crunch
BCP	Big Crunch Phase
CFE	Continuous Fractional Expansion
Clk	Clock
CM	Chaotic Map
CoM	Center of Mass
DE₁	Differential Equation
DE₂	Differential Evolution
DL	Deep Learning
EA	Evolutionary Algorithms
EM	Energy Management
FD	Frequency Domain
FDT	Frequency Domain Technique
FIS	Fuzzy Inference System
FiT	Fitness function
FL	Fuzzy Logic
FOC	Fractional Order Calculus
FOO	Fractional Order Operator
FOPID	Fractional Order PID controller
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GA	Genetic Algorithm
GAD	Global Adequacy Degree
GPUs	Graphics Processing Units
Hum	Humidity
HVAC	Heating, Ventilation and Air Conditioning
IAE	Integral Absolute Error
ISE	Integral Square Error
ITAE	Integral Time Absolute Error
ITSE	Integral Time Square Error
LAMDA	Learning Algorithm Multivariate Data Analysis
LSFR	Linear Feedback Shift Register
LUT	Look Up Table

MAD	M arginal A dequacy D egree
MF	M embership F unction
MIMO	M ultiple- I ntput and M ultiple- O utput
ML	M achine L earning
NB	N egative B ig
NS	N egative S mall
OWA	O rdered W eighted A veraging
PB	P ositive B ig
PGAs	P arallel G enetic A lgorithms
PID	P roportional I ntegrating D erivative
PS	P ositive S mall
PSO	P article S warm O ptimization
RandGen	R andom G enerator
REG	R egister
RNG	R andom N umber G enerator
RTL	R egister T ransfer L evel
SBF	S igned B inary F ractional
SOPTD	S econd O rders P lus T ime D elay
SP	S et P oint
SSE	S tate S pace E quation
SST	S teady S tate T ime
STD	S tandard D eviation
Temp	T emperature
TS	T aylor S eries
VHDL	V ery H igh-Speed I ntegrated C ircuit H ardware D escription L anguage
ZE	Z ero

CHAPTER 1

Introduction

1 Introduction

1.1 Evolutionary algorithms

Evolutionary Algorithms (EAs) are a class of optimization techniques that are inspired by the process of natural evolution. They are used to find solutions to complex problems by mimicking the process of natural selection. These algorithms are particularly useful in large, single or multi-dimensional search spaces, where traditional optimization methods may struggle. EAs are widely applied across various fields, including Artificial Intelligence (AI), machine learning, industrial control systems, resource management, etc., due to their ability to handle complex, non-linear problems efficiently. These algorithms aim not to find the best solution to a problem, but to explore the solution space with minimal effort and without being constrained by initial conditions. This allows researchers and practitioners to find solutions that would be difficult to reach using other search methods. The main advantage of these algorithms is that they allow for testing a large number of potential solutions to identify the best one. There are several common examples of EAs, each with its own unique approach and applications. One of the most well-known EAs is the Genetic Algorithm (GA), which simulates the process of natural selection by mimicking the process of genetic mutation and crossover. Other popular EAs are the Particle Swarm Optimization (PSO), which simulates the behavior of a flock of birds or a swarm of bees in order to find the optimal solution to a problem [1] and the Ant Colony Optimization (ACO), which is a probabilistic technique used to solve combinatorial optimization problems, inspired by the behavior of ants searching for food [2-4]. This natural process is modeled in ACO algorithms to find optimal or near-optimal solutions [5]. Another EA example is the Big Bang-Big Crunch (BB-BC) algorithm [6], which combines both global and local search, exploiting the principles of genetic crossover and mutation. All these EA algorithms, such as GA, PSO, ACO, and BB-BC, are widely used in real-world applications, in tasks like optimizing control parameters, automating industrial processes, and solving combinatorial optimization problems. Figure 1 shows a short classification of the EAs studied in this thesis based on their key techniques. It is worth noting that there are additional optimization algorithms not mentioned in this introduction, which present only the algorithms studied in this thesis, such as GA and PSO.

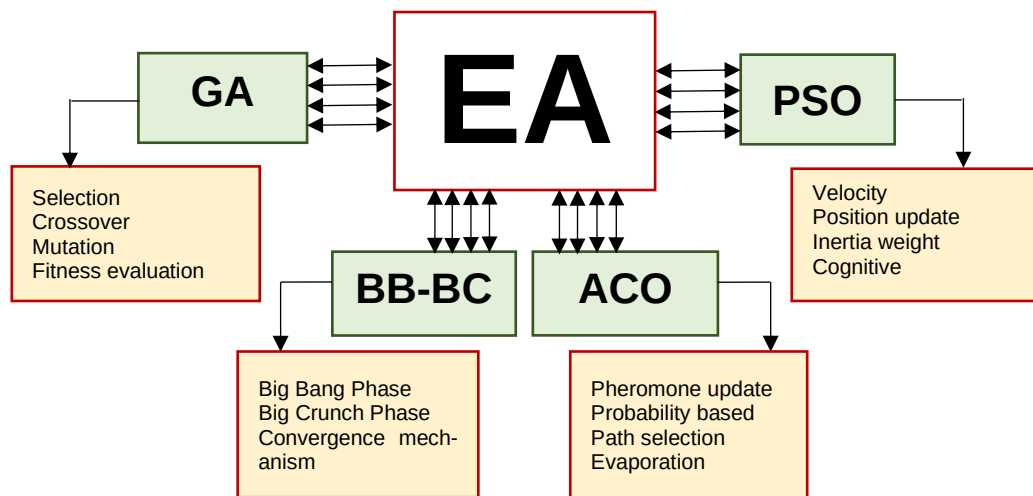


Figure 1. Classification of EAs

1.2 Cutting-Edge in EA

The main branch of metaheuristic algorithms is EA, which is widely popular among researchers. Many research efforts have been conducted on various hybrid EAs to solve different optimization problems. Several existing EA algorithms have been merged and applied to various industrial applications; these algorithms can be classified as hybrid. The EAs have seen significant advancements with the introduction of hybrid techniques that combine EA with Machine Learning (ML) methods. For instance, Qin et al., [7] integrated Deep Learning (DL) models with PSO to improve efficiency in solving large-scale optimization problems. Additionally, Cavallaro et al., [8] showed that combining GA with ML achieves high effectiveness, as demonstrated in a case study on image reconstruction. In general, when combined with other techniques, EAs provide a promising strategy for control systems and design applications. This thesis presents novel EAs addressing complex computational and non-linear problems, such as the BB-BC algorithm for tuning LAMDA controllers, and proposes hardware implementations.

1.3 Evolutionary Algorithms: Advances and Applications

Over the past decade, EAs have experienced significant advancements, particularly in parameter optimization, integration with other computational methods, and hardware implementation. Additionally, the idea of using hardware, such as Field Programmable Gate Arrays (FPGAs) and Graphics Processing Units (GPUs), to implement EAs has boosted

their processing capabilities, enabling real-time applications in fields such as robotics and autonomous systems. These advancements have expanded the scope and practical use of EAs in solving complex, real-world problems, particularly in industrial applications.

We can summarize the most important recent advances in EAs:

- *Hybrid optimization approaches*: Combining multiple algorithms, such as GA, PSO, etc., improves solution accuracy and addresses specific weaknesses, including convergence speed and local optima avoidance. Recent hybrid methods, like PSO-GA, leverage the strengths of different EAs to enhance performance in complex real-world problems, as presented in [9] and [10].
- *Multi-objective optimization*: Techniques that help researchers define solutions for complex, multi-objective problems, allowing trade-offs between competing objectives [11].
- *Real-time engineering and hardware implementation*: EAs are effectively applied [12, 13] to optimize systems in noisy, dynamic environments with multiple constraints. Implementing these algorithms on hardware accelerator platforms, such as FPGAs and GPUs, further enhances processing speed and efficiency.

Real-world applications of EAs span several industrial areas. For example, Sun et al., [14] presented a combined ML-EA approach to optimize Energy Management, Ali et al., [15] proposed a database query optimization, Fathi et al., [16] suggested an optimization approach based on artificial intelligence and EAs, and Larsen and Kim [17] studied robot motion planning. Some representative EA use cases are as follows:

- *Optimization*: EAs have been widely used in solving complex optimization problems.
- *Machine learning*: EAs are increasingly integrated with ML, particularly for tasks like feature selection, hyperparameter tuning, and model training optimization.
- *Artificial intelligence*: EAs are combined with artificial intelligence when traditional optimization methods struggle. By mimicking the process of natural evolution, EAs can effectively explore large solution spaces and adapt to changing environments, making them useful for complex, dynamic problems.
- *Databases*: EAs are used to optimize query performance in large-scale databases. Techniques such as Differential Evolution (DE) and GA help in minimizing query time and resource usage.

- *Pattern recognition*: EAs are used in image analysis and computer vision to optimize classifiers and improve accuracy in recognizing patterns, from medical imaging to facial recognition technologies.
- *Robot motion planning*: EAs are applied in mobile robot navigation and path planning.
- *Forecasting*: EAs have been used in predictive models, where their adaptability to changing data patterns makes them suitable for real-time and dynamic forecasting scenarios.

These applications demonstrate the versatility of EAs in solving a wide range of complex problems across various industries.

1.4 Key factors of EA

In the following, we summarize the basic characteristics and the main advantages and disadvantages of the state-of-the-art EA strategies in order to justify the choice of the BB-BC algorithm as the central optimization algorithm of this thesis:

- *GA*: They utilize a population-based search with crossover and mutation, which allows exploration and exploitation of the search space. They are generally slower due to the iterative nature of genetic operations, but effective in finding global optima [18].
- *PSO*: They employ a swarm-based approach where particles are guided by collective and individual knowledge, leading to fast convergence in continuous spaces. Typically, they are faster than GAs, especially in continuous domains, due to the direct adjustment of particle positions based on fitness feedback [19].
- *ACO*: They use pheromone trails to reinforce promising paths, making it particularly effective in discrete, combinatorial spaces. The limitations of ACO are not always the best choice, especially for large or continuous optimization problems [20].
- *BB-BC*: They combine random exploration (BB with focused exploitation BC), balancing diversity and convergence speed. They often outperform GA in convergence speed [21], especially when executed in parallel architectures like FPGAs.

In conclusion, the choice of an EA depends on the specific requirements of the target optimization problem. GAs provide a robust framework for a wide range of problems, but

may require more computational time. The PSO algorithm offers a fast convergence rate for continuous problems, while the ACO algorithm excels at discrete combinatorial optimization. Based on its balance between exploration and exploitation, the BB-BC algorithm is particularly advantageous for real-time applications and can outperform traditional methods in convergence speed, primarily when implemented on parallel architectures.

1.5 Motivation and objectives of the thesis

The BB-BC algorithm is a robust mathematical optimization technique, one member of the family of EA, such as GA, PSO, and DE. The BB-BC employs the concept of the bang phase and the crunch phase to provide optimal solutions. The idea is to conduct an efficient search over open spaces by maintaining a well-organized pool of populations that span the search space. The BB-BC has better intelligence than random search algorithms because it uses gathered data to guide the search to the best-performing region within the solution space and to reach the optimal solutions from the first iterations. On the other hand, the BB-BC algorithm has an additional advantage, particularly in terms of its intensification feature, which becomes more pronounced as the algorithm progresses, while diversification is notable at the beginning. This technique increases the likelihood of avoiding getting stuck in local optima compared to other algorithms, such as GA or PSO.

In recent years, the use of the BB-BC algorithm has shown a varied pattern of adoption in the fields of optimization and computational intelligence compared to more established algorithms such as GA, PSO, DE, and ACO. Despite its relatively recent inception, the BB-BC algorithm has found a niche in applications that require global optimization, thanks to its unique approach of combining an expansive search phase (BB) with a convergence phase (BC). When analyzing academic and research publications over the past recent years (2014-2025), the figure indicates that the BB-BC algorithm has an increasing frequency of use side by side with other popular evolutionary algorithms such as GA, PSO, DE, and ACO, which can be attributed to several factors such as its simplicity and reported efficiency in various scenarios, although it does not necessarily imply broader acceptance or more extensive documentation compared to more established methods like GA and PSO.

The analysis of publication counts reveals that BB-BC, GA, PSO, ACO, and DE continue to dominate the optimization landscape, reflecting their entrenched roles and continuous

innovation in their methodologies. On the other hand, the BB-BC algorithm, as frequently cited in recent literature available on Google Scholar, continues to garner interest for its potential in solving specific optimization problems, indicating a steady but niche growth in research interest and application [Google Scholar analyzed]. Figure 2 shows the growth of research on BB-BC algorithm optimization methods as analyzed by Google Scholar in recent years.

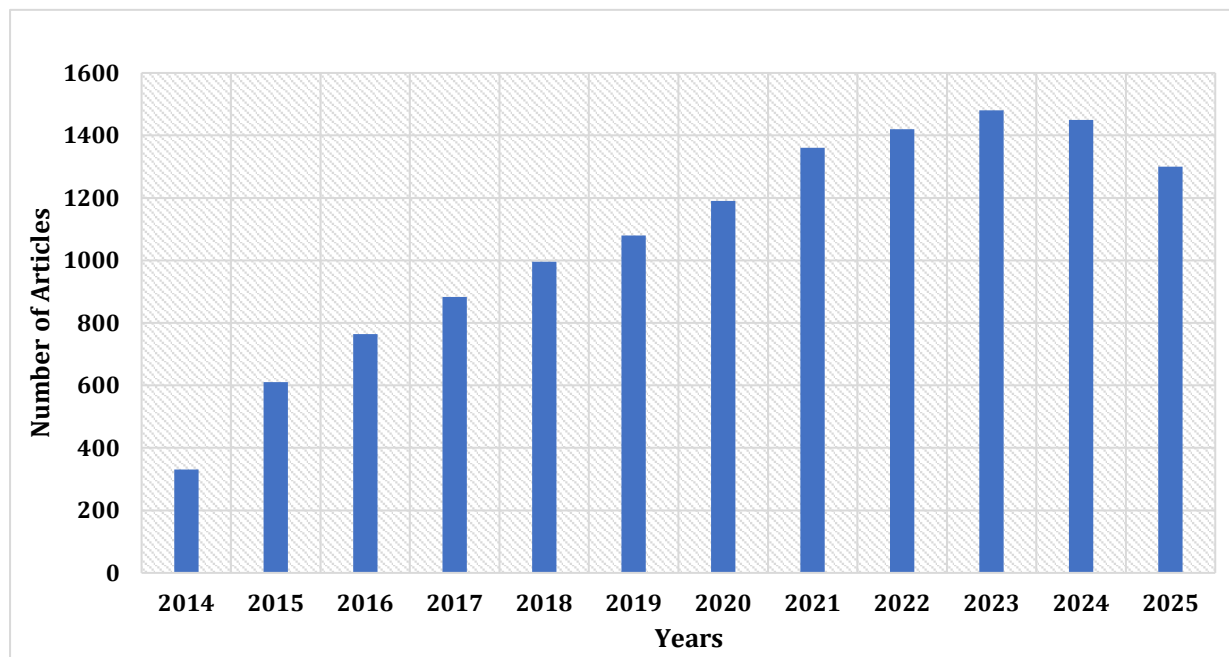


Figure 2. The utilization of the BB-BC algorithm in recent years

The main goal of this thesis is to explore advanced optimization techniques by integrating the BB-BC algorithm with various control methods, including Proportional-Integral-Derivative (PID) and Fractional Order PID (FOPID). It aims to address key challenges in the control and optimization of HVAC systems by implementing innovative strategies across multiple hardware platforms (FPGA). It also involves developing and implementing the LAMDA controller as a control-unit accelerator.

The main objectives of this dissertation are the following:

- Develop an *FPGA*-based implementation of the *BB-BC* optimization algorithm and assess its performance using representative benchmarks and hardware-driven case studies.
- Integrate the *BB-BC* algorithm with *PID/FOPID* controllers to obtain optimized parameters, and validate the approach on HVAC systems, including *MIMO* control scenarios.

- Extend the *BB-BC* optimization strategy to *LAMDA* controllers and enhance its performance through new search operators such as the Ordered Weighted Average (*OWA*).
- Design an *FPGA*-oriented framework capable of solving constrained optimization problems using the enhanced *BB-BC-LAMDA* architecture for industrial control applications.
- Improve real-time operation and energy efficiency by leveraging hardware-accelerated optimization through the proposed *FPGA*-based solutions.

1.6 Contributions of the thesis

The thesis contributions are summarized as follows:

- *Design of a novel hardware architecture for accelerating the BB-BC optimization algorithm for real-time applications.* The thesis designed a generic and flexible hardware architecture to accelerate the *BB-BC* optimization algorithm, addressing key limitations of conventional optimization algorithms when implemented in hardware. The *BB-BC* algorithm inherently eliminates data dependencies that typically hinder pipelining and parallelization, enabling a fully pipelined architecture in which multiple stages operate in parallel, thereby significantly reducing execution time. The proposed hardware architecture supports various configurations, including adjustable arithmetic operators and alternative Center of Mass (*CoM*) calculation methods, thus balancing computational accuracy and hardware resource utilization. Furthermore, a parallel scheme that integrates multiple *BB-BC* engines enables concurrent candidate processing, further boosting performance by up to 15x compared with software implementations. These features make the proposed *BB-BC* hardware accelerator well-suited for real-time, large-scale, and latency-sensitive applications, ensuring faster convergence and more efficient resource utilization, as reported in. These contributions were presented in publication [P1].
- *Implementation of the BB-BC accelerator in modern FPGA devices and evaluation for various use cases.* The thesis implemented the *BB-BC* accelerator on an Xilinx Virtex-5 *FPGA* device and evaluated its performance for three different fitness functions (*FiT*). Performance comparisons with software and *CUDA GPU* implementations demonstrated significant speedups. In addition, an Adaptive Neuro-fuzzy Inference System (*ANFIS*) was integrated to reduce execution latency for complex

fitness functions. The effectiveness of the BB-BC algorithm was further validated in an image search application, where it exhibited faster convergence than typical optimization algorithms. These contributions were presented in publications [P1] and [P3].

- *Use the FPGA BB-BC accelerator to tune and optimize PID controller parameters on an HVAC system.* The thesis proposed the use of the FPGA BB-BC accelerator to efficiently tune the parameters of a PID controller for an HVAC system modeled as a Second-Order Plus Time Delay (SOPTD) process; the Integral Time Absolute Error (ITAE) plus the control signal was selected as the objective function. The fast convergence property of the BB-BC algorithm, together with the acceleration achieved through hardware implementation, enables significantly faster optimization compared to traditional methods. FPGA-in-the-Loop experiments were conducted by integrating the proposed scheme with MATLAB/Simulink models for real-time evaluation. Experimental results showed that the FPGA-based BB-BC approach outperforms software implementations, as well as other optimization techniques like GA and PSO, in terms of both accuracy and convergence speed. These contributions were presented in publication [P2].
- *Enhance the FPGA BB-BC accelerator to optimize the control parameters of an FOPID controller on an HVAC system.* The thesis extended the FPGA PID/BB-BC approach to optimally tune the Fractional-Order PID (FOPID) control parameters for six different objective functions. A disturbance input applied where the BB-BC algorithm compensates for this, showcasing its proficient tracking and disturbance signal rejection capabilities. To demonstrate the effectiveness of the proposed FPGA-based scheme, two FOPID controllers, namely the D-FOPID and the GL-FOPID models, were designed and implemented on an Xilinx FPGA device. The proposed approach successfully reduced oscillations in the HVAC output signal and achieved rapid convergence. These contributions were presented in publication [P4].
- *Use the FPGA BB-BC accelerator to optimize the control parameters of a LAMDA controller on a MIMO system.* The thesis also applied the FPGA-based BB-BC accelerator to optimize the control parameters of a Learning Algorithm for Multivariate Data Analysis (LAMDA) controller in a Multi-Input Multi-Output (MIMO) system. Modifications to the LAMDA controller were introduced by replacing the

traditional T- or S-norm aggregation functions with an Ordered Weighted Averaging (OWA) operator. The adoption of FPGA technology significantly improved execution speed and energy efficiency compared to conventional computing platforms. A MIMO HVAC system was modeled using state-space equations, and the complete hardware-based control and optimization framework was implemented on an AMD/Xilinx Virtex-7 FPGA, demonstrating the effectiveness of the proposed approach. These contributions were presented in publication [P5].

1.7 Organization of the thesis

The rest of this thesis is organized as follows:

Chapter 2 introduces the Evolutionary Algorithms and the basics of the Optimization Controllers (PID, FOPID, and LAMDA), and also discusses the deployment of these control strategies in HVAC systems. *Chapter 3* presents the FPGA-based acceleration of the BB-BC optimization algorithm and compares it with software and GPU-based implementations. Subsequently, *Chapter 4* presents novel techniques based on the BB-BC optimization algorithm for tuning the PID and FOPID controllers in second-order HVAC systems. *Chapter 5* describes an FPGA-based implementation of the LAMDA controller, which in conjunction with the BB-BC algorithm, improves the performance of MIMO HVAC systems. Finally, *Chapter 6* provides a summary of the thesis and discusses future work directions.

CHAPTER 2

Evolutionary Algorithms and Optimization Controllers

2 Evolutionary Algorithms and Optimization Controllers

This chapter first provides an overview of Evolutionary Algorithms (EAs), focusing on the Big Bang–Big Crunch (BB-BC) optimization algorithm, which is the central focus of this thesis. Then, it explores various optimization controllers, including the Proportional-Integral-Derivative (PID) controller, the FOPID controller, and the most sophisticated LAMDA controller. Moreover, it discusses implementation issues of these controllers and the benefits of implementing them using the FPGA technology. Finally, the chapter presents the deployment of these control strategies in HVAC systems and compares the PID/FOPID and LAMDA-based controllers to highlight their effectiveness in system optimization. An HVAC system is presented and modeled as an SOPTD; it will be used in the following chapters as a demonstration vehicle for evaluating the optimization controllers under study.

2.1 Introduction

Evolutionary Algorithms (EAs) are adaptive heuristic search algorithms that employ an evolutionary process to derive the best solution from a large population. EAs are typically used in systems designed to solve optimization and search problems. Various EAs have been successfully applied across a wide range of industrial systems to address the increasing complexity of underlying applications. An industrial field in which various classical or intelligent EA-based control units have recently been proposed is the Heating, Ventilation, and Air Conditioning (HVAC) systems. Thus, it is imperative to build sophisticated control units that are more effective, robust, and faster than the state-of-the-practice. In the following, we briefly review the literature on the three optimization controllers studied in this thesis: PID, FOPID, and LAMDA. Razvarz, S., et al., dealt with the PID controller, which are widely used in industrial applications due to their simple structure and efficient implementation [22]. However, the performance of a conventional PID controller may be insufficient when applied to industrial processes of complex dynamics [23]. As an alternative, Podlubny [24] proposed the FOPID controller, which has been studied by several researchers [28] and can improve dynamic performance and system robustness. The FOPID controller adds two extra parameters, the integration order (λ) and the derivative order (μ), to the three parameters of the PID controller (proportional, k_p ; integral, k_i ; and derivative, k_d); this enables the designer to

model the system response specifications more efficiently. For example, MIMO systems that must meet four or five robustness criteria can benefit from the existence of the two extra tuning knobs in FOPID controllers. Given the complexity of FOPID controller design and their parameters optimization, it is essential to develop efficient controller implementations that reduce control unit latency and improve system responsiveness [25]. One of the major advantages of applying FOPID controllers to industrial processes is their potential to minimize control magnitude, leading to reduced power consumption [23]. FOPID controllers have been used in various control applications, such as synthesizing FO filters in industrial electrical drives [26], controlling the trajectory of robotic manipulators in medical robots [27], controlling the nonlinear attitude dynamics in unmanned aerial vehicles [28], controlling the indoor temperature in an air-conditioning room for HVAC systems in buildings [29], and more. Overall, integrating FOPID controllers can improve the performance and robustness of HVAC systems, particularly in complex dynamic systems. A new intelligent controller, based on the machine learning algorithm LAMDA, was proposed by L. Morales, et al. [30]. Although LAMDA is a fuzzy classification algorithm, it lacks an inference method. The thesis addresses this limitation by assigning a numerical value to the controller's output when the system is in one of the classes. L. Morales [31] tested the LAMDA in two case studies, a cart-inverted pendulum and a mixing tank with variable dead time, demonstrating its versatility in implementing PD and PI controllers.

2.2 Evolutionary Algorithms

EAs are optimization techniques inspired by the principles of natural evolution. They operate on a population of candidate solutions that evolve over generations to optimize a given objective. Key mechanisms include *mutation*, where small random changes are introduced to individuals to maintain diversity, and *crossover*, where two parent solutions are combined to create offspring, promoting the exchange of beneficial traits. Selection processes favor the survival and reproduction of the fittest individuals, driving the population toward optimal solutions. This iterative process mimics natural selection and genetic variation, enabling evolutionary algorithms to solve complex optimization problems effectively. As shown in Figure 3, the optimization process of an EA is a loop.

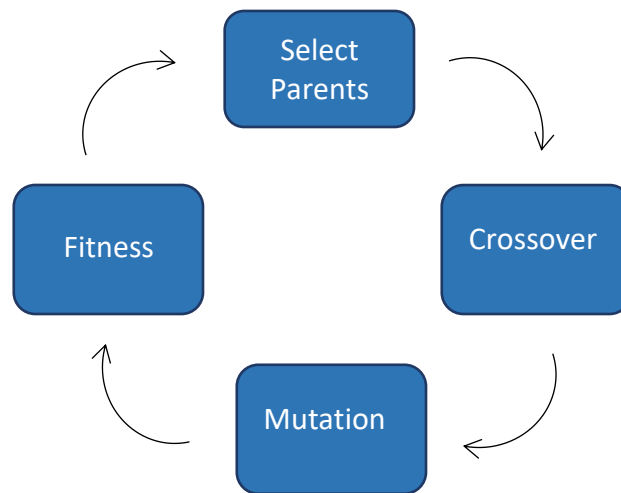


Figure 3. The EA process as a loop

These algorithms are particularly effective for solving complex, non-linear, and multi-modal optimization problems where traditional optimization methods may struggle. EAs do not require gradient information and are better suited for global optimization tasks, where the search space is vast and contains multiple local optima.

Key terms in EA terminology are:

- *Evolution*: Gradual improvement of candidate solutions over generations, guided by selection mechanisms.
- *Population*: A set of candidate solutions that evolves collectively.
- *Mutation*: A random change in a candidate solution to maintain diversity in the population.
- *Crossover*: Combines parts of two parent solutions to generate new offspring, promoting the exchange of information.
- *Selection*: Chooses the best solutions from the current population to pass their characteristics to the next generation.
- *Fitness Function*: Evaluates the quality of each candidate solution based on the optimization objective.
- *Generation*: One complete cycle of evaluation, selection, mutation, and crossover within the algorithm.

2.3 Big Bang – Big Crunch Algorithm

The Big Bang – Big Crunch (BB-BC) algorithm has been developed by Erol and Eksin in [32]. It consists of two essential phases: the *big bang phase (BBP)* and the *big crunch phase (BCP)*. In the BBP, candidate solutions are uniformly distributed over the search space, which is limited by boundary constraints, as in most meta-heuristic optimization techniques. The uniform distribution of candidates in the search space is analogous to energy dissipation in nature or the universe. The BBP is followed by the BCP, which can be visualized as a transformation from a disordered state of energy (*uniform distribution*) to an ordered state of energy. The BCP acts as a concurrence operator, producing a single output for many inputs (the *solution*). More precisely, uniformly distributed solutions in the search space are drawn into order, forming what can be named the Center of Mass (CoM). The point representing the CoM is denoted by x^c and calculated according to:

$$x^c = \frac{\sum_{i=1}^N \frac{1}{f^i} x^i}{\sum_{i=1}^N \frac{1}{f^i}} \quad (2.1)$$

where x^i is a point within an L -dimensional search space, f^i is the fitness function value of this point, and N is the size of the population generated in the BBP. After the BCP, the algorithm creates new members to be used as the BB of the next iteration according to the following formula:

$$x^{new} = x^c + r * l/k \quad (2.2)$$

where r is a random number with a normal distribution, k is the number of BB iterations and l is a problem-dependent constant, $l = a * (x_{max} - x_{min})$; parameter a limits the search space where $0 < a < 1$, while x_{max} and x_{min} are the upper and lower values of the optimization function variables. After the second explosion, the CoM is recalculated. These successive explosion and contraction steps are carried out repeatedly until a stopping criterion is met. Since normally distributed numbers can exceed ± 1 , it is necessary to keep candidate values within the predefined search space bounds. Eq. (2.3) is used to generate chaotic sequences for enhancing randomness in optimization algorithms.

$$[x(n+1) = r * x(n) * (1 - x(n))] \quad (2.3)$$

The steps of the BB-BC algorithm are summarized as follows:

1. Form an initial generation of N candidates in a random manner.
2. Calculate the fitness function values of all the candidate solutions. This step depends on the target optimization problem.
3. Find the CoM according to Eq. (2.1). The best fitness individual of each generation can also be chosen as the CoM instead of using Eq. (2.2), which reduces the computation time.
4. Calculate new candidates around the CoM using Eq. (2.3). Notice that the random value added or subtracted to the CoM decreases as the iterations elapse.
5. Return to Step 2 until the stopping criteria have been met.

The primary challenge in the BB-BC algorithm is that it may sometimes get trapped in local optima, making it difficult to reach the global optimum. Figure 4 shows the flow chart of the BB-BC algorithm.

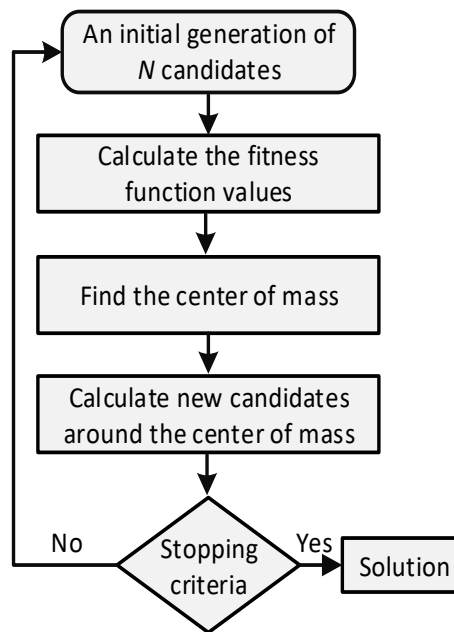


Figure 4. BB-BC optimization algorithm flow chart.

The BB-BC algorithm is summarized in pseudocode as follows:

BEGIN

INPUTS: Initial parameters

Initialize:

$$k = 1, c_i = [c_0, c_1, c_2, \dots, c_n]$$

Calculate:

Fitness function (f^i)

While the Minimum Error criterion is not met, DO

Calculate CoM x^c using Eq.(2.1)

Evaluate f^i (fitness function)

$$k \leftarrow k + 1$$

Calculate new candidates x^{new} using Eq.(2.2)

Evaluate α using Eq. (2.3)

End while

Output: Updated parameters in x^{new}

END

Kumbasar, T. and H. Hagnas [33] have implemented the BB-BC algorithm in various engineering applications, particularly where computational and convergence times are important factors. Some representative optimization problems, where the BB-BC efficiency has been demonstrated, are the following: optimal design of truss structures [34], Schwedler and ribbed domes [35], and concrete retaining walls and [36]. Kumbasar, T., et al., implemented a fuzzy inverse controller [37], and Yesil, E., applied a fuzzy PID controller [38]. Moreover, the BB-BC algorithm has been adopted in real-time applications where the optimization task is adaptive or must be solved at relatively short sampling intervals. Besides its low computational time and high global convergence speed, another advantage of BB-BC, essential for auto-tuning processes, is that it does not require gradient information and, therefore, can operate to minimize naturally defined cost functions without complex mathematical operations. Hardware implementations using FPGA technology have been presented in the literature for industrial controllers [39-41]. The FPGA-based implementations reduce the execution time of the optimization algorithms and enhance their flexibility, i.e., enabling

quick redesign and runtime reconfiguration. These advantages are mainly achieved because FPGA technology supports parallel execution and hardware reconfigurability. Towards this direction, this thesis studies novel FPGA-based implementations of the BB-BC algorithm.

2.4 Optimization Controllers

Optimization controllers aim to enhance the performance of a process by minimizing or maximizing a specific objective, such as energy consumption, cost, or error. These controllers use optimization algorithms to determine the optimal control actions in real time, ensuring the system operates efficiently under varying conditions. In other words, optimization controllers are algorithms that drive a system toward an optimal state by adjusting control variables. They are widely applied in industries such as energy management (EM), robotics, and process control, where achieving optimal performance is crucial. For example, optimization controllers such as Model Predictive Control (MPC), PID, and FOPID controllers, and Learning Multi-variable Data Analysis (LAMDA) controllers, are widely used in industrial processes due to their efficiency in handling real-time control tasks. They rely on mathematical models of the system and use techniques such as gradient descent to minimize or maximize a specific objective function [42].

Optimization controllers based on evolutionary algorithms have emerged as a powerful approach to improve the efficiency and performance of control systems.

2.5 PID Control Algorithm

The PID controllers have been widely used in control techniques since their invention in 1910 by Åström, K.J., and T. Hägglund [43] in both academia and industry. Their primary purpose is to provide a robust and simple control mechanism that minimizes errors between the system's desired and actual outputs. PID controllers are widely used across domains such as process control, robotics, automotive systems, and HVAC systems due to their ability to ensure stability, improve transient response, and achieve precise control. The PID controller's benefits include ease of implementation, cost-effectiveness, and adaptability to a wide range of dynamic systems. The mathematical foundation of the PID controller relies on three gains: proportional (P), integral (I), and derivative (D), which respond to the error

between a system's set-point and its actual output. There are many PID control configurations, but the most common implementation is shown in Figure 5.

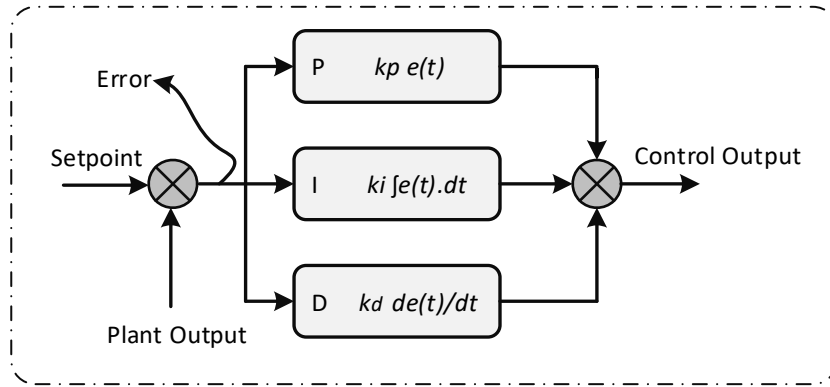


Figure 5. Block diagram of a typical PID control structure

In general, the synthesis of the PID controller can be mathematically described by the following time-domain equation:

$$u(t) = k_p e(t) + k_i \int_0^t e(t) dt + k_d \frac{d}{dt} e(t) \quad (2.4)$$

where k_p is the proportional gain, k_i is the integral gain, k_d is the derivative gain, and the error $e(t)$ is the difference between the set-point and the plant output $y(t)$.

In addition, the PID control law can be expressed in the Laplace domain or in other digital forms in order to be implemented on hardware platforms, such as FPGAs.

Specifically, Equation (2.1) can be rewritten in Laplace form as:

$$u(s) = e(s) \left[k_p + \frac{k_i}{s} + k_d s \right] \quad (2.5)$$

Finally, under the above scheme, the transfer function of the PID controller or the control law is specified by:

$$C(s) = \frac{u(s)}{e(s)} = k_p + \frac{k_i}{s} + k_d s \quad (2.6)$$

For digital PID controllers, Equation (2.6) can be transformed via the z-domain into a difference equation for a recursive digital filter with an infinite impulse response (IIR):

$$\frac{U(z)}{E(z)} = k_p + \frac{k_i T_s z}{z-1} + \frac{K_d N(z-1)}{(1+NT_s)z-1} \quad (2.7)$$

where T_s is the sampling period, and N is the filter value. To be implemented on a hardware platform, a digital PID controller must be transformed into a difference equation, which relates the controller's gains to the digital implementation as shown below:

$$\frac{U(z)}{E(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \quad (2.8)$$

A differential equation can be derived from equation (2.8) as follows:

$$u[n] = k_1 u[n-1] + k_2 u[n-2] - k_3 e[n] - k_4 e[n-1] - k_5 e[n-2] \quad (2.9)$$

Therefore, the relationship between the PID gains and their implementation in a digital system, in other words, the equivalence of parameters between continuous and digital forms, is shown in Table 1.

Table 1. Equivalence of parameters for PID controllers

$C(z)$	k_1	k_2	k_3	k_4	k_5
$C(s)$	$-\frac{a_1}{a_0}$	$-\frac{a_2}{a_0}$	$-\frac{b_0}{a_0}$	$-\frac{b_1}{a_0}$	$-\frac{b_2}{a_0}$

where the differential equation variables equal:

$$\begin{aligned} a_0 &= (1 + N * T_s). \\ a_1 &= -(2 + N * T_s); \\ a_2 &= 1. \\ b_0 &= k_p * (1 + N * T_s) + k_i * T_s * (1 + N * T_s) + k_d * N. \\ b_1 &= -(k_p * (2 + N * T_s) + k_i * T_s + 2 * k_d * N). \\ b_2 &= k_p + k_d * N. \end{aligned} \quad (2.10)$$

2.6 Fractional Order $PI^\lambda D^\mu$ (FOPID) Controller

Fractional calculus is a generalization of integration and differentiation to a non-integer order fundamental operator ${}_a D_t^\alpha$ as defined by the continuous integro-differential fundamental operator presented by Chen, Y., et al. [44] :

$${}_a D_t^\alpha \equiv \begin{cases} \frac{d^\alpha}{dx^\alpha}, & \text{Re}(\alpha) > 0 \\ 1, & \text{Re}(\alpha) = 0 \\ \int_a^t (d\tau)^{-\alpha}, & \text{Re}(\alpha) < 0 \end{cases} \quad (2.11)$$

where r and t are the limits of the operation and α (Alpha) is the order of the operation. The Grunwald-Letnikov (GL) and the Riemann-Liouville (RL) are two definitions used for the general fractional differ-integral. This thesis focuses on the GL definition introduced by Podlubny, I., [45] and briefly described in the following subsection. Over the past few decades, the introduction of fractional calculus has enabled the development of non-classical controllers described by differential equations of non-integer order. FOPID controllers, also known as $PI^\lambda D^\mu$ controllers have been developed in the time domain [46]. The time-domain definition of a FOPID controller is given as:

$$u(t) = K_p e(t) + K_i D_i^{-\lambda} e(t) + K_d D_t^\mu e(t) \quad (2.12)$$

where $e(t)$ is the error signal and $u(t)$ is the control signal (output of the controller). Hence, the parameters of a typical FOPID controller include the proportional gain K_p , integral gain K_i , derivative gain K_d , and two additional fractional operators, namely the integral order λ and the derivative order μ . Figure 6 shows the block diagram of a typical FOPID controller.

The implementation of the FOPID controller is a critical issue for achieving satisfactory trajectory tracking performance. A rational approximation of the FOPID controller may be obtained through continuous approximation functions (S-domain) for analog circuit implementation. Podlubny, I., et al., realized the continuous fractional expansion (CFE) method that is used for continuous approximation [47]. On the other hand, El-Khazali, R., et al., presented a discrete approximation function (Z-domain) for implementation on digital platforms (such as FPGAs) using the Finite Impulse Response (FIR) approximation method [48]. Two of the most widely used algorithms for the digital implementation of the fractional order operator (FOO) are presented in [49] and [50]. In [50] Merrikh-Bayat, F., et al., described the digital implementation using the Taylor series, one of the most common Power Series Expansion (PSE) methods for discrete approximation via several generating functions.

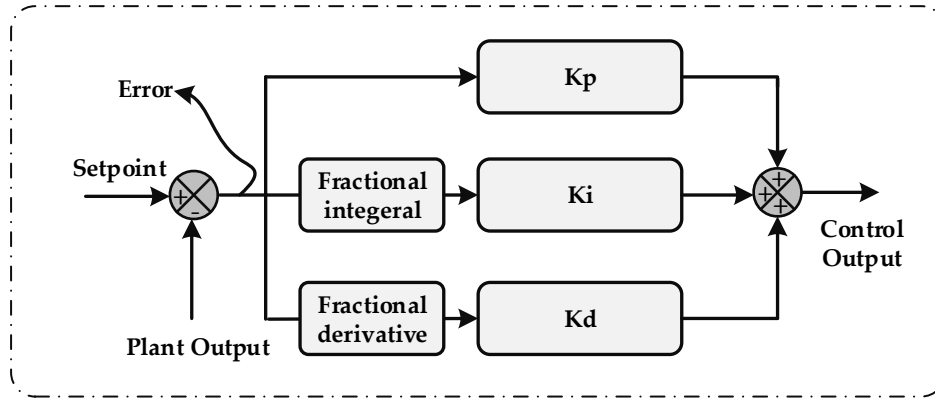


Figure 6. Block diagram of a typical FOPID controller

2.6.1 GL-based FOPID

The Grunwald-Letnikov (GL) definition of the Fractional-Order (FO) calculus is widely used for analytically finding fractional differ-integrals. The α^{th} -order integration of the control system input (error) to implement the PID controller can be calculated according to the GL formula:

$${}^{GL}_r D_t^\alpha f(t) = \frac{1}{h^\alpha} \sum_{j=0}^{[L]} w_j^{(\alpha)} f(t - jh) \quad (2.13)$$

where D is the differ-integral operator, h is the step size, $w_j^{(\alpha)}$ are the binomial coefficients, function $f(t)$ denotes the control input (error), L is the window size, and h is the step size. Figure 7 shows the block diagram of a GL-based FOPID controller.

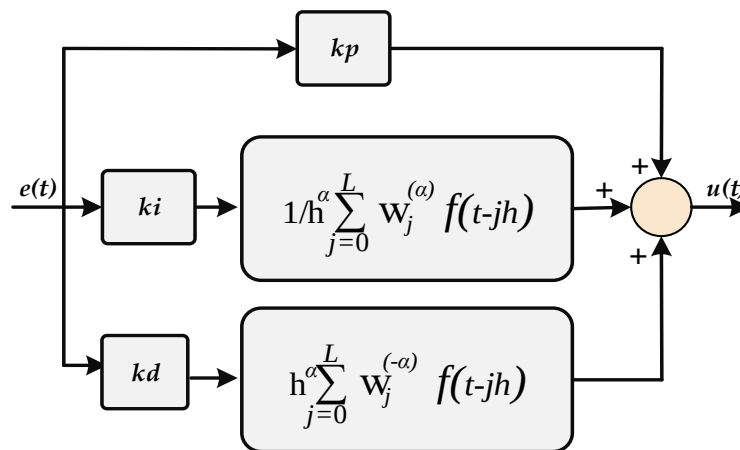


Figure 7. Block diagram of a GL-based FOPID controller

2.6.2 Discretized (D) FOPID Controller

The simplest fractional differ-integration method is the direct discretization, based on the finite memory length expansion from the GL definition proposed by Sundriyal, N., [51]. In general, the direct discretization of a fractional order differentiator-integrator $S^{\pm\alpha}$ ($\alpha \in \mathbb{R}$) uses the Tustin transform and the power series expansion (PSE) to express the result in terms of z . To obtain a discrete fractional-order system, Merrikh-Bayat, F., et al. suggested using a limited-memory approach. [50]. The output of the discretized FOPID controller can be calculated as:

$$C_d(z) = k_p + k_d \alpha^\mu \sum_{k=0}^{\infty} f_k(\mu) z^{-k} + k_i \alpha^{-\lambda} \frac{1+z^{-1}}{1-z^{-1}} \sum_{k=0}^{\infty} f_k(1-\lambda) z^{-k} \quad (2.14)$$

Figure 8 presents the block diagram of a discretized (D)-FOPID controller with a limitation in memory size (M).

Note that according to (2.14), the difference equation relating $e[n]$ to $u[n]$ in Figure 8. Block diagram of a discretized (D)-FOPID controller is as follows:

$$u[n] = u[n-1] + k_p(e[n] - e[n-1]) + \sum_{k=0}^M k_d f_k(\mu) (e[n-k] - e[n-k-1]) + \sum_{k=0}^M k_i f_k(1-\lambda) (e[n-k] - e[n-k-1]) \quad (2.15)$$

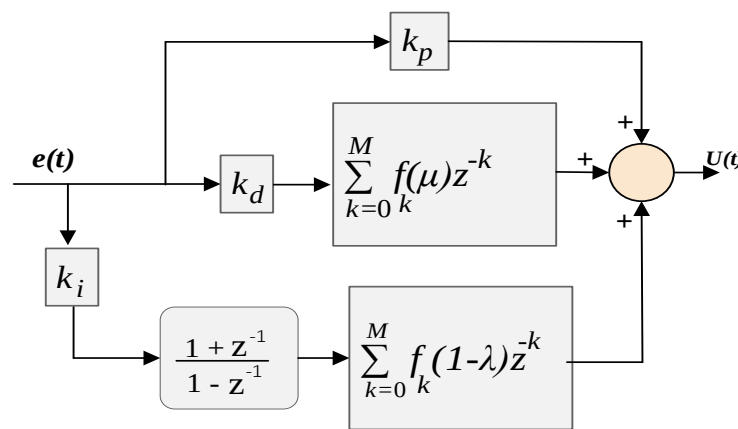


Figure 8. Block diagram of a discretized (D)-FOPID controller

2.6.3 Advantages of the FOPID Controller over the PID Controller

The FOPID controller differs from the PID controller in several key aspects, including lower steady-state error, reduced overshoot, fewer ripples, and greater robustness against variations in the controlled plant parameters and system disturbances. Table 2 summarizes the advantages of FOPID controllers over PID controllers.

Table 2. FOPID vs. PID feature comparison		
Physical properties	Features & Description	
	FOPID	PID
Structure	<i>Complex:</i> Involves additional FO parameters (λ, μ).	<i>Simple:</i> Involves only proportional, integral, and derivative terms.
No. of parameters	$k_p, k_i, k_d, \lambda, \mu$: Involves more parameters ($k_p, k_i, k_d, \lambda, \mu$).	k_p, k_i, k_d : Involves fewer parameters (k_p, k_i, k_d).
Power of operation	<i>Any positive integer:</i> Allows any positive integer value, offering more flexibility in control customization.	<i>0 or 1:</i> Limited to values of 0 or 1, which simplifies control behavior but reduces customization options.
Power consumption	<i>Low:</i> Exhibits lower power consumption due to its advanced control mechanism.	<i>High:</i> Tends to have higher power consumption, especially in situations where its control efforts are more intensive.
Effectiveness	<i>Complex system:</i> Particularly effective for handling complex systems with non-integer order dynamics, offering improved control performance.	<i>Simple system:</i> Effective for simpler systems and situations where FO dynamics are not critical.

This approach offers potential improvements in the overall performance and effectiveness of the controller to meet the performance requirements. It enables a more comprehensive and targeted approach to achieving the desired outcomes. The FOPID controller provides an excellent variable-frequency response because its parameter set covers the problematic area better than that of the PID controller. Furthermore, FOPID controllers exhibit a more complex structure with greater parameter diversity, making them suitable for complex systems and situations that require advanced control mechanisms. On the other hand, PID controllers offer simplicity and are effective for simpler systems, trading customization options for straightforward control operation. Additionally, FOPID controllers often demonstrate better power efficiency and can accommodate a broader range of operation values, offering a more versatile control experience. The popularity of FOPID controllers is a crucial factor that strengthens our motivation and stands as a significant contribution to this thesis. Its notable advantage lies in controlling fractional power, enabling precise adjustment of the derivative

or integral components based on specific requirements. This feature becomes especially valuable when combined with optimization algorithms, further optimizing its performance. Abumeteir, H.A. et al., optimized the FOPID controller yielding a noteworthy advantage in terms of energy efficiency [52]. This advantage stems from the sensitivity of the FOPID controller parameters, which respond favorably to the rapid convergence of the optimal solution achieved by the optimization algorithm.

2.7 LAMDA-Based Controller

Learning Algorithm for Multivariable Data Analysis (LAMDA), proposed by Joseph Aguilar-Martin [53], is a non-iterative fuzzy clustering and classification algorithm that utilizes the degree of adequacy to classify each object.

The LAMDA algorithm, which relies on machine learning methods, is a good candidate for various control applications due to several reasons: i) it can work as a clustering and classification approach, which allows evaluating the autonomous cycles with only one technique, ii) it gives information about the descriptors of the classes or cluster that it uses, which is very important in the context of the generation of corrective actions, iii) improvements to this algorithm have been developed, such as those proposed in [54], which uses OWA and achieves outstanding performance in classification and clustering benchmarks.

The main steps of the LAMDA algorithm are listed below:

- *Input Data and Preprocessing*: Collects and normalizes input data for consistency.
- *Feature Membership Calculation*: Computes the degree of membership for each feature concerning each class.
- *Aggregation of Feature Contributions*: Combines membership values of all features for each class using aggregation operators (e.g., weighted sum, arithmetic mean).
- *Class Assignment*: Identifies the class with the maximum Generalized Aggregated Distance (*GAD*) and assigns the object to this class.
- *Recurrent Computation*: Repeats the *GAD* computation for multi-step or hierarchical classification, if needed.
- *Output*: Provides the final classification decision and optional confidence measures.

Doncescu, A., et al. [55] introduced the intelligent controller, which is used in the LAMDA algorithm and is described below. The features of any object $X = [x_1, \dots, x_n]$ are compared with the features of the existing classes $C = [C_1, \dots, C_m]$, to determine their similarity; the classes are defined based on the system's behavior. To improve the performance, the features of the object are normalized to span the interval $[0, 1]$ using the formula:

$$[\bar{x}_j = \frac{x_j - x_{jmin}}{x_{jmax} - x_{jmin}}]. \quad (2.16)$$

where x_j and $\bar{x}_j$ are the original and normalized values, respectively, of the j_{th} feature and x_{jmin} and x_{jmax} are the minimum and maximum values, respectively, of x_j . Then, the marginal adequacy degree (*MAD*) of the j_{th} feature to the k_{th} class is calculated using the probability density function of the normal distribution, i.e.:

$$[MAD_{k,j}(\bar{x}_j | \rho_{k,j}) = e^{-\frac{1}{2} \left(\frac{\bar{x}_j - \rho_{k,j}}{\sigma_{k,j}} \right)^2}] \quad (2.17)$$

where $\rho_{k,j}$ and $\sigma_{k,j}$ are the mean and the standard deviation, respectively, of the j_{th} feature in the k_{th} class. $MAD_{k,j}$ is a measure of the similarity of the j_{th} feature class with the corresponding feature in the k_{th} class. Subsequently, the *GAD* for the k_{th} class is calculated as:

$$[GAD_{k,\bar{X}}(MAD_{k,1}, \dots, MAD_{k,n}) = \alpha \times T(MAD_{k,1}, \dots, MAD_{k,n}) + (1 - \alpha) \times S(MAD_{k,1}, \dots, MAD_{k,n})]. \quad (2.18)$$

where $\bar{X}$ is the normalized object X , $\alpha \in [0,1]$ is a parameter that represents the level of exactitude, T is the t-norm (intersection), and S is the t-conorm (union).

$$T(a, b) = \frac{ab}{p + (1-p)(a+b-ab)} \quad (2.19)$$

$$S(a, b) = \frac{a+b-ab-(1-p).ab}{p+(1-p)(a+b-ab)} \quad (2.20)$$

Both a and b are constrained to $[0,1]$, as they represent normalized measures in Eq. (2.19) and Eq. (2.20). Generally $p=1$ is used to modify the sensitivity, which typically adjusts the operator to low or high input values.

Aggregation is the process of combining several numerical values into a single representative one. In other words, aggregation serves a critical role in decision-making by combining multiple partial evaluations or degrees of membership into a single output that determines the final classification of a given object. Note that we used in our work, the OWA operator is employed as the primary aggregation mechanism to enhance flexibility and robustness in the decision-making process.

2.7.1 OWA Operator

The *OWA* operator, used for decision-making, requires careful determination of weights. The *OWA* operator is an aggregation function proposed by Yager, R.R. [56] to create a generic model for the operation of union with classes instead of using T-norm and S-norm aggregation functions. The *OWA* operator allows us to model a wide range of aggregation functions, spanning from the maximum to the arithmetic mean and even the minimum. When associated with mapping F , the *OWA* operator of dimension n is a mapping $F : R_n \rightarrow R$, where a weighting vector W_v is defined as:

$$[W_v = [w_{v1} \quad w_{v2} \quad \dots \quad w_{vn}]] \quad (2.21)$$

such as:

$$\sum_{j=1}^n W_{v_j} = w_{v1} + w_{v2} + \dots + w_{vn} = 1 \quad (2.22)$$

Furthermore

$$[F(a_1, a_2, \dots, a_n) = \sum_{j=1}^n W_{v_j} b_j]. \quad (2.23)$$

where b_j is the j_{th} largest of a_i and F is an *OWA* operator.

Notice: One can implement different aggregation operators by choosing different W_v (w_v range). As α increases, the classification becomes more selective (more precise). When two or more features are considered, the *GADs* are computed recurrently [55]. In classification tasks, the normalized object $\bar{X}$ is assigned to the class with the maximum *GAD*, i.e.:

$$GAD = OWA(MAD, w_v)$$

$$[index = \max(GAD_{1,\bar{x}}, \dots, GAD_{m,\bar{x}})] \quad (2.24)$$

where $index$ is the identifier of the class selected.

Up to this point, the LAMDA algorithm has been able to identify the current operating state of the system. For control tasks, an inference method is needed to drive the system to the desired state. For this purpose, rules are expressed in the form of the following LAMDA inference mechanism:

$$[R^{(l)}: \text{if } \{\bar{x}_1 \text{ is } F_1^k \text{ and } \dots \text{ and } \bar{x}_n \text{ is } F_n^k\} \text{ then } \{y_l \text{ is } G^l\}] \quad (2.25)$$

where $\bar{x}_j$ takes values from the universe of discourse U_j , the linguistic variable y_l is defined in the universe of discourse V_l , F_j^k ($j = 1, \dots, n$) are fuzzy sets in U_j and G^l ($l = 1, \dots, m$) are fuzzy sets in V_l , n is the number of features and m is the number of rules, also known as LAMDA classes. Note that we have used GAD for each class to obtain the crisp output, since LAMDA lacks an interface mechanism. The crisp control output is calculated as:

$$[u = \beta \sum_{k=1}^n q^k GAD_{k,\bar{x}}] \quad (2.26)$$

where q^k is the weight applied to the k_{th} class and β is a parameter calculated in the training phase according to the formula:

$$[\beta = \frac{\max(q^k)}{\sum_{k=1}^n q^k GAD_{k,\max(\bar{x})}}] \quad (2.27)$$

to guarantee that u is within the class boundaries. Figure 9 shows a general scheme of the LAMDA algorithm. In all phases, we can summarize the LAMDA algorithm in four basic stages: 1) *Descriptors*: the set of objects is described by $X = [x_1, \dots, x_n]$, and presented in vector; 2) *Fuzzification*: MAD calculation is applied to each descriptor x with each class c ; 3) *Inference*: the grade of membership of each input vector X to each class c is calculated using OWA ; 4) *Defuzzification*: the index of the class to which the input vector belongs is finally generated.

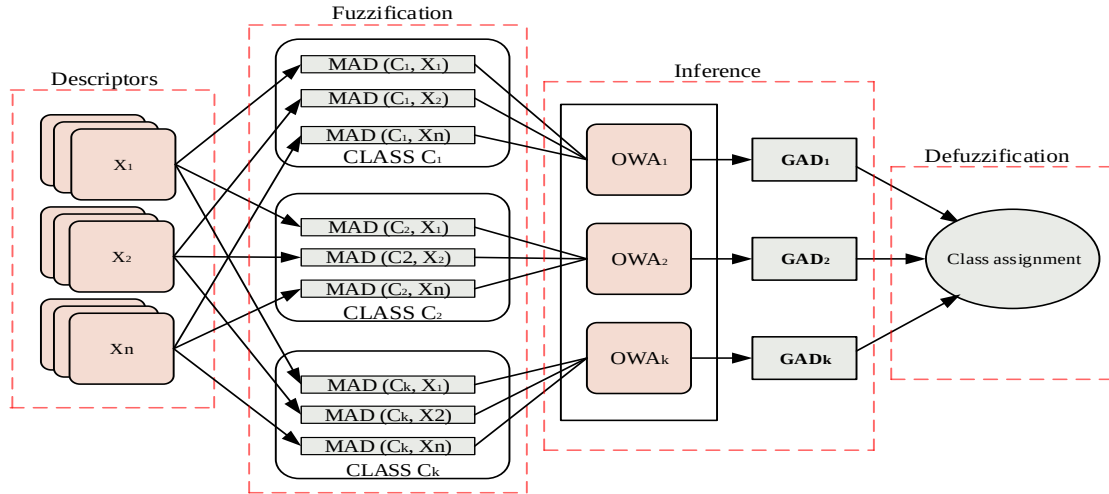


Figure 9. General scheme of the LAMDA algorithm

Figure 10 shows a block diagram of the LAMDA controller, whose inputs are the error (e) and the derivative of the error (de), and its output is the control action. Note that the proportional gain Kp and the derivative gain Kd multiply e and de , respectively, before being passed as inputs to the LAMDA algorithm described above, and then the calculated U is multiplied by the integral gain Ki and is then integrated.

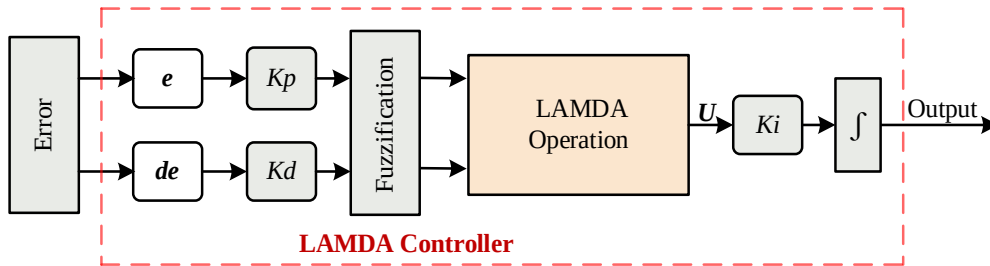


Figure 10. Block diagram of the LAMDA controller

2.8 Integration of EAs into FPGAs

The integration of EAs into embedded processing systems using FPGA technology drives significant advancements in various industries by enhancing real-time data processing capabilities. FPGAs are flexible, reconfigurable integrated circuits that can be programmed to perform specific tasks in hardware, making them ideal for applications that require low latency and high performance. This integration is particularly impactful in industrial applications,

such as telecommunications, automotive, and other fields, where real-time decision-making is critical [57].

Moreover, the continuous evolution of FPGA technology, such as the development of high-level synthesis tools and the availability of various families tailored to different requirements, is making it easier to integrate with various EA systems. These advancements not only improve performance but also reduce power consumption, which is essential for deploying FPGAs in energy-sensitive environments such as mobile and embedded systems.

The combination of EA, controllers, and FPGA technologies is poised to reshape the future of industries that demand rapid data processing and real-time responsiveness, pushing the boundaries of what is possible in automation and digital design [58].

The EA, when integrated into hardware-based optimization controllers for any control application, can enhance the ability to find global optima, improve energy efficiency, and adapt to changing conditions or system uncertainties. FPGA technology provides a flexible and high-performance hardware platform for implementing complex control algorithms with real-time processing capabilities, making it ideal for demanding applications. Moreover, it offers additional advantages, including parallel processing, low latency, and the ability to handle complex computations in real time.

2.9 HVAC System Integration with Control Strategies

HVAC systems are automatic systems that control and provide a comfortable environment in terms of temperature, humidity, ventilation, and other parameters (power consumption, pressure, and time control). HVAC systems are equipped with various sensors to monitor these parameters (e.g., temperature and humidity sensors, anemometers for measuring the airflow speed, etc.) and optimize system performance and power consumption.

HVAC systems provide an acceptable thermal environment in buildings, but conventional HVAC systems result in significant energy consumption [59]. Arguello-Serrano and Velez-Reyes in [60] defined a mathematical model of an HVAC system. This model describes the system equipment, which consists of sensors that provide data to the control units for regulating variables such as area temperature, supply air temperature, supply air fan speed, duct static pressure, and chilled water temperature at their set-points. Figure 11 shows the block

diagram of a typical HVAC system. The system must simultaneously control variables T_3 and W_3 based on thermal loads, varying the fan speed u_1 to regulate the air flow rate, and the cold-water pumping rate u_2 from the chiller to the heat exchanger. Escobar, L.M., et al. implemented an integrating model of the sub-processes of the HVAC system resulting in the following set of differential equations that describe the dynamics of the system [60, 61]:

$$\begin{aligned}
 [\dot{T}_3 &= \frac{f}{V_s} (T_2 - T_3) - \frac{h_{fg}}{C_p V_s} (W_s - W_3) + \frac{1}{0.25 C_p V_s} (Q_0 - h_{fg} M_0) \\
 W_3 &= \frac{f}{V_s} (W_s - W_3) + \frac{M_0}{\rho V_s}], \\
 [\dot{T}_2 &= \frac{f}{V_{he}} (T_3 - T_2) - \frac{0.25f}{V_{he}} (T_0 - T_3) \\
 &- \frac{f h_w}{C_p V_{he}} (0.25W_0 + 0.75W_3 - W_s) - 6000 \frac{gpm}{\rho C_p V_{he}}]. \tag{2.28}
 \end{aligned}$$

where h_w is the enthalpy of liquid water, W_0 is the humidity ratio of outdoor air, h_{fg} is the enthalpy of water vapor, V_{he} is the volume of the heat exchanger, W_s is the humidity ratio of the supply air, W_3 is the humidity ratio of the thermal space, C_p is the specific heat of air, T_0 is the temperature of outdoor air, M_0 is the moisture load, Q_0 is the sensible heat load, T_2 is the temperature of the supply air, T_3 is the temperature of Zone 3, V_s is the volume of the thermal space, ρ is the air mass density, f is the volumetric flow rate of air ($ft^3 = min$) and gpm is the flow rate of chilled water ($gal=min$). Defining the inputs $u_1 = f$ and $u_2 = gpm$, the state variables $z_1 = T_3$, $z_2 = W_3$ and $z_3 = T_2$, the outputs $y_1 = T_3$ and $y_2 = W_3$, and the parameter $\alpha_1 = 1/V_s$, $\alpha_2 = h_{fg}/C_p V_s$, $\alpha_3 = 1/\rho C_p V_s$, $\alpha_4 = 1/\rho V_s$, $\beta_1 = 1/V_{he}$, $\beta_2 = 1/\rho C_p V_{he}$, and $\beta_3 = h_w/C_p V_{he}$, the model can be equivalently represented by the MIMO state space model:

$$\begin{aligned}
 [\dot{z}_1 &= u_1 \alpha_1 60 (z_3 - z_1) - u_1 \alpha_2 60 (W_s - z_2)], \\
 [\dot{z}_2 &= u_1 \alpha_1 60 (W_s - z_2) + \alpha_4 M_0], \\
 [\dot{z}_3 &= u_1 \beta_1 60 (z_1 - z_3) + u_1 \beta_1 15 (T_0 - z_1) \\
 &- u_1 \beta_1 60 (0.25W_0 + 0.75z_2 - W_s) - 6000 u_2 \beta_2] \tag{2.29}
 \end{aligned}$$

where $y_1 = T_3 = z_1$, $y_2 = W_3 = z_2$.

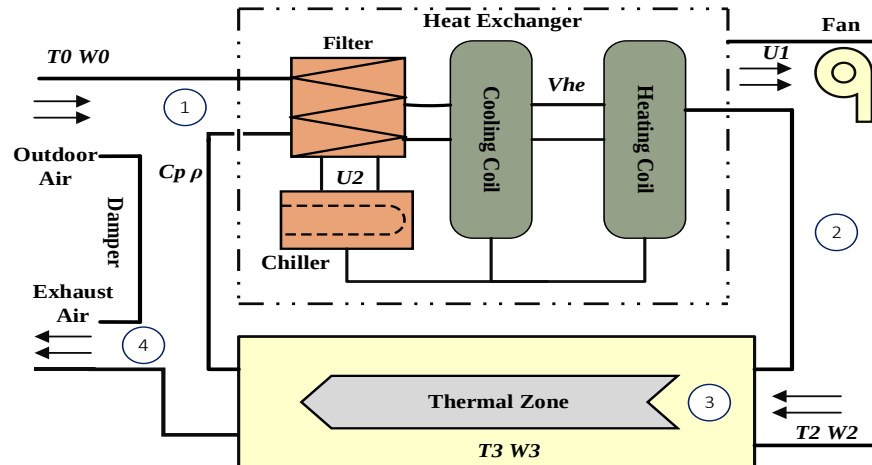


Figure 11. Block diagram of a typical HVAC system

The dynamics of the system are represented by the set of differential equations in Eq. (2.29), which incorporate various parameters, such as the enthalpies of air and water, the volumes of the thermal space and the heat exchanger, and the airflow rates. These equations are further simplified into a MIMO state-space model that links the system inputs and outputs to the state variables, allowing for precise control of temperature and humidity within the HVAC system.

2.9.1 HVAC Systems using PID, FOPID, and LAMDA Controllers

Maasoumy, M., and A. Sangiovanni designed an HVAC system where the PID controller is used in the control loop of the supply air pressure [62]. Figure 12 shows the block diagram of a conventional single-zone HVAC system merged with a PID control unit. The control inputs of the HVAC system are the air flow rate, which is adjusted by a variable-speed fan, and the water flow rate from the chiller to the heat exchanger. The PID controller adjusts motor speed as necessary to maintain a set point compared to the system output signal.

The motor speed should be adjusted to maintain a set-point difference for HVAC system operations, including valve and damper operation, ambient temperature, and flow rate. The PID/FOPID system performance depends on its ability to maintain a set point and to speed up actions.

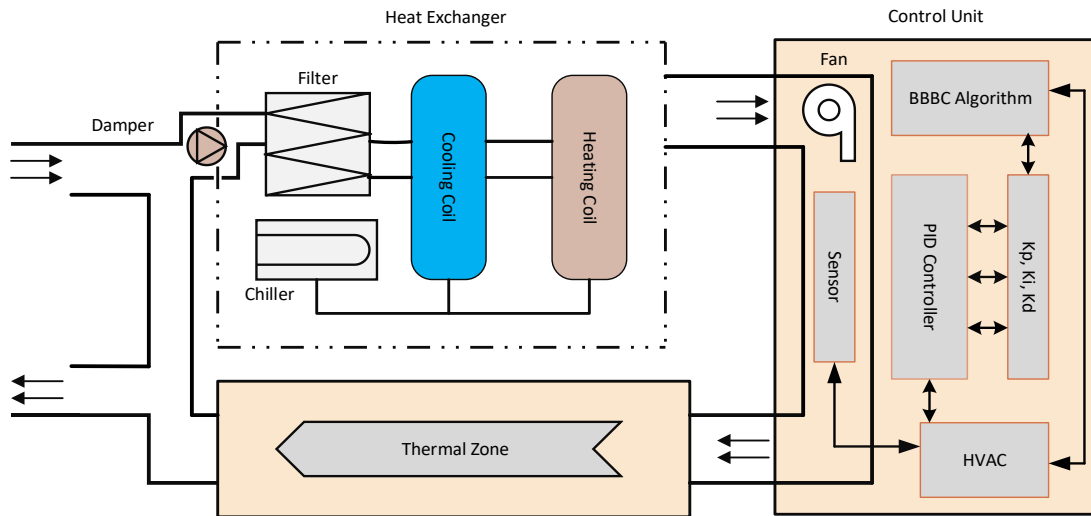


Figure 12. Block diagram of a conventional HVAC system and PID control structure

The tuning process in large HVAC systems is complex and time-consuming [63, 64] due to their non-linearity and time-varying characteristics. Consequently, tuning the PID controller of an HVAC system to achieve the optimum tracking control performance is a challenging task. In this thesis, we propose the use of the BB-BC algorithm and FPGA technology to tune the PID/FOPID parameters, as shown in Figure 12, and thus, speed up the re-tuning process of the HVAC system when needed. Moreover, we propose the LAMDA controller to maintain a steady-state error in MIMO of the HVAC system, as shown in Figure 11, since the control objective in this process is to maintain both temperature and humidity.

CHAPTER 3

FPGA-based Acceleration of the Big Bang-Big Crunch Optimization Algorithm

3 FPGA-based Acceleration of the BB-BC Optimization Algorithm

This chapter discusses FPGA-based acceleration techniques for evolutionary algorithms in industrial applications. First, it surveys the literature related to the hardware-based implementations of genetic algorithms, highlighting design challenges and performance issues. Next, it presents an efficient and novel FPGA-based accelerator architecture for the Big Bang-Big Crunch (BB-BC) algorithm featuring various performance mechanisms, such as pipelined scheme, parallel execution engines, efficient fitness function approximation and high-speed reduction circuits. Finally, it provides a detailed evaluation of the proposed FPGA accelerator architecture and compares it with software and GPU-based implementations to demonstrate its superior performance. The contents of this chapter are mostly based on the publications [P1] and [P3].

3.1 Introduction

Generic Algorithms (GAs) have been proven effective in solving complex optimization and search problems. However, due to their large exploration space and heuristic nature, GAs typically suffer from long execution times. Hardware platforms, such as GPUs and FPGAs, have been used to accelerate the execution of GAs. Due to their high performance and reconfigurability, modern FPGAs offer a flexible and effective platform for accelerating the evolution process. Exploiting the hardware parallelism to perform complex computation tasks and the inherent parallelism of these algorithms yields a significant speedup. Since Scott, S.D., first proposed the FPGA-based acceleration of GA systems [65], several researchers have presented efficient FPGA implementations of both application-specific [66],[67] and general-purpose GAs [68],[69]. FPGA-based GAs may achieve an average speedup of around 5x compared to their software counterparts, as shown in [69].

FPGAs enable the parallel execution of multiple genetic operations, such as selection, crossover, and mutation, resulting in significant improvements in throughput. Their hardware customization potential enables designs to be tailored for specific applications, optimizing both resource usage and energy efficiency. Another notable advantage is their adaptability; FPGA architectures allow dynamic adjustments to algorithm parameters without requiring time-consuming recompilation, enhancing flexibility and real-time responsiveness. The

main advantages of FPGA technology include reduced engineering costs, shorter time-to-market, high performance enabled by fine-grained hardware parallelism, and the ability to adapt to control system requirements. Especially for control systems that employ advanced computational intelligence techniques, the high performance of FPGAs, combined with their low cost and high adaptability, makes them an attractive computing platform.

Moreover, hardware acceleration techniques, such as pipelining [70],[71] and parallel execution [72], have been applied to the FPGA designs to further improve GA performance. Pipelined and parallel GA architectures aim to eliminate pipeline stalls and increase the execution parallelism during the various stages of the evolution process (e.g., random population generation, fitness calculation, genetic operations). However, some features of the evolutionary process do not map efficiently to hardware accelerators, such as the inherent data dependency between successive generations and the large, shared population memory. In other words, these features introduce design bottlenecks in the FPGA architecture that limit the maximum achievable speedup. To overcome these limitations, recent approaches study the implementation of “hardware-friendly” GA approaches, such as pipelined genetic propagation (PGN) and parallel (pGA) or distributed (dGA) algorithms. For example, Shackleford, B., et al. [70, 73] and Tang, W.K.-S. and L. Yip [70, 73] focus on the optimal design of hardware accelerators to achieve high performance and flexibility, while Tachibana, T., et al. [74] satisfy other GA features, such as multi-objective optimization.

In this thesis, we accelerate the BB-BC optimization algorithm using FPGA technology. The main features of the BB-BC algorithm, i.e., simplicity and efficiency, global optimization, fast convergence, scalability, and implementation feasibility, mainly motivated us to study the acceleration of this specific optimization algorithm. These features ensure that an FPGA-based BB-BC accelerator can significantly reduce computational time, providing an ideal platform for large-scale adaptive or latency-sensitive optimization problems.

Towards this end, we propose a *novel pipelined FPGA design* for the BB-BC algorithm. All the stages of the algorithm are implemented in a fully pipelined fashion to reduce latencies. We present a parameterized FPGA BB-BC architecture that offers tradeoffs among algorithm accuracy, circuit area, and clock frequency. The configurable design parameters of the BB-BC architecture are: i) the function for the calculation of the CoM: it can be either a light minimal

cost function or a complex equation (inverse of fitness function), ii) the arithmetic circuits: they can be composed of either synthesized HDL operators or highly optimized IP cores (Core Generator), iii) the length of integer and fractional parts of fixed point arithmetic, iv) the parameters of the BB-BC algorithm (i.e. number of iterations, population size, candidate length, LFSR seeds for the random candidate generators).

We implemented the FPGA BB-BC accelerator on an AMD/Xilinx Virtex-5 device and demonstrated its effectiveness for three different optimization problems: a simple two-dimensional function, the Rosenbrock function, and the Sphere function. The experimental results showed that the proposed FPGA accelerator achieved average speedups of 170x and 104x compared with custom GA s/w and BB-BC s/w programs running on an ARM Cortex-A9 embedded processor. The proposed FPGA-based approach was compared with its software counterpart (written in C) and a GPU (Graphics Processing Unit) implementation using the CUDA language. It was also compared with an FPGA accelerator generated via a high-level synthesis (HLS) design flow to demonstrate the benefits of the proposed well-optimized architecture. Given that the proposed architecture focuses on accelerating the two primary operations of the algorithm, i.e., the Big Band and the Big Crunch phases, complex, time-consuming fitness functions that dominate the total execution latency of the algorithm, may negatively affect the effectiveness of the methodology. To alleviate this problem, various fitness function approximation methods have been proposed in the past based on surrogate models, fitness inheritance, support vector machines, and neural networks [75],[76]. In this thesis, we adopted an approximation method, which is based on the adaptive neuro-fuzzy inference system (ANFIS) technique proposed in [77] to reduce the execution time of complex fitness functions. The ANFIS technique integrates neural network and fuzzy logic principles, and it has been shown to efficiently approximate nonlinear functions. We demonstrated the proposed approximation method for a complex fitness function (i.e., the Ackley function) on an AMD/Xilinx Kintex-7 design. The ANFIS-based Ackley FPGA design was generated by the AMD/Xilinx Vivado HLS tool.

We also demonstrated the applicability of our FPGA architecture on an image search problem, where the BB-BC algorithm is used to find the darkest pixel of a grey image. The experimental results showed that the proposed FPGA BB-BC architecture provides a significant

speedup over the competitive GPU technology across all cases and problem dimensions. Given also that the BB-BC algorithm converges faster than other typical GAs, the proposed FPGA BB-BC architecture provides an ideal optimization solver for real-time embedded applications.

3.2 Related work

As mentioned above, several researchers have investigated the adoption of FPGA technology to implement and accelerate GAs. In a pioneering work [65], Scott, S.D., et al., introduced general-purpose FPGA-based GA engines that combined coarse-grained pipelining and parallelization to achieve notable performance improvements. In [70], Shackleford, B., et al., proposed pipelining the execution of almost all GA states, i.e., parent selection, crossover, mutation, and fitness functions to maximize clock frequency. In [78], Guo, L., D.B. Thomas, and W. Luk proposed an automated FPGA framework based on a scalable and customizable hardware architecture for building and executing general-purpose GAs in FPGAs. In [69], Fernando, P.R. et al., presented a parameterized GA IP core synthesizable at both the register transfer (RT) level and gate-level that can be easily customized in terms of various GA parameters, e.g., population size, number of generations, crossover and mutation rates, random number generator (RNG) seed, and fitness function. Some approaches, such as [68] by Tachibana, T., et al., dealt with improving the performance of FPGA-based GA architectures using hardware speed-boosting techniques. Given that the FPGA-based GAs performance is often limited by the generational features inherited by the software GAs, which are not mapped efficiently to FPGA architecture, another category of approaches has been proposed in [71],[78],[72],[79] where the authors explore a new hardware-oriented GA algorithm style, which enables the development of massively parallel FPGA-based architectures. Towards this end, in [80] Nambiar, V.P., et al., proposed a hardware-software co-design methodology which improves system flexibility and reconfigurability by moving problem-dependent processes into software; the system partitioning is driven by various design aspects such as system constraints, operational intensity, process sequencing, hardware logic utilization, and reconfigurability. Despite these advances, typical FPGA-based GAs introduce design bottlenecks in FPGA architectures that are actually inherited by their sequential software implementations; these design bottlenecks impose significant performance barriers

[72]. More specifically, the selection and reproduction operations of the evolution process are actually two-by-two combination calculations. Thus, for the evolution process to proceed, the algorithm must wait until the entire population is generated. These data dependencies between the population generation stage and the parent selection/reproduction stage cause stalls in the pipelined execution of these two stages slowing down the system performance. Another issue is that the typical GAs require a large amount of memory to store the population in each generation [78]. The selection and reproduction operators can run in parallel but require access to the shared population memory. This memory sharing introduces a memory-access bottleneck in FPGA implementations, limiting the level of execution parallelism. To overcome these limitations, recent approaches [71],[79],[81] studied the implementation of “hardware-friendly” GA approaches. Specifically, Guo, L., et al., [71], proposed an intrinsically distributed and pipelined GA approach, called Pipelined Genetic Propagation (PGP). PGP consists of loosely coupled genetic operators and can be structured as a graph of parallel, pipelined components with local storage, thereby breaking the data-dependency limitation of typical generational GAs and eliminating the shared-memory bottleneck. In [79], Zhuo, L., et al., introduced a new GA based on a monogenetic strategy that retains only the optimal individual, significantly reducing the on-chip memory requirements, since only two individuals must be stored each time. In [81], Letras, M., et al., presented a hardware architectural framework for implementing cellular GAs (cGAs). In these GAs, the population is decentralized, and individuals are typically placed on a grid with a toroidal connectivity.

It is worth noting that in this thesis, we chose to study an optimization algorithm, namely the BB-BC algorithm, which does not suffer from the design bottlenecks of typical GAs discussed above, thereby restricting the parallelization efficiency of FPGA accelerators and consequently the maximum achievable speedup. More specifically:

Data dependencies between the BB and BC phases do not cause pipeline stalls. The calculations performed during the Big Crunch phase can commence before the completion of the Big Bang phase. This is because the CoM can be calculated on-the-fly by processing every newly generated candidate. This applies to both alternatives for the CoM calculation. In the case of the best fitness candidate, a reduction operation (comparison) is performed in the series of sequentially generated candidates. Similarly, in the case of Eq. (2.1) (see Chapter 2), the

numerator and denominator of the equation can be separately calculated by a reduction operation (accumulation) in the sequentially generated candidates and their fitness values. Thus, the operations performed during both the BB and BC phases can be fully pipelined to improve speedup.

Buffering of the population is not required. In the case of the BB-BC algorithm, the calculation of the CoM can be performed on-the-fly as explained in the above paragraph, and thus it does not require any buffering of the population. Only in the case that the reduction operation (comparison or accumulation) has a long latency (i.e., data reduction is slower than the data generation), the reduction circuit requires a small buffer for rate compensation.

3.3 Proposed FPGA architecture

In this section, we describe the proposed FPGA architecture of the BB-BC algorithm, which primarily relies on a pipelined BB-BC engine tailored to the computing needs of the algorithm. We also present all the performance-aware design techniques of the FPGA accelerator: the high-speed reduction circuits for the CoM calculation described in Section 3.3.2, the integration of parallel BB-BC engines described in Section 3.3.3 and the Adaptive Neuro-Fuzzy Inference Systems (ANFIS) we adopted for the fitness function approximation method described in Section 3.3.4

3.3.1 Pipelined BB-BC engine

The main building block of the FPGA architecture is the BB-BC engine shown in Figure 13. The BB-BC engine implements the three processing stages (i.e., random generation, fitness function, center-of-mass calculation) of each iteration of the algorithm in a fully pipelined fashion. Figure 14 presents the pipelined execution of two consecutive BB-BC iterations. Each BB-BC iteration includes three stages: Generation of candidates (Cand), Calculation of fitness function (Fit), and Calculation of CoM. Notice that in order to commence an iteration cycle, the previous one must have been completed (i.e., pipelined execution is not feasible at the iteration level). This is due to the fact that the BB Phase, i.e., random generation, requires the value of the CoM obtained by the BC Phase of the previous iteration. The description of the pipeline stages of the BB-BC engine follows:

- **Generation of candidate:** The RandGen module generates a candidate ($\vec{x}^i$), either to form an initial population (iteration 0) or to create new candidates around the CoM (iteration > 0). The candidate value is stored in a pipelined register. A typical linear feedback shift register (LFSR) is used to generate random numbers based on a configurable seed. The random number r is multiplied by the value l/k , as depicted in Eq. (2.1) (see Chapter 2). The value l/k is calculated statically and stored in a ROM with M entries, where M is the number of iterations. Thus, its calculation does not impose delays in the generation of candidates. The stage delay is actually dominated by the multiplier latency. In each cycle, the RandGen module generates L random numbers, where L is the number of features of each candidate. In each iteration, the module generates N candidates sequentially.

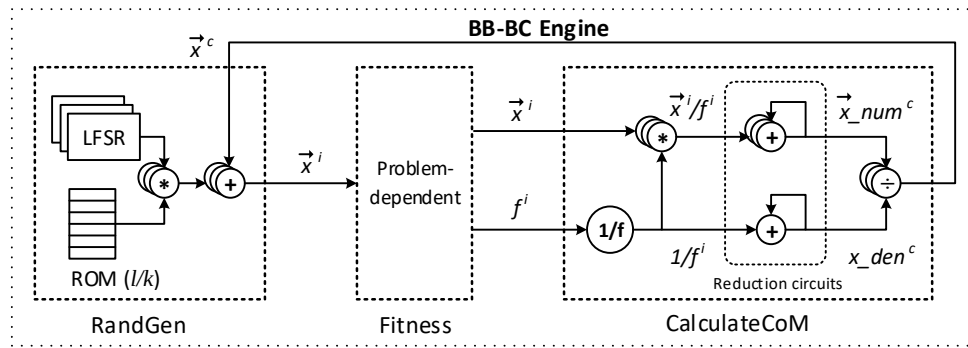


Figure 13. Block diagram of the BB-BC engine

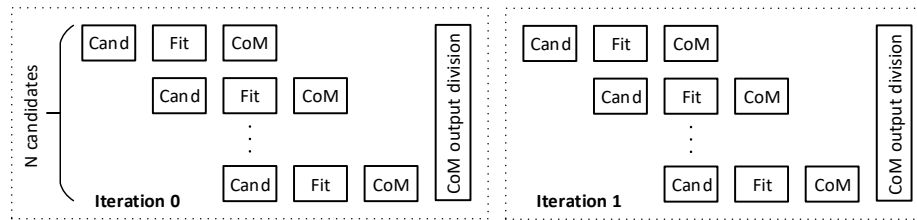


Figure 14. Pipelined execution of the BB-BC engine

- **Calculation of fitness function:** The Fitness module calculates the fitness function (f^i) for each candidate. The fitness function is application dependent. The stage latency depends on the fitness function complexity. For complex fitness functions, the stage should be pipelined to achieve high clock frequency. An approximation method is also proposed in the next subsections to reduce the latency of complex fitness functions.
- **Calculation of CoM:** The CalculateCoM module calculates the CoM ($\vec{x}^c$) for each iteration. The module runs N consecutive cycles to sample and process the N candidates of the

current iteration. The CoM calculation finishes when the last candidate has been processed. The module supports two alternatives for the CoM calculation (Figure 13 depicts, only the second option):

a. minimal cost function: $\vec{x}^c = \vec{x}^i \ni \min_{i=1..N} f^i$ which chooses the candidate of the generation with the best (smallest) fitness function value. In this case, the CalculateCoM module integrates a simple reduction (comparison) circuit to perform this function. The stage delay depends on the comparator latency.

b. complex inverse fitness function (see Eq. (2.1)): $\vec{x}^c = \vec{x}_{num}^c / x_{den}^c$. The module calculates the numerator separately ($\vec{x}_{num}^c$) and the denominator (x_{den}^c) of the CoM. It integrates a divider to calculate the reciprocal of f^i and L multipliers for the products $x^i * 1/f^i$. Also, $L+1$ reduction (accumulation) circuits are used to calculate the numerator (L points) and the denominator and L dividers to calculate the final result. The output dividers are activated when the last candidate has been processed. The module has been designed in a fully pipelined fashion (assuming that the arithmetic circuits are also pipelined), and its total delay depends on the latency of the arithmetic circuits (i.e., divider, multiplier, accumulator). Note that the design of reduction circuits requires special attention when the reduction operation has a long latency. For more information about the design of high-speed reduction circuits, refer to the following subsection.

Our FPGA BB-BC architecture is parameterized to support various alternatives regarding the function for the calculation of CoM, the implementation of arithmetic operators (multipliers), the number representation (fixed-point arithmetic) and the parameters of the BB-BC algorithm. The main parameters of the FPGA BB-BC architecture are presented in Table 3.

Table 4 presents the latency and the arithmetic operators of the pipeline stages of the BB-BC engine. Note that the output divider for the calculation of the complex CoM is activated only for the last candidate of every iteration (see the asterisk in the last row of the Table). Assuming population size N and number of iterations M , the total execution time (ET) of the proposed FPGA architecture for the minimal cost function (ET-MCF_{total}) and the complex CoM equation (ET-CCE_{total}) are (T_A , T_M , T_D , T_R , and T_c are the Adder, Multiplier, Divider, Reciprocal, and Comparator latencies):

$$ET-MCF_{\text{total}} = M \times ((1+T_M+T_A) + T_F + T_C + N) \quad (3.1)$$

$$ET-CCE_{\text{total}} = M \times ((1+T_M+T_A) + T_F + (T_R+T_M+T_A) + N + T_D) \quad (3.2)$$

Table 3. FPGA BB-BC architecture parameters

Parameter	Values	Comments
CoM calculation function	(a) minimal cost function (b) complex inverse fitness function	Option (a) reduces system complexity, while option (b) improves convergence speed/accuracy
Arithmetic Circuits	(a) simple non-pipelined HDL operators (b) use of highly optimized IP blocks	Option (b) is selected for speed optimization
Number representation	Fixed point arithmetic (variable length of integer and fractional parts)	Future versions will support floating-point arithmetic
BB-BC fitness function	Problem-dependent	Future versions will support dynamic reconfiguration of fitness function
BB-BC algorithm parameters	(a) number of iterations (M) (b) population size (N) (c) number of candidate features (L) (d) LFSR seeds	Algorithm parameters are configured statically. Future versions will support dynamic reconfiguration of all parameters

Table 4. BB-BC engine latency and arithmetic operations

Pipeline stage	CoM function	Latency	Arithmetic operations
Candidate generation	-	$1+T_M+T_A$	L LFSR, L Multipliers, L Adders
Fitness calculation	-	T_F Problem-dependent	Problem-dependent
CoM calculation	Minimal cost	T_C	1 Comparator
	Complex CoM	$(T_R+T_M+T_A) + T_D^{(*)}$	1 Reciprocal, L Multipliers, $L+1$ Adders, L Dividers

3.3.2 High-speed reduction circuits for the CoM calculation

High-speed reduction circuits are employed to address the bottleneck associated with the calculation of the CoM in the BB-BC algorithm. This is particularly significant in hardware-based implementations, where parallel and pipelined operations are leveraged to maximize speed. The CoM calculation is a sequential process and can become a critical performance barrier if it is not optimized because: a) it constitutes a substantial portion of the total execution time in the BB-BC algorithm, especially when an increased number of candidates or dimensions, and b) it is performed at every iteration of the algorithm and directly impacts the overall execution time. Any delay in the CoM calculation disrupts the pipeline, diminishing the benefits of parallelism and potentially stalling the entire optimization process. Thus, a slow CoM calculation can create a performance bottleneck, delaying subsequent operations in the BC phase and reducing the achievable throughput of the system. Finally, by reducing

its latency, we ensure that the CoM computation does not dominate the execution time, thereby improving the overall performance and responsiveness of the algorithm.

The reduction method refers to reducing a series of sequentially generated input values (in our case, the candidate values and their inverse fitness functions) to a single value (the numerator and the denominator of the complex CoM equation) using a single binary operator. In our case, the binary operator is $\sum d_i$, where d_i equals $\frac{1}{f^i} x^i$ and $\frac{1}{f^i}$ for the numerator and the denominator, respectively. Note that for the accumulation, the reduction operation is a 2-input addition. If the latency of the primitive operation (comparison or accumulation) is one clock cycle, then the design of the reduction circuit is straightforward. However, for multiple-clock-cycle latency (e.g., due to high-speed pipelined operators or floating-point arithmetic), the design of the reduction circuit requires special attention. For example, suppose a simple accumulator is employed to perform the reduction for an operator with t -cycle latency. In that case, the circuit must wait for t clock cycles before feeding the next value to the accumulator, introducing pipeline stalls during the reduction process. The CoM reduction could be implemented using a full binary tree with P leaf operators. A pipelined implementation of the tree provides a high-speed reduction circuit that can input up to P candidate values per clock cycle. However, this solution requires a large amount of hardware resources (i.e., $P-1$ operators) and is practically infeasible even for moderate values of P . Instead, high-speed reduction circuits as proposed in [79],[82] can be utilized. These circuits reduce multiple sets of sequentially generated input data without stalling the pipeline or imposing large buffer requirements. In this thesis, reduction operations (comparison and accumulation) with one-cycle latency are assumed, and thus, no sophisticated reduction circuits are required.

3.3.3 Parallel FPGA BB-BC architecture

To improve performance, several BB-BC engines can be integrated into our hardware accelerator. Figure 15.d shows four BB-BC engines running in parallel. Each engine encapsulates a RandGen module to generate random candidates $\vec{x}^i$, a Fitness module to calculate the fitness function f^i and a CalcCoM module (Figure 15.a), which calculates the reciprocal of f^i and the product $\vec{x}^i * 1/f^i$. The values produced by all parallel engines are processed by a full

binary tree of 2-cand-add operators. Each 2-cand-add module (Figure 15.b) produces the sum of the values ($\vec{x}^i/f^i$ and $1/f^i$) of two candidates. The final output of the binary tree is the sum of the values $\sum_{i=1}^4 \vec{x}^i/f^i$ and $\sum_{i=1}^4 1/f^i$ of all four engines. The output of the tree is fed to the CoM-reduction module (Figure 15.c), which reduces the sequentially generated values of the partial sums of four candidates to calculate the numerator ($\vec{x}_{num}^c$) and the denominator (x_{den}^c) of Eq. (2.1) (see Chapter 2) and finally the CoM of the generation. The latency and the arithmetic operators of the extra modules used in the parallel scheme are presented in Table 5. Assuming G parallel engines, the full binary tree includes $G-1$ 2-cand-add modules. The total execution time (ET) of the parallel FPGA architecture for the minimal cost function (ET-MCF_{total}) and the complex CoM equation (ET-CCE_{total}) are:

$$ET-MCF_{total} = M \times ((1+T_M+T_A) + T_F + T_C + N/G) \quad (3.3)$$

$$ET-CCE_{total} = M \times ((1+T_M+T_A) + T_F + (T_R+T_M) + (\log_2 G \times T_A) + T_A + N/G + T_D) \quad (3.4)$$

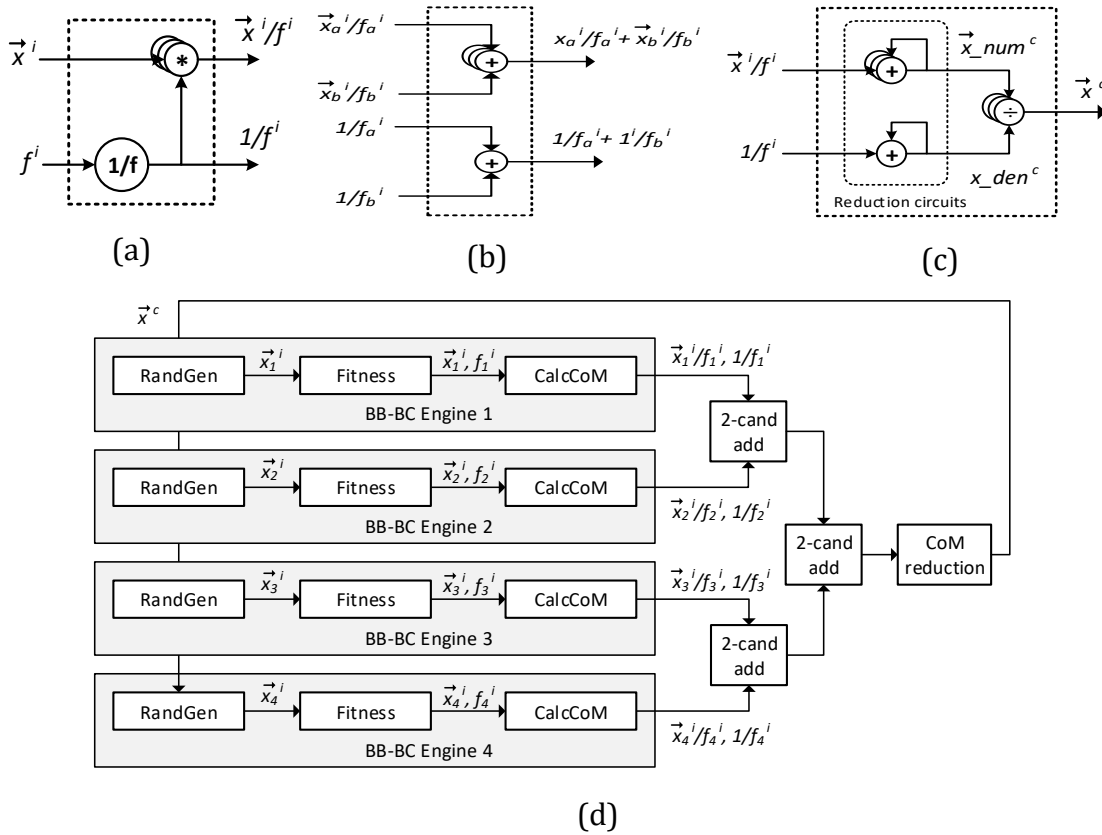


Figure 15. Parallel BB-BC architecture: a) CalcCoM module, (b) 2-cand-add module, (c) CoM-reduction module, (d) Four parallel BB-BC engines

Table 5. Parallel BB-BC scheme latency and operations

Module	CoM function	Latency	Arithmetic operations
CalcCoM	Minimal cost	-	-
	Complex CoM	$T_R + T_M$	1 Reciprocal, L Multipliers
2-cand-add tree	Minimal cost	$\log_2 G \cdot T_C$	$G-1$ Comparators
	Complex CoM	$\log_2 G \cdot T_A$	$(G-1) \cdot (L+1)$ Adders
CoM-reduction	Minimal cost	T_C	1 Comparator
	Complex CoM	$T_A + T_D^{(*)}$	$L+1$ Adders, L Dividers

3.3.4 Fitness function approximation method

We adopted the ANFIS approximation method to reduce the execution time of complex fitness functions. ANFIS is an adaptive neuro-fuzzy inference system proposed by Jang in [77], which combines neural networks and fuzzy logic to model complex functions. It is a class of adaptive networks functionally equivalent to first-order Takagi–Sugeno fuzzy reasoning [83]. The ANFIS approach replaces complex calculations with fast, approximate evaluations and learns the consequent and premise parameters of membership functions (MFs) of the rules from data. For a first-order Takagi–Sugeno fuzzy model [84], where two inputs x_1, x_2 and one output y is assumed Figure 17. The common rule set for two fuzzy *IF-THEN* rules can be expressed as follows:

Rule 1: if x_1 is A_{11} and x_2 is A_{21} , then $y_1 = a_{11} \cdot x_1 + a_{21} \cdot x_2 + b_1$

Rule 2: if x_1 is A_{12} and x_2 is A_{22} , then $y_2 = a_{12} \cdot x_1 + a_{22} \cdot x_2 + b_2$

Where x_1 and x_2 represent the linguistic variables of MFs and $A_{11}, A_{21}, A_{12}, A_{22}, b_1$ and b_2 are the parameters of the consequent first-order polynomial function to be determined during the training stage. Figure 16 illustrates the Takagi–Sugeno fuzzy reasoning for two inputs and two rules. The $\mu_{A_{i1}}(x_1), \mu_{A_{i2}}(x_2)$ represents the membership grade of the inputs x_1 or x_2 of the fuzzy MF ($A_{11}, A_{21}, A_{12}, A_{22}$).

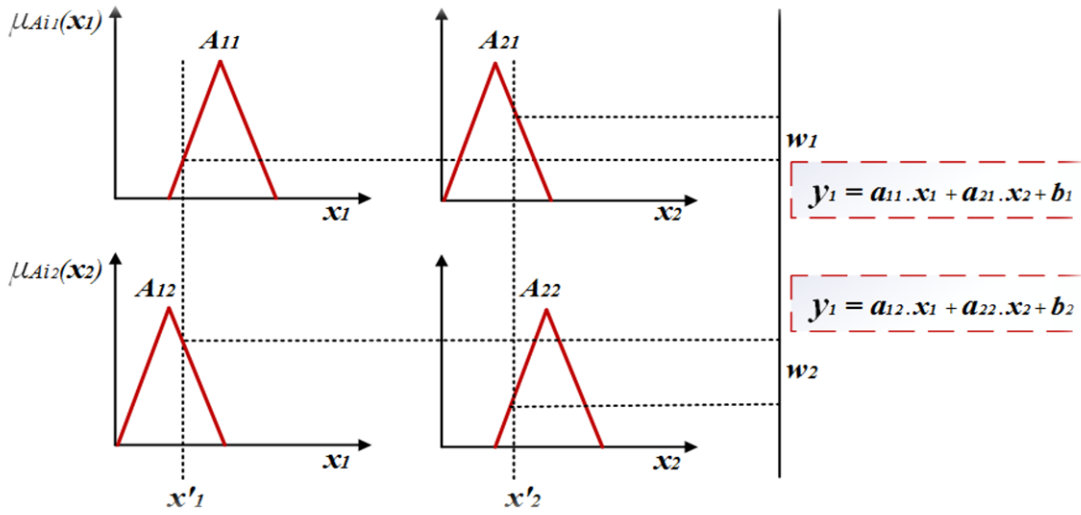


Figure 16. A two-input first-order Takagi-Sugeno fuzzy reasoning

The equivalent ANFIS architecture is shown in Figure 17. The nodes in the same layer of the network have similar functions. The ANFIS structure consists of five layers, excluding the input layer.

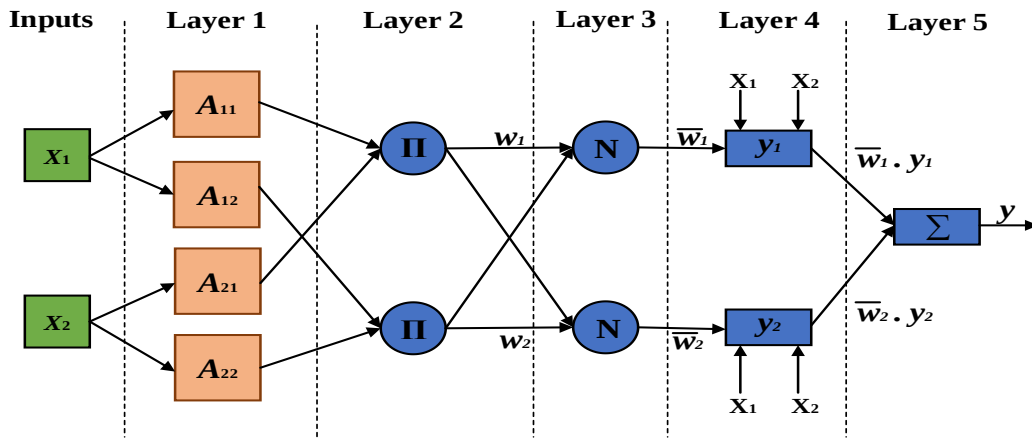


Figure 17. An equivalent ANFIS structure to a first-order Takagi-Sugeno fuzzy model with two inputs and two rules

Layer 1: Every square node in this layer in the ANFIS structure is an adaptive node and performs fuzzification. They are labelled Π to indicate that they multiply the incoming signals to produce the firing strength of the rule. The MF A_{i1} are parameterized triangular functions:

$$\mu_{A_{i1}}(x) = \begin{cases} 0, & x \leq \rho_i \\ \frac{x-\rho_{i1}}{m_{i1}-\rho_{i1}}, & \rho_{i1} \leq x \leq m_{i1} \\ \frac{\varphi_{i1}-x}{\varphi_{i1}-m_{i1}}, & m_{i1} \leq x \leq \varphi_{i1} \\ 0, & x \geq \varphi_{i1} \end{cases} \quad (3.5)$$

where A_{i1} are the linguistic variables, the set $\rho_{i1}, m_{i1}, \varphi_{i1}$ are local adaptive parameters also known as premise parameters. Their values are updated by gradient descent during the learning phase. The node output is the fuzzy membership grade $\mu_{A_{i1}}(x)$.

Layer 2: Product Layer: This layer has as many nodes as the fuzzy rules. Each node output represents the degree of fulfilment of a fuzzy rule. Every node can perform any T-norm operators, that is, perform a multiplication operator on fuzzy membership grades:

$$w_i = \mu_{A_{i1}}(x_1) \cdot \mu_{A_{i2}}(x_2), \quad \text{for } i = 1, 2 \quad (3.6)$$

Layer 3: Normalization Layer: In the third layer, every node is fixed (circle node). They are labelled N to indicate the calculation of the ratio of the i^{th} rules firing strength to the sum of rules firing strengths, and calculate the normalized value of the degree of fulfilment according to the following formula:

$$\bar{w}_i = \frac{w_i}{w_1 + w_2} \quad \text{for } i=1, 2 \quad (3.7)$$

Layer 4: Defuzzify Layer: The adaptive nodes in the fourth layer are the conclusive parts of fuzzy rules. Every node computes the product of the normalized degree of fulfilment and an affine function (first-order Takagi-Sugeno model) as follows:

$$y_i = a_{1i} \cdot x_1 + a_{2i} \cdot x_2 + b_i \quad (3.8)$$

The adaptive parameters a_{1i} and b_i in this layer are called consequent parameters. The linear function $y(\cdot)$ is called local linear model in the areas of interpolation and approximation theory.

Layer 5: Output Later: Finally, in the last layer, a single fixed node is labeled Σ to indicate that it calculates the overall network output as the summation of all incoming signals.

$$y = \sum_{i=1}^M \bar{w}_i \cdot y_i(x) = \frac{\sum_i w_i y_i}{\sum_i w_i} = \frac{w_1 y_1 + w_2 y_2}{w_1 + w_2} \quad \text{for } i=1, 2 \quad (3.9)$$

The fuzzy rules are actually local linear models. The interpolation of these local linear models achieves global mappings, that is, identification of nonlinear relationships.

The ANFIS employs a hybrid learning algorithm to train the network. It is a two-phase algorithm. In the first phase (forward pass) of the hybrid learning, the least-square method is used to estimate the consequent parameters of Takagi–Sugeno type rules (*Layer 3*). In the second phase (backward pass), the gradient-descent method is used to adjust the parameters of the antecedents (*Layer 1*).

From the ANFIS architecture shown in Figure 17 and Eq. (3.9), we found that the ANFIS output y can be expressed as a linear combination of the consequent parameters when the premise parameters are fixed. In symbols, the output y can be rewritten as:

$$y = \frac{w_1 \cdot y_1 + w_2 \cdot y_2}{w_1 + w_2} = \frac{w_1}{w_1 + w_2} \cdot y_1 + \frac{w_2}{w_1 + w_2} \cdot y_2 = \bar{w}_1 \cdot y_1 + \bar{w}_2 \cdot y_2 \quad (3.10)$$

3.3.5 Application of the ANFIS method

The ANFIS method has been demonstrated on the Ackley function:

$$f(x) = -\alpha \exp\left(-b \sqrt{\frac{1}{d} \sum_{i=1}^d x_i^2}\right) - \exp\left(\frac{1}{d} \sum_{i=1}^d \cos(cx_i)\right) + a + \exp(1) \quad (3.11)$$

The Ackley function is widely used for testing optimization algorithms. In its two-dimensional form, it is characterized by a nearly flat outer region and a large hole at the center, as shown in Figure 21. Ackley is a minimization problem with a global minimum $f(x) = 0$, at $\vec{x} = (0, 0, \dots, 0)$ and is usually evaluated on the hypercube $x_i \in [-32.768, +32.768]$, $i = 1, \dots, d$. The recommended variable values are: $a = 20$, $b = 0.2$ and $c = 2\pi$.

A total dataset with 4,225 elements has been used for training the network. The ANFIS model has been implemented using MATLAB ANFIS toolbox [85]. Figure 18 illustrates the ANFIS model structure used to approximate the Ackley function. The rule base consists of 25 rules, corresponding to the 5 MFs of the two inputs (x_1, x_2) .

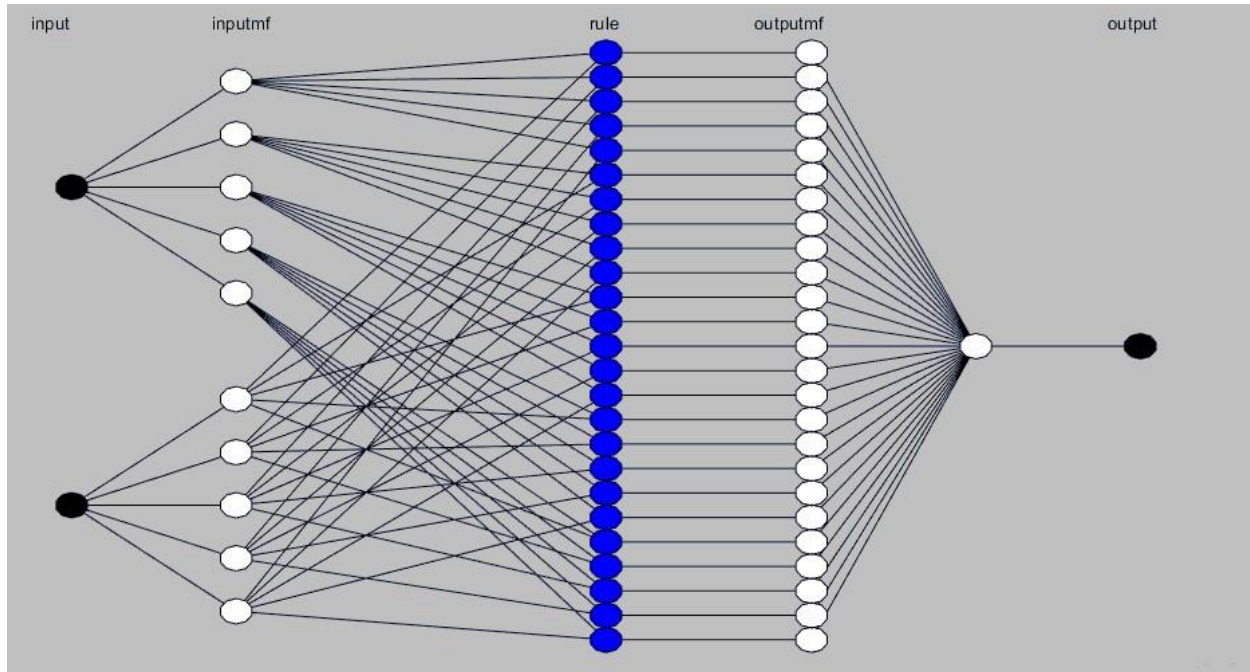


Figure 18. ANFIS model structure with two inputs and one output for Ackley function approximation

Triangular MFs are popular for specifying fuzzy sets, providing concise notation. There are five fuzzy values (rules) A_{ij} for both x_1, x_2 variables as shown in Figure 19.

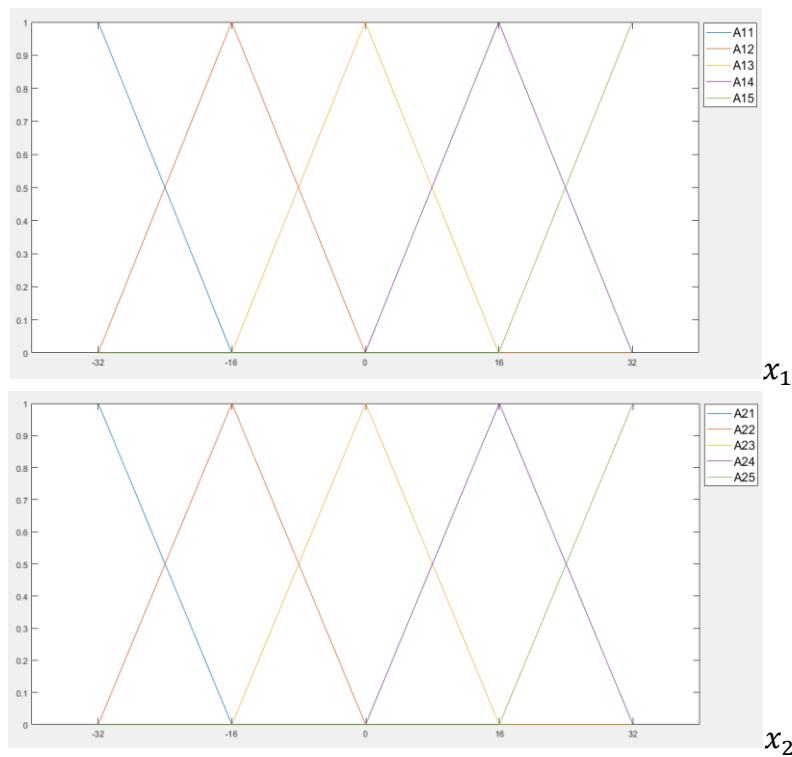


Figure 19. T-S membership functions plot for inputs x_1, x_2

The Sugeno-type fuzzy inference, also known as the Takagi-Sugeno method, is a type of fuzzy inference system that utilizes fuzzy logic to model complex systems by combining fuzzy rules and mathematical functions for reasoning. The Sugeno inference outputs are crisp values derived from mathematical functions. The total fuzzy rules used in the ANFIS-based Ackley FPGA design are presented in Table 6.

Table 6. The Sugeno-type if-then rules

	A₁₁	A₁₂	A₁₃	A₁₄	A₁₅
A₂₁	$y_1 = 0.05852 x_1 + 0.05853 x_2 + 23.75$	$y_6 = 0.05721 x_1 - 0.1361 x_2 + 16.36$	$y_{11} = 0.0005896 x_1 + 0.8159 x_2 + 45.61$	$y_{16} = -0.05625 x_1 - 0.1364 x_2 + 16.34$	$y_{21} = -0.05759 x_1 + 0.05865 x_2 + 23.72$
A₂₂	$y_2 = -0.1361 x_1 + 0.05721 x_2 + 16.36$	$y_7 = -0.1546 x_1 - 0.1546 x_2 + 14.53$	$y_{12} = -0.001428 x_1 + 0.675 x_2 + 28.95$	$y_{17} = 0.1521 x_1 - 0.1548 x_2 + 14.57$	$y_{22} = 0.1337 x_1 + 0.05731 x_2 + 16.44$
A₂₃	$y_3 = 0.8159 x_1 + 0.0005936 x_2 + 45.61$	$y_8 = 0.6749 x_1 - 0.001438 x_2 + 28.95$	$y_{13} = 0.0005628 x_1 + 0.0005799 x_2 + 0.03777$	$y_{18} = -0.6779 x_1 - 0.001437 x_2 + 28.99$	$y_{23} = -0.819 x_1 + 0.0005931 x_2 + 45.71$
A₂₄	$y_4 = -0.1364 x_1 - 0.05624 x_2 + 16.34$	$y_9 = -0.1548 x_1 + 0.1521 x_2 + 14.57$	$y_{14} = -0.001427 x_1 - 0.6779 x_2 + 28.99$	$y_{19} = 0.1523 x_1 + 0.1523 x_2 + 14.6$	$y_{24} = 0.1339 x_1 - 0.05635 x_2 + 16.41$
A₂₅	$y_5 = 0.05865 x_1 - 0.05759 x_2 + 23.72$	$y_{10} = 0.05731 x_1 + 0.1337 x_2 + 16.44$	$y_{15} = 0.0005892 x_1 - 0.819 x_2 + 45.71$	$y_{20} = -0.05635 x_1 + 0.1339 x_2 + 16.41$	$y_{25} = -0.05771 x_1 - 0.05771 x_2 + 23.7$

The inputs of the ANFIS model, the number of MFs used, the number of rules, and the number of parameters are given in Table 7.

Table 7. ANFIS parameter counts for approximating the Ackley function

Inputs	Membership functions	Rules	Parameters	Number of train data
x_1, x_2	10 [5, 5]	25	225	4225

The Sugeno-type ANFIS was designed using the proposed method, and the system's training was performed in *MATLAB*. To generate the Fuzzy Inference System (FIS), we employed grid partitioning and trained it using a hybrid optimization method over 100 total epochs. Figure 20 demonstrates the results of the ANFIS training, i.e., how the sum of squared errors decreases with the epochs.

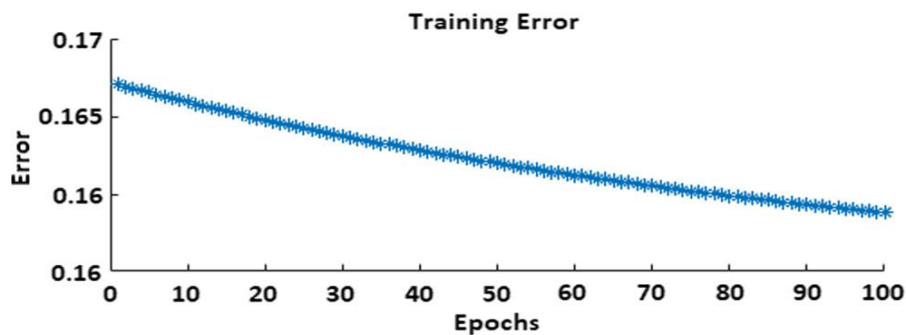


Figure 20. Sum of square error versus the number of epochs for training dataset

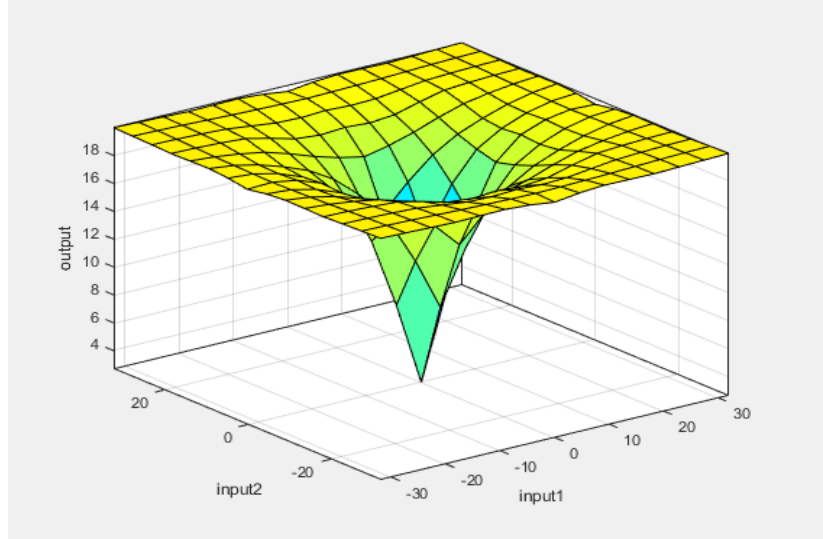


Figure 21. Surface graph showing approximation of the Ackley function

The surface graph in Figure 21 indicates that the solution obtained from ANFIS is continuous for all the training data and forms a good approximation of the Ackley function. The global minimum of the Ackley function was approximately “0”. The ANFIS-based Ackley function has been implemented and optimized using an HLS design flow and the implementation details are presented in Section 3.4.4 and Table 13.

Here, we summarize all the innovative hardware speed-boosting techniques suggested for the FPGA-based acceleration of the BB-BC algorithm:

- It provides flexible implementation of arithmetic operators using synthesized HDL operators or optimized IP cores.
- It is compatible with real number approximation techniques.
- It supports configurable BB-BC parameters.
- It supports two alternatives for the calculation of CoM
- It implements all the stages of the BB-BC algorithm in a fully pipelined fashion.
- It enables parallel operation with multiple pipelined BB-BC engines; these engines act independently during the BB phase but are synchronized and collaborate to produce a merged CoM during the BC phase.
- It implements the CoM reduction using a full binary tree (a pipelined implementation of the tree provides a high-speed reduction circuit).

- It provides an ideal platform for large-scale adaptive or latency-sensitive optimization problems.
- It integrates the ANFIS-based approximation method to reduce the execution time of complex fitness functions.

3.4 Evaluation of the FPGA BB-BC architecture

We present a large set of experiments to demonstrate the efficiency of the proposed FPGA BB-BC architecture. First, we implemented it on an AMD/Xilinx Virtex-5 XC5VLX10T device (Xilinx XUPV505 development board) and demonstrated its efficiency for three different fitness functions in terms of hardware area and system performance. Next, we compared the proposed accelerator architecture with an HLS-based accelerator generated and optimized using the state-of-the-art AMD/Xilinx Vivado HLS tool for an AMD/Xilinx Kintex-7 XC7K325T device (Xilinx KC705 development board). Next, we demonstrated the efficiency of the ANFIS approximation method on a complex fitness function using again the Vivado HLS tool. Finally, we studied the applicability of the proposed architecture in a real-time image processing application, where the BB-BC algorithm is used to solve an image search problem.

3.4.1 Comparison with GA for different fitness functions

We evaluated the proposed FPGA BB-BC architecture for the following three fitness functions:

$$f1(x_1, x_2) = (a - x_1)^2 + b(x_2 - x_1^2)^2 \quad (3.12)$$

$$f2(x_1, x_2) = (a - x_1)^2 + (x_2 - b)^2 \quad (3.13)$$

$$f3(x_1, x_2, \dots, x_n) = \sum_{i=1}^n x_i^2 \quad (3.14)$$

The non-convex Rosenbrock function, given in Eq. (3.12) is a minimization problem with a global minimum in a long, narrow valley. For the two-dimensional Rosenbrock function of Eq. (3.12) the global minimum is $(x_1, x_2) = (a, a^2)$. The simple continuous, unimodal function given in Eq. (3.13) is a minimization problem, having a global minimum at $(x_1, x_2) = (a, b)$. The convex and unimodal Sphere function, given in Eq. (3.14) is a minimization problem and has n local minima except for the global one located at $(0,0,\dots,0)$. We have

implemented, herein, the two-dimensional form of the Sphere function. All the fitness functions have been implemented in FPGA using fixed-point arithmetic.

The performance of the BB-BC algorithm has been evaluated on these three benchmarks and compared with that of a typical GA. The results from the performance evaluation experiments are presented in Table 8 and Figure 21. For the Rosenbrock function, it is assumed that $\alpha = 1$ and $b = 1$, while for the Unimodal function $\alpha = 5$ and $b = 10$. The Rosenbrock function has been evaluated in the range $x_1, x_2 \in [-2, 2]$, the Unimodal function in $[0, 12]$ and the Sphere function in $[-5.2, 5.2]$. For both BB-BC and GA algorithms, the population size is 256 and the maximum number of iterations is 1000. The tolerance has been set at 10^{-6} , which corresponds to the minimum acceptable variation in the estimation of the optimal value. Table 8 compares the two algorithms based on their proximity to the optimal solution. The Estimated Optimal Solution, the Distance from the optimal solution and the Estimated Optimal Value after 1000 iterations are presented. Note that since all optimal (minimum) values are equal to 0, the Estimated Optimal Values are equal to the error with respect to the true minimum. The experiments proved that the BB-BC algorithm outperforms the GA algorithm in terms of proximity to the true optimal value for all three benchmarks. Figure 22 compares the convergence speed of the two algorithms presenting their intermediate estimated optimal values, from iteration #100 to iteration #1000. It is evident that the BB-BC converges significantly faster to the proximity solution than the GA algorithm for all three benchmarks.

Table 8. Performance comparison between BB-BC and GA

Fitness Function	Algorithm	Estimated Optimal Solution		Distance	Estimated Optimal Value
		x_1	x_2		
Rosenbrock	BB-BC	0.9992	0.9989	0.0013	2.1704e-05
	GA	0.9954	0.9904	0.0106	3.1741e-05
Unimodal	BB-BC	5.0194	10.0034	0.0360	0.0013
	GA	5.0576	10.0294	0.0647	0.0420
Sphere	BB-BC	0.0000	0.0168	0.0168	1.3797e-11
	GA	0.0138	0.4245	0.4247	1.9138e-04

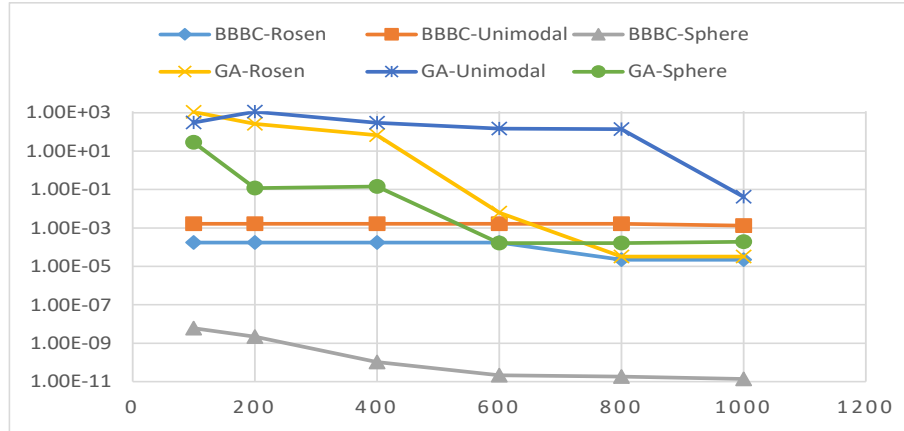


Figure 22. Performance comparison between BB-BC and GA (Convergence speed)

3.4.2 Evaluation of FPGA area and frequency

We implemented several versions of the FPGA BB-BC architecture to evaluate how fitness functions and other design parameters (i.e., CoM function, arithmetic circuits' synthesis, single or parallel engine) affect system area and performance. First, we studied the effects of the various design parameters in the FPGA BB-BC architecture. Table 9 presents the device utilization (LUTs, registers, DSP slices) and the operating frequency of the system using the Rosenbrock fitness function and: i) single engine or dual (parallel) engine, ii) for the calculation of CoM: minimal cost function (MCF) or complex CoM equation (CCE), ii) for the arithmetic circuits (multipliers): non-pipelined HDL operators or highly optimized IP cores (Core Generator). Note that for all the BB-BC implementations, we utilized fixed-point arithmetic (32-bit width) to represent the function variables.

From the experimental results, we can observe the following:

- The parallel engine significantly increases the reconfigurable resources ($\sim 1.8X$ in the case of MCF and less than $1.4X$ in the case of CCE) with a slight decrease in the operating frequency ($\sim 15\%$). In the case of the CCE function, the CoM-Reduction module, which integrates L dividers that occupy a large portion of the device area, is not replicated across the parallel engines; thus, the resulting area increase is lower.
- The use of the CoM function, i.e., MCF or CCE, significantly affects the device area since the CCE integrates complex arithmetic circuits, but does not affect the clock frequency. Given that the design of the Calculate-CoM modules is fully pipelined, the more complex CCE function does not lengthen the critical paths. The static timing analysis of the system

revealed that the maximum clock frequency primarily depends on the latency of the arithmetic circuits.

- The use of CoreGen IP cores dramatically improves the operating frequency with a minor penalty in the device area (especially the registers) due to the highly optimized pipelined designs of the IP arithmetic cores.

Table 9. Reconfigurable resources and maximum operating frequency of the FPGA BB-BC architecture for the Rosenbrock fitness function

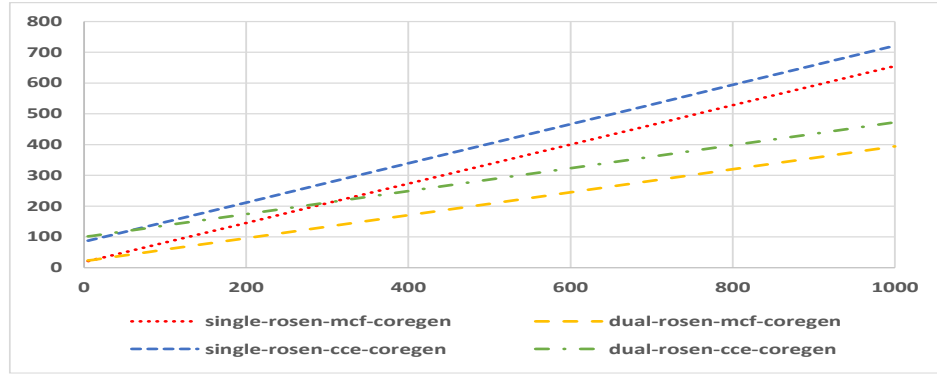
CoM function	Scheme	Arithmetic circuits	LUTs	Registers	DSP48Es	Clock freq (MHz)
MCF	Single engine	HDL	1690	982	24	74.0
		CoreGen	1736	1245	24	314.2
	Dual engine	HDL	3079	1783	48	61.7
		CoreGen	3076	2296	48	268.3
CCE	Single engine	HDL	7477	12996	32	74.0
		CoreGen	7468	13247	32	314.2
	Dual engine	HDL	9935	15291	64	61.7
		CoreGen	9880	15795	64	268.3

Next, we studied the effects of the fitness function in the FPGA BB-BC architecture. Table 10 presents the device utilization and the operating frequency of the single BB-BC engine for the three fitness functions: i) Rosenbrock, ii) Simple, and iii) Sphere using the MCF function and HDL operators or CoreGen IP cores for the arithmetic circuits. Note that the selection of the CoM function and the parallel engines are not correlated with the functionality of the Fitness module. From the experimental results, it is observed that the most complex fitness function (i.e., Rosenbrock) requires more reconfigurable resources but does not affect the maximum operating frequency or, consequently, the system throughput.

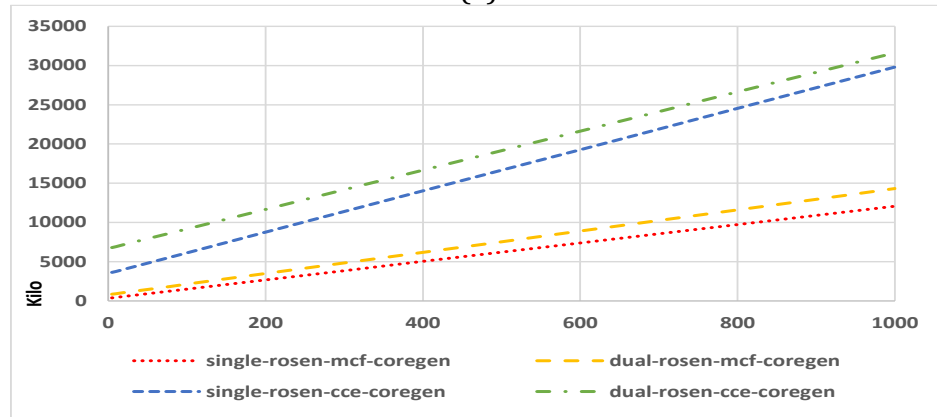
Table 10. Reconfigurable resources and maximum operating frequency of the FPGA BB-BC architecture for the three fitness functions (Rosenbrock, Simple, and Sphere)

Fitness function	Arithmetic circuits	LUTs	Registers	DSP48Es	Clock freq (MHz)
Rosenbrock	HDL	1690	982	24	74.0
	CoreGen	1736	1245	24	314.2
Simple	HDL	1289	756	16	74.0
	CoreGen	1297	960	16	314.2
Sphere	HDL	1228	627	16	74.0
	CoreGen	1231	831	16	314.2

Next, we investigated the impact of design parameters on the execution time of the BB-BC algorithm. Figure 23.a compares the execution time (in μs) of the single engine and the dual engine versus population size (N) for the Rosenbrock function assuming: MCF or CCE function, CoreGen building blocks and number of iterations (M) 200. The arithmetic circuits' latencies, when the CoreGen IPs are used, are $T_A=1$, $T_M=6$, $T_D=68$, $T_R=30$, $T_c=1$ and $T_{F1}=20$ (Rosenbrock). Figure 23.b presents the product “area $\times$ execution time” of the same setup with Figure 23.a. For the area calculation, a 644-LUT equivalent area for the DSP48E has been assumed [86]. Thus, the product “area $\times$ execution time” is $(\text{LUTs} + \text{Regs} + 644 \times \text{DSP48E}) \times \text{ET}$ (in μs). The curves of Figure 23.a clearly demonstrate the performance improvement achieved by the parallel scheme (dual engine) compared to the single engine, which increases with the population size. Moreover, the CCE function consistently increases the total execution time compared with the MCF due to its higher complexity; however, it enables faster convergence of the optimization algorithm, as expected. On the other hand, Figure 23.b proves that the single engine achieves a better tradeoff between area and execution time than the dual engine for both the MCF and CCE functions. It should be noted that results for more parallel engines are not presented because they exceed the available number of DSP48Es of the target Virtex-5 device.



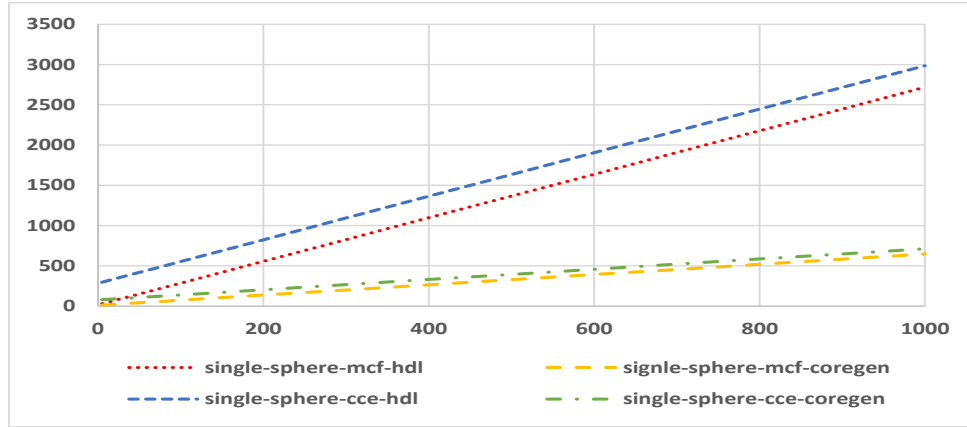
(a)



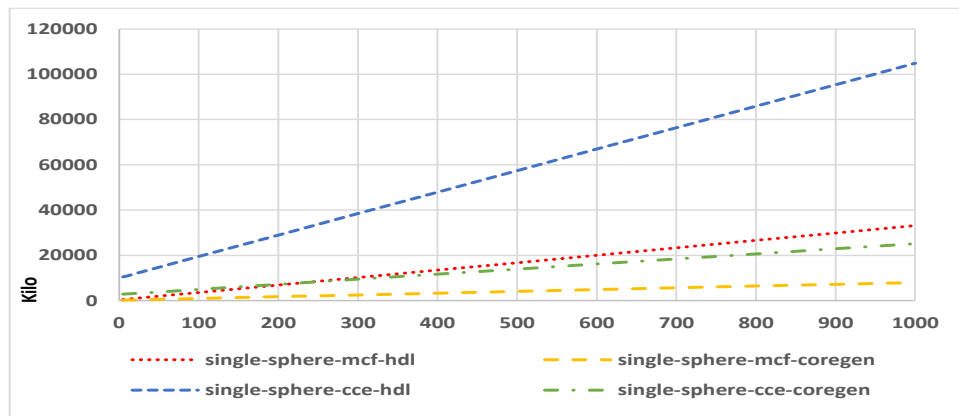
(b)

Figure 23. Comparison between single and dual engines for the Rosenbrock function:
a) execution time (μs), b) execution time $\times$ area

Figure 24.a and Figure 24.b compare the execution time and the product “area $\times$ execution time”, respectively, of the single BB-BC engine using HDL operators or CoreGen IP cores versus population size (N) for the Sphere function assuming: MCF or CCE function and number of iterations (M) 200. The arithmetic circuits’ latencies are $T_A=1$, $T_M=1$ (HDL) or 6 (CoreGen), $T_D=68$, $T_R=30$, $T_c=1$ and $T_{F3}=2$ (HDL) or 7 (CoreGen) (for the Sphere function). Figure 24.a clearly demonstrates that: (a) the highly optimized multipliers (GenCore IPs) significantly improve the max operating frequency and consequently the system performance for both cost functions, and (b) the higher complexity of the CCE functions slightly affects the execution time for the case of the CoreGen cores. On the other hand, the curves of Figure 24.b show that combining the CoreGen IPs and the MCF function achieves the best tradeoff between area and performance.



(a)



(b)

Figure 24. Comparison between HDL operators and CoreGen IP cores for the Sphere function:
a) execution time (μs), b) execution time $\times$ area

3.4.3 Comparison with an HLS-based accelerator

To demonstrate the advantages of the proposed well-optimized FPGA BB-BC architecture compared to a hardware accelerator generated by a state-of-the-art high-level synthesis (HLS) design flow, we utilized the AMD/Xilinx Vivado HLS tool to generate a Register Transfer Level (RTL) design from the original C++ code of the BB-BC algorithm. Our aim in this experiment was twofold: a) to identify whether specific features of the BB-BC algorithm are hard for an HLS tool to infer and accelerate and b) to determine the steps of the algorithm that can be pipelined and parallelized by an HLS flow.

We first developed a baseline C++ software implementation directly based on the operations of the BB-BC algorithms. Various code alterations were performed in this baseline software

program to ensure synthesizability and enable latency and throughput optimizations. More specifically:

- The C++ pseudo-random number generator was replaced by a linear feedback shift register (LSFR) C++ routine: the `rand()` function used in the original C code is not synthesizable, and for that reason, it was replaced by a C++ routine implementing an LFSR-based number generator.
- The float data types were altered into fixed-point types: to reduce the complexity of the arithmetic circuits and the required programmable resources (i.e., to avoid the use of complex floating-point arithmetic units), we changed the data types for the representation of the candidate features, the fitness function and the CoM from float to fixed-point representation (32-bit). The precision of the fixed-point representation is programmable to guarantee the minimum accuracy required by the target application. In the following experiments, the fixed-point precision is 16 bits for the decimal part and 16 bits for the fractional part.
- The pointer-linked array structures were replaced by fixed-length arrays: the pointer-based data structures are too complex to be implemented by the state-of-the-art HLS tools [87]. Transforming the pointer-linked array structures into fixed-length arrays enabled us to apply performance mechanisms, such as array partitioning, pipelining and unrolling, to improve system performance, as described below.

Next, we guided the RTL generation using manually invoked synthesis directives. We applied several directives to the functions, loops and arrays of the baseline software, such as `INLINE`, `PIPELINE`, `UNROLL` and `ARRAY PARTITION`. After several trials and performance analyses, we concluded on an optimized RTL design. Despite our efforts, we were unable to further optimize the design using synthesis directives.

Table 11 presents the area utilization (LUTs, FFs, DSP slices) of five different BB-BC implementations for the minimal cost function (MCF) and the complex CoM equation (CCE) for the Rosenbrock fitness function: i) non-optimized HLS-based BB-BC accelerator (baseline software code), ii) optimized HLS-based BB-BC version-1 (with `PIPELINE` directives), iii) optimized HLS-based BB-BC version-2 (with `PIPELINE` and `UNROLL` directives), iv) proposed

single engine, and v) proposed dual engine. The AMD/Xilinx Kintex-7 XC7K325T device (Xilinx KC705 development board) was chosen as the target device for these experiments, as the AMD/Xilinx Virtex-5 device previously used for evaluating the proposed architecture is not supported by the Vivado HLS tool.

Table 11. Area comparison between the HLS-based and the proposed FPGA BB-BC accelerators for different cost functions

Cost Function		Non Optimized	Optimized V1	Optimized V2	Proposed Single	Proposed Dual
MCF	DSP48E	30	33	49	24	48
	FF	1822	2999	4485	1243	1346
	LUT	3563	5144	25340	1457	2795
CCE	DSP48E	50	53	86	32	64
	FF	3026	4645	8586	14566	15790
	LUT	6258	8470	31655	6631	9360

Table 12 presents the performance of the four different implementations in terms of latency and clock frequency. The performance analysis has been conducted assuming a population size (N) 128 and a number of iterations (M) 256.

Table 12. Performance comparison between the HLS-based and the proposed FPGA BB-BC accelerators for different cost functions

Cost Function		Non Optimized	Optimized V1	Optimized V2	Proposed Single	Proposed Dual
MCF	Latency	1017345	73473	59649	40192	23808
	Clock period (ns)	8.43	8.43	8.52	2.17	2.72
CCE	Latency	1556737	92417	61697	66816	50688
	Clock period (ns)	8.43	8.43	8.52	2.17	2.72

The experimental results showed that the HLS flow can significantly shorten the design cycle of the hardware accelerator, but cannot achieve the performance levels of the manually optimized BB-BC accelerator. An in-depth performance analysis of the HLS-based RTL design revealed that although the different stages of the algorithm are efficiently pipelined and parallelized, the three separate pipelines (i.e., RandGen, Fitness and CalcCoM) cannot be merged into an integrated pipelined scheme due to interdependencies. These dependencies, resolved by the carefully optimized design of the proposed BB-BC architecture, are the main limitations to further performance improvement in the HLS design flow.

3.4.4 Design optimization considerations

To demonstrate the efficiency of the approximation method presented in Section 3.3.4, we implemented the complex Ackley fitness function using the Vivado HLS design flow. We implemented and compared the original Ackley fitness function and the ANFIS-based approximated Ackley function in the AMD/Xilinx Kintex-7 XC7K325T device. Table 13 presents the synthesis report for the two functions. The experimental results proved the efficiency of the proposed approach: the ANFIS-based approximation of the Ackley function requires significantly fewer reconfigurable resources and shorter latency. Note that we did not devote much effort to the well-optimized design of the ANFIS-based function and we relied on the performance optimizations of the HLS tool. However, someone can adopt an existing optimized ANFIS implementation, such as the one proposed in [88], to improve the system performance.

Table 13. Synthesis report for the original Ackley and the ANFIS-based Ackley functions

Function	DSP48E	FF	LUT	Latency		Clock period (ns)
				MIN	MAX	
Original Ackley	86	10828	45882	300	848	8.68
ANFIS Ackley	192	3826	5816	41	41	8.65

3.4.5 Comparison with s/w and GPU-based approaches

To demonstrate the speedup achieved by the proposed FPGA accelerator, we also compared the FPGA BB-BC architecture with a software version of the BB-BC (written in the C language) and a CUDA program running on a GPU (Graphics Processing Unit) device (FPGA and GPU technology are competing to provide acceleration platforms for compute-intensive applications). Table 14 compares the execution time (in ms) between the proposed single BB-BC engine (assuming CoreGen building blocks), the C program and the CUDA version of the BB-BC algorithm for the Rosenbrock function for both CoM functions (MCF and CCE), various population sizes (N) and numbers of generations (M) equal to 500. The C program runs on an Intel Core i7-4510U CPU at 2 GHz, while the CUDA version runs on an NVidia GeForce 840M board. The experimental results showed that the proposed approach outperforms GPU-based acceleration by a large margin. The CUDA approach achieves a speedup compared with the serial program (in C) only for large population sizes. For population sizes up to 1024, the problem does not provide sufficient parallelism to be well-suited to the massive

parallelism of GPU devices. On the contrary, the proposed FPGA accelerator achieves a speedup of around 10X for large population sizes (e.g., 8192) for both the MCF and CCE functions compared to the CUDA program and very high speedups for smaller population sizes. Notice that the CCE function's complexity imposes significant overhead on the GPU execution time, which is due to the low parallelism of the reduction function of the CoM calculation, while the latter is handled more efficiently by the pipelined engine of the proposed approach.

Table 14. Execution time of the C-program, CUDA version and FPGA approach of the BB-BC algorithm for the Rosenbrock function for $M=500$ (in ms)

N	MCF			CCE		
	C	CUDA	FPGA	C	CUDA	FPGA
64	8.11	24.20	0.15	8.31	31.12	0.31
256	17.11	26.54	0.45	16.47	34.38	0.62
1024	45.44	50.39	1.68	48.16	59.85	1.84
8192	318.43	119.76	13.08	335.54	136.07	13.25

To further evaluate the performance of the proposed approach in a real-time embedded environment, we developed C versions of the BB-BC algorithm and a typical GA algorithm on a BeagleBone Black board with an ARM Cortex-A8 processor running at 1 GHz. Table 15 compares the execution time between the proposed single and dual BB-BC engines (assuming CCE function and CoreGen building blocks) and the embedded s/w implementations of the GA and BB-BC algorithms for the Rosenbrock function for different numbers of generations (M) and population size (N) equal to 256. The single engine achieves average speedups of 157x and 96x, while the dual engine achieves average speedups of 198x and 121x against the GA and BB-BC embedded s/w programs, respectively. The results prove both the superiority of the proposed BB-BC algorithm over GAs for real-time optimization problems and the efficiency of the proposed FPGA accelerator.

M	GA s/w	BB-BC s/w	FPGA: Single-CCE			FPGA: Dual-CCE		
			Exec. time	Speedup vs GA	Speedup vs BB-BC	Exec. time	Speedup vs GA	Speedup vs BB-BC
100	21.33	12.25	0.12	178	99	0.10	218	125
200	39.17	23.50	0.25	157	95	0.20	201	120
500	92.00	58.75	0.62	148	95	0.49	188	120
1000	181.00	116.63	1.24	146	94	0.98	185	119

3.4.6 Case study: image search problem

The GAs have been used in the past to solve various image analysis problems, such as feature extraction in geoscience and remote sensing [89], template matching for eye tracking [90], video segmentation [91] and medical image segmentation [92],[93]. Although GAs may not be suitable for real-time image and video processing applications due to their large search space, they can be accelerated using FPGA or GPU technology, thereby providing an effective optimization tool for such applications [94]. In this chapter, we utilize the BB-BC optimization algorithm to identify the darkest area/pixel in an image. The problem of finding the darkest pixel arises in several real-time image analysis applications, such as tissue image segmentation [100] and iris localization [95]. Here, we searched for the darkest area of an image to locate the region where the two images differ. Image difference detection and location are commonly used for quality control in the manufacturing industry [96]: images of the product are recorded at consecutive time points during the manufacturing process and compared with the “golden” image. Let’s assume two grayscale images of dimensions $m \times n$: the golden image G and the tested image T (assumed geometrically matched). The steps of our image difference detection application are described in the following:

1. Subtract the two images, $D = G - T$
2. Convert the grayscale image D to a binary image BD
3. Dilate the binary image BD to expand the difference area
4. Smooth the image using a 2D Gaussian filter
5. Locate the darkest area, i.e., the area with the darkest pixels, of the smoothed image (this is the area where the G and T images differ).

The problem of searching the darkest area is a maximization problem that we solve herein using the BB-BC algorithm. The population includes pixels of the image $p(i, j)$, $i = 0 \dots m - 1$, $j = 0 \dots n - 1$, and the fitness function is the gray intensity of the pixel. The result (CoM) is a pixel (coordinates) near the center of the darkest area. We implemented the entire image analysis application (steps 1 to 5) in MATLAB and validated it across several pairs of images. The image is 600x600 pixels, the range of values for x and y is [1..600], and the fitness function should be as close to zero as possible. Some of the most essential BB-BC parameters are the particles (any possible solution) and the iterations. In our case study, we used 600

particles and 10 iterations. We modified all versions of the BB-BC algorithm evaluated in Section 3.4.5 (C program, CUDA program and FPGA BB-BC) to integrate the darkest pixel/area search problem (Step 5 of the image analysis application).

Table 15 compares the execution time (in ms) between the proposed single BB-BC engine (assuming CoreGen building blocks), the C program and the CUDA version of the BB-BC algorithm for the darkest pixel problem for both CoM functions (MCF and CCE), various population sizes (N), and numbers of generations (M) equal to 500. The C program ran on an Intel Core i7-4510U CPU at 2GHz, while the CUDA version ran on an NVidia GeForce 840M board. The experimental results are similar to those presented in Table 14 for the Rosenbrock function. The CUDA approach achieves speedup compared with the C program only for large population sizes. In contrast, the proposed FPGA accelerator achieves a speedup of around 11X for large population sizes compared to the CUDA program and very high speedups for smaller population sizes.

Note that this image analysis problem may be conditioned by real-time requirements depending on the target application; for example, in the case of a machine vision system of a manufacturing process that automatically discards defective pieces from the production flow, the image difference search problem should check the tested images within a specified period that may range between a few ms up to several seconds/minutes depending on the product image capture frequency. In such cases, by accelerating the BB-BC algorithm, the proposed approach may provide the means to meet the real-time requirements of the image analysis application.

Table 15. Execution time of the C-program, CUDA version and FPGA approach of the BB-BC algorithm for the darkest area problem for $M=500$ (in ms)

N	MCF			CCE		
	C	CUDA	FPGA	C	CUDA	FPGA
64	7.32	21.56	0.12	9.61	28.43	0.29
256	13.15	23.47	0.43	15.29	31.66	0.59
1024	37.63	45.29	1.65	42.61	52.27	1.82
8192	225.14	103.47	13.06	244.19	125.91	13.22

3.5 Summary

In this chapter, we studied the FPGA-based acceleration of a low-complexity optimization algorithm, namely BB-BC. We proposed a novel, parameterized, pipelined FPGA design of the

BB-BC algorithm and a parallel scheme that integrates multiple BB-BC engines to improve performance. CoM calculation significantly affects the performance of the algorithm, so a high-speed reduction circuit was implemented to reduce latency. We compared the proposed FPGA BB-BC acceleration platform with its software counterpart (C program) and an optimized CUDA program running on a GPU for different fitness functions. We demonstrated the benefits of the proposed well-optimized accelerator compared to an HLS-based accelerator generated by a state-of-the-art HLS tool. We evaluated the efficiency of the ANFIS approximation method on a complex fitness function. We also assessed the applicability of our approach on an image analysis application, where the BB-BC algorithm is used to solve an optimization problem. The experimental results showed that the proposed FPGA BB-BC architecture achieves a significantly higher speedup than the competitive GPU technology across all cases. Given the high-convergence speed of the BB-BC algorithm compared to typical GAs, the proposed FPGA accelerator provides an ideal platform for large-scale adaptive or latency-sensitive optimization problems.

CHAPTER 4

Efficient Implementations of PID Controllers for HVAC Systems

4 Efficient Implementations of PID Controllers for HVAC Systems

This chapter introduces novel techniques based on the BB-BC optimization algorithm for tuning the PID and FOPID controllers in second-order HVAC systems. The BB-BC algorithm, when implemented on an FPGA, significantly accelerates the optimization process, resulting in rapid convergence and improved system performance. The BB-BC-based tuning approach was validated for the case of the PID controller using the FPGA-In-the-Loop (FIL) technique, demonstrating superior optimization accuracy and tuning speed compared to other optimization methods, such as GA and PSO. Additionally, the experimental results showed that adopting an FPGA-based implementation in a FOPID controller improves dynamic performance and robustness, providing substantial reductions in oscillations, control signal output, and error. The contents of this chapter are mostly based on the publications [P2] and [P4].

4.1 Introduction

Although novel and practical theories and design methodologies are continually being developed in the field of digital control, PID controllers remain among the most widely adopted solutions for solving control problems in both academic and industrial sectors. The tuning of the parameters of the PID controllers has been proven to be a challenging task and several approaches have been investigated [97, 98], using analytical methods [99], heuristic methods, such as the Ziegler–Nichols (Z–N) rule, frequency response methods [100], artificial intelligence, such as fuzzy logic [101] and iterative learning approaches [102]. Evolutionary Algorithms (EAs), such as the Genetic Algorithm (GA) [103] and Particle Swarm Optimization (PSO) [104], used in intelligent control applications, have also been proposed as efficient solutions for tuning the PID parameters. The simplicity and convergence speed of EAs force the controller to quickly calculate the optimal output.

In this thesis, we first studied the use of an EA, named Big Bang-Big Crunch (BB-BC), to optimize PID controller gains. The BB-BC optimization algorithm minimizes performance control measures, including integral time absolute error (*ITAE*), integral absolute error (*IAE*), integral square error (*ISE*), and integral time square error (*ITSE*), while also reducing the system overshoot. The adoption of the BB-BC algorithm is motivated by its advantages: (a)

it converges quickly due to its iterative approach, which is crucial for PID real-time tuning where latency is a concern, (b) it can deal with various performance indices (e.g., ITAE, IAE, ISE, ITSE) to minimize control errors; this adaptability allows designers to meet diverse control system requirements, and (c) it has demonstrated superior performance in various engineering applications, such as structural design and control systems. Additionally, the authors in [105, 106] have also proposed adopting the BB-BC algorithm for tuning PID/FOPID controllers used in load-frequency control or for higher-order oscillatory systems.

We evaluated the efficiency of the BB-BC algorithm for tuning the PID controller parameters of a second-order HVAC system. In an HVAC system, the supply air pressure is regulated by the speed of a supply air fan. Increasing the fan speed increases the supply air pressure and vice versa. The dynamics from the fan variable speed drive to the supply air pressure can be modeled as a second-order plus dead time [38]. In real applications, however, where both fans and dampers exhibit nonlinear behaviour at different operating points, even a well-tuned PID controller may not achieve the desired performance across all set points and process variations. Thus, we investigated how fast and accurately the BB-BC algorithm can tune the PID controller parameters in this HVAC system.

It is worth noting that embedded computers, built around programmable microcontrollers, are the dominant computing platform for implementing PID controllers in industrial environments. However, field programmable hardware technology has emerged as an alternative for controlling industrial applications, as discussed in Chapter 3. Thus, several approaches have proposed the use of FPGA technology to implement PID controllers [107, 108] and/or optimize the PID controller parameters in real-time industrial applications, such as DC motor speed control systems [109]. For example, in [110] Kim implemented a fuzzy logic controller on an FPGA platform and validated it on a truck backer-upper control system. In [111], Khan et al. presented a runtime reconfigurable PID controller that can dynamically switch between controller variants, e.g., PI, PD, PI-PD, etc., as control requirements change. The inherent hardware parallelism and customization of the FPGA technology make it well-suited for executing complicated control algorithms in industrial applications, particularly where conventional computing platforms may not be able to meet the real-time requirements of the target applications [112].

Towards this end, and considering the real-time requirements of our HVAC system, this thesis proposes FPGA-based acceleration of the BB-BC optimization algorithm and the PID controller to speed up the parameter tuning process, and consequently, improve system performance.

We developed the RTL models of the BB-BC algorithm and the PID controller in VHDL and implemented them on an AMD/Xilinx Zynq-7000 FPGA device. The Xilinx Zybo (Zynq) development board [113], integrating the Zynq XC7Z010 FPGA device, was chosen for demonstration purposes. Moreover, we used the FPGA-In-the-Loop (FIL) technique to enable the emulation of the plant (HVAC) in MATLAB/Simulink environment in conjunction with the execution of the PID controller and the optimization process in the real hardware; this way, the FIL technique helped us to evaluate the entire system in higher abstraction layers [114]. To demonstrate the efficiency of the proposed approach, we conducted various experiments on a second-order system with a two-second time delay and compared it with GA and PSO optimization algorithms in terms of PID tuning and convergence speed. The experimental results showed that the FPGA BB-BC algorithm tunes the PID controller parameters $\sim 430X$ faster than the software version of the GA algorithm, $\sim 539X$ faster than the software version of the PSO algorithm, and $\sim 270X$ faster than the software version of the BB-BC algorithm. Furthermore, we tested the proposed approach with various fitness functions (*error*) for different numbers of iterations. We chose the *ITAE* metric because it offers several advantages, particularly for HVAC systems, such as sensitivity to transient behavior, improved stability, and settling time.

On the other hand, Fractional-Order PID (FOPID) controllers offer improved dynamic performance and system robustness, as discussed in Chapter 2. By adding two extra parameters, the integration order and the derivative order, to the three parameters of the PID controller (proportional, integral, and derivative), the system response specifications can be modeled more efficiently. In [115], De Castro, F.A., et al. compared the performance of fractional and integer PID controllers tuned by GAs and showed that the former significantly improves control signal stabilization. However, due to the larger number of control parameters, optimal tuning of FOPID controllers is a more complex task that often requires the use of effective optimization algorithms running alongside the control unit to dynamically calculate the

optimal parameter values for a specific target application [116]. In [117], Moreira, W.E.M. and C. Garcia recommended that enhancing the efficiency of the FOPID controller can be accomplished by integrating second-order models and using optimization algorithms. Despite the existence of several algorithms such as DE [118], GA [119], [120], and PSO [121],[122], for optimizing the performance of PID and FOPID controllers, the BB-BC algorithm has recently gained significant popularity [123].

Similar to PID controllers, FPGA-based computing platforms can also offer efficient implementations for FOPID controllers in terms of latency, connectivity, and energy efficiency. Thus, FPGA technology reinforces FOPID control systems, making them an optimal choice for real-time industrial control applications. For example, Jiang, C.X., et al. [124] proposed a generalized class of FO systems and presented effective FPGA-based implementations for the FO operators.

The FOPID controller, along with the BB-BC algorithm, can directly compensate for disturbances in the HVAC system caused by undesired nonlinearities, such as time delays and environmental variations, thereby improving system precision and stability. In this thesis, we propose the optimization of settings for a PID controller integrated into an HVAC system using the BB-BC algorithm. Many design parameters based on optimization significantly impact the heating and cooling of buildings [125]. Since the BB-BC algorithm generates new solutions by generating new candidates around the center of mass (CoM) of the best initial candidate at each iteration, it yields better convergence when combined with the FOPID controller. It is worth noting that this property of the BB-BC algorithm is not met in other optimization algorithms. Another benefit of using the BB-BC algorithm is that its intensification feature becomes more pronounced as the algorithm progresses, while its diversification is more pronounced at the beginning. This property of the BB-BC algorithm increases the likelihood of not getting stuck at local optima compared to other algorithms, such as GA or PSO. To evaluate the performance of the proposed FOPID-BB-BC approach, we compared it with GA and H_∞ algorithms in terms of time response, and the experimental results showed that it outperforms both.

To further emphasize the problem statement (Complexity of FOPID tuning, Effective optimization, and Real-time requirements and hardware limitations) and the objective formulation

(Optimize FOPID controller parameters, Minimize performance control measures, Accelerate tuning process based on hardware implementation, Demonstrate superiority of proposed approaches, and Handle various performance indices and system dynamics) through soft computing techniques, we extended our approach by employing four objective functions to optimize the parameters of the FOPID controller ($k_p, k_i, k_d, \lambda, \mu$). Compared to the conventional PID controller and the MPC, the FOPID controller offers additional degrees of freedom, enabling more precise and flexible control of the HVAC system. Through experiments, we showed that the FOPID-BB-BC controller outperforms alternative optimization-based controllers, including PSO-FOPID, GA-FOPID, and conventional PID controllers, with *ITAE* emerging as the most effective objective function, with *ITAE* emerging as the most effective objective function. Additionally, we examined the performance of two FOPID controllers, D-FOPID and GL-FOPID, and evaluated their effectiveness in controlling an HVAC system when their parameters are optimized using the BB-BC algorithm. The Discretized FOPID (D-FOPID) controller depends on the Tustin transform and the Power Series Expansion (PSE), while the GL-FOPID controller follows the Grünwald-Letnikov (GL) definition. Furthermore, the second use case highlights the effectiveness of the BB-BC algorithm in optimizing the performance of SOPTD systems. To analyze the robustness of the proposed approach, we conducted several experiments on an SOPTD system for different values of time delay θ and time constant τ . All experiments were performed under various objective functions (J_n) for different numbers of iterations, with the integral performance index (J) being a linear function of the *ITAE*. To compare the control performance of the proposed approach with previous studies, several error criteria, such as *ITAE*, *IAE*, *ITSE* and *ISE*, were examined, while two performance indices (J_n) were used to optimize the FOPID parameters.

Moreover, we implemented the control unit of the target HVAC system on an FPGA, including: (a) the FOPID controller to improve the system latency and energy efficiency, and (b) the BB-BC evolutionary algorithm to optimally tune its control parameters. Experiments on the FPGA implementation of the FOPID-BB-BC scheme showed that the FPGA-based solution effectively achieves the desired balance among performance, power consumption, and resource usage. To improve the effectiveness of the FPGA FOPID-BB-BC scheme, we applied various hardware design techniques, including pipelining, resource sharing, and clock

gating. Furthermore, the optimization features offered by synthesis and implementation tools, such as AMD/Xilinx Vivado, can yield optimal results for the intricate design requirements and constraints. Moreover, we integrated the D-FOPID and GL-FOPID controllers, along with the BB-BC algorithm, into the FPGA design to optimize HVAC system performance. We applied various optimization techniques to the FPGA design, which can handle the large storage requirements for calculating the PID integral and derivative parts. We also investigated the impact on system precision and hardware complexity of representing the values of the FO parameters (λ , μ) as 32-bit or 64-bit numbers. The entire system was described using VHDL language and implemented on an AMD/Xilinx Zynq UltraScale+ device. This design enables real-time adjustments to control parameters, step size, and constant system parameters without requiring changes to the hardware implementation. To the best of our knowledge, no prior work has studied the integration of an optimization algorithm, such as BB-BC, into the FOPID controller and implemented the system using FPGA technology to improve its performance.

4.2 Optimizing the PID controller of an HVAC with the BB-BC algorithm

This section presents a novel technique for fast tuning the PID controller parameters of a second-order HVAC system. HVAC systems vary significantly in size, control functions and energy consumption. The optimal design and power efficiency of an HVAC system depend on how quickly the integrated controller, e.g., a PID controller, is adapted to changes in environmental conditions. In this thesis, we relied on BB-BC, a fast-converging evolution algorithm, to achieve high tuning speed. We implemented the BB-BC algorithm, along with a PID controller, in an FPGA device, to further accelerate the optimization process. Moreover, we utilized the FIL technique to connect the FPGA board (i.e., the PID and BB-BC subsystems) with the plant (i.e., MATLAB/Simulink models of HVAC) to emulate and evaluate the entire system. The experimental results demonstrated the efficiency of the proposed technique in terms of optimization accuracy and convergence speed compared to other optimization approaches for tuning PID parameters: the SW implementation of the BB-BC, GA, and PSO.

4.2.1 Tuning the PID controller parameters using the BB-BC algorithm

Evolutionary algorithms can be applied to tune the PID controller gains and parameters, ensuring optimal control performance under stable operating conditions. In this thesis, we employed the BB-BC algorithm to tune offline the PID parameters (kp , ki , kd) in an HVAC model. Figure 12 shows how the control unit, consisting of the BB-BC unit and the PID controller, is connected to the HVAC system. In the design methodology of a PID controller, one of the most critical performance criteria is the difference (*error*) between the plant output and the set point signal. Using this error criterion as the fitness function of the optimization algorithm results in a small overshoot with a long settling time. In general, fitness functions are based on error equations. The following four equations, *ITAE*, *ITSE*, *IAE*, and *ISE* indicate the most commonly used fitness functions:

$$J_1 = ITAE = \int t \cdot |e(t)| dt \quad (4.1)$$

$$J_2 = IAE = \int |e(t)| \cdot dt \quad (4.2)$$

$$J_3 = ISE = \int e(t)^2 \cdot dt \quad (4.3)$$

$$J_4 = ITSE = \int t \cdot [e(t)^2] \cdot dt \quad (4.4)$$

Fitness functions are not actually limited to the above equations. Engineers can provide custom fitness functions depending on the target design and control system. The overall performance (convergence speed and optimization accuracy) of evolutionary algorithms depends on the fitness function adopted to monitor the optimization search. To choose the optimal fitness function, i.e., the objective function of the BB-BC algorithm, we conducted a series of experiments for the four typical error equations. We computed the four error formulas (*ITAE*, *ITSE*, *IAE*, and *ISE*) for increasing iteration numbers. As the number of iterations increases from 100 to 1000, *ITAE* and *IAE* exhibit a clear downward trend, indicating improvements in transient response and error minimization over time. Specifically, *ITAE* improves from 533 to 396 and *IAE* from 310 to 286, suggesting enhanced system performance and faster settling. In the objective function used, the added term of the control output $u * u^T$ (where u^T is the transpose of u) seeks to reduce the magnitude of the control signal. When the objective function includes the control signal term (u), the algorithm aims to minimize it. Therefore, the term $u * u^T$ is equivalent to squaring every element in u . The objective

function that achieved the best performance is the *ITAE* with control signal (u), as reported below in Eq. (4.8). The results of these experiments are shown in Table 16.

Table 16. Comparison of different errors, *ITAE*, *ITSE*, *IAE*, *ISE* as performance index

Iteration No	ITAE	ITSE	IAE	ISE
100	5.33e+02	2.47e+03	3.10e+02	2.25e+03
200	4.85e+02	2.58e+03	2.99e+02	2.24e+03
500	4.40e+02	2.62e+03	3.46e+02	2.34e+03
1000	3.96e+02	2.20e+03	2.86e+02	2.45e+03

Figure 25 presents the convergence and time response of the PID controller using four objective functions (*ITAE*, *ITSE*, *IAE*, *ISE*) for parameters' tuning with different numbers of iterations (100, 200, 500, and 1000). The curves show how the fitness function response (error) changes during the optimization process of the PID controller using the BB-BC algorithm. Among the four traditional objective functions, the *ITAE* provides the minimum settling time, peak overshoot, and rise time. As mentioned above, any objective function can be adopted to tune the PID parameters. However, it is hard to reduce the rise time without increasing overshoot. By reducing the rise time, the system attempts to follow the set-point faster, increasing inertia and the risk of overshoot. As shown in Figure 25, although the *ITAE* response signal declines faster, its convergence velocity is better than that of the other methods. Note that the integral absolute error (*IAE*) provides the best convergence, but its time response (overshoot, rise time, and settling time) is not optimal. Thus, in all cases (number of iterations), the *ITAE* provides the best solution. Overall, the results from Table 16, and Figure 25 indicate that increased iterations enhance system accuracy, particularly in terms of absolute error criteria.

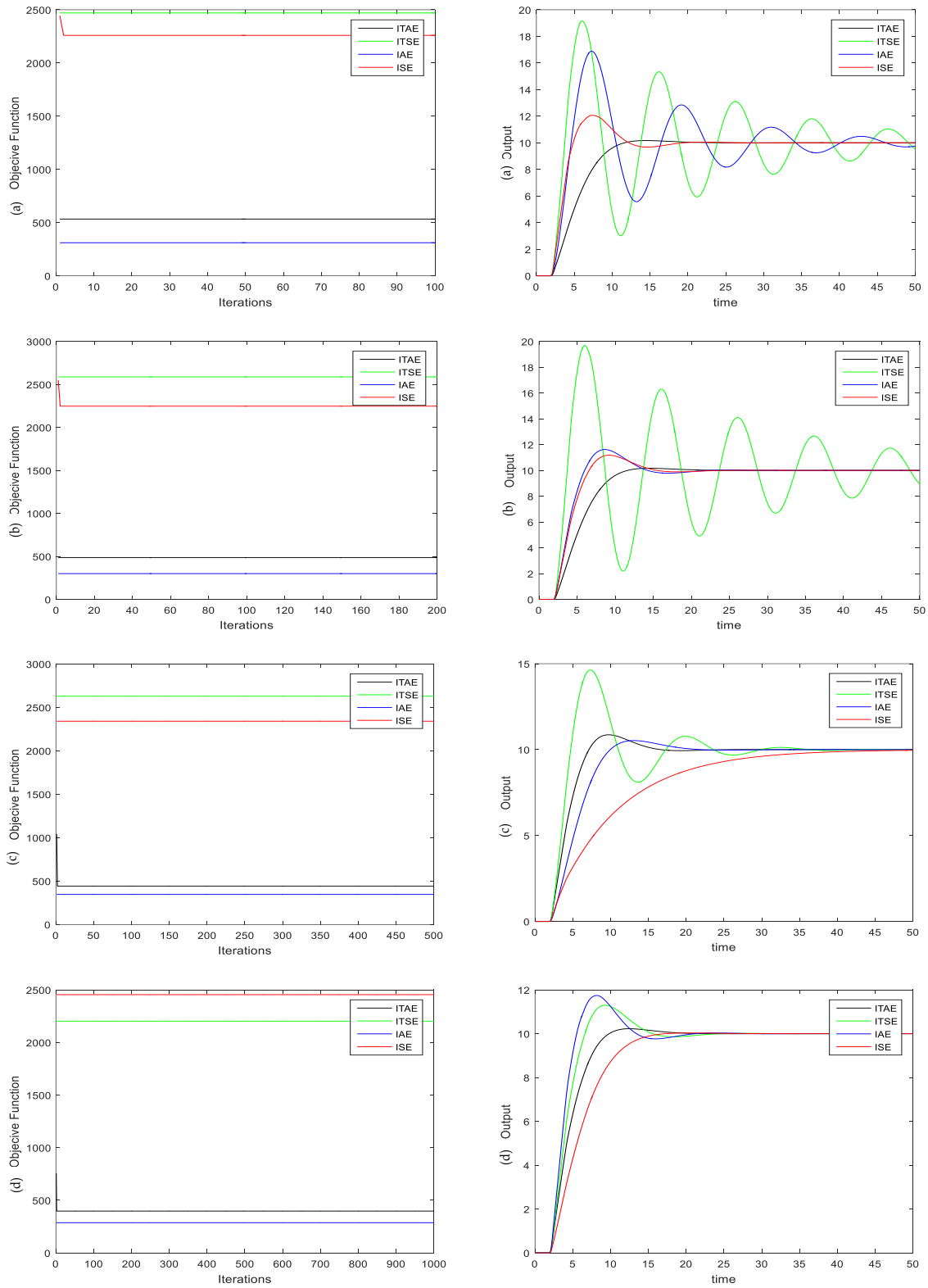


Figure 25. Convergence and time response in seconds of the objective functions (*ITAE*, *ITSE*, *IAE*, *ISE*) for different numbers of iterations: a) 100, b) 200, c) 500 and d) 1000 iterations

4.2.2 Implementing the PID Controller using FPGA

The FPGA technology is well-suited for industrial control applications due to its low cost, flexibility, energy efficiency, and real-time performance enabled by the hardware parallelism. In this section, we present the FPGA implementation of the PID controller along with the BB-BC algorithm that calculates and optimizes the PID control parameters (k_p , k_i , k_d) to accelerate the control system process.

Figure 26 shows the block diagram of a typical PID controller hardware (FPGA) architecture based on the (differential) Eq. (2.9) and the parameters of Table 1.

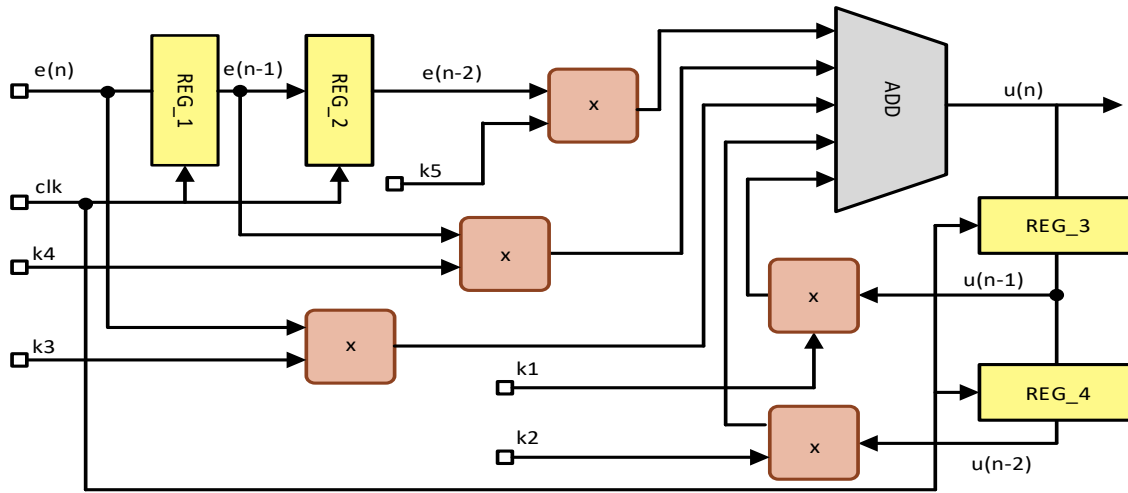


Figure 26. Block diagram of a typical PID controller FPGA architecture

The PID controller diagram consists of four registers that store the current and previous error values, i.e., $e(n-1)$, $e(n-2)$, $u(n-1)$, and $u(n-2)$. Note that the PID controller uses a five-stage pipeline structure to increase the clock frequency. Five arithmetic operators (i.e., multipliers) are used to multiply the proportional part, derivative part and integral part with the corresponding parameters (k_3 , k_4 , and k_5 , respectively) and the output signal with k_1 and k_2 as described in Equation (2.9), while the adder at the final stage sums all the factors. Embedded DSP slices in the FPGA device are used to implement the arithmetic operators, improving design area and speed.

4.2.3 Implementing the BB-BC algorithm using FPGA

In this subsection, we describe the FPGA design architecture of the BB-BC algorithm. It is worth noting that we chose to develop a new, area-optimized FPGA design for the BB-BC

algorithm tailored to the needs of this system, rather than using the high-performance FPGA architecture described in Section 3. The latter FPGA architecture incorporates parallel, latency-optimized pipelined engines to accelerate the execution of the BB-BC algorithm, making it particularly suitable for control applications with strict real-time requirements. However, this performance optimization has been achieved at the expense of extra hardware area and power consumption, and thus, the latency-optimized FPGA architecture of Section 3 is recommended only for time-sensitive control applications.

Figure 27 depicts the top-level design diagram of the area-optimized FPGA BB-BC architecture consisting of the following blocks (for reader convenience, we shortly describe the operations of the BB-BC algorithms analyzed with more details in Chapters 2 and 3):

- **Random Number Generation:** It generates random numbers, either to form an initial population (iteration 0) or to create new candidates around the CoM (iteration > 0). In each iteration, the module generates N candidates sequentially, where N is the population size, and each candidate consists of L random numbers, where L is the number of features of each candidate. The random numbers generated in each iteration are stored in RAM for later processing in the calculation of the CoM.
- **Objective Function:** After the random number generation, this module calculates the objective function (f) for each candidate ($x(i)$) using the *ITAE* equation, which is also stored in the RAM. Both the random number generation and objective function modules have been implemented following a simple, serial processing model, where a single element is processed per clock cycle, resulting in a low-cost FPGA circuit. Note that both modules can be parallelized (i.e., generating random numbers and calculating the objective functions concurrently) and pipelined to improve performance at the cost of extra area, as proposed in Chapter 3.
- **Calculate CoM:** It reads the candidates and their objective function values from the RAM and calculates the CoM, which is forwarded to the random number generation module to generate the population in the next iteration cycle.
- **Evaluate Function:** It reads the objective function values of all candidate solutions from RAM, selects the candidate with the best fitness, and compares it to the previous output signal while the FPGA-BB-BC algorithm controls the HVAC model.

- Control Unit: It orchestrates the operation of the other modules and repeats the BB-BC steps until the stopping criteria are met.
- Solution: It includes a register that stores the outcome of the BB-BC algorithm.

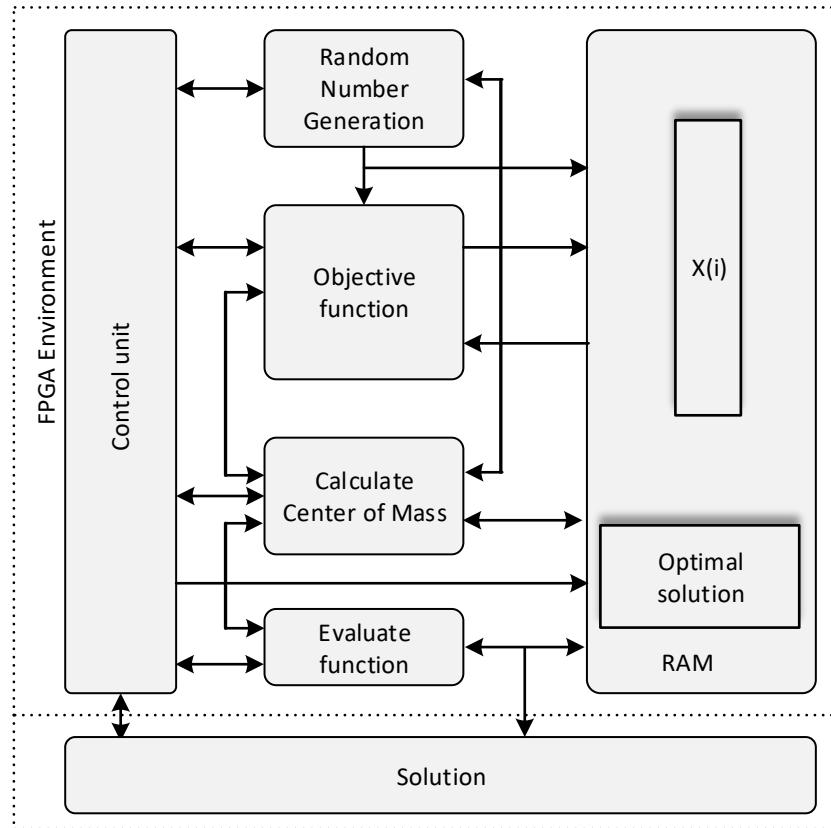


Figure 27. Top-level diagram of the PID-BB-BC FPGA architecture

Table 17 presents the device utilization summary (LUTs, Registers, DSP48Es, and RAM/FIFO modules) of the PID-BB-BC scheme, consisting of the BB-BC components and the PID controller presented in Section 2.2, on an AMD/Xilinx Zynq XC7Z010 FPGA device. This BB-BC architecture emphasizes area optimization, making it more suitable for resource-constrained FPGA implementations. This shows that the BB-BC architecture prioritizes energy efficiency, whereas the design presented in Chapter 3 prioritizes performance.

Table 17. FPGA resource utilization summary of the PID-BBBC control unit

Unit	LUTs	Registers	DSP48Es	RAM/FIFO
PID	57	80	--	--
BB-BC	7994	273	32	2
PID-BB-BC	16998	306	54	2

4.2.4 FPGA-In-The-Loop (FIL)

MATLAB/Simulink is a high-level language and interactive environment that is popular among system and algorithm designers for various applications and scientific areas. To accommodate developers to interface with real hardware (e.g., components implemented in FPGA devices), *MathWorks®* introduced the FPGA-In-The-Loop (FIL) concept. Based on this idea, data can be exchanged between the *MATLAB/Simulink* modules and the FPGA device. The FIL scheme can be utilized for multiple tasks during the design, evaluation, and verification phases. For example, it can be used to configure and tune FPGA design parameters at runtime, such as tuning data pipe gains and modifying filter parameters [126], or to monitor hardware components by reading specific status registers. Another powerful capability of the FIL approach is that it allows the engineers to generate complex stimulus datasets on the host PC and download them to the FPGA target in real-time. Using the FIL approach and connecting the hardware target via a standard protocol interface (e.g., Ethernet or JTAG), one can significantly improve simulation runtime compared to cycle-accurate host simulation.

The proposed approach, in conjunction with the FIL technique, achieves a speedup of around 253X compared to the MATLAB 2015b software simulation. Moreover, when design complexity increases, the performance advantages of the FIL methodology over software simulation will be more evident. Figure 28 presents the structure of the FIL acceleration scheme used in our system, where MATLAB/Simulink provides a working environment for the user.

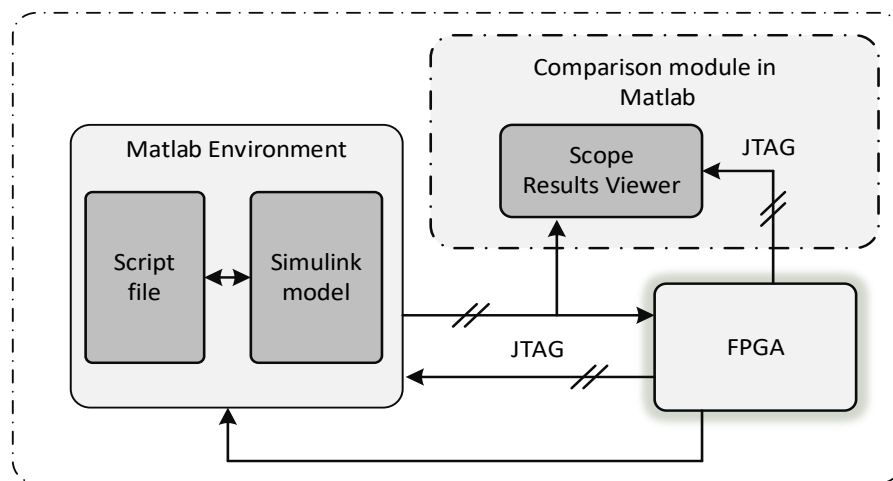


Figure 28. The FIL acceleration scheme

4.2.5 HVAC system architecture

Figure 29 depicts the HVAC system architecture, including the FPGA device and the HVAC model implemented in the MATLAB environment. The FPGA design, which comprises the PID controller and the BB-BC algorithm, is written in VHDL hardware description language and implemented on an AMD/Xilinx development platform. The processing steps of the proposed architecture are briefly described below.

1. The FPGA BB-BC module generates a set of random PID parameters (kp , ki , kd), to form either the initial candidate or the new candidate around the CoM in the next iteration cycles.
2. The FPGA PID controller module receives the values of the parameters and controls the HVAC plant.
3. The FPGA BB-BC module calculates the fitness functions for this candidate (i.e., PID parameters) using the *ITAE* equation and then calculates the new CoM.
4. The HVAC model runs for a short period.
5. The evaluate function checks the system output and decides whether the BB-BC module must re-run to improve system performance, i.e., when the system has been affected by external disturbances that cause the output to exceed some predefined thresholds. Note that we use the HVAC plant as part of the FPGA BB-BC module to generate the initial optimization parameters (kp , ki , kd) of the PID controller for the HVAC model in the first run.

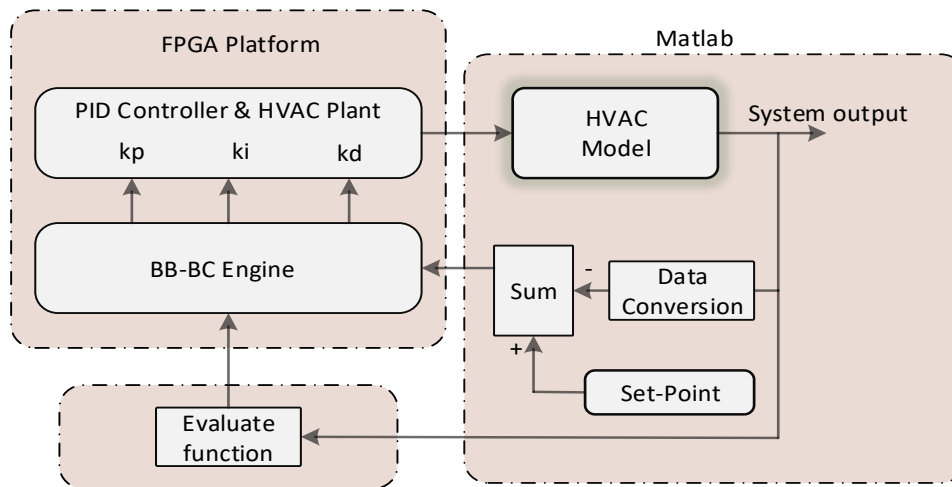


Figure 29. Proposed system architecture

Given that evolutionary algorithms may consume a significant amount of time to reach the optimal solution, even when the number of iterations is small, they are not suitable for latency-sensitive real-time applications. However, accelerating the evolutionary algorithm using an FPGA-based platform, as proposed in this thesis, can reduce the execution time and achieve convergence to the optimal solution under specific timing constraints. Therefore, the FPGA-based BB-BC algorithm can be executed significantly faster, enabling the optimization algorithm to meet the real-time requirements of the target industrial application. The total execution time of the *MATLAB/Simulink* model (T_{MS}) incorporates the *MATLAB* processing time, the FPGA execution time and the time required for the data exchange between the host PC (*MATLAB* model) and the FPGA board and is given by the equation:

$$T_{MS} = T_{MP} + T_{M2F} + T_{FPGA} + T_{F2M} \quad (4.5)$$

where T_{MS} is the total execution time of the *MATLAB/Simulink* model, T_{MP} is the *MATLAB* processing time, T_{M2F} is the time required for transferring data from the *MATLAB* model to the FPGA platform, T_{FPGA} is the FPGA processing time, and T_{F2M} is the time required for copying data from the FPGA platform to the *MATLAB/Simulink* environment.

Assuming population size N and number of iterations M , the total execution time T_{FPGA} of the proposed FPGA BB-BC circuit is given by the equation:

$$T_{FPGA} = M \times N \times (T_{CG} + T_{FC} + T_{COM} + T_{PID+Plant}) \quad (4.6)$$

where T_{CG} , T_{FC} , T_{COM} are the latencies of the FPGA BB-BC stages, i.e., candidate generation, fitness function calculation, and CoM calculation, respectively and $T_{PID+Plant}$ is the latency of the FPGA PID controller and the plant. Figure 30 shows the top-level diagram of our *MATLAB/Simulink* model, which includes the FPGA BB-BC/PID module (using the FIL concept), the HVAC module, an analog-to-digital converter to feed the digital error value into the FPGA BB-BC module, a digital-to-analog converter to provide the PID controller output to the HVAC model, an adder to calculate the error and the set point.

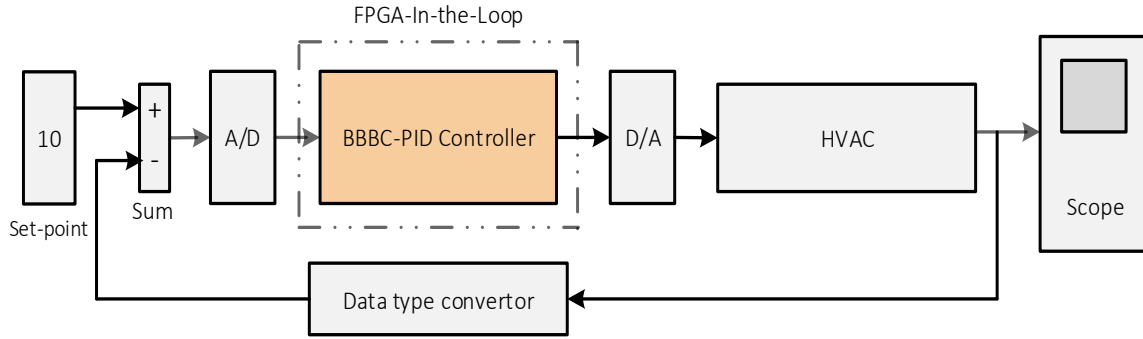


Figure 30. Block diagram of our MATLAB/Simulink model

4.2.6 Experimental evaluation of the proposed PID-HVAC system

4.2.6.1 HVAC system experimental setup

The HVAC dynamics system, which includes the control signal output $u(t)$ from the fan variable-speed drive to the supply air pressure, can be modeled as a second-order plus time-delay system. The process response is shown in Figure 31. For our experiments, we adopted the HVAC model proposed by Bi, Q., et al., in [127]. In [127], the authors derived this HVAC model from experimental tests on supply air pressure and room pressure in an HVAC pilot. Thus, our target system is an HVAC model using the transfer function:

$$G(s) = \frac{e^{-2s}}{0.12s^2 + 1.33s + 1.24} \quad (4.7)$$

The system response performance criterion used as the objective function in our optimization algorithm is the *ITAE* plus the control signal (u) given by the formula:

$$ITAE = \int t. |e(t)| dt + u^T * u \quad (4.8)$$

Note that the *ITAE* is widely used in control processing because it is simple to implement and natural to define the energy of a signal that is symmetric and differentiable. The efficiency of the proposed approach was demonstrated through a set of experiments conducted on a second-order system with a two-second delay. Through our case study (HVAC) simulation experiments, we observed that when the BB-BC runs and the error value is calculated at the beginning, the plant output becomes equal to the set point. On the contrary, when the optimal values for the K_p , K_i , and K_d parameters are used to calculate the current plant output, the plant output equals the set point when the error reaches near zero, hence, the plant starts

running at the set point. The final outputs of the proposed design are the optimized PID controller parameters (k_p, k_i, k_d) used in the HVAC control system. The experiments showed that our approach considerably reduces overshoot without compromising latency sensitivity, compared to the GA and PSO algorithms, which do not account for the desired overshoot, even when the error and response time are reduced [128].

4.2.6.2 HVAC system performance and convergence

To evaluate the robustness of the proposed approach against other optimization methods, we simulated our HVAC model running: i) BB-BC algorithm in MATLAB software, ii) BB-BC in FPGA (our approach), iii) PSO method in MATLAB software and iv) GA in MATLAB software. The simulations were performed for different numbers of iterations ($M = 100, 200, 500, 1000$), and the results are illustrated in Figure 31.a. The BB-BC and BB-BC-FPGA methods consistently demonstrate faster rise times and quicker settling to the desired output, with minimal overshoot, compared to the PSO- and GA-based methods. Their response curves are very similar, indicating that the FPGA implementation maintains the performance of the BB-BC algorithm. Their responses are already sufficient even at 100 iterations (Figure 31.a), and they remain stable and performant as the number of iterations increases to 1000 (Figure 31.d). The PSO method shows a slower rise time and more pronounced oscillations before settling than the BB-BC methods. The GA method exhibits the slowest response, with a significantly longer rise time and prolonged settling behavior, even with a large number of iterations.

The BB-BC and BB-BC-FPGA methods achieve superior time responses, producing faster, more stable outputs with minimal overshoot across all tested iteration counts. This demonstrates their effectiveness in tuning the PID controller for optimal transient performance.

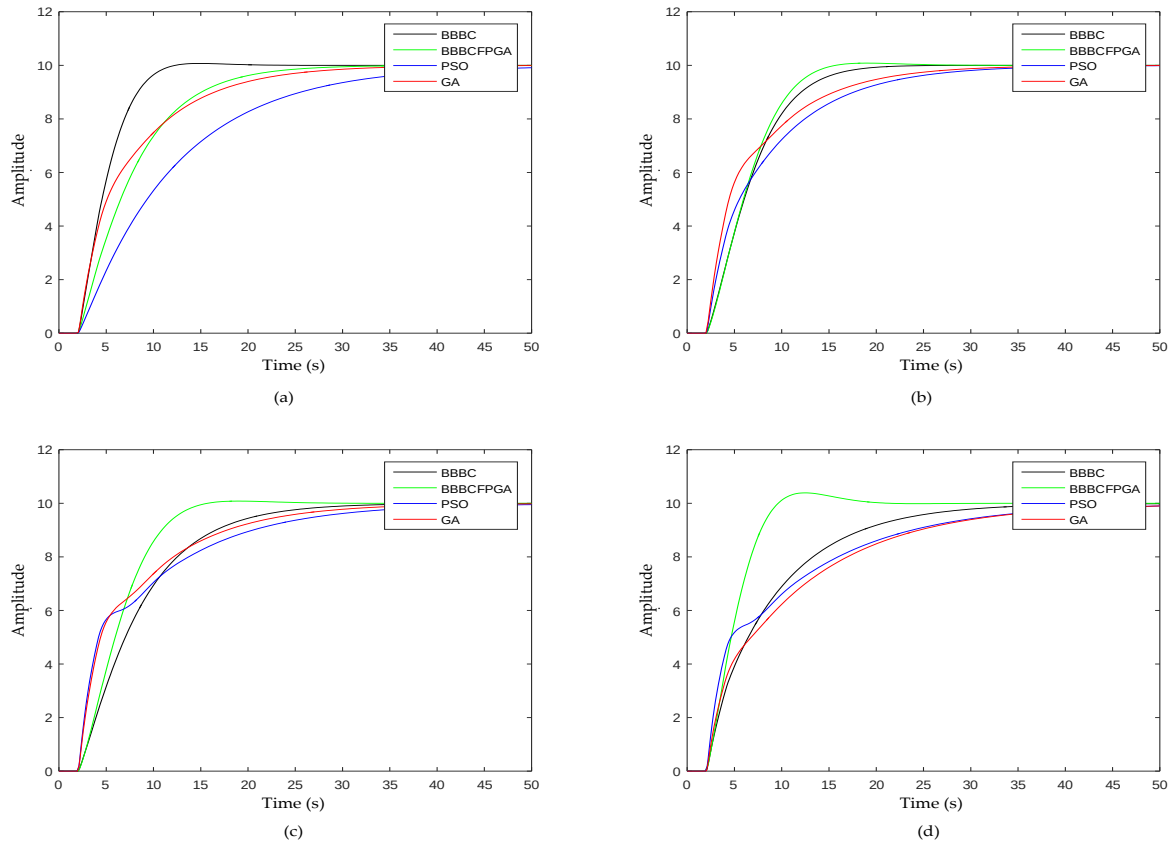


Figure 31. Time responses in seconds of four optimization methods (BB-BC, BB-BC-FPGA, PSO, GA) for different numbers of iterations (a, b, c, and d for 100, 200, 500, and 1000 iterations, respectively).

The experimental simulations for the software models (BB-BC, PSO, GA) run on an Intel *Core* i7-3612QM CPU at 2.10 GHz (64-bit). The transient performance measured in seconds is presented in Table 18. All algorithms achieve very low overshoot percentages, with values around 0.5%. Both the BB-BC and BB-BC-FPGA methods have an $Os\%$ of 0.505%, while the GA method has 0.493% and the PSO method 0.499%. This indicates that all optimization methods are effective in minimizing overshoot. The BB-BC (8.632 sec) and BB-BC-FPGA (9.498 sec) methods exhibit significantly shorter settling times compared to the GA (23.196 sec) and PSO (15.430 sec) methods. This confirms that the BB-BC-based methods allow the system to reach and maintain its steady state more quickly. Similar to settling time, the BB-BC (7.708 sec) and BB-BC-FPGA (7.959 sec) methods have faster rise times, while the PSO (9.364 sec) method is slower, and the GA (14.014 sec) method is the slowest one. This highlights the superior responsiveness of the BB-BC and BB-BC-FPGA optimized systems.

Table 18. Transient performance comparison

Method	PID Parameters			Os%	Ts (s)	Tr (s)
	kp	ki	kd			
BB-BC	0.205	0.063	0.046	0.505	8.632	7.708
BB-BC-FPGA	0.236	0.076	0.020	0.505	9.498	7.959
GA	0.341	0.154	0.014	0.493	23.196	14.014
PSO	0.479	0.217	0.006	0.499	15.430	9.364

Figure 32 shows the control system action i.e., system output, control signal, error, and objective function for all optimization methods, assuming a population size of 10 ($N = 10$) and 100 iterations ($M = 100$). Regarding the system output (Figure 32.a), the BB-BC and BB-BC-FPGA methods achieve a faster and smoother rise to the desired output with minimal overshoot, settling quickly, while the PSO and GA methods exhibit slower responses and more pronounced transient behaviors before reaching steady state. Figure 32.b illustrates the control signals generated by each method. The BB-BC and BB-BC-FPGA methods produce more stable, rapidly settling control inputs, reaching steady-state values around 12-12.5 units, whereas the PSO and GA methods exhibit more varied, less stable control efforts, taking longer to stabilize. Figure 32.c illustrates the error between the set point and the system output. The BB-BC and BB-BC-FPGA methods exhibit a rapid decay of the error signal to near zero, indicating precise control. In contrast, the PSO and GA methods exhibit slower convergence of the error to zero, implying less accurate or slower responses. Figure 32.d illustrates how fast the objective function converges over iterations. The BB-BC and BB-BC-FPGA methods converge very quickly to a low objective function value, often within the first few iterations. In contrast, the PSO and GA methods exhibit slower and more oscillatory convergence, requiring significantly more iterations to approach their optimal objective function values. As clearly depicted in Figure 32, the BB-BC and BB-BC-FPGA approaches are more robust than both the PSO and GA approaches.

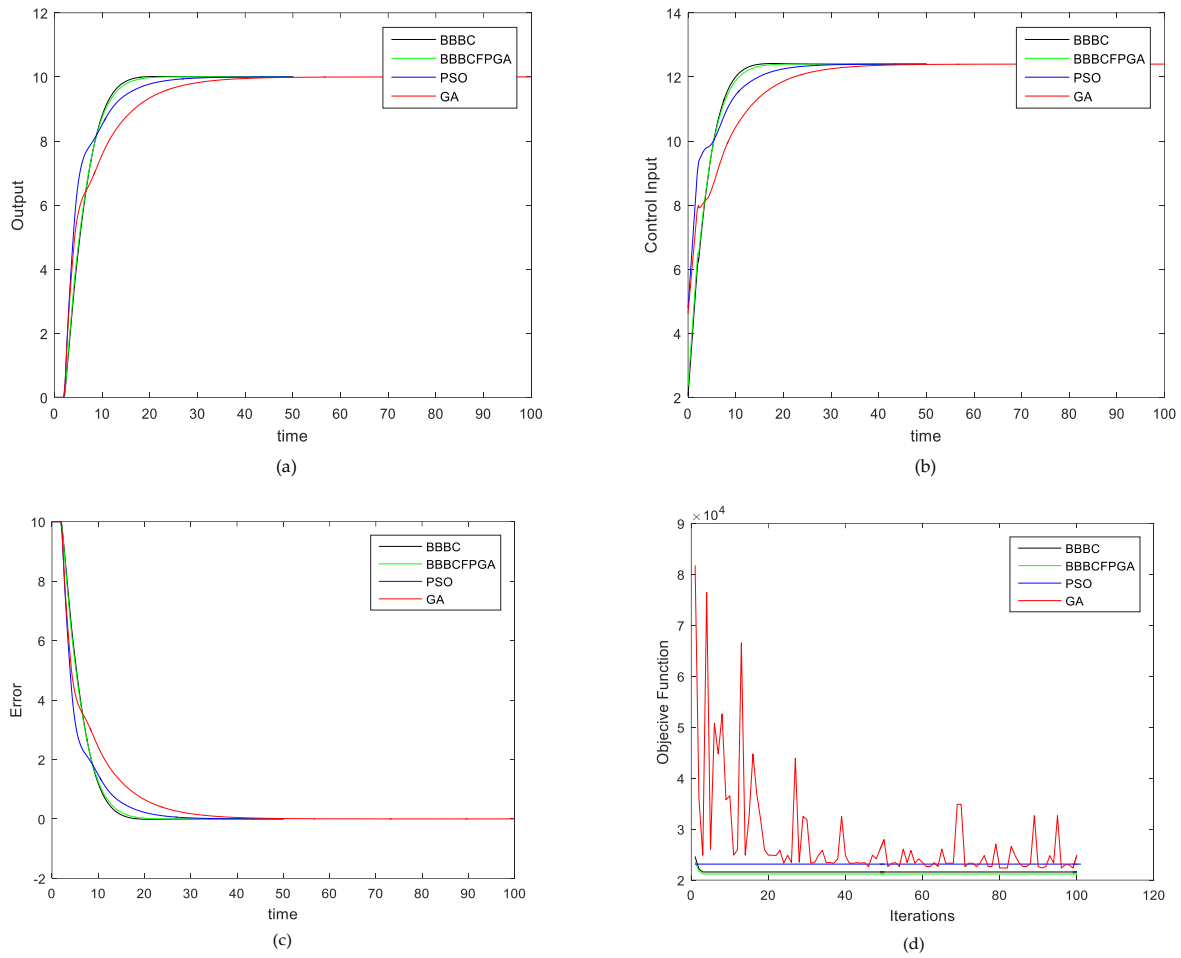


Figure 32. The (a) system output, (b) control signal, (c) error signal, and (d) objective function for all four optimization methods

Figure 33 shows the best objective function values per iteration for all four optimization algorithms across different numbers of iterations. In all cases, the BB-BC and BB-BC-FPGA methods converge faster (to local minima) than the GA and PSO methods, which is mostly expected from the literature.

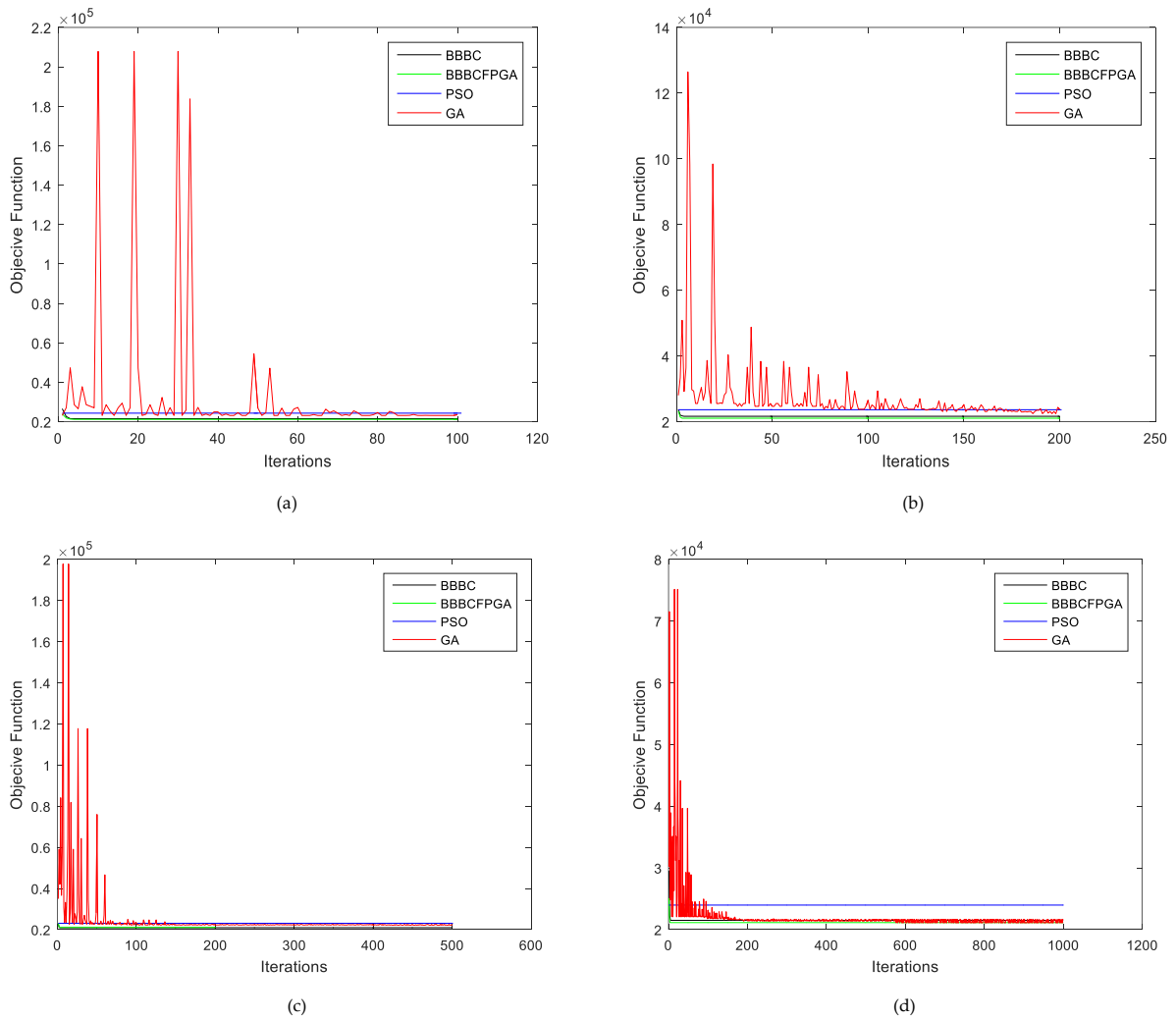


Figure 33. Objective function convergence for all optimization methods and for different numbers of iterations (a, b, c, and d for 100, 200, 500, and 1000 iterations, respectively)

With a new run of all optimization methods (BB-BC, BB-BC-FPGA, GA, PSO), Table 19 presents the objective function convergence for each method and varying numbers of iterations. Specifically, it provides the number of iterations to achieve convergence of the objective function, the Steady State Time (SST) in seconds, and the running time in seconds assuming a population size of 10 ($N=10$) and 100 iterations ($M=100$). Both the BB-BC and BB-BC-FPGA methods converge in just two iterations, demonstrating an exceptionally fast convergence rate. The PSO method also converges quickly, requiring only two iterations, whereas the GA method requires more iterations, specifically 73. The BB-BC method has an SST of 10.642 seconds, while the BB-BC-FPGA has a slightly higher SST of 11.456 seconds. The GA method shows an SST of 25.116 seconds, which is the longest among the methods and the PSO

method shows an SST of 19.440 seconds. The BB-BC-FPGA method stands out with a low runtime of 0.016 seconds, demonstrating its efficiency when implemented on an FPGA. In contrast, all the other methods have significantly higher running times: 4.061 seconds for the BB-BC, 6.887 seconds for the GA, and 8.634 seconds for the PSO method.

Table 19. Objective function convergence details (iterations and execution time) for all four optimization methods

Tuning Method	Convergence Iteration	Steady State Time (s)	Running Time (s)
BB-BC	2	10.642	4.061
BB-BC-FPGA	2	11.456	0.016
GA	73	25.116	6.887
PSO	2	19.440	8.634

Figure 34 illustrates the convergence behavior of the objective function for four different optimization methods (BB-BC, BB-BC-FPGA, PSO, and GA) over 120 iterations. Both the BB-BC and BB-BC-FPGA methods exhibit very rapid convergence, with their objective functions dropping to low values almost immediately, converging by iteration 2. This indicates their efficiency in finding an optimal or near-optimal solution early in the optimization process. The PSO method shows less stable convergence than the BB-BC and BB-BC-FPGA methods, exhibiting oscillations in the objective function value before convergence. The PSO method converges around the 2nd iteration. The GA method shows the slowest convergence among the four methods, with its objective function values fluctuating significantly across many iterations before settling. Finally, Figure 34 reinforces the observations from Table 20 regarding convergence speed. The BB-BC and BB-BC-FPGA methods are clearly superior in terms of iteration count to convergence and the running time highlights the significant advantage of the FPGA implementation, with BB-BC-FPGA being orders of magnitude faster than the other methods, including its software counterpart, BB-BC. While the BB-BC and BB-BC-FPGA methods converge quickly, their SSTs are comparable to or better than those of the PSO method and significantly better than those of the GA method. The remarkable reduction in running time for the BB-BC-FPGA method strongly suggests the benefits of hardware acceleration for this specific optimization task.

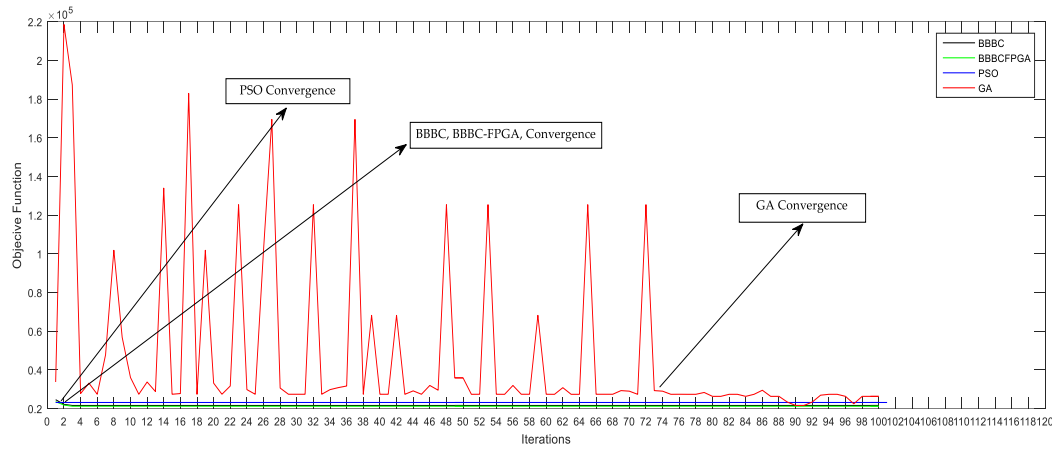


Figure 34. Objective function convergence for all four optimization methods

4.2.6.3 Computation Time

Table 20 compares the computation time of the proposed approach (BB-BC-FPGA) with the GA and BB-BC algorithms implemented in software (MATLAB model). The proposed approach achieves a speedup of more than 430X compared to the GA (in sw) approach and more than 253X compared to the BB-BC (in sw) approach.

Table 20. Computation time in (s) for GA, BB-BC (in sw) and BB-BC-FPGA (proposed approach)

M	GA sw (MATLAB)	BB-BC sw (MATLAB)	BB-BC-FPGA		
			Exec. Time	Speedup vs GA	Speedup vs BB-BC
100	6.887	4.061	0.016	430	253
200	12.008	7.990	0.028	428	285
500	28.728	18.023	0.065	441	277
1000	59.205	40.302	0.140	422	287

4.3 Optimizing the FOPID controller of an HVAC with the BB-BC algorithm

This section presents how we optimized the FOPID control parameters of an HVAC system using the BB-BC algorithm and enhanced system performance by accelerating various components using FPGA technology. Compared to classical PID controllers, the FOPID model improves dynamic performance and system robustness. However, tuning the FOPID control parameters ($k_p, k_i, k_d, \lambda, \mu$) is a challenging task. To address this, we adopted the BB-BC algorithm to tune the FOPID controller parameters optimally by minimizing six different performance indices. The HVAC system is modeled as an SOPTD system (General-purpose transfer function) yielding six different cases, obtained by combining three delay values and two

gain values. Furthermore, we designed a high-performance pipelined model (Chapter .3) of the BB-BC algorithm and the FOPID controller, and implemented it on an FPGA, as detailed in chapter 3. Experimental results demonstrated that the proposed FOPID-BB-BC scheme achieves significant improvements: it reduces HVAC output oscillations through auto-tuning of the control parameters and enhances rapid convergence. Additionally, a substantial reduction in both the output control signal and the error was observed when utilizing a larger window size, highlighting the impact of parameter tuning on system performance. The proposed approach uses FOPID controllers to optimize building HVAC systems. The study focuses on the optimal tuning of the FOPID control parameters to minimize energy consumption while maintaining system stability. The application and testing of the BB-BC algorithm demonstrated its functionality and robustness, even in the presence of time delays and external disturbances in building HVAC systems.

Finally, we introduced the BB-BC evolution algorithm to the domain of EM building offering a novel optimization strategy, demonstrating its effectiveness in optimizing FOPID control parameters for HVAC systems and enhancing transient stability under pulse disturbances. A comparative analysis with GA, PSO, and H_∞ algorithms showcased the superior performance of FOPID-BB-BC controllers compared to conventional PID, GA, and PSO implementations.

4.3.1 Regulating an HVAC system using the FOPID controller

An HVAC system provides suitable air quality by conditioning the flow of air in the indoor space of a closed building to ensure the comfort of its occupants, by diluting and removing humidity from indoor air through ventilation and control of building pressurization [60]. The transfer function of an HVAC system can be expressed as:

$$G(s) = \frac{ke^{-\theta s}}{(\tau_1 s + 1)(\tau_2 s + 2)} \quad (4.9)$$

where the time delay θ can be 2, 3, or 4, the gain k can be 0.81 or 1.2 and the time constants are $\tau_1 = 0.97$, $\tau_2 = 0.1$ [129]. The six possible combinations of θ and k values in the SOPTD generate six different HVAC system models (use cases). The block diagram of the HVAC system has been depicted in Figure 11.

4.3.2 Tuning the FOPID controller parameters using the BB-BC algorithm

An FOPID-based control unit offers improved HVAC performance. In this thesis, we combined the FOPID controller with the BB-BC algorithm to optimize its control parameters, improving the system performance and dynamic responsiveness. The integral performance index was considered as the minimization objective of the BB-BC algorithm and the results were obtained by equating the values of all control parameters. In addition, the closed-loop unit-step responses of the proposed method were plotted for all HVAC systems modeled by an SOPTD transfer function. The transfer function of the entire system was implemented using the BB-BC algorithm to emulate a full closed-loop control system plus various time delays (θ) and a setpoint reference of 1. The FO modeling and control (FOMCON), an open-source MATLAB toolbox and extension to the mini toolbox FOTF [130], was used to emulate and analyze the FOPID controller. A set of six performance indexes ($J_1, \dots, J_6$) was used to demonstrate the robustness of the proposed FOPID-BB-BC scheme. This set includes four common objective functions, IAE , $ITAE$, ISE , and $ITSE$ mentioned in Subsection 4.2.1 plus the two following objective functions:

$$J_5 = [(1 - \text{abs}.e^{-\beta}) * (M_p + E_{ss}) + \text{abs}.e^{-\beta} * (t_s - t_r)] \quad (4.10)$$

$$J_6 = [(w_1 * M_p) + (w_2 * t_r) + (w_3 * t_s) + (w_4 * E_{ss})] \quad (4.11)$$

Figure 35 shows the architecture of the entire system, including the FOPID controller, the BB-BC optimizer and the HVAC system.

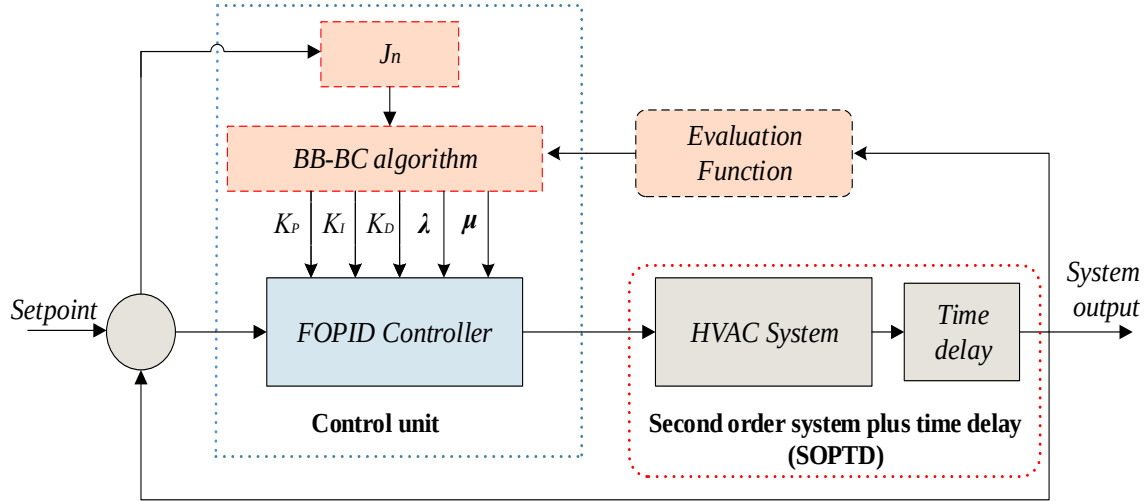


Figure 35. System architecture: FOPID controller, BB-BC & HVAC

We tested the robustness of the proposed method for six HVAC use cases: (a) $\theta=2$ and $k=0.81$, (b) $\theta=2$ and $k=1.2$, (c) $\theta=3$ and $k=0.81$, (d) $\theta=3$ and $k=1.2$, (e) $\theta=4$ and $k=0.81$, f) $\theta=4$ and $k=1.2$. We calculated various time response parameters, such as overshoot (M_p), steady state error (E_{ss}), absolute error order (β), settling time (t_s) and rise time (t_r) for every use case. Regarding the BB-BC algorithm, we used a uniform random number generator to build an initial population of size 10. The algorithm was run until it reached the maximum number of 100 iterations. Table 21 presents the control parameters and the time responses for the six HVAC use cases.

Table 21. Control parameters and time response for the six HVAC use cases ($\theta = 2, 3, 4$ and $k = 0.81, 1.2$) and the six performance indices

Values	Index	Control parameters					Time response				
k	J_n	k_p	k_i	k_d	λ	μ	E_{ss}	t_s	t_r	$M_p\%$	
θ_1	0.81	J_1	0.3208	0.3071	0.0888	0.9710	0.1066	1.0001	3.4860	3.635	00.505
		J_2	0.2957	0.3024	0.0805	0.9733	0.1097	0.9927	4.8310	3.910	-00.149
		J_3	0.3188	0.3053	0.0962	0.9389	0.1006	0.9875	14.501	3.069	13.068
		J_4	0.3184	0.3207	0.0855	0.9909	0.1062	1.0001	6.1090	3.422	2.5770
		J_5	0.3145	0.3078	0.0963	0.9963	0.1186	0.9956	3.4970	3.605	00.486
		J_6	0.2966	0.3141	0.1058	0.9865	0.1027	0.9946	3.3210	3.538	00.456
	1.2	J_1	0.3143	0.2817	0.1057	0.9978	0.1267	0.9969	8.5060	2.034	14.368
		J_2	0.2967	0.2878	0.0974	0.9534	0.1211	0.9929	9.8070	2.088	13.068
		J_3	0.3042	0.3106	0.0855	0.9338	0.1093	0.9911	9.7410	1.961	18.452
		J_4	0.3060	0.2902	0.1022	0.8482	0.1292	0.9781	31.459	2.001	34.459
		J_5	0.3139	0.2983	0.0922	0.9905	0.0995	0.9964	9.7240	1.986	19.880
		J_6	0.2963	0.2768	0.0913	0.8745	0.1129	0.9802	16.688	2.138	05.851

(a)

Values	Index	Control parameters					Time response				
k	J_n	k_p	k_t	k_d	λ	μ	E_{ss}	t_s	t_r	$M_p\%$	
θ_2	0.81	J_1	0.3276	0.2884	0.1086	0.9928	0.1109	1.0003	14.118	3.159	15.698
		J_2	0.3187	0.3091	0.0910	0.9387	0.1162	0.9877	14.368	14.44	03.051
		J_3	0.3058	0.3071	0.0971	0.9695	0.1302	0.9923	8.6060	3.817	-00.041
		J_4	0.3152	0.3112	0.0834	0.8686	0.1032	0.9719	22.399	3.125	06.989
		J_5	0.3153	0.2855	0.0852	0.9935	0.1071	0.9949	14.003	3.265	15.698
		J_6	0.3181	0.2989	0.0874	0.8936	0.1180	0.9772	14.671	3.206	06.989
	1.2	J_1	0.3095	0.3044	0.1061	0.9996	0.1326	0.9976	19.784	1.950	57.937
		J_2	0.3131	0.2909	0.0817	0.9052	0.1120	0.9842	35.647	2.076	38.194
		J_3	0.3127	0.2914	0.0906	0.8896	0.1317	0.9834	35.156	2.061	36.301
		J_4	0.3060	0.2902	0.1022	0.8482	0.1292	0.9781	31.459	2.001	34.459
		J_5	0.1071	0.3012	0.0975	0.9857	0.1007	0.9924	19.505	1.961	55.469
		J_6	0.3058	0.2796	0.0925	0.8852	0.1324	0.9824	24.417	2.152	30.921

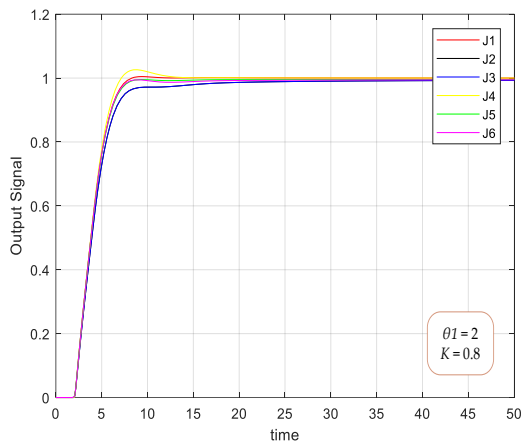
(b)

Values	Index	Control parameters					Time response				
k	J_n	k_p	k_i	k_d	λ	μ	E_{ss}	t_s	t_r	$M_p\%$	
θ_3	0.81	J_1	0.3151	0.2941	0.1056	0.9913	0.1022	1.0083	19.697	3.068	44.203
		J_2	0.2976	0.2996	0.0949	0.9336	0.1072	0.9862	35.849	3.249	29.221
		J_3	0.3209	0.2965	0.1020	0.9065	0.1131	0.9877	32.640	3.352	18.452
		J_4	0.3059	0.2919	0.0856	0.9377	0.1193	0.9872	32.829	3.262	29.221
		J_5	0.3035	0.2847	0.0928	0.9775	0.1257	0.9959	19.499	3.252	34.459
		J_6	0.3049	0.2918	0.0845	0.9267	0.1297	0.9872	32.829	3.253	29.221
	1.2	J_1	0.3126	0.2900	0.0983	0.9942	0.1145	0.9973	100.07	2.033	84.256
		J_2	0.3100	0.2948	0.0821	0.9495	0.1378	0.9954	97.184	2.053	77.679
		J_3	0.3136	0.2842	0.0910	0.8651	0.1175	0.9902	82.015	2.080	57.937
		J_4	0.3078	0.3124	0.0978	0.8103	0.1374	1.1033	69.222	1.995	57.937
		J_5	0.3299	0.2815	0.1021	0.9965	0.1130	0.9970	105.98	2.011	80.909
		J_6	0.3077	0.3123	0.0978	0.8103	0.1273	1.1101	68.931	1.985	57.937

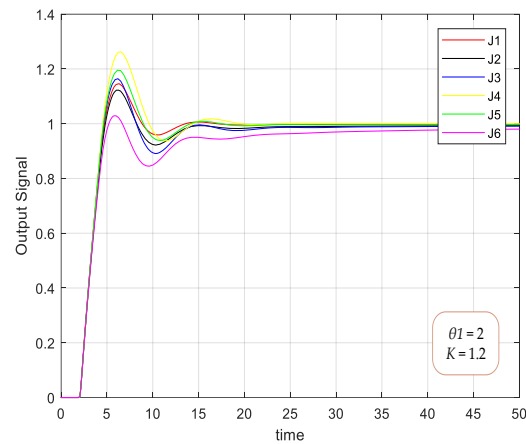
(c)

These time responses demonstrate the impact of optimizing the FOPID parameters, specifically on the overshoot, rise time, and settling time of the system, as shown in Figure 36. From the analysis of the results, we observed the following:

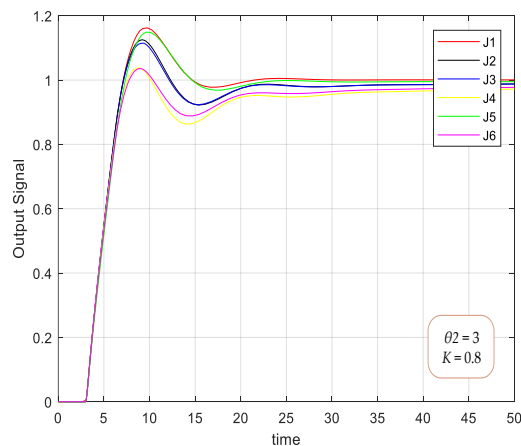
- The proposed approach deals effectively with all six performance indexes, taking more aggressive control actions even in process plants with longer delay times. By observing the settling time and the maximum overshoot of the output signal, it is concluded that the controller effectively processes the error.
- It greatly tracks the setpoint, reduces the rise and settling times, and eliminates the overshoot problems in all the process plants.
- There is a considerable peak overshoot and late settling time for the cases where k is equal to 1.2 and performance indices (J_1, J_2, J_3, J_4, J_5 , and J_6) are used, whereas the FOPID controller smoothly achieves the desired reference point in all the other cases.



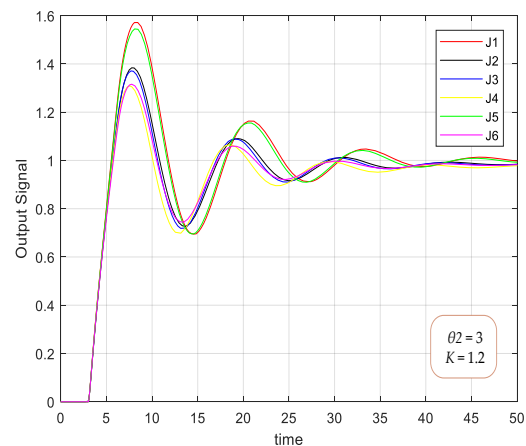
(a)



(b)



(c)



(d)

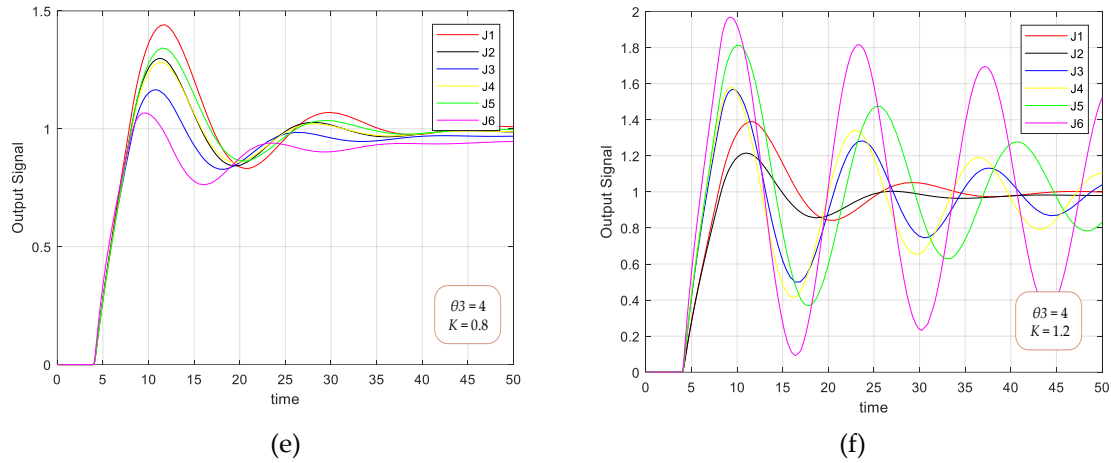


Figure 36. Time responses (*a, b, c, d, e, and f*) of the FOPID controller using the six HVAC cases for the six performance indexes (J_1 , J_2 , J_3 , J_4 , J_5 , and J_6)

Furthermore, we assessed the functionality and robustness of the FOPID-BB-BC scheme in the presence of time delay and external disturbances on an HVAC system. Figure 37 illustrates the ability of the proposed approach to effectively track and reject a disturbance signal applied after 30 seconds.

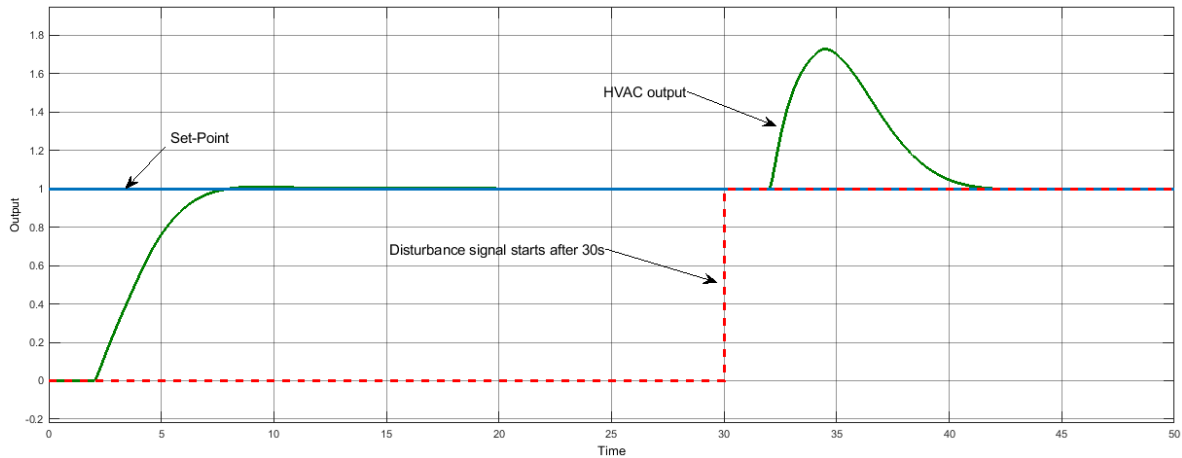


Figure 37. The effectiveness of the FOPID-BB-BC scheme against a disturbance signal

We also compared the proposed approach, i.e., the BB-BC algorithm, with other optimization algorithms, namely the GA and the H_∞ algorithms, in terms of time response performance, when they are used for tuning the FOPID parameters. Table 22 compares the results for the GA and H_∞ algorithms as obtained from the literature [131] with the experimental results for our FOPID controller.

Moreover, we experimentally calculated the time-response performance of the GA and PSO algorithms for tuning the FOPID parameters and compared them with the proposed BB-BC-based approach. Table 23 presents the experimental results along with the FOPID control parameters used in the simulations. For the experiments, we assumed the *ITAE* criterion, the unit step as the reference signal, and 100 iterations for all three algorithms. The simulation results demonstrated that the proposed FOPID-BB-BC approach outperforms the GA and PSO optimization algorithms in terms of rise time, settling time and overshoot.

Table 22. Time response performance for the BB-BC, GA, H_∞ algorithms

Methods	t_s	t_r	M_p	E_{ss}
BB-BC	3.486	3.635	0.505	1.0001
GA	150	>30	0.22	>250
H_∞	250	>30	0.42	>250

Table 23. FOPID control parameters for the BB-BC, GA, PSO algorithms

Method	FOPID Parameters					$Os\%$	T_s	Tr
	K_p	K_i	K_d	μ	λ			
BB-BC	0.20	0.06	0.04	0.06	0.19	0.42	7.19	3.56
GA	0.714	0.496	0.198	0.865	0.032	10.33	31.73	1.36
PSO	0.52	0.26	0.03	0.21	0.09	32.9	33.16	1.48

Then, we compare the performance of an HVAC system regulated by conventional PID and FOPID controllers optimized by the BB-BC algorithm with an HVAC system regulated by an MPC controller. The FOPID-BB-BC scheme compensates for disturbances in the HVAC system caused by undesired nonlinearities, such as time delays and environmental variations, thereby improving system precision and stability. A thorough comparison of the FOPID-BB-BC and the PID-BB-BC over 100 control cycles (i.e., 100 iterations) demonstrates that the BB-BC-FOPID controller offers the best overall balance for the transient performance, with a minimal overshoot (0.688%) and a relatively short settling time (7.315 sec). In contrast, the BB-BC-PID controller exhibits a higher overshoot (4.450%) and a significantly longer settling time (11.58 sec), though it achieves a faster rise time (2.308 sec). The MPC controller has the lowest peak value (0.8100) and the shortest settling time (5.9002 sec), but its overshoot is not reported, making it difficult to fully evaluate its transient response. Table 24 compares the time responses for the three schemes, while Table 25 outlines the specifications of the MPC controller, including the prediction horizon (ranging from 10 to 50), control horizon

(ranging from 1 to 10), input constraints (minimum and maximum set to -1), sample time (0.1), and input weights (ranging from 0 to 2).

Table 24. Transient performance comparison

Method	Peak	Os%	Ts	Tr
BB-BC-FOPID	1.007	0.688	7.315	3.653
BB-BC-PID	1.044	4.450	11.58	2.308
MPC	0.810	--	5.900	2.152

Table 25. Specifications of MPC

MPC specifications	Values
Prediction Horizon	10 - 50
Control Horizon	1 - 10
Input Constraints	Min= -1, Max= -1
Sample Time	0.1
Input Weights	0 - 2

4.3.3 Implementing the FOPID-BB-BC controller using FPGA

We implemented the FOPID controller along with the BB-BC algorithm on an FPGA platform. In this section, we describe the specific FPGA architecture used for the BB-BC within the context of FOPID optimization, which supports high-performance parallel processing. The description begins with the D-FOPID controller, focusing on its discrete transfer function, and continues with the hardware realization of the GL-FOPID model, which integrates FO control with optimized parameters. We also provide a detailed technical description of the FPGA-based design and implementation of both the D-FOPID and GL-FOPID controllers, integrated with the BB-BC optimization algorithm. It must be noted that in the FPGA implementation, we selected $J1$ ($ITAE$) as the performance index and θ_1 with $k=1.2$ for the transfer function of the HVAC system; these parameters were chosen based on the simulation results.

4.3.3.1 BB-BC hardware model

We customized the FPGA design of the BB-BC algorithm to meet the computational demands and architectural constraints of each controller. The FOPID controller involves more complex computations, including fractional-order calculus, which increases the computational issues. Thus, we used the high-performance, pipelined, parameterized FPGA architecture for the BB-BC algorithm described in Chapter 3. The reader can find more technical details about the FPGA implementation of the BB-BC algorithm in Chapter 3.

4.3.3.2 D-FOPID hardware model

For the hardware model of the digital FOPID controller, we adopted the discrete transfer function given by Eq. (2.14). To reduce the memory storage requirements, we set parameter M to 5. Note that for higher values of M , the computational cost and memory capacity of the system increase. On the other hand, the controller output is not sensitive to the value assigned to M , provided that the other parameters are appropriately selected via an optimization method. The controller parameters $(k_p, k_i, k_d, \lambda, \mu)$ are computed according to Eq. (2.14) and Eq. (2.15) using the Tustin transform and the PSE approximation method as:

$$C_d(z) = \frac{w_0 + w_1 * z^{-1} + w_2 * z^{-2} + w_3 * z^{-3} + w_4 * z^{-4} + w_5 * z^{-5} + w_6 * z^{-6}}{1 - z^{-1}} \quad (4.12)$$

where $w_0 \dots w_6$ are coefficients of the control parameters. Figure 38 shows the circuit that implements the operations of Eq. (4.12). The parallel calculation of the seven products and the fast execution of the arithmetic operations, such as multiply, add, and multiply-accumulate (MAC), by the hardwired DSP logic integrated into the FPGA devices applying the Tustin transform to each fractional term, expanding the resulting as a truncated power series (PSE) with $M=5$, guarantee the fast calculation of the discrete transfer function.

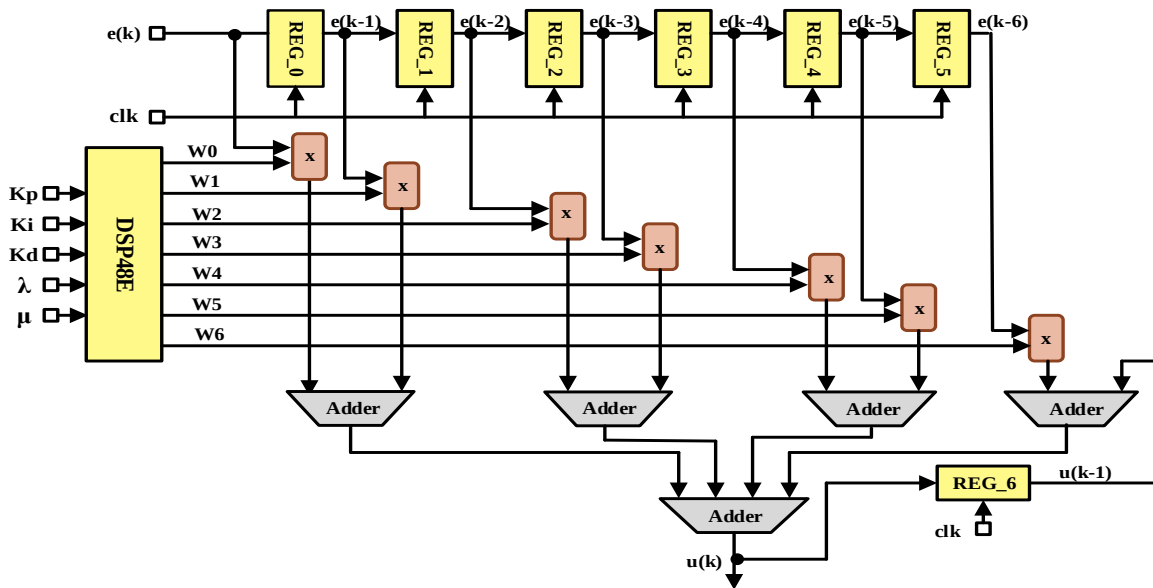


Figure 38. D-FOPID controller parameters (K_p , K_i , K_d , λ , and μ), error $e(k)$, and output $u(k)$

Figure 39 shows the structure of the D-FOPID controller consisting of the following: the BB-BC module. The control loop begins with the error process module that calculates the system error signal in real time. This error signal is forwarded both to the multiplication and accumulation unit and to the BB-BC evolutionary module, which adaptively tunes the controller. These updated gains are sent to the control unit that responsible for parameter management and coordination across FPGA blocks. A DSP-based computation module performs the fractional-order derivative and integral computations using digital signal processing techniques. It interacts with the coefficient module, which stores and updates the precomputed weights ($w_0, w_1, \dots, w_6$) necessary for approximating the fractional-order operators. The control output signal is the HVAC system input. The multiplication and accumulation unit combines the incoming error signal with the coefficient set ($w_0, w_1, \dots, w_6$) to generate the weighted fractional terms. The resulting output is the finalized control signal sent to the system. The architecture allows real-time adaptation, hardware-level parallelization, and high-speed execution suitable for HVAC optimization control.

Note that the D-FOPID controller parameters (K_p, K_i, K_d, λ , and μ), the error $e(k)$ and the output $u(k)$ are represented by 16-bit signed fixed-point real numbers, with the 4 upper bits denoting the integer part and the 12 lower bits the fractional part.

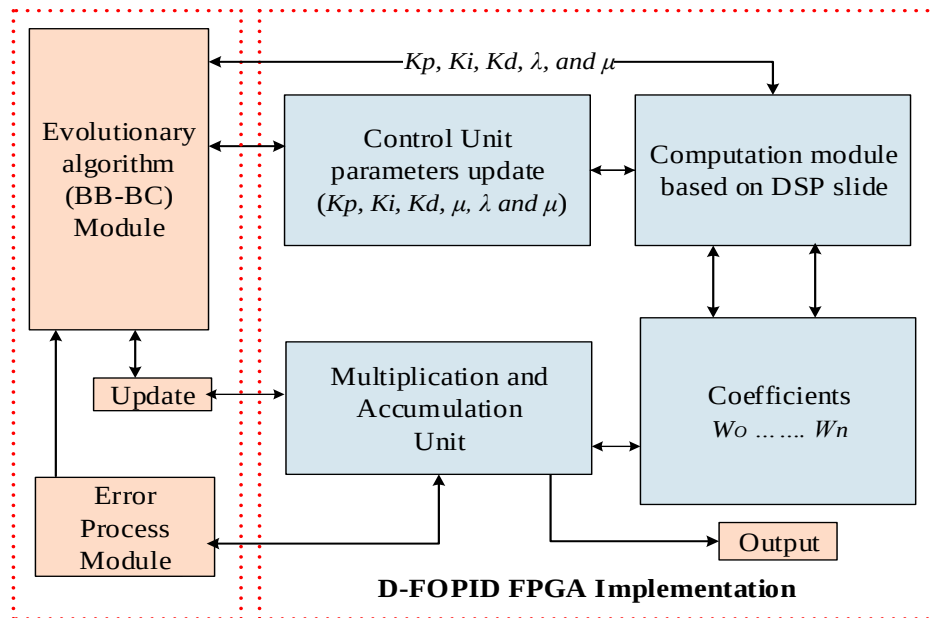


Figure 39. Discretized (D)-FOPID controller structure

4.3.3.3 GL-FOPID hardware model

For the hardware GL-FOPID model, we adopted the transfer function of Eq. (2.13). Figure 40 shows the structure of the GL-FOPID controller consisting of three main stages: powering function operation, coefficient computation and GL definition, which are described below:

- **Powering function operation:** This module is composed of a ROM that stores the values of the term $\frac{1}{h^\alpha}$ of Eq. (2.13). These values, which have been statically calculated on MATLAB, depend on α , which equals λ in the derivative part and μ in the integral part. In other words, when the value λ (μ) is fed to the ROM input, the module outputs the value of $\frac{1}{h^{-\lambda}}$ ($\frac{1}{h^\mu}$) for the integer (derivative) part. The BB-BC module generates the ROM inputs (i.e., parameters λ and μ).
- **Coefficient computation:** This module calculates the terms $w_j^{(\alpha)}$ of Eq. (2.13), which are the binomial coefficients calculated recursively as $w_0^{(\alpha)} = 1, w_1^{(\alpha)} = \left(1 - \frac{\alpha+1}{1}\right)w_0^{(\alpha)}, w_j^{(\alpha)} = \left(1 - \frac{\alpha+1}{j}\right)w_{j-1}^{(\alpha)}$, where $j = 1, 2, 3, \dots$
- **GL definition:** This module completes the computation of Eq. (2.13), namely the binomial coefficients (w), which are stored in a LookUp Table (LUT_1) and the expression $e_{in}w_0 + d_0$ to $e_{in}w_n + d_n$, and stores the results in a second LookUp Table (LUT_2) as suggested in [132].

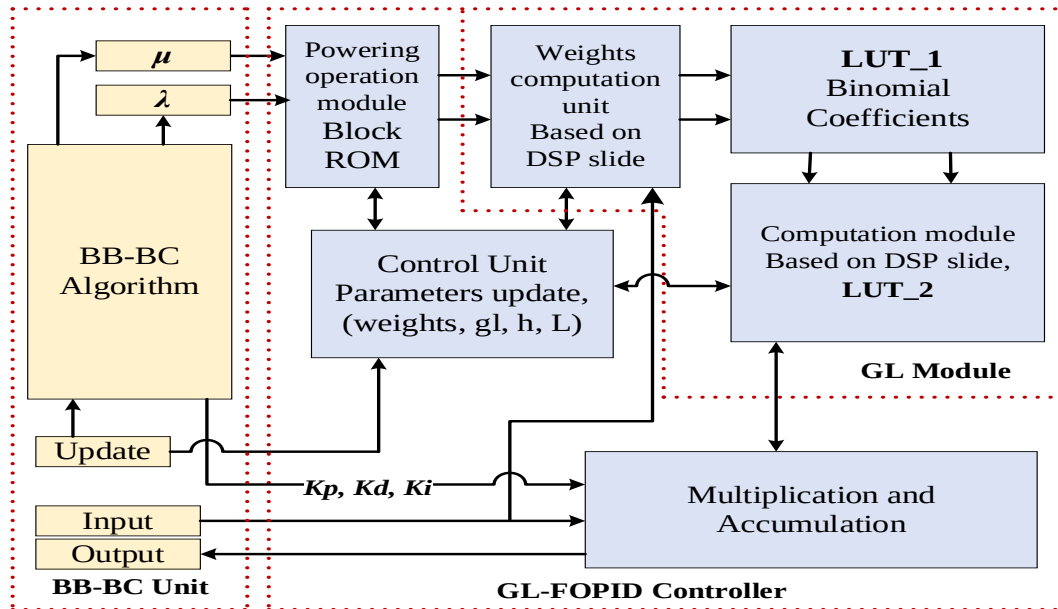


Figure 40. GL-FOPID controller design structure

The values of the parameters L and h in Eq. (2.13) should be selected according to the availability of resources of the FPGA device and the accuracy specifications of the target system. In our FPGA implementation, the step size h was set to 0.01, and window sizes L of 32, 64, 128, and 256 bits were tested. Table 26 demonstrates how the window size (L) affects the FOPID controller parameters and the output control signal of the system. It is evident that increasing the window size (L) results in lower error and a lower output control signal, thus improving system stability, but at the cost of more hardware resources.

Table 26. GL-FOPID performance vs. window size (L)

Control parameters	Window size L			
	$L=32$	$L=64$	$L=128$	$L=256$
K_p	1.0010	1.0010	1.0010	1.0010
K_i	0.1510	0.1510	0.1510	0.1510
K_d	0.9010	0.9010	0.9010	0.9010
λ	0.5010	0.5010	0.5010	0.5010
μ	0.1510	0.1510	0.1510	0.1510
Error	0.1010	0.0781	0.0551	0.0272
Output control signal	0.5560	0.3353	0.2365	0.1089

Figure 41 shows the structure of the FPGA design for the case of the GL-FOPID controller, including the BB-BC module and the HVAC function. This figure illustrates the FPGA-based implementation of the control system. It integrates the BB-BC algorithm, which optimizes the control system parameters; the control unit, which updates the system parameters (weights and fractional-order coefficients); and the GL-FOPID unit, which serves as core controller. The GL-FOPID controller receives the error signal, which is the difference between the set point and the Output of the SOPTD system and adjusts the necessary control action to minimize this error based on updated parameters and the SOPTD system. The system processes error inputs, updates control signals, and generates outputs to achieve optimal performance. The multiplication and accumulation unit plays a role in processing signals related to the control output. It takes inputs from the control unit parameters update block, the GL-FOPID controller, and the system output signal. Evaluation function is to perform multiplications and accumulate results, contributing to the overall control signal that feeds back into the system by combining the GL-FOPID's output with system parameters or feedback for a refined control action. Note that the evaluation function unit provides feedback to the BB-BC algorithm, enabling it to assess the current parameter settings and guide the optimization process.

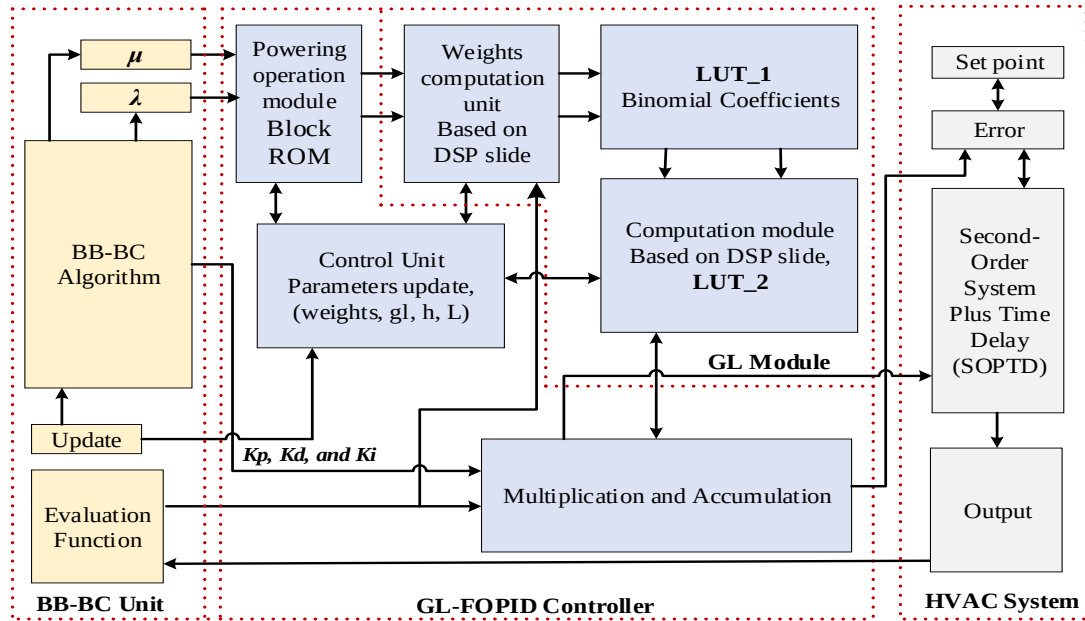


Figure 41. GL-FOPID & BB-BC FPGA design structure and HVAC system

4.3.4 Experimental evaluation of the FOPID-HVAC system

We developed two BB-BC optimized controllers (D-FOPID and GL-FOPID) and implemented them on an AMD/Xilinx Zynq UltraScale+ FPGA device (xczu7ev-ffvc1156). It is worth noting that the optimization algorithm employed the objective function J_1 . Table 27 presents a summary of resource utilization for the D-FOPID, GL-FOPID, and GL-FOPID-BB-BC designs. It must be noted that the GL-FOPID model achieves a higher frequency than the D-FOPID model, leading us to integrate the former with the BB-BC unit to create the complete system (GL-FOPID-BB-BC).

Table 27. FPGA resource utilization of the FOPID-BB-BC schemes on a Zynq UltraScale+ device

Category	Resource utilization					
	DSP		LUT		FF	
	(%)	Used	(%)	Used	(%)	Used
D-FOPID	0.52	9	0.13	289	0.01	39
GL-FOPID	59.95	1036	51.80	119345	1.78	8201
GL-FOPID-BB-BC	62.62	1082	54.37	125275	2.16	9945

Table 28 presents the GL-FOPID control parameters, errors, control, and output signals for different numbers of iterations of the BB-BC algorithm (100, 200, 500, and 1000). The results demonstrate a significant improvement in the controller's performance, as evidenced by reduced steady-state error and optimal signal values with increasing BB-BC iterations,

resulting in faster convergence of the output signal. On the other hand, our FPGA design is susceptible to even slight changes in the controller parameter values. It must be noted that the GL-FOPID controller parameters are obtained by checking the parameter sets on the available stability curve against the robustness requirements (as shown in Table 28), ensuring the feasibility of the GL-FOPID-BB-BC controller design technique.

Table 28 Performance of the FPGA implementation of the GL-FOPID-BB-BC controller

GL-FOPID	Number of iterations of BB-BC			
	100	200	500	1000
K_p	0.25587	0.25166	0.25374	0.26532
K_i	0.76725	0.81327	0.69811	0.77599
K_d	0.25691	0.21138	0.21162	0.23505
λ	0.78880	0.81708	0.73971	0.74407
μ	0.69353	0.59580	0.66384	0.64870
Error	-0.00081	-0.00074	-0.00069	-0.00033
Control signal	-0.00020	-0.00018	-0.00017	-0.00009
Output signal	1.00081	1.00075	1.00069	1.00034

Figure 42 shows the control and error signals for different numbers of iterations of the BB-BC algorithm. As the absolute value of the control signal decreases (i.e., the control signal goes closer to zero), so does the error, and vice versa.

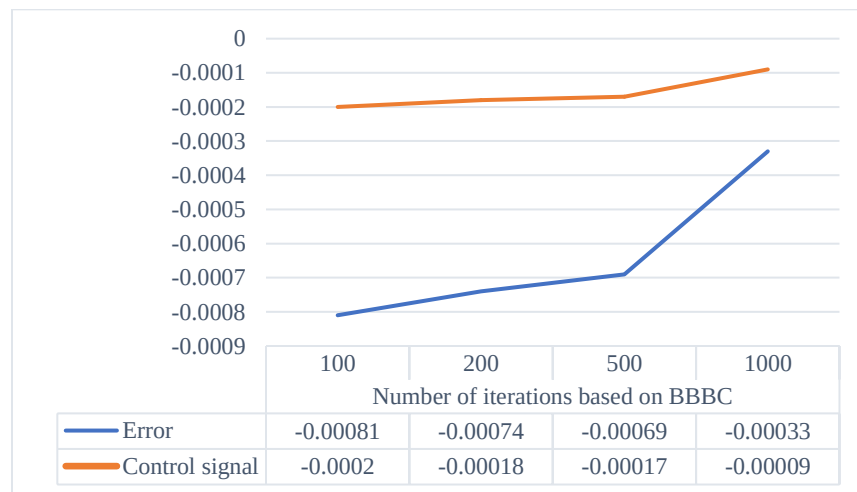


Figure 42. Control and error signals for different numbers of BB-BC iterations

4.4 Summary

In this chapter, we presented novel FPGA-based solutions for fast and efficient optimization of the BB-BC algorithm and the HVAC system controller. First, we proposed a low-cost, fast-converging optimization approach for tuning the PID controller parameters of a second-

order HVAC system, leveraging the BB-BC optimization algorithm. To further accelerate the process, we implemented both the BB-BC algorithm and the PID controller on an FPGA. The FIL technique was adopted to emulate and evaluate the system, in which the plant (developed as a MATLAB/Simulink model) was connected to the optimization controller implemented on an FPGA device. Extensive experiments were conducted to compare the proposed FPGA-based BB-BC scheme with software implementations of the BB-BC, GA, and PSO algorithms. The experimental results demonstrated superior convergence speed and computation time for the FPGA-based approach.

Additionally, we introduced a novel FPGA-based FOPID accelerator, implemented and evaluated for controlling HVAC systems in buildings. The BB-BC optimization algorithm was again used to optimally tune the FOPID control parameters for six different objective functions. We assessed the effectiveness and robustness of the BB-BC algorithm to optimize building HVAC systems in the presence of time delays and external disturbances. We tested the FPGA design for various values of L (window size) and input signals to ensure its validity. The results showed up to an 80% reduction in the output control signal and 73% in the error across different values of L , with rapid convergence to the optimal solution. It must be noted that the performance attained by the FPGA-based FOPID & BB-BC designs is comparable to and even better than the performance obtained through simulations of the *MATLAB/Simulink* models. The optimized FPGA design of the BB-BC algorithm, along with the hardware implementation of the GL-FOPID controller, provides a high-performance, energy-efficient solution for demanding, real-time industrial control applications.

CHAPTER 5

**An FPGA-based LAMDA
controller with BB-BC
tuning for HVAC systems**

5 An FPGA-based LAMDA controller with BB-BC tuning for HVAC systems

This chapter presents an efficient implementation of a Learning Algorithm for Multivariate Data Analysis (LAMDA) controller, integrated with the BB-BC algorithm, to enhance the performance of a Multi-Input Multi-Output (MIMO) HVAC system. The approach involves evaluating input descriptors using the LAMDA controller in conjunction with the BB-BC algorithm for fitness evaluation. The chaotic map function (CM) is also presented to enhance the exploration capability of LAMDA-BB-BC. The CM function is incorporated into the generation of new candidate solutions, instead of relying on conventional fuzzy aggregation operators such as T -norms and S -norms, which often drive the search toward smooth and predictable. The integration of the LAMDA controller with the HVAC system relies on simulation data, enthalpy calculations via MATLAB, and static control gains (K_p , K_i , K_d). The experimental results include system parameters, equilibrium points, and control loops and demonstrate the superior performance of the proposed approach when compared to solutions based on GA and PSO algorithms, confirming its superiority for the tested HVAC control scenario. An FPGA-based implementation of the LAMDA-BB-BC controller is introduced to meet the requirements of real-time applications and evaluated in terms of system performance and area utilization. The pipelined FPGA architecture of the BB-BC algorithm, as presented in Section 4, has been adopted, while the LAMDA controller's fuzzification and inference operations have been optimized using fixed-point arithmetic and parallel processing. The contents of this chapter are mostly based on the publication [P5].

5.1 Introduction

The stability of HVAC systems is affected by fluctuating environmental parameters and complex system characteristics, including nonlinearity, time lags, and temporal variations. Due to the complexity of modern HVAC systems, which can be modeled as MIMO systems, Anastasiadi, C., and A.I. Dounis presented modeling and optimization methods for the system components and operation, offering significant potential to reduce energy consumption [133].

Classical controllers struggle to meet the complex control requirements of modern HVAC systems. To overcome these challenges, several researchers have investigated alternative

solutions. For example, Ma, J., and X. Ma, studied a distributed algorithm for optimizing EM in microgrids by coordinating HVAC operation [134]. Nagpal, H., et al. examined model predictive controllers [135], and M. Anvar. Adibhesami focused on optimizing HVAC energy efficiency in low-energy buildings, a comparative analysis of reinforcement learning control strategies [136]. Moreover, traditional control strategies (e.g., PID, On-Off control) for MIMO systems cannot adapt to the changing operating conditions of HVAC systems. Consequently, self-tuning control units can effectively handle the dynamically changing parameters of these systems. Artificial Intelligence (AI) has been proposed to handle complex computational and dynamic control problems in various industries [137], [138], [139, 140]. Fuzzy logic techniques are employed to achieve complex control objectives cost-effectively. Machine Learning (ML), a subset of the AI field, provides computational tools for analyzing and controlling data. The benefits of applying ML to solve real-time control problems include its flexibility to model the dynamic behavior of nonlinear systems with MIMO [141]. The ML-based method Learning Algorithm for Multivariate Data Analysis (LAMDA) [142] is based on fuzzy logic (fuzzification and defuzzification) and, unlike other techniques, is not iterative. In recent years, several researchers have studied the LAMDA algorithm in various applications. Ruiz, F.A., et al., improved the classification process of the LAMDA algorithm applied to diesel engines [143]. Morales, L., et al. presented an intelligent controller based on LAMDA for Single-Input Single-Output (SISO) systems [144], and an adaptive LAMDA approach for modeling and controlling nonlinear systems [145]. This adaptability stems from LAMDA's ability to handle changes in multiple data variables, as shown by Morales, L., and J. Aguilar [146].

Optimization encounters three common challenges: *sensitive environments*, *uncertain outcomes*, and *cost considerations*. Various optimization techniques have been proposed to address these challenges, including an optimized prediction model [147], a combination of an artificial neural network (ANN) and a multi-objective genetic algorithm [148], and a particle optimization-based optimal LQR-PID controller [149]. The complexity of implementing and controlling MIMO systems (HVAC) can be reduced by using the LAMDA controller when the classical controllers fail to cope with it. The operation of the LAMDA controller has been tested and validated across different systems [154], including HVAC temperature and humidity control, which are complex to model due to their large number of parameters.

Previous studies demonstrated that the LAMDA controller outperformed various state-of-the-art approaches and methodologies (e.g., PID, Fuzzy-PID) [150, 151].

In industrial applications, a fully integrated control unit automates holistic system management, optimizing efficiency by continuously analyzing system data to enhance performance. In this chapter, we propose an *improved version of the LAMDA method* by incorporating two key approaches. First, we employ the full reinforcement operator in the calculation of the Generalized Aggregated Distance (*GAD*) function using the Ordered Weighted Averaging (*OWA*) method to effectively handle the function dimension (n) and the weighted range (W), instead of relying on the T and S norms. On the other hand, we can control W through any optimization method. Second, we integrate an evaluation algorithm to obtain optimal parameter values for the LAMDA controller. It is worth noting that combining an optimization algorithm with other applicable decision-making criteria is seldom considered in previous works. This integration allows us to achieve superior performance and accuracy. By combining these two approaches, we aim to enhance the effectiveness of the LAMDA method in practical applications. Additionally, we incorporate the powerful optimization algorithm BB-BC to tune the LAMDA controller, as we did in the previous chapter to optimize the performance of the PID/FOPID controllers. Moreover, several researchers have used the BB-BC algorithm in various applications with different objectives; for example, to model order reduction in Linear Time-Invariant (LTI) systems [152], or in SISO commensurate FO systems [153], to control energy management systems in hybrid isolated micro-grids [154], or to achieve maximum power point tracking [155].

To improve the responsiveness of the optimization algorithms and make them better suited for real-time applications, we accelerate them using FPGA technology. FPGAs provide significant advantages over traditional industrial computing platforms (e.g., microcontrollers) in terms of circuit speed, performance-to-watt ratio and system integration [156]. The benefits of using an FPGA have been analyzed in detail in Chapter 2. Modern SRAM FPGAs, with built-in arithmetic, memory, and connectivity modules, enable efficient mapping of intelligent neural control while also reducing the design cycle [157]. Hardware-based implementations have also been presented in the past for learning-assisted multi-stage autotuning functions [158], and Ustun, E., et al. has been proposed the LAMDA framework, a learning-assisted

multi-stage autotuning method for accelerating FPGA design closure in [159]. In this thesis, we propose a novel controller design that combines the LAMDA algorithm (based on the *OWA* aggregation function) with the BB-BC algorithm for optimizing the control input (classes). Integrating the LAMDA controller and the BB-BC algorithm into a single FPGA chip offers high speed and reduces the list of discrete components. The efficiency of the FPGA implementation is demonstrated on an AMD-Xilinx Virtex-7 FPGA device.

Concluding, this chapter provides the following contributions addressing the aforementioned key challenges of the optimization problems:

- The proposed LAMDA/BB-BC approach has been validated through simulations and compared with other well-known algorithms, including GA and PSO. The experimental results showed that it significantly improves optimization accuracy and convergence speed (*uncertain outcomes*).
- Reinforcement operator *OWA* has been used instead of S, T norms to avoid computational complexity and break the constraints of selecting W_v .
- A novel FPGA-based accelerator has been developed for the LAMDA/BB-BC controller. The proposed FPGA-based implementation enables faster tuning and improves system responsiveness. This reduces development time and allows efficient iteration, crucial for quickly adapting to changing conditions (*sensitive environments*).
- The FPGA-based solution enables real-time evaluation, allowing cost-effective tuning of LAMDA parameters (*cost considerations*).

5.2 Tuning the LAMDA Controller of a MIMO HVAC System using the BB-BC Algorithm

Our approach optimally tunes the LAMDA control unit of a MIMO HVAC system. A key feature evaluates initial values (memberships) twice using the BB-BC and once using the LAMDA algorithms. The first evaluation relies on the BB-BC fitness function and the number of iteration count, while the second step applies the LAMDA algorithm, which assesses the membership values directly based on the incoming control inputs, without requiring any prior modifications.

5.2.1 Integration of the LAMDA Controller into HVAC

This section describes the integration of the LAMDA controller into the HVAC system. To improve the LAMDA controller performance, the descriptors (membership functions) of the control unit inputs range between $[0, 1]$, $[0, 5]$, and $[0, 10]$ for each simulation. For the classification part, the class boundaries are between $[-1, 1]$, $[-5, 5]$, $[-10, 10]$, resulting in 25 classes; constant values are used for the three gains (kp, ki, kd) as in [160]. The proposed LAMDA controller was tested with different descriptor values (membership functions). The nonlinear MIMO HVAC system was implemented and simulated using reference points [61]. The enthalpy of liquid water (h_w) and the enthalpy of water vapor (h_{fg}) were calculated using the *MATLAB* X-Steam tables (m.file). The X-Steam provides a fast, reliable way to compute steam-table properties such as liquid water enthalpy (h_w) and latent heat of vaporization (h_{fg}) directly inside *MATLAB* without requiring external libraries. These functions are based on the IAPWS-IF97 steam formulation and allow the nonlinear MIMO HVAC model to calculate thermodynamic variables at each simulation step with high numerical accuracy.

Table 29 lists the chosen numerical values, and

Table 30 shows the numerical values of the system parameters at the operating equilibrium point.

Table 29. Numerical values of the system parameters

$\rho = 0.0074 \text{ [lb/ft}^3\text{]}$	$C_p = 0.24 \text{ [Btu/lbF}^\circ\text{]}$
$T_{ref} = 55 \text{ [F}^\circ\text{]}$	$T_{ref} = 71 \text{ [F}^\circ\text{]}$
$W_{ref} = 0.0088 \text{ [lb/lb]}$	$V_{he} = 60.75 \text{ [ft}^3\text{]}$
$f_{ref} = 17.000 \text{ [ft}^3\text{/min]}$	$V_s = 58.464 \text{ [ft}^3\text{]}$
$W_s = 0.007 \text{ [lb/lb]}$	-----

Table 30. Numerical values of the parameters at the operating equilibrium point

$x_1^0 = 71 \text{ [F}^\circ\text{]}$	$x_2^0 = 0.00092 \text{ [lb/lb]}$
$x_3^0 = 55 \text{ [F}^\circ\text{]}$	$T_0^0 = 85 \text{ [F}^\circ\text{]}$
$W_0^0 = 0.0018 \text{ [lb/lb]}$	$M_0^0 = 166.06 \text{ [lb/hr]}$
$u_1^0 = 17,000 \text{ [ft}^3\text{/min]}$	$u_2^0 = 58 \text{ [gpm]}$
$Q_0^0 = 289,897.52$	$W_s^0 = 0.007 \text{ [lb/lb]}$

Figure 43 illustrates the block diagram of the LAMDA control unit merged with the BB-BC algorithm unit and HVAC system, which includes two separate control loops, each dedicated to maintaining one of the two variables T and W .

To evaluate the performance of the LAMDA controller for the MIMO HVAC system, we analyze its behavior for various Membership Functions (MFs), where the maximum value is at the center point of the classes for the same rules and values of (kp, ki, kd) . Table 31 shows the different ranges of humidity (W) and temperature (T), along with the corresponding MFs. It also displays the maximum and minimum output signals for humidity and temperature.

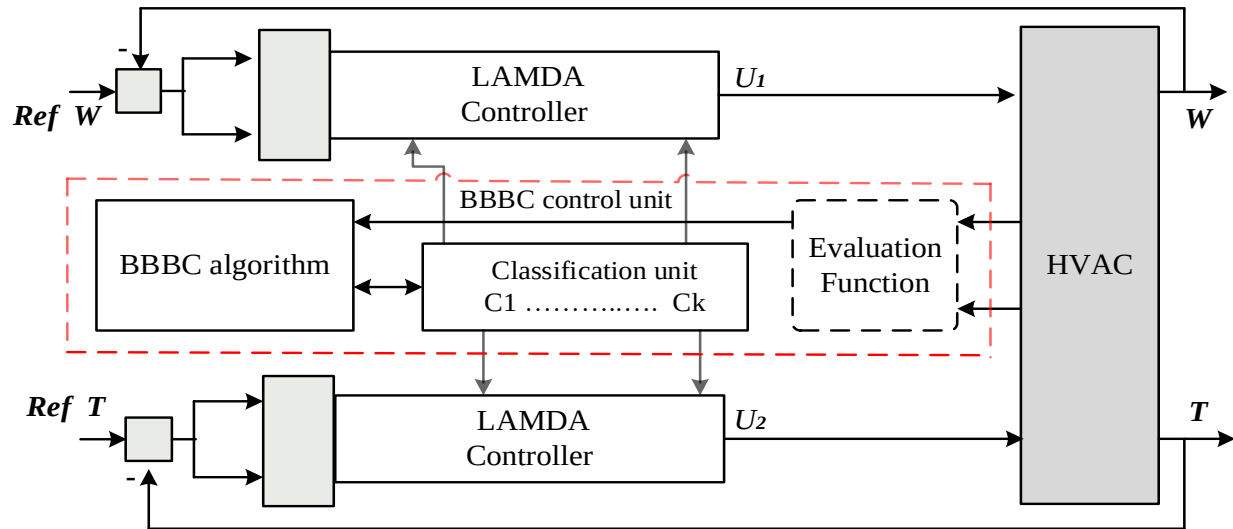


Figure 43. Block diagram of the LAMDA / BB-BC scheme for the HVAC system

Table 31. Membership functions and min & max outputs of LAMDA & HVAC system

Range	Temp (T)			Hum (W)			Membership functions				
	T	Min	Max	W	Min	Max	NB	NS	ZE	PS	PB
[0, 1]	T ₁	6.484e+ 01	7.144e+ 01	W ₁	9.200e-03	9.200e-03	-1	-0.5	0	0.5	1
[0, 5]	T ₂	6.498e+ 01	7.174e+ 01	W ₂	9.200e-03	9.200e-03	-5	-2.5	0	2.5	5
[0, 10]	T ₃	6.501e+ 01	7.185e+ 01	W ₃	9.200e-03	9.200e-03	-10	-5	0	5	10

Figure 44 (a) and (b) display the output signals for temperature and humidity, respectively, for the entire control of the MIMO HVAC system using the LAMDA controller for three different ranges of MFs ([0, 1], [0, 5], [0, 10]). The initial experiment aimed to compare the effectiveness of different membership function ranges to demonstrate the LAMDA controller's robustness. It is clear from Table 31, which shows the MFs and their classifications for the three sets of this experiment, that small descriptor values yield better output signals for temperature (T) and humidity (W), as shown in Figure 44 (a) and (b), respectively. The objective of using the LAMDA optimization for the MIMO of HVAC is to maintain the temperature (Temp) at 71 to 65 [F°] and the relative humidity (Hum) at 0.0092 [lb/lb] based on the

reference point. In other words, the results in Table 31 and Figure 44 show that for the temperature $[0, 1]$ membership function range (T_1) provides the most stable tracking around the initial reference of 71 °F, exhibiting the smallest overshoot and smoothest settling. In contrast, the $[0, 10]$ range (T_3) responds fastest to the step change toward 65 °F but does so with more aggressive behavior, while $[0, 5]$ (T_2) offers a balanced, intermediate performance. The humidity results show that all three membership-function ranges converge smoothly to the reference value of 0.0092 lb/lb, with only negligible differences in transient behavior. The $[0, 1]$ range (W_1) settles slightly faster and with the least deviation, while $[0, 5]$ (W_2) and $[0, 10]$ (W_3) follow almost identically, confirming that the LAMDA controller maintains robust humidity regulation across all descriptor ranges.

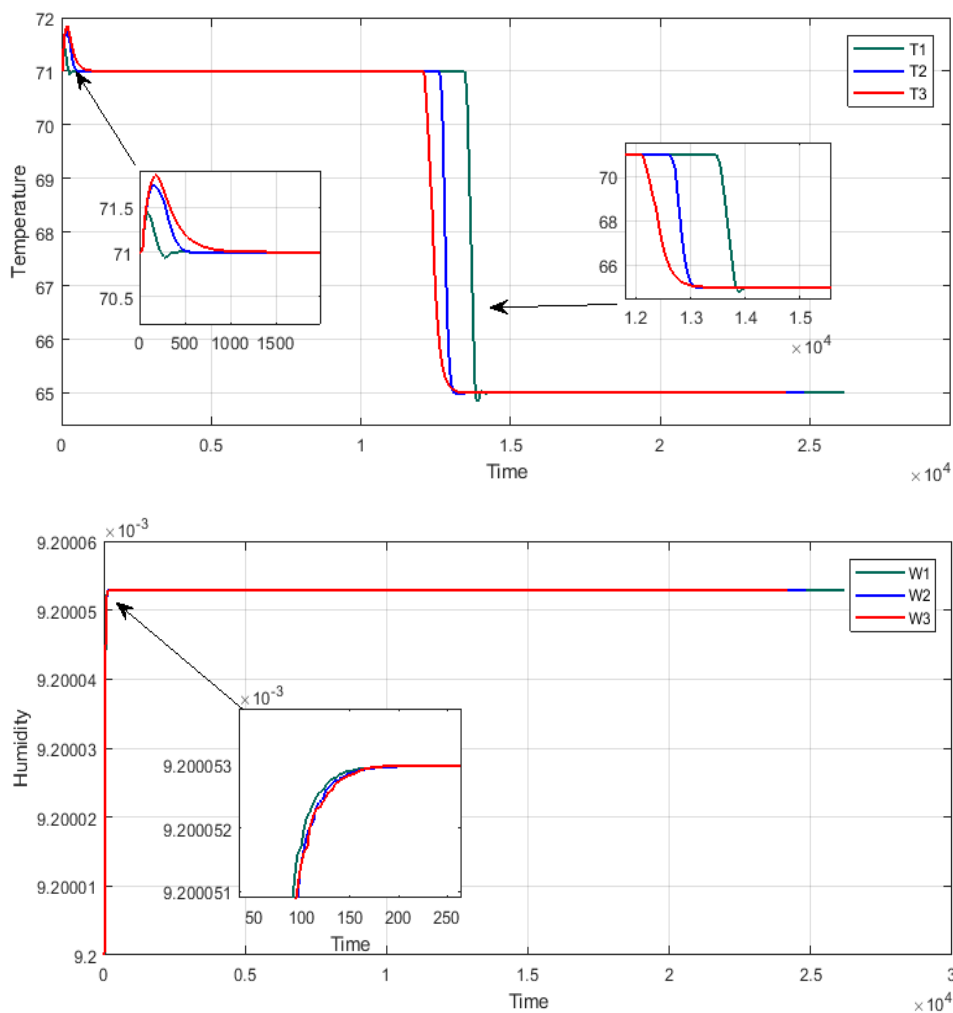


Figure 44. Output signals of (a) temperature and (b) humidity

5.3 Evaluation of the BB-BC-LAMDA Control Unit

The BB-BC-LAMDA control unit was implemented and evaluated, taking into account the limitation of the classes (descriptor range). The experiments demonstrate that it yields efficient results and ensures the stability of the MIMO HVAC system for the tuning parameters. Moreover, the tests demonstrate how the BB-BC algorithm efficiently finds global minima in a few iterations. The results of the BB-BC technique in adjusting the parameters (NB, NS, ZE, PS, PB) of the LAMDA controller for the MIMO HVAC system are discussed below. Table 32 shows the parameters used by the BB-BC to optimize the MFs (classes) for the experiments performed.

Table 32. BB-BC algorithm parameters for LAMDA controller optimization

Algorithm	Parameters	Value
BB-BC	Number of particles	10
	Number of iterations	100, 200, 500, 1000
	Boundaries of (NS, NB, ZE, PS, PB)	0 – 10
	Boundaries of α	0.01– 0.1
	Number of LAMDA parameters	5

The time response parameters (overshoot, rise time and settling time) of the proposed BB-BC-LAMDA control system for temperature and humidity are presented in Table 33 and Table 34, respectively. It is evident that the BB-BC algorithm can achieve rapid convergence. The LAMDA controller constants, $kp_1, ki_1, kd_1, kp_2, ki_2$ and kd_2 are equal to 0.02, 0.0005, 200, 1, 10, and 0.5, respectively. In the literature, numerous works have explored the LAMDA algorithm design, enhancing it as an optimal controller [61, 150, 161]. By integrating BB-BC as an optimization unit, we achieve satisfactory performance.

Table 33. Time response for temperature with different iteration numbers

Search space	No of Iteration	Temperature (T) in ms		
		O.shoot	R.Time	S.Time
[0, 10]	100	0.575%	76.220	70.920
	200	0.575%	76.168	70.710
	500	0.575%	68.135	63.590
	1000	0.571%	66.857	62.070

Table 34. Time response for humidity with different iteration numbers

Search space	No of Iteration	Humidity (W) in ms		
		O.shoot	R.Time	S.Time
[0, 10]	100	0.501%	123.400	172.010
	200	0.501%	143.440	172.130
	500	0.501%	123.430	172.090
	1000	0.501%	123.400	172.010

Table 35 provides a comprehensive comparison of the time responses between the original LAMDA and the LAMDA enhanced with the GA, PSO, and BB-BC algorithms, reinforcing our findings and approach (for 200 iterations).

Table 35. Time response: original LAMDA vs. optimization algorithms (GA, PSO, and BB-BC)

Algorithm	Time response in ms					
	Temperature (T)			Humidity (W)		
	O.shoot	R.Time	S.Time	O.shoot	R.Time	S.Time
Original LAMDA	0.505 %	263.333	02.591	0.501 %	123.437	172.054
GA-LAMDA	0.581 %	085.752	79.533	0.501 %	123.438	172.169
PSO-LAMDA	0.581 %	104.880	97.102	0.502 %	123.411	172.089
BB-BC-LAMDA	0.568 %	065.203	60.995	0.501 %	123.405	172.012

The parameter values (NS, NB, ZE, PS, PB) for the LAMDA controller inputs, along with the corresponding fitness values, are presented in Table 36. The membership values were selected based on the [0, 10] range of the experiments across various numbers of iterations (100, 200, 500, 1000). The *ITAE* was used as the objective function for evaluating the output. Comparing the output signal of the BB-BC-LAMDA for a MIMO HVAC system (shown in Table 36) with a LAMDA-based (w/o BB-BC) HVAC system (shown in Table 31), demonstrates the effectiveness and robustness of the BB-BC algorithm as a method to tune the LAMDA controller.

Table 36. Number of iterations, MFs and constant gains of the LAMDA controller

No of Iteration	Membership Functions (MFs)					Output Signal evaluation			
	NB	NS	ZE	PS	PB	Ref (T)	Out (T)	Ref (W)	Out (W)
100	-3.881	-0.872	0	0.872	3.881	71.0	7.101e-03	9.200e-03	9.2001e-03
200	-7.275	-0.473	0	0.473	7.275	71.0	7.101e-03	9.200e-03	9.2001e-03
500	-8.849	-0.300	0	0.300	8.849	71.0	7.101e-03	9.200e-03	9.2001e-03
1000	-9.172	-0.174	0	0.174	9.172	71.0	7.101e-03	9.200e-03	9.2001e-03

Based on the number of iterations, Figure 45 shows that the step response curves for the humidity (W) output signal are very similar, exhibiting slight overshoot, a quick rise time, and a small settling time. In contrast, the step responses of temperature (T) are faster, with smaller overshoot and rise times. Simulation results demonstrate that the membership range of 0 to 1 is suitable for the inputs of the LAMDA controller for the MIMO HVAC system. The step size control, ranging from 0.1 to 0.001, is more accurate and effective.

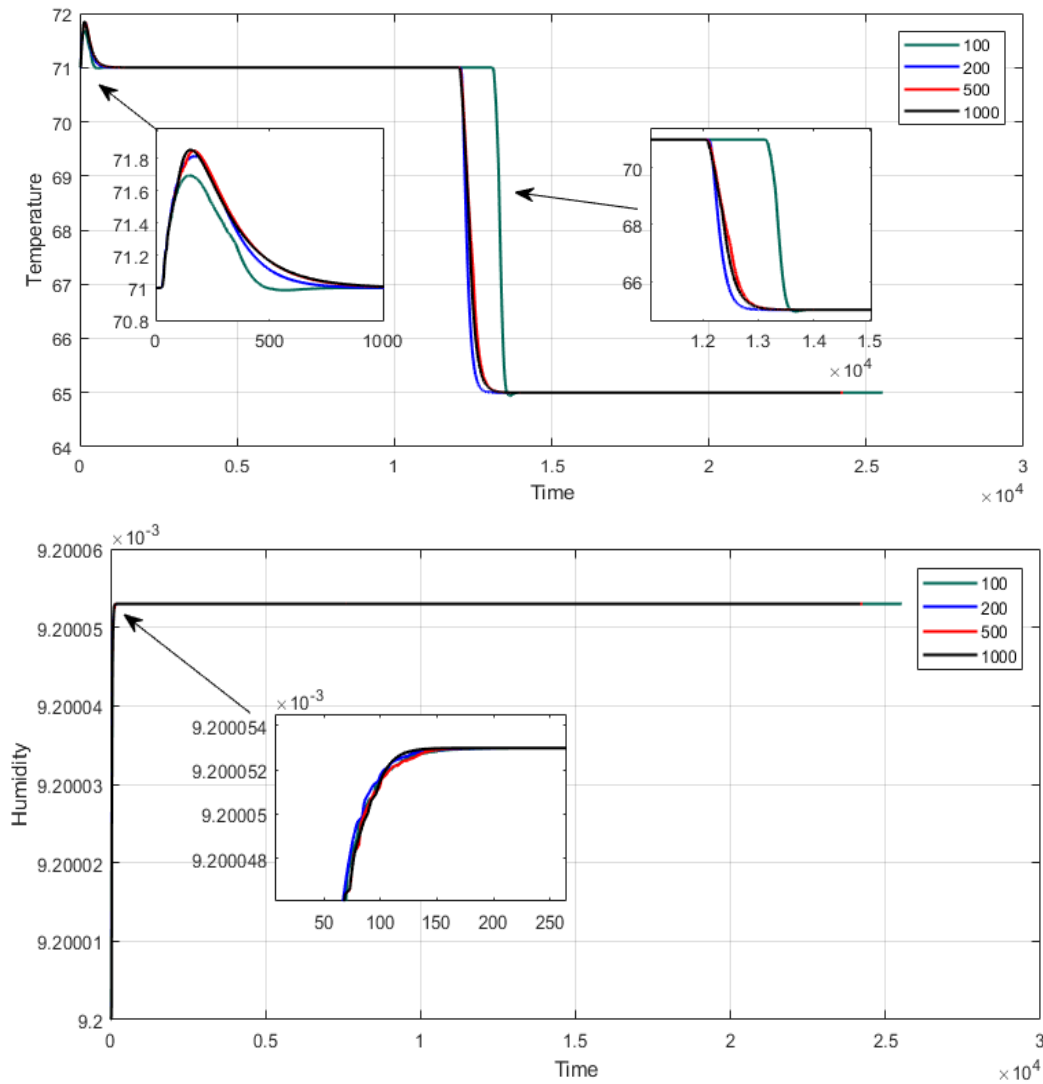


Figure 45. The step response curves of (a) temperature and (b) humidity

5.4 FPGA-based Implementation of the System

The hardware (FPGA) implementation significantly increases the effectiveness of the proposed approach for real-time applications. Our design includes two subsystems: the LAMDA controller integrated with the BB-BC algorithm, and the HVAC subsystem. The entire system has been implemented on an FPGA device.

5.4.1 FPGA-based BB-BC engine

The BB-BC engine architecture consists of three main blocks (processing stages) that implement the iterations of the algorithm in a fully pipelined fashion: the random number

generator (RNG), the fitness function evaluation (FiT), and the center-of-mass calculation (CoM). The stages of the BB-BC engine are described in Chapter 3. Table 37 shows the main arithmetic formulas and parameters of the FPGA BB-BC-LAMDA architecture. It highlights how mathematical formulations, hardware choices, and optimization parameters influence computational efficiency, convergence performance, and control accuracy.

Table 37. Hardware BB-BC-LAMDA architecture

	Module	Mathematical formula	Values	Comments
BB-BC algorithm	CoM Function	$x^c = \frac{\sum_{i=1}^N \frac{1}{f^i} x^i}{\sum_{i=1}^N \frac{1}{f^i}}.$	(a) minimal cost function (MCF) (b) complex inverse fitness function, complex CoM equation (CCE)	(a) reduces system complexity, while (b) improves convergence speed/accuracy
	Arithmetic Circuits	Customized Arithmetic Units	(a) simple non-pipelined HDL operators (b) use of highly optimized IP blocks	Option (b) is selected for speed optimization
	Fitness Function	ITAE	(a) Problem-dependent	Future versions will support dynamic reconfiguration of fitness function
	BB-BC Parameters	The parameters we have used in our design	(a) number of iterations (M) (b) population size (N) (c) number of candidate features (L) (d) LFSR seeds	Algorithm parameters are configured statically. Future versions will support dynamic reconfiguration of all parameters
LAMDA algorithm	Descriptor	$\min(\max(\text{classed}), \max(\min(\text{classed}), [e \text{ de}]))$	(a) stored in two arrays [25*1] (b) error and derivative of error (e, de).	(a) Select classes from RAM for parallel processing. (b) Training phase using (e, de) processing
	Fuzzification	$cd = \frac{\text{classed} - \min(\text{classed})}{\max(\text{classed}) - \min(\text{classed})}$ $MAD = \exp(-0.5*na/nb)$	(a) classification (b) normalization (c) marginal adequacy degree (MAD)	(a) The first phase is the training (b) To improve the performance (c) To evaluate fuzzy membership function (Taylor Series)
	Inference	$w_v = [1-\alpha, \alpha]$ $GAD = OWA(MAD, w)$	(a) weighted calculation process (b) aggregation process	(a) Can be optimized in the future (b) Using DSP48Es for calculating ordered weighted averaging (OWA)
	Defuzzification	$uu = GAD * od$ $ud = \text{sum}(uu)*abs(Beta),$	(a) output control (uu, ud)	(a) od : to prevent the controller's output from exceeding established limits.
	LAMDA Parameters	The parameters we have used in our design	(a) classes (NB, NS, ZE, PS, PB) (b) nb (c) $kp, ki, kd, na, \alpha, od, x, y, \text{and } go$	(a) Obtained from BB-BC (b) Obtained from normalization of classes (c) Constant values

5.4.2 LAMDA Controller Architecture

The LAMDA algorithm is decomposed into two basic operations, the fuzzification and the inference, which are described below:

▪ **Fuzzification:** it consists of three sub-units:

- *Classification:* it starts with the LAMDA MFs (classes), which are defined based on the knowledge and behavior of the HVAC system. These MFs are named Negative

Big (*NB*), Negative Small (*NS*), Zero (*ZE*), Positive Big (*PB*), and Positive Small (*PS*). The MF classification is performed in a $[25 \times 2]$ matrix. This matrix is composed of two arrays (vectors) of size $[25 \times 1]$ in a parallel processing operation, where both vectors are processed simultaneously, using a 16-bit fixed-point data type. The upper 4 bits represent the integer part, and the lower 12 bits represent the fractional part (the precision of the fixed-point representation is programmable to guarantee the minimum accuracy required by the target application).

- (ii) *Normalization*: In this step, the control of the power and search space range of the vectors are modified by multiplying them with the variables $[x \ y]$ to improve the performance by scaling vector used to resize the search space of the optimization algorithm.
- (iii) *Marginal Adequacy Degree (MAD)* unit calculates the *MAD* using probability density functions, and the pipelined log generators for 16-bit utilizing a combination of lookup and Taylor Series.
- *Inference*: it uses the aggregation operator *OWA* to calculate the *GAD*. An ordinary matrix is used to prevent the controller's output from exceeding specific values. The block diagram of the LAMDA controller architecture is shown in Figure 46. The data flow of the LAMDA algorithm starts when the timing analysis indicates that the controller is able to receive 1) inputs (memberships), which are outputs of the BB-BC algorithm, and 2) the error and derivative of the error (*e*, *de*), which are outputs of the error signal processing unit. The fuzzification module starts with the classification unit, which assigns memberships to a $[25 \times 2]$ matrix and formats them into two $[25 \times 1]$ arrays (vectors). The normalization unit, which normalizes the array to $[25 \times 1]$. To improve signal performance, the standard deviation (STD) is calculated through the average and dispersion unit. The last operation of the fuzzification module is *MAD*, which evaluates the fuzzy membership function using the Gaussian function, and the signal passes to the next main stage. The aggregate computation unit of the inference module receives the signal and calculates it using the *OWA* method. The inference mechanism uses the *GAD* for each class via *OWA*, then stores it in memory. Finally, the class assignment unit calculates the controller output (*U*).

We summarize the innovative hardware speed-boosting techniques suggested for the FPGA-based acceleration of the LAMDA algorithm as follows:

- All stages of the LAMDA algorithm were implemented in a fully pipelined fashion.
- Several pipelined LAMDA processes run in parallel, are synchronized, and collaborate with all modules.
- A mathematical approximation method was proposed to reduce the execution time of the complex fuzzification part.

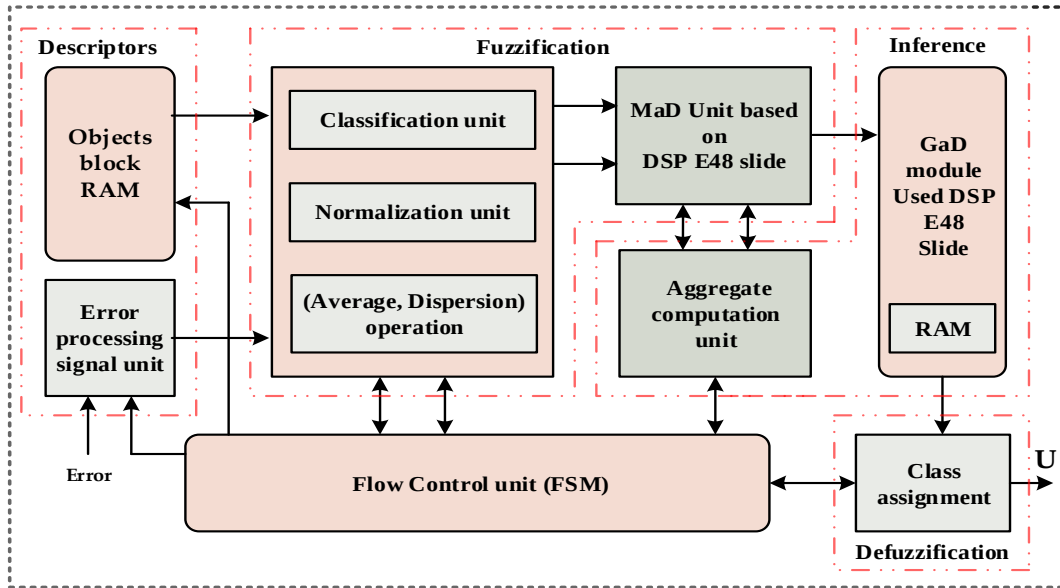


Figure 46. Block diagram of the LAMDA controller architecture

5.4.3 BB-BC-LAMDA Controller Design

Figure 47 depicts the main modules of the BB-BC-LAMDA control unit. The combined BB-BC & LAMDA control unit for the MIMO HVAC system can be divided into three modules. In the first module, the BB-BC block provides the candidates, i.e., the MFs (NB, NS, ZE, PS, PB), as input data to the second module, the LAMDA controller. The LAMDA controller receives and stores the data, then classifies it and calculates the MAD and the GAD . The third module includes the HVAC system, which receives as input the action (U_1) from the integrator module of the LAMDA controller for controlling the HVAC's temperature (T). The next step relies on the results from the first step after the training dataset (*the data that the BB-BC algorithm uses to train/evaluate new candidates*) is created. These results will be used to train the proposed new candidate provided by the BB-BC algorithm and to get the final output and the

optimal solution of the entire control system. The same steps apply to the LAMDA controller (U_2) for controlling the HVAC's humidity (W). Note that the HVAC system has been modeled in VHDL based on the state space equations of Eq. (2.29).

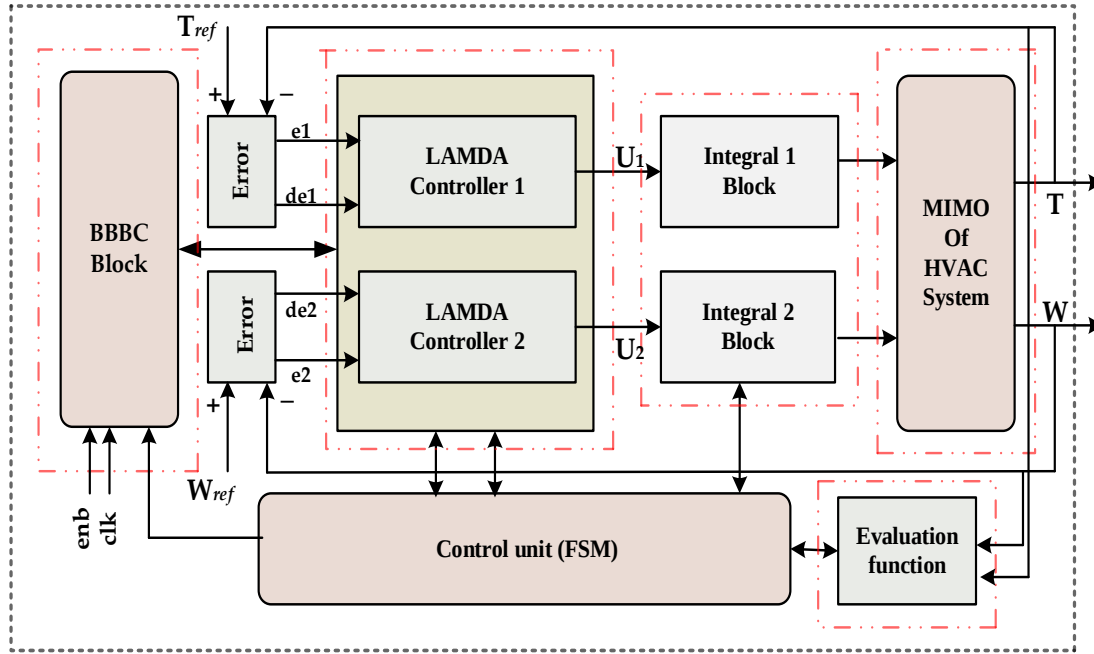


Figure 47. Design of the BB-BC-LAMDA for the MIMO HVAC system

In the hardware design, classic mechanisms, such as pipelining and parallel execution units, are used to improve system performance. Using parallel engines and pipelining on an FPGA greatly increases the throughput of the optimization algorithm by allowing multiple candidate solutions to be evaluated simultaneously while overlapping computation stages, resulting in much faster convergence and lower latency. Examples such as Duan, M., et al., included a parallel multiclassification algorithm for big data using an extreme learning machine [162], and used for both modules, i.e., BB-BC, that presented by Abdoalnasir, A., et al., [P5] and LAMDA, which significantly reduces the system response time. The latency and the arithmetic operators of the pipeline stages of the BB-BC-LAMDA are presented in Table 38. Assuming population size N and number of iterations M , the total execution time (ET) of the proposed FPGA for the ET-MCF-LAMDA and the ET-CCE-LAMDA (T_{AS} , T_M , T_D , T_R , T_C , T_L , and T_{LAMDA} are the Adder-Subtractor, Multiplier, Divider, Reciprocal, Comparator, and Look-Up-Table latencies) is calculated as follows:

$$ET-MCF-LAMDA_{Total} = M \times ((1 + T_M + T_{AS}) + T_F + T_C + N + (T_{LAMDA})) \quad (5.1)$$

$$ET-CCE-LAMDA_{Total} = M \times ((1 + T_M + T_{AS}) + T_F + (T_R + T_M + T_{AS}) + N + T_D + (T_{LAMDA})) \quad (5.2)$$

Table 38. BB-BC-LAMDA latency and arithmetic operations

Pipeline stage	Latency	Arithmetic operations
Candidate generation	$1 + T_M + T_{AS}$	L LFSR, L Multipliers, L Adders-Subtractors
Fitness calculation	T_F	Problem-dependent
CoM calculation	T_C for MCF $(T_R + T_M + T_{AS}) + T_D$ (*) for CCE	1 Comparator 1 Reciprocal, L Multipliers, $L+1$ Adders-Subtractors, L Dividers
Fuzzification	$((T_M + T_{AS}) + T_D) + T_L$	L Multipliers, L Adders-Subtractors, L Dividers
Interface	$(T_M + T_{AS})$	L Multipliers, L Adders-Subtractors
Defuzzification	$(T_M + T_{AS}) + T_L$	L Multipliers, L Adders-Subtractors, 1 LUT

The finite state machine (FSM) generates control signals to manage the data path modules and control data flow through the various stages. The main state machine of the BB-BC-LAMDA control unit is shown in Figure 48, and is based on two data-flow processes: the BB-BC flow (blue FSM) and the LAMDA flow (orange FSM). The BB-BC engine flow works as follows: a start signal triggers the FSM to enter (from the idle state, S_0) to the RNG state (S_1), where the BB-BC module provides the LAMDA controller with the initial candidates (MFs). When the LAMDA output is ready, the state machine goes to state S_2 , and the output is applied to the plant. At this step, the LAMDA FSM is activated and goes to the idle state (S_0). The BB-BC FSM enters the CoM state (S_3), and when the generation number is reached, it goes to the last state (S_4) before returning to the idle state. When the LAMDA FSM leaves the idle state, it enters S_1 , where it calculates the minimum and maximum values of the MFs. Then, it goes to the training operation stage (S_2). When the training values are ready to be used, the LAMDA module calculates the MAD (S_3). Next, it enters state S_4 , where the calculation operations are executed, and the HVAC system goes into normal operation. Before it goes back to the idle state, LAMDA FSM passes through state S_5 , and continues using the error and the derivative of the error (evaluation function) for new calculations. Finally, it repeats the process based on the iteration number.

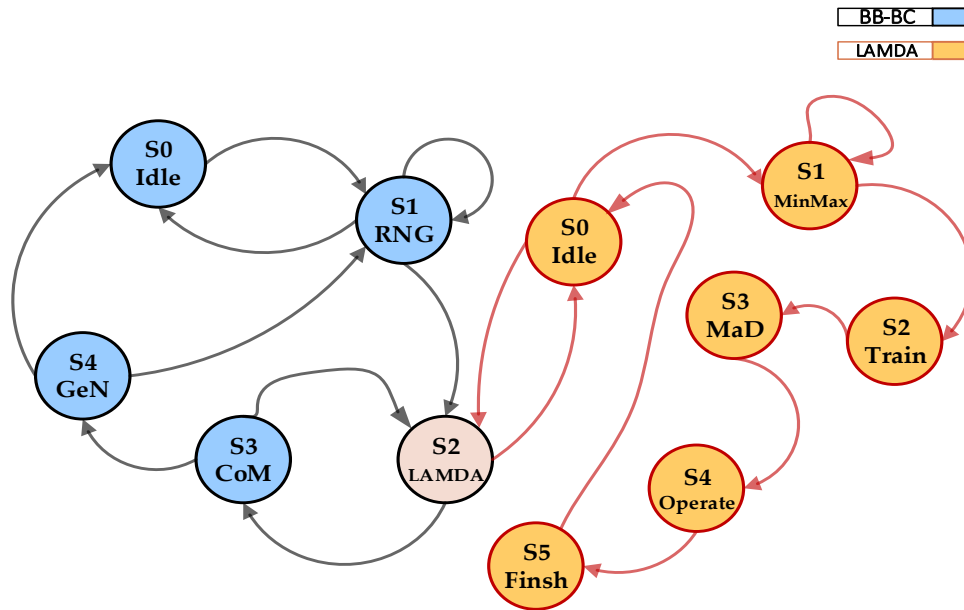


Figure 48. FSM of the BB-BC-LAMDA controller

5.4.4 Simulation Interpretation and Results

The FPGA design is highly flexible and configurable, as the MFs and inference rules can be altered by changing the VHDL code parameters (od, x, y, etc.). The feasibility of the hardware architecture of the BB-BC-LAMDA control unit has been demonstrated through analog simulations using Vivado tools and compared with MATLAB/Simulink. The digital architecture uses fixed-point arithmetic formats (8, 24, 32, 64) to represent input/output signals as well as inner operations. Table 39 summarizes device utilization for the BB-BC-LAMDA-HVAC control system. The design was successfully implemented on an AMD/Xilinx Virtex7 XC7VX330T FPGA evaluation platform, achieving an operating frequency of 100.1 MHz and requiring 74 clock cycles to compute one output sample (0.74 μ s). Note that the main reason for selecting this FPGA, despite its large resource capacity, is the wide search space range required by the LAMDA-BB-BC controller, which benefits from the higher DSP count, faster logic fabric, and larger on-chip bandwidth of the Virtex-7 family.

Table 39. FPGA resources utilization for the BB-BC-LAMDA design

Digital Elements	Resources	Used	%
Register only	408000	3928	0.960%
LUT	204000	5727	2.810%
DSP48s	1124	46	4.110%
No of bonded IOBs	600	259	43.170%

5.4.5 Limitations and considerations

The limitations of the proposed LAMDA design when employing optimization algorithms may include challenges related to algorithm complexity, hardware resource constraints, and the need for efficient implementation techniques to achieve desired performance levels. Despite the promising aspects of our approach, and considering the merging of two algorithms (BB-BC, LAMDA) and their computational complexity, several challenges emerge:

- As the number of parameters and elements in the clusters increases, the computational complexity rises, demanding more time and power. (Our proposed hardware scheme partly addressed this issue.)
- An additional drawback is finding the bounds range (*limitation of determining*) of the parameters (K_i, K_p, K_d), which we established experimentally and tested individually.

5.5 Summary

In this chapter, we presented a novel hardware-based LAMDA controller optimized by the BB-BC algorithm. The goal of using the BB-BC algorithm is to optimize the parameters (membership functions) of the LAMDA controller for an MIMO HVAC system. The experimental results demonstrate that the proposed BB-BC-LAMDA approach achieves faster optimization of system output, bringing it closer to the set-point. They also highlight improved performance with limited hardware resources and reduced power consumption compared to a software version implemented in MATLAB/Simulink. The FPGA-based solution speeds up the computing operations required by the LAMDA algorithm.

CHAPTER 6

Conclusions and outlook

6 Conclusions and Outlook

This thesis focuses on EAs, a crucial extension in the field of industrial application systems control and analysis. We conducted a thorough review of the literature on evolutionary algorithms. We introduced a novel design of the BB-BC algorithm to assess the performance of various controllers, including PID, FOPID, and LAMDA, each comprising several sub-processes.

6.1 Summary of the Work Done

The work done in this thesis can be summarized as follows:

Chapter 1 introduces EAs, emphasizing their nature-inspired optimization approach for solving complex problems across diverse domains, such as artificial intelligence, industrial control systems, and machine learning. Prominent EA types, including GA, PSO, ACO, and the BB-BC algorithm, are discussed with their unique strengths and real-world applications, particularly in FPGA-based implementations. Advances in EAs include hybrid techniques that combine EAs with ML, multi-objective optimization, and hardware acceleration via FPGAs and GPUs for real-time applications. This chapter highlights the BB-BC algorithm's significant role in optimizing HVAC control systems, emphasizing its advantages in rapid convergence and efficient hardware implementation. It outlines the thesis objectives, focusing on developing novel FPGA frameworks for the BB-BC algorithm to optimize controllers like PID, FOPID, and LAMDA, and addresses challenges in real-time industrial systems, and highlights the contributions of the thesis.

Chapter 2 describes the related technology and key points for integrating evolutionary algorithms into optimization controllers, especially for HVAC systems. It provides an in-depth overview of EAs, emphasizing the BB-BC algorithm and its integration with advanced control strategies, including PID, FOPID, and LAMDA-based controllers. Key components, such as the implementation of EAs on FPGAs, are explored. This chapter also highlights the application of these optimization techniques in HVAC systems that are modeled as SOPTD, and compares the performance of PID/FOPID and the implementation of the LAMDA-based controllers to demonstrate their effectiveness in a system optimization-based MIMO HVAC system.

Chapter 3 presents a novel FPGA implementation of the BB-BC optimization algorithm and demonstrates that the proposed BB-BC approach outperforms classical GA approaches. The proposed scheme integrates an efficient, fully pipelined FPGA design that overcomes the limitations of traditional GAs. Experimental results show a significant speedup compared to software versions, positioning the FPGA-based BB-BC scheme as an ideal solution for real-time embedded applications, applicable to ANFIS and various fitness functions.

Chapter 4 presents a technique for fast tuning the PID controller in a second-order HVAC system, utilizing the BB-BC algorithm that is accelerated through FPGA technology. Using the FIL technique, experimental results demonstrate the efficiency of the proposed method in terms of optimization accuracy and convergence speed compared to other approaches, such as software implementation of BB-BC, GA, and PSO for PID parameters tuning. Moreover, this chapter presents an FPGA implementation of an FOPID controller for HVAC systems, enhancing dynamic performance and system robustness. The tuning challenge of FOPID controllers is tackled using the BB-BC evolutionary algorithm, yielding improved control parameters and reduced oscillations in HVAC output signals. Its pipelined implementation on an Xilinx FPGA achieves high performance and significant reductions in control signal and error with optimal tuning.

Chapter 5 presents a MIMO control system that utilizes the BB-BC optimization algorithm and the LAMDA algorithm. The system was developed using the calculation function (aggregation function) OWA. The entire system was implemented on an FPGA to improve speed and energy efficiency. Experimental results confirmed the feasibility of the proposed optimization strategy.

Chapter 6 concludes the thesis summarizing the work done, the contributions and the orientations for future research.

6.2 Summary of the Contributions

The main contributions of this thesis are summarized in bullets:

- Initially, the optimization algorithm BB-BC was implemented using FPGA, offering a novel strategy for optimization.

- Based on the results obtained from our experiments and the current state of the art, a generic hardware design framework was presented from both a technical and algorithmic perspective. This framework addresses the complexities associated with mathematical operation functions (Rosenbrock, unimodal, Sphere) and optimization problems in control units. It also enhances the transient stability of control systems when subjected to pulse disturbances at various time points.
- The hardware-based BB-BC scheme enables the adjustment of tuning parameters of various controllers in multiple applications. The proposed approach was evaluated in optimizing the PID, FOPID, and LAMDA controllers. A thorough comparison was conducted between BB-BC, GA, PSO, and the H_∞ algorithm.

6.3 Future Work

This thesis addresses the challenges of real-time optimization in industrial applications, focusing on the implementation of the BB-BC algorithm using FPGA technology. This approach is expected to yield valuable insights, warranting further research and development to effectively address the stability requirements of optimization problems.

Future work involves adapting the proposed FPGA architecture in heterogeneous programmable SoCs, such as the Xilinx Zynq-7000 APSoC, in order to build a more flexible and configurable framework for accelerating the BB-BC algorithm, which will combine the processing power of embedded application processors and SIMD coprocessors with the benefits of programmable logic. Doing this, the system designers can more easily integrate the BB-BC accelerator into large-scale optimization problems and provide integrated and effective acceleration solutions for demanding real-time applications. The proposed FPGA architecture can be applied in real-time engineering problems, such as path planning of autonomous mobile robots, intelligent control systems in nuclear plants, and object detection in remote sensing applications. Moreover, future work could attempt to enhance the BB-BC scheme efficiency applying specific modifications, such as incorporating chaotic maps to generate the initial population instead of relying on random numbers. Such modifications will aim to reduce dependency on extensive source spaces and improve overall performance. For the

LAMDA controller, the system can be expanded by increasing the number of membership functions (classes) and fuzzy logic rules, as well as exploring a broader search space. Additionally, incorporating real-world constraints into the models is expected to further refine the optimization process.

CHAPTER 7

References

7 References

1. Gad, A.G., *Particle swarm optimization algorithm and its applications: a systematic review*. Archives of computational methods in engineering, 2022. **29**(5): p. 2531–2561.
2. Katoch, S., S.S. Chauhan, and V. Kumar, *A review on genetic algorithm: past, present, and future*. Multimedia Tools and Applications, 2021. **80**(5): p. 8091–8126.
3. Marini, F. and B. Walczak, *Particle swarm optimization (PSO). A tutorial*. Chemometrics and Intelligent Laboratory Systems, 2015. **149**: p. 153–165.
4. Dorigo, M., M. Birattari, and T. Stutzle, *Ant colony optimization*. IEEE computational intelligence magazine, 2006. **1**(4): p. 28–39.
5. Shan, D., et al., *Path-Planning Strategy: Adaptive Ant Colony Optimization Combined with an Enhanced Dynamic Window Approach*. Electronics, 2024. **13**(5): p. 825.
6. Kumar, V. and S. Sharma, *Extensive Review of Big Bang Big Crunch Optimization Algorithm*. Research Journal of Engineering Sciences ISSN. **2278**: p. 9472.
7. Qin, J., et al., *Deep learning-driven particle swarm optimisation for additive manufacturing energy optimisation*. Journal of Cleaner Production, 2020. **245**: p. 118702.
8. Cavallaro, C., et al., *Machine Learning and Genetic Algorithms: A case study on image reconstruction*. Knowledge-Based Systems, 2024. **284**: p. 111194.
9. Kumar, D.G., et al., *Evolutionary Algorithms for Real Time Engineering Problems: A Comprehensive Review*. Ingénierie des Systèmes d'Inf., 2021. **26**(2): p. 179–190.
10. Ghamisi, P. and J.A. Benediktsson, *Feature selection based on hybridization of genetic algorithm and particle swarm optimization*. IEEE Geoscience and remote sensing letters, 2014. **12**(2): p. 309–313.
11. Fonseca, C.M. and P.J. Fleming, *An overview of evolutionary algorithms in multiobjective optimization*. Evolutionary computation, 1995. **3**(1): p. 1–16.
12. Indrusiak, L.S., R.I. Davis, and P. Dziuranski, *Evolutionary optimisation of real-time systems and networks*. arXiv preprint arXiv:1905.01888, 2019.
13. Rahamneh, S.Z., *Efficient Hardware/Software Partitioning Techniques for a Cloud-scale CPU-FPGA Platform*. 2020: Western Michigan University.
14. Sun, X., et al., *Machine Learning-Evolutionary Algorithm Enabled Design for 4D-Printed Active Composite Structures*. Advanced Functional Materials, 2022. **32**(10): p. 2109805.
15. Ali, Z., H.M. Kiran, and W. Shahzad, *Evolutionary Algorithms for Query Optimization in Distributed Database Systems: A review*. 2018.
16. Fathi, M., et al., *Artificial intelligence and evolutionary algorithms-based optimization*. Optimization in Electrical Engineering, 2019: p. 137–169.
17. Larsen, L. and J. Kim, *Path planning of cooperating industrial robots using evolutionary algorithms*. Robotics and Computer-Integrated Manufacturing, 2021. **67**: p. 102053.
18. Alam, T., et al., *Genetic algorithm: Reviews, implementations, and applications*. arXiv preprint arXiv:2007.12673, 2020.
19. Koyuncu, H. and R. Ceylan, *A PSO based approach: Scout particle swarm algorithm for continuous global optimization problems*. Journal of Computational Design and Engineering, 2019. **6**(2): p. 129–142.
20. Eskandari, L., et al., *A modified and enhanced ant colony optimization algorithm for traveling salesman problem*. Mathematical Methods in Engineering: Theoretical Aspects, 2019: p. 257–265.

21. Kiziloluk, S. and A.B. Özer, *Hybrid parliamentary optimization and big bang-big crunch algorithm for global optimization*. Turkish Journal of Electrical Engineering and Computer Sciences, 2019. **27**(3): p. 1954–1969.
22. Razvarz, S., et al., *Flow control of fluid in pipelines using PID controller*. IEEE Access, 2019. **7**: p. 25673–25680.
23. Tepljakov, A., et al., *FOPID controllers and their industrial applications: A survey of recent results*. IFAC-PapersOnLine, 2018. **51**(4): p. 25–30.
24. Podlubny, I., *Fractional-order systems and PI/sup/spl lambda//D/sup/spl mu//-controllers*. IEEE Transactions on automatic control, 1999. **44**(1): p. 208–214.
25. Sadati, N., M. Zamani, and P. Mohajerin. *Optimum design of fractional order PID for MIMO and SISO systems using particle swarm optimization techniques*. in *2007 IEEE International Conference on Mechatronics*. 2007. IEEE.
26. Lino, P., et al., *Synthesis of fractional-order PI controllers and fractional-order filters for industrial electrical drives*. IEEE/CAA Journal of Automatica Sinica, 2017. **4**(1): p. 58–69.
27. Jayaswal, K., D. Palwalia, and S. Kumar, *Performance investigation of PID controller in trajectory control of two-link robotic manipulator in medical robots*. Journal of Interdisciplinary Mathematics, 2021. **24**(2): p. 467–478.
28. Aziz, K., *A Comparative Study on the Tuning of the PID Flight Controllers Using Swarm Intelligence*. International Journal of Aviation Science and Technology, 2020. **1**(02): p. 80–91.
29. Li, S., et al., *A fractional order PID controller using MACOA for indoor temperature in air-conditioning room*. Journal of Building Engineering, 2021. **44**: p. 103295.
30. Morales, L., et al., *LAMDA control approaches applied to trajectory tracking for mobile robots*. IEEE Access, 2021. **9**: p. 37179–37195.
31. Morales, L., et al. *An intelligent controller based on LAMDA*. in *2019 IEEE 4th Colombian Conference on Automatic Control (CCAC)*. 2019. IEEE.
32. Erol, O.K. and I. Eksin, *A new optimization method: big bang–big crunch*. Advances in Engineering Software, 2006. **37**(2): p. 106–111.
33. Kumbasar, T. and H. Hagsras, *Big Bang–Big Crunch optimization based interval type-2 fuzzy PID cascade controller design strategy*. Information Sciences, 2014. **282**: p. 277–295.
34. Kaveh, A. and S. Talatahari, *Size optimization of space trusses using Big Bang–Big Crunch algorithm*. Computers & structures, 2009. **87**(17-18): p. 1129–1140.
35. Kaveh, A. and S. Talatahari, *Optimal design of Schwedler and ribbed domes via hybrid Big Bang–Big Crunch algorithm*. Journal of Constructional Steel Research, 2010. **66**(3): p. 412–419.
36. Camp, C.V. and A. Akin, *Design of retaining walls using big bang–big crunch optimization*. Journal of Structural Engineering, 2012. **138**(3): p. 438–448.
37. Kumbasar, T., et al., *Adaptive fuzzy model based inverse controller design using BB-BC optimization algorithm*. Expert Systems with Applications, 2011. **38**(10): p. 12356–12364.
38. Yesil, E., *Interval type-2 fuzzy PID load frequency controller using Big Bang–Big Crunch optimization*. Applied Soft Computing, 2014. **15**: p. 100–112.
39. Monmasson, E. and M.N. Cirstea, *FPGA design methodology for industrial control systems—A review*. IEEE transactions on industrial electronics, 2007. **54**(4): p. 1824–1842.
40. Yang, H., et al., *Review of advanced FPGA architectures and technologies*. Journal of Electronics (China), 2014. **31**(5): p. 371–393.

41. Kok, J., L.F. Gonzalez, and N. Kelson, *FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning*. IEEE transactions on evolutionary computation, 2012. **17**(2): p. 272–281.
42. Eiben, A.E. and J.E. Smith, *Introduction to evolutionary computing*. 2015: Springer.
43. Åström, K.J. and T. Hägglund, *PID controllers: theory, design, and tuning*. 1995: ISA-The Instrumentation, Systems and Automation Society.
44. Chen, Y., I. Petras, and D. Xue. *Fractional order control-a tutorial*. in *2009 American control conference*. 2009. IEEE.
45. Podlubny, I., *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*. 1998: elsevier.
46. Zhang, J. and W. Cao. *Analysis of PID control based on the balance equation*. in *2017 36th Chinese Control Conference (CCC)*. 2017. IEEE.
47. Podlubny, I., et al., *Analogue realizations of fractional-order controllers*. Nonlinear dynamics, 2002. **29**(1): p. 281–296.
48. El-Khazali, R., I.M. Batiha, and S. Momani. *Approximation of Fractional-Order Operators*. in *International workshop on Advanced Theory and Applications of Fractional Calculus*. 2018. Springer.
49. Maione, G., *High-speed digital realizations of fractional operators in the delta domain*. IEEE Transactions on Automatic Control, 2010. **56**(3): p. 697–702.
50. Merrikh-Bayat, F., N. Mirebrahimi, and M.R. Khalili, *Discrete-time fractional-order PID controller: Definition, tuning, digital realization and some applications*. International journal of control, automation and systems, 2015. **13**(1): p. 81–90.
51. Sundriyal, N., P. Yadav, and M. Chaturvedi. *A Novel Approach for Designing PID Controller for Set-Point Tracking for a HVAC Process*. in *Proceeding of International Conference on Intelligent Communication, Control and Devices*. 2017. Springer.
52. Abumeteir, H.A. and A.M. Vural, *Design and Optimization of Fractional Order PID Controller to Enhance Energy Storage System Contribution for Damping Low-Frequency Oscillation in Power Systems Integrated with High Penetration of Renewable Sources*. Sustainability, 2022. **14**(9): p. 5095.
53. Gull, C.Q., J. Aguilar, and R. García. *Study of Explainability Analysis Methods for the LAMDA Family Algorithms in Classification and Clustering Tasks*. in *2024 International Joint Conference on Neural Networks (IJCNN)*. 2024. IEEE.
54. Morales, L., et al., *Experimental comparison of the diagnostic capabilities of classification and clustering algorithms for the QoS management in an autonomic IoT platform*. Service Oriented Computing and Applications, 2019. **13**(3): p. 199–219.
55. Doncescu, A., J. Aguilar-Martin, and J.-C. Atine, *Image color segmentation using the fuzzy tree algorithm T-LAMDA*. Fuzzy Sets and Systems, 2007. **158**(3): p. 230–238.
56. Yager, R.R., *On ordered weighted averaging aggregation operators in multicriteria decisionmaking*. IEEE Transactions on systems, Man, and Cybernetics, 1988. **18**(1): p. 183–190.
57. Rojas-Muñoz, L.F., et al., *Embedded system implementation of an evolutionary algorithm for circle detection on programmable devices*. Computers and Electrical Engineering, 2022. **99**: p. 107714.
58. Noordin, N.H., P.S. Eu, and Z. Ibrahim, *FPGA Implementation of Metaheuristic Optimization Algorithm*. E-Prime-Advances in Electrical Engineering, Electronics and Energy, 2023. **6**: p. 100377.

59. Cho, J., et al., *Energy-cost analysis of HVAC system for office buildings: Development of a multiple prediction methodology for HVAC system cost estimation*. Energy and Buildings, 2018. **173**: p. 562–576.
60. Arguello-Serrano, B. and M. Velez-Reyes, *Nonlinear control of a heating, ventilating, and air conditioning system with thermal load estimation*. IEEE Transactions on Control Systems Technology, 1999. **7**(1): p. 56–63.
61. Escobar, L.M., et al., *Advanced fuzzy-logic-based context-driven control for HVAC management systems in buildings*. IEEE Access, 2020. **8**: p. 16111–16126.
62. Maasoumy, M. and A. Sangiovanni-Vincentelli, *Total and peak energy consumption minimization of building HVAC systems using model predictive control*. IEEE Design & Test of Computers, 2012. **29**(4): p. 26–35.
63. Wang, L., et al., *Fuzzy self-tuning PID control of the operation temperatures in a two-staged membrane separation process*. Journal of Natural Gas Chemistry, 2008. **17**(4): p. 409–414.
64. Dounis, A.I. and C. Caraiscos, *Advanced control systems engineering for energy and comfort management in a building environment—A review*. Renewable and Sustainable Energy Reviews, 2009. **13**(6-7): p. 1246–1261.
65. Scott, S.D., A. Samal, and S. Seth. *HGA: A hardware-based genetic algorithm*. in *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*. 1995.
66. Tang, W. and L. Yip. *Hardware implementation of genetic algorithms using FPGA*. in *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS'04*. 2004. IEEE.
67. dos Santos, P.V., J.C. Alves, and J.C. Ferreira. *A framework for hardware cellular genetic algorithms: An application to spectrum allocation in cognitive radio*. in *2013 23rd International Conference on Field programmable Logic and Applications*. 2013. IEEE.
68. Tachibana, T., et al. *General architecture for hardware implementation of genetic algorithm*. in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*. 2006. IEEE.
69. Fernando, P.R., et al., *Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine*. IEEE Transactions on Evolutionary Computation, 2009. **14**(1): p. 133–149.
70. Shackelford, B., et al., *A high-performance, pipelined, FPGA-based genetic algorithm machine*. Genetic Programming and Evolvable Machines, 2001. **2**(1): p. 33–60.
71. Guo, L., et al. *Pipelined genetic propagation*. in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. 2015. IEEE.
72. Guo, L., et al., *Parallel genetic algorithms on multiple FPGAs*. ACM SIGARCH Computer Architecture News, 2016. **43**(4): p. 86–93.
73. Tang, W.K.-S. and L. Yip, *Hardware implementation of genetic algorithms using FPGA*. The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS '04., 2004. **1**: p. I–549.
74. Tachibana, T., et al. *A hardware implementation method of multi-objective genetic algorithms*. in *2006 IEEE International Conference on Evolutionary Computation*. 2006. IEEE.
75. Jin, Y., *A comprehensive survey of fitness approximation in evolutionary computation*. Soft computing, 2005. **9**(1): p. 3–12.

76. Shi, L. and K. Rasheed, *A survey of fitness approximation methods applied in evolutionary algorithms*, in *Computational intelligence in expensive optimization problems*. 2010, Springer. p. 3–28.
77. Jang, J.-S., *ANFIS: adaptive-network-based fuzzy inference system*. IEEE transactions on systems, man, and cybernetics, 1993. **23**(3): p. 665–685.
78. Guo, L., D.B. Thomas, and W. Luk. *Automated framework for general-purpose genetic algorithms in FPGAs*. in *European Conference on the Applications of Evolutionary Computation*. 2014. Springer.
79. Zhuo, L., G.R. Morris, and V.K. Prasanna, *High-performance reduction circuits using deeply pipelined operators on FPGAs*. IEEE Transactions on Parallel and Distributed Systems, 2007. **18**(10): p. 1377–1392.
80. Nambiar, V.P., et al., *HW/SW co-design of reconfigurable hardware-based genetic algorithm in FPGAs applicable to a variety of problems*. Computing, 2013. **95**(9): p. 863–896.
81. Letras, M., A. Morales-Reyes, and R. Cumplido, *A scalable and customizable processor array for implementing cellular genetic algorithms*. Neurocomputing, 2016. **175**: p. 899–910.
82. Huang, M. and D. Andrews, *Modular design of fully pipelined reduction circuits on FPGAs*. IEEE Transactions on Parallel and Distributed Systems, 2012. **24**(9): p. 1818–1826.
83. Tahmasebi, S., M.A. Khanesar, and M. Teshnehlab. *Fuzzy reference model for adaptive indirect Takagi-Sugeno model reference control*. in *2016 IEEE 13th International Conference on Networking, Sensing, and Control (ICNSC)*. 2016. IEEE.
84. Jin, Y., J. Jiang, and J. Zhu, *Neural network based fuzzy identification and its application to modeling and control of complex systems*. IEEE transactions on systems, man, and cybernetics, 1995. **25**(6): p. 990–997.
85. Matlab manual, F.L.T.U.s.G. 2009.
86. Jaiswal, M.K. and H.K.-H. So. *DSP48E efficient floating point multiplier architectures on FPGA*. in *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*. 2017. IEEE.
87. Winterstein, F., S. Bayliss, and G.A. Constantinides. *High-level synthesis of dynamic data structures: A case study using Vivado HLS*. in *2013 International conference on field-programmable technology (FPT)*. 2013. IEEE.
88. Saldaña, H.J.B. and C.S. Cárdenas. *Design and implementation of an adaptive neuro-fuzzy inference system on an FPGA used for nonlinear function generation*. in *2010 IEEE ANDESCON*. 2010. IEEE.
89. Daida, J.M., et al., *Algorithm discovery using the genetic programming paradigm: Extracting low-contrast Curvilinear Features from SAR images of Arctic ice*. Advances in genetic Programming, 1996. **2**: p. 417–442.
90. Akashi, T., et al., *Using genetic algorithm for eye detection and tracking in video sequence*. Journal of Systemics, Cybernetics and Informatics, 2007. **5**(2): p. 72–78.
91. Kim, E.Y. and S.H. Park, *Automatic video segmentation using genetic algorithms*. Pattern Recognition Letters, 2006. **27**(11): p. 1252–1265.
92. Mohamed Ben Ali, Y., *Edge-based segmentation using robust evolutionary algorithm applied to medical images*. Journal of Signal Processing Systems, 2009. **54**(1): p. 231–238.
93. Ghosh, P., et al., *Incorporating priors for medical image segmentation using a genetic algorithm*. Neurocomputing, 2016. **195**: p. 181–194.

94. Li, C., et al., *Fast FPGA prototyping for real-time image processing with very high-level synthesis*. Journal of Real-Time Image Processing, 2019. **16**(5): p. 1795–1812.
95. Shamsi, M., et al. *Fast algorithm for iris localization using Daugman circular integro differential operator*. in *2009 International Conference of Soft Computing and Pattern Recognition*. 2009. IEEE.
96. Feng, L. and P. Qiu, *Difference detection between two images for image monitoring*. Technometrics, 2018. **60**(3): p. 345–359.
97. Cominos, P. and N. Munro, *PID controllers: recent tuning methods and design to specification*. IEE Proceedings-Control Theory and Applications, 2002. **149**(1): p. 46–53.
98. Li, Y., K.H. Ang, and G.C. Chong, *PID control system analysis and design*. IEEE Control Systems Magazine, 2006. **26**(1): p. 32–41.
99. Hu, W., G. Xiao, and X. Li, *An analytical method for PID controller tuning with specified gain and phase margins for integral plus time delay processes*. ISA transactions, 2011. **50**(2): p. 268–276.
100. Kalangadan, A., N. Priya, and T.S. Kumar. *PI, PID controller design for interval systems using frequency response model matching technique*. in *2015 International Conference on Control Communication & Computing India (ICCC)*. 2015. IEEE.
101. Kumar, A. and F.D. JL. *A novel self-tuning fuzzy based PID controller for speed control of induction motor drive*. in *2013 International Conference on Control Communication and Computing (ICCC)*. 2013. IEEE.
102. Xu, J.-X., D. Huang, and S. Pindi, *Optimal tuning of PID parameters using iterative learning approach*. SICE Journal of Control, Measurement, and System Integration, 2008. **1**(2): p. 143–154.
103. Mitsukura, Y., T. Yamamoto, and M. Kaneda, *A design of self-tuning PID controllers using a genetic algorithm*. Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251), 1999. **2**: p. 1361–1365 vol.2.
104. Chung, I.-Y., et al. *Control parameter optimization for a microgrid system using particle swarm optimization*. in *2008 IEEE International Conference on Sustainable Energy Technologies*. 2008. IEEE.
105. Jain, S., Y.V. Hote, and S. Saxena, *Fractional Order PID Design Using Big Bang–Big Crunch Algorithm and Order Reduction: Application to Load Frequency Control*. Electric Power Components and Systems, 2022. **49**(6-7): p. 624–636.
106. Anand, S., R. Kumar, and A.N. Jha. *Tuning of PID Controller using BBBC algorithm for Higher Order Oscillatory Systems*. in *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*. 2020. IEEE.
107. Lima, J., et al. *A methodology to design FPGA-based PID controllers*. in *2006 IEEE International Conference on Systems, Man and Cybernetics*. 2006. IEEE.
108. Chan, Y.F., M. Moallem, and W. Wang, *Design and implementation of modular FPGA-based PID controllers*. IEEE transactions on Industrial Electronics, 2007. **54**(4): p. 1898–1906.
109. Raju, S.S.K.N. *Implementation of fpga based pid controller for dc motor speed control system*. in *Proceedings of the World Congress on Engineering and Computer Science*. 2010.
110. Kim, D., *An implementation of fuzzy logic controller on the reconfigurable FPGA system*. IEEE Transactions on industrial Electronics, 2000. **47**(3): p. 703–715.
111. Khan, S., et al. *A Reconfigurable PID Controller*. in *International Symposium on Applied Reconfigurable Computing*. 2018. Springer.

112. Rodríguez-Andina, J.J., M.D. Valdes-Pena, and M.J. Moure, *Advanced features and industrial applications of FPGAs—A review*. IEEE Transactions on Industrial Informatics, 2015. **11**(4): p. 853–864.
113. <https://digilent.com/shop/zybo-z7-zynq-7000-arm-fpga-soc-development-board>.
114. Ządek, P., et al., *Improving efficiency of FPGA-in-the-loop verification environment*. IFAC-PapersOnLine, 2015. **48**(4): p. 180–185.
115. de Castro, F.A., et al. *Comparison of fractional and integer PID controllers tuned by genetic algorithm*. in *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*. 2016. IEEE.
116. Maddi, D., et al. *Optimization of PID Controller Gain Using Evolutionary Algorithm and Swarm Intelligence*. in *2019 10th International Conference on Information and Communication Systems (ICICS)*. 2019. IEEE.
117. Moreira, W.E.M. and C. Garcia, *Performance comparison between IOPID and FOPID controllers in an industrial flow pilot plant*. IFAC-PapersOnLine, 2021. **54**(4): p. 232–237.
118. Taher, S.A. and S. Mansouri, *Optimal PI controller design for active power in grid-connected SOFC DG system*. International journal of electrical power & energy systems, 2014. **60**: p. 268–274.
119. Kumar, S., et al., *Artificial intelligence power controller of fuel cell based DC nanogrid*. Renewable Energy Focus, 2020. **34**: p. 120–128.
120. Patra, A.K., et al. *Genetic Algorithm based FOPID Controller Design for Balancing an Inverted Pendulum (IP)*. in *2020 International Conference on Computational Intelligence for Smart Power System and Sustainable Energy (CISPSSE)*. 2020. IEEE.
121. Vasanthakumar, S., et al. *Cuckoo search algorithm based environmental economic dispatch of microgrid system with distributed generation*. in *2015 international conference on smart technologies and management for computing, communication, controls, energy and materials (ICSTM)*. 2015. IEEE.
122. Parnianifard, A., et al., *Robust optimal design of FOPID controller for five bar linkage robot in a Cyber-Physical System: A new simulation-optimization approach*. PloS one, 2020. **15**(11): p. e0242613.
123. Jain, S., Y.V. Hote, and S. Saxena, *Fractional Order PID Design Using Big Bang–Big Crunch Algorithm and Order Reduction: Application to Load Frequency Control*. Electric Power Components and Systems, 2021: p. 1–13.
124. Jiang, C.X., et al., *A systematic approach for implementing fractional-order operators and systems*. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2013. **3**(3): p. 301–312.
125. Bichiou, Y. and M. Krarti, *Optimization of envelope and HVAC systems selection for residential buildings*. Energy and Buildings, 2011. **43**(12): p. 3373–3382.
126. Altera. *Hardware in the Loop from the MATLAB/Simulink Environment*.
127. Bi, Q., et al., *Advanced controller auto-tuning and its application in HVAC systems*. Control Engineering Practice, 2000. **8**(6): p. 633–644.
128. Pal, A. and R. Mudi, *Self-tuning fuzzy PI controller and its application to HVAC systems*. International journal of computational cognition, 2008. **6**(1): p. 25–30.
129. Khooban, M.H., et al., *Optimal type-2 fuzzy controller for HVAC systems*. automatika, 2014. **55**(1): p. 69–78.
130. Xue, D., T. Li, and L. Liu. *A MATLAB toolbox for multivariable linear fractional-order control systems*. in *2017 29th Chinese Control And Decision Conference (CCDC)*. 2017. IEEE.

131. Moradi, M., *A genetic-multivariable fractional order PID control to multi-input multi-output processes*. Journal of Process Control, 2014. **24**(4): p. 336–343.
132. Tolba, M.F., et al., *FPGA implementation of two fractional order chaotic systems*. AEU-International Journal of Electronics and Communications, 2017. **78**: p. 162–172.
133. Anastasiadi, C. and A.I. Dounis, *Co-simulation of fuzzy control in buildings and the HVAC system using BCVTB*. Advances in Building Energy Research, 2018. **12**(2): p. 195–216.
134. Ma, J. and X. Ma, *Distributed Control of HVAC-BESS under Solar Power Forecasts in Microgrid System*. IEEE Transactions on Industrial Informatics, 2023.
135. Nagpal, H., A. Staino, and B. Basu, *Robust model predictive control of HVAC systems with uncertainty in building parameters using linear matrix inequalities*. Advances in Building Energy Research, 2020. **14**(3): p. 338–354.
136. Adibhesami, M.A. and A. Hassanzadeh, *Optimizing HVAC energy efficiency in low-energy buildings: a comparative analysis of reinforcement learning control strategies under Tehran climate conditions*. Data-Centric Engineering, 2025. **6**: p. e40.
137. Seo, B., et al., *Application of artificial neural network for the optimum control of HVAC systems in double-skinned office buildings*. Energies, 2019. **12**(24): p. 4754.
138. Menghal, P. and A.J. Laxmi. *Adaptive neuro fuzzy based dynamic simulation of induction motor drives*. in *2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2013. IEEE.
139. Carpanzano, E. and D. Knüttel, *Advances in Artificial Intelligence Methods Applications in Industrial Control Systems: Towards Cognitive Self-Optimizing Manufacturing Systems*. Applied Sciences, 2022. **12**(21): p. 10962.
140. Sodhro, A.H., S. Pirbhulal, and V.H.C. De Albuquerque, *Artificial intelligence-driven mechanism for edge computing-based industrial applications*. IEEE Transactions on Industrial Informatics, 2019. **15**(7): p. 4235–4243.
141. Abouheaf, M., et al., *Real-time measurement-driven reinforcement learning control approach for uncertain nonlinear systems*. Engineering Applications of Artificial Intelligence, 2023. **122**: p. 106029.
142. Zambrano, J.G., E. Guzmán-Ramírez, and O. Pogrebnyak, *Search algorithm for image recognition based on learning algorithm for multivariate data analysis*, in *Search Algorithms for Engineering Optimization*. 2013, IntechOpen.
143. Ruiz, F.A., et al., *A new criterion to validate and improve the classification process of LAMDA algorithm applied to diesel engines*. Engineering Applications of Artificial Intelligence, 2017. **60**: p. 117–127.
144. Morales, L., et al., *An intelligent sliding mode controller based on LAMDA for a class of SISO uncertain systems*. Information Sciences, 2021. **567**: p. 75–99.
145. Morales, L., et al., *Modeling and control of nonlinear systems using an Adaptive LAMDA approach*. Applied Soft Computing, 2020. **95**: p. 106571.
146. Morales, L. and J. Aguilar, *An automatic merge technique to improve the clustering quality performed by LAMDA*. IEEE Access, 2020. **8**: p. 162917–162944.
147. Si, Q., et al., *Optimization for the Model Predictive Control of Building HVAC System and Experimental Verification*. Buildings, 2022. **12**(10): p. 1602.
148. Satrio, P., et al., *Optimization of HVAC system energy consumption in a building using artificial neural network and multi-objective genetic algorithm*. Sustainable Energy Technologies and Assessments, 2019. **35**: p. 48–57.

149. Rene, E.A. and W.S.T. Fokui, *A quantum particle swarm optimization-based optimal LQR-PID controller for load frequency control of an isolated power system*. Journal of Engineering and Applied Science, 2023. **70**(1): p. 97.
150. Morales, L.A., P. Fabara, and D.F. Pozo, *An Intelligent Controller Based on LAMDA for Speed Control of a Three-Phase Inductor Motor*. Emerging Science Journal, 2023. **7**(3): p. 676–690.
151. Morales, L., et al., *Approaches based on LAMDA control applied to regulate HVAC systems for buildings*. Journal of Process Control, 2022. **116**: p. 34–52.
152. Biradar, S., Y.V. Hote, and S. Saxena, *Reduced-order modeling of linear time invariant systems using big bang big crunch optimization and time moment matching method*. Applied Mathematical Modelling, 2016. **40**(15-16): p. 7225–7244.
153. Jain, S., Y.V. Hote, and S. Saxena, *Model order reduction of commensurate fractional-order systems using big bang–big crunch algorithm*. IETE Technical Review, 2020. **37**(5): p. 453–464.
154. Kalki, K.M. and H. Rajaguru, *Chaotic-Based Big Bang-Big Crunch Algorithm (CBCA) for Energy Management System of Hybrid Isolated Micro-grid: An Inherent Control Scheme*. Energy Sources, Part A: Recovery, Utilization, and Environmental Effects, 2020: p. 1–21.
155. Bavarinos, K., A. Dounis, and P. Kofinas, *Maximum Power Point Tracking Based on Reinforcement Learning Using Evolutionary Optimization Algorithms*. Energies, 2021. **14**(2): p. 335.
156. Yalçın, O., et al. *Robust tuning of pid controller using differential evolution algorithm based on fpga*. in *2022 9th International Conference on Electrical and Electronics Engineering (ICEEE)*. 2022. IEEE.
157. Blaiech, A.G., et al., *A survey and taxonomy of FPGA-based deep learning accelerators*. Journal of Systems Architecture, 2019. **98**: p. 331–345.
158. Khongprasongsiri, C., et al. *Hardware Implementation of PID Autotuning with Efficient Particle Swarm Optimization*. in *2023 20th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. 2023. IEEE.
159. Ustun, E., et al. *LAMDA: Learning-assisted multi-stage autotuning for FPGA design closure*. in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2019. IEEE.
160. Dounis, A.I., et al. *Fuzzy-PID controller for MPPT of PV system optimized by Big Bang-Big Crunch algorithm*. in *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2015. IEEE.
161. Quintero Gull, C., J.L. Aguilar Castro, and M.D. Rodríguez Moreno, *A multi-label approach for diagnosis problems in energy systems using LAMDA algorithm*. 2022.
162. Duan, M., et al., *A parallel multiclassification algorithm for big data using an extreme learning machine*. IEEE transactions on neural networks and learning systems, 2017. **29**(6): p. 2337–2351.