MULTI-OBJECTIVE DESIGN SPACE EXPLORATION IN HIGH LEVEL SYNTHESIS FOR APPLICATION SPECIFIC COMPUTING

Ph.D. Thesis

By

VIPUL KUMAR MISHRA

DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MARCH 2015

MULTI-OBJECTIVE DESIGN SPACE EXPLORATION IN HIGH LEVEL SYNTHESIS FOR APPLICATION SPECIFIC COMPUTING

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree

of

DOCTOR OF PHILOSOPHY

by

VIPUL KUMAR MISHRA

DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

MARCH 2015

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **MULTI-OBJECTIVE DESIGN SPACE EXPLORATION IN HIGH LEVEL SYNTHESIS FOR APPLICATION SPECIFIC COMPUTING** in the partial fulfillment of the requirements for the award of the degree of **DOCTOR OF PHILOSOPHY** and submitted in the **DISCIPLINE OF COMPUTER SCIENCE & ENGINEERING, INDIAN INSTITUTE OF TECHNOLOGY INDORE**, is an authentic record of my own work carried out during the time period from January 2013 to March 2015 under the supervision of Dr. Anirban Sengupta, Assistant Professor, Indian Institute of Technology Indore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Signature of the student with date
VIPUL KUMAR MISHRA

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

Signature of Thesis Supervisor with date

(Dr Anirban Sengupta)

Vipul Kumar Mishra has successfully given his/her Ph.D. Oral Examination held on **July 11, 2015**.

Signature(s) of Thesis Supervisor(s)
Date:

Convener, DPGC
Date:

Signature of PSPC Member #1
Date:

Signature of PSPC Member #1
Date:

Signature of External Examiner
Date:

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor Dr. Anirban Sengupta, who was a constant source of inspiration during my work, without his constant guidance and research directions this research work could not be completed. His continuous support and encouragement has motivated me to remain streamlined in my research work. I am grateful to Dr. Abhishek Srivastava, Head, CSE Discipline, for all his extended help and support.

I would like to thank my family, especially my mother, father and my sweet wife for always believing in me, for their continuous source of inspiration and their support in my decisions. Without them I could not have made it here.

I am thankful to Dr. Surya Prakash and Dr. Vivek Kanhangad, for taking out some valuable time to evaluate my research progress all these years. Their expert comments and suggestions helped me to improve my work at various stages.

I wish to thank all my colleagues, and staff from the Discipline of Computer Science and Engineering for their suggestions and friendship. A special thanks to Tanveer and Saumya for their constructive opinions, efficiency and assistance on various matters, understanding and constant encouragement.

I extend my sincere thanks to many government bodies like DST, MHRD and IIT Indore to help me with financial support.

DEDICATED TO MY FAMILY

Abstract

High level synthesis (HLS) has gained rapid dominance in the design flow of application specific computing. In HLS, design space exploration (DSE) is an indispensable part, which plays a vital role during design process. Due to advancement in DSE, the designing of an optimal digital circuit for highly complex applications has become possible. Therefore, this thesis proposed four novel automated DSE methodologies for designing application specific systems (ASP) or hardware accelerators. This thesis solves four different types of problem in DSE: **a)** Design space exploration problem for data intensive application during power performance trade-off by proposing a novel DSE methodology employing particle swarm optimization (PSO). In addition, a novel model for power metric, a novel fitness function used for design quality assessment, a novel mutation algorithm, a novel end terminal perturbation algorithm to handle boundary outreach problem during exploration have also been proposed through this solution. Moreover, sensitivity analysis of different PSO parameters such as swarm size, inertia weight, acceleration coefficient, and termination condition on multi objective DSE have also been presented in this solution. **b)** Multi-objective DSE problem for single loop based control and data intensive application by proposing a novel automated methodology for simultaneous exploration of data path and loop unrolling factor (UF) through an integrated multi-dimensional particle encoding process using swarm intelligence. Moreover, to enhance exploration process an estimation model for computation of execution delay of a loop unrolled control and data flow graph (CDFG) (based on a resource configuration visited) without requiring to tediously unroll the entire CDFG for the specified loop value for single loop based application has also been presented. **c)** DSE problem during area performance trade-off for single loop based CDFG by proposing automated exploration of data path and loop UF together through PSO. **d)** DSE problem for perfectly nested loop based applications during power performance trade-off by proposing a novel methodology for automated exploration of architecture and UFs for nested loop using particle swarm optimization. Moreover, a model has been derived which directly estimates

the execution time of nested loop, based on resource constraint and UFs without necessity of tediously unrolling the entire CDFG for the specified UFs values in most cases. The proposed exploration approaches can be applied for designing application specific systems, standalone application specific integrated circuits (ASIC's), hardware accelerators, or DSP cores. Results of the experiments for proposed approaches on the standard benchmarks indicated improvements in terms of exploration runtime and enhancement of quality of final solution (final cost) when compared to recent approaches.

LIST OF PUBLICATIONS

International Journals (5)

1. Vipul Kumar Mishra, Anirban Sengupta, "'Swarm Inspired Exploration of Architecture and Unrolling Factors for Nested Loop Based Application in Architectural Synthesis" ***IEEE/IET Electronics Letters*** , Volume 51, Issue: 2, pp. 157 – 159, Jan 2015 **(5yr Impact Factor = 1.1).**

2. Anirban Sengupta, Vipul Kumar Mishra "Automated Exploration of Datapath and Unrolling Factor during Power-Performance Tradeoff in Architectural Synthesis Using Multi-Dimensional PSO Algorithm", ***Elsevier Journal on Expert Systems With Applications***, Volume 41, Issue 10, pp 4691-4703, August 2014 **(5yr Impact Factor = 2.339).**

3. Vipul Kumar Mishra, Anirban Sengupta "MO-PSE: Adaptive Multi Objective Particle Swarm Optimization Based Design Space Exploration in Architectural Synthesis for Application Specific Processor Design", ***Elsevier Journal on Advances in Engineering Software***, Volume 67, Issue: C, pp. 111–124, January 2014. **(5yr Impact Factor = 1.5).**

4. Anirban Sengupta, Vipul Mishra. Simultaneous Exploration of Optimal Datapath and Loop Based High level Transformation during Area-Delay Tradeoff in Architectural Synthesis Using Swarm Intelligence, IOS Press, ***International Journal of Knowledge-based and Intelligent Engineering Systems***, vol. 19, no. 1 pp.47-61, 2015

5. Anirban Sengupta, Reza Sedaghat, Vipul Kumar Mishra, "Execution Time – Area Tradeoff in GA using Residual Load Decoder: Integrated Exploration of Chaining Based Schedule and Allocation in HLS for Hardware Accelerators, ***Journal of Electronics and Energetics: Facta Universitatis***, Volume 27, No. 2, pp. 235-249, February 2014.

Peer Reviewed Conferences (9)

6. Anirban Sengupta, Vipul Kumar Mishra, " Integrated Particle Swarm Optimization (i-PSO): An Adaptive Design Space Exploration Framework for Power-Performance Tradeoff in Architectural Synthesis ", Proceedings of ***IEEE 15th International Symposium on Quality Electronic Design (ISQED 2014)***, Santa Clara, California, USA, pp. 60 – 67, March 2014 (BLIND REVIEW). *Note- Amongst top 10 EDA/VLSI conferences*

7. Anirban Sengupta, Vipul Kumar Mishra, " Swarm Intelligence Driven Simultaneous Adaptive Exploration of Datapath and Loop Unrolling Factor during Area-Performance Tradeoff ", ***Proceedings of 13th IEEE Computer Society Annual***

International Symposium on VLSI (ISVLSI), Florida, USA, pp. 106-112, July 2014 (BLIND REVIEW). *Note- Amongst top 10 EDA/VLSI conferences*

8. Vipul Mishra, Anirban Sengupta "Swarm Intelligence Driven Design Space Exploration: An Integrated Framework for Power-Performance Trade-off in Architectural Synthesis ", ***Proceedings of 25th IEEE International Conference on Microelectronics (ICM 2013)***, pp. 1- 4, Sep 2013.

9. Anirban Sengupta, Vipul Mishra, " Automated Parallel Exploration of Datapath and Unrolling Factor in High Level Synthesis using Hyper-Dimensional Particle Swarm Encoding ", ***Proceedings of 27th IEEE Canadian Conference on Electrical and Computer Engineering,*** Toronto, pp. 069 - 073, May 2014.

10. Vipul Kumar Mishra, Anirban Sengupta, "PSDSE: Particle Swarm Driven Design Space Exploration of Architecture and Unrolling Factors for Nested Loops in High Level Synthesis" ***Proceedings of 5th International Symposium On Electronic System Design (ISED 2014)*** pp. 10- 14, Dec 2014 (DOUBLE BLIND REVIEW).

11. Anirban Sengupta and Vipul Kumar Mishra, "Time Varying vs. Fixed Acceleration Coefficient PSO Driven Exploration during High Level Synthesis: Performance and Quality ", ***Proceedings of 13th IEEE International Conference on Information Technology,*** pp. 281- 286, Dec 2014 (DOUBLE BLIND REVIEW).

12. Anirban Sengupta, Vipul Kumar Mishra, Pallabi Sarkar, "Rapid Search of Pareto Fronts using D-logic Exploration during Multi-Objective Tradeoff of Computation Intensive Applications ", ***Proceedings of IEEE 5th Asian Symposium on Quality Electronic Design (ASQED)***, Malaysia, pp. 113-122, August 2013.

13. Anirban Sengupta, Vipul Mishra, "D-logic Exploration: Rapid Search of Pareto Fronts during Architectural Synthesis of Custom Processors", ***IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI-2013),*** Mysore, pp. 586 - 593, August 2013.

14. Anirban Sengupta, Vipul Mishra, "Multidimensional Encoding Based Evolutionary Exploration Approach: Adaptive Methodology for Parametric Trade-offs in High Level Synthesis for Control flow Graphs; ***Proceedings of 3rd IEEE CALCON***, IEEE Kolkata, pp. 43 – 46, Nov 2014.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Symbol	Explanation
X_i	Particle /Position of a particle
R_x	Candidate resource combination
UF_N	N^{th} Unrolling factor
P_T	Power consumed by a resource combination
P_D	Dynamic power
P_s	Static power
P_{max}	Maximum power consumption
P_{min}	Minimum power consumption
T_E	Execution time consumed by a resource combination
T_{max}	Maximum execution time
T_{min}	Minimum execution time
P_{cons}	Power constraint specified by the user
T_{cons}	Execution time constraint specified by the user
R_d	d^{th} Resource type
$N(R_d)$	Number of instances of resource type *'R_d'*
CS	Control Step
C_T	Total CSs required to execute the loop completely
C_{body}	Number of CSs required to execute loop body once
C_{first}	Number of CSs required to execute first iteration
I	Maximum number of iteration (loop count)
C_{II}	Number of CSs required between initiations of consecutive iterations
Δ	Delay of one CS in nanoseconds
L	Latency of a scheduling solution

T_c	Cycle time of a scheduling solution
N	Number of input samples to be processed by a functionally pipelined data-path
α	$\left(\frac{I}{UF}\right)^{quotient}$
Z	Stopping criterion
W_1	User defined weightage to power
W_2	User defined weightage to execution time
E_{FU}	Energy consumption of the resources
$E_{MUX/DMUX}$	Energy consumed by a multiplexer and demultiplexer
P_c	Power dissipated per area unit (e.g. transistors).
S	Swarm size
D	Number of dimension
V_{d_i}	Velocity of i^{th} particle in d^{th} dimension
$V_{d_i}^{+}$	New velocity of i^{th} particle in d^{th} dimension
R_{d_i}	Resource value or UF value of particle X_i in d^{th} dimension
$R_{d_i}^{+}$	New resource value or UF value of particle X_i in d^{th} dimension
$R_{d_{lbi}}$	Resource value or UF value of X_{lbi} in d^{th} dimension
$R_{d_{gb}}$	Resource value or UF value of X_{gb} in d^{th} dimension
ω	Inertia weight
b_1,b_2	Acceleration coefficients
r_1,r_2	Random numbers between [0-1].
$C_f^{X_i}$	Fitness of particle X_i
$C_{f_{lbi}}^{X_i}$	Local best fitness of particle X_i

X_{lbi}	Local best position of i^{th} particle
X_{gb}	Global best position of the population
N_{R_i}	Number of instance of resource R_i
K_{R_i}	Area occupied by resource R_i
$N_{MUX/DMUX}$	Number of the multiplexer or demultiplexer
$K_{MUX/DMUX}$	Area occupied by the multiplexer or demultiplexer
v	Number of resource types
φ_1	User defined weightage to power
φ_2	User defined weightage to execution time
$\min(R_d)$	minimum values of resources or UF of d^{th} dimension
$\max(R_d)$	maximum values of resources or UF of d^{th} dimension

ACRONYMS

DSE	Design space exploration
HLS	High level synthesis
PSO	Particle swarm optimization
GA	Genetic algorithm
ACO	Ant colony optimization
DFG	Data flow graph
CDFG	Control and data flow graph
IC	Integrated circuit
RTL	Register transfer level
VHDL	VHSIC hardware description language
ASP	Application specific processor
ASIC	Application specific integrated circuit
DSP	Digital signal processing
ALU	Arithmetic and logical unit
PSGA	Problem specific genetic algorithm
WSPSO	Weighted sum particle swarm optimization
PFA	Pareto front arithmetic
UF	Unrolling factor
FPGA	Field programmable gate array
EA	Evolutionary algorithm
SA	Simulated annealing
QoR	Quality of result
MO-PSE	Multi objective particle swarm exploration
PSDSE	Particle swarm design space exploration

H-SI	Hyper dimension swarm intelligence
AS	Architectural synthesis
RC	Resource combination/configuration
FU	Functional unit
ARF	Auto regressive filter
BPF	Band pass filter
DCT	Discrete cosine transformation
IDCT	Inverse discrete cosine transformation
DWT	Discrete wavelet transformation
WDF	Wave digital filter
FIR	Finite impulse response
EWF	Elliptic wave filter
FFT	Fast Fourier transformation
MMV	MPEG motion vectors

Chapter 1

Introduction

1.1. Preamble

As per *Moore's law,* the density of transistors in an integrated circuit (IC) has been growing consistently since the era of 70s. Such an exponential growth envisioned to produce more capable and powerful devices. However, this exponential growth increased the complexity of the circuit even more. Although, full custom designs of ICs were possible, however, full custom design process was considered a tedious task. It was evident from the fact that, full custom designing was used to design very small chips or highly regular chips like memory. After recognizing this problem in its early stages, a feasible solution was to start the design process at a higher level of abstraction. To complement this approach, researchers in early 80s came up with the idea of fully automated synthesis tools for layout design. As a result of these efforts, the design process became much simpler and helped the designers to focus on the logic level design. But, these solutions also fell short on several fronts. This was mainly due to the fact that the era saw a continued growth in circuit complexity. This increased design complexity, again motivated the designers to move to yet another abstraction level called as the register transfer level (RTL). Thus, with the development of RTL synthesis tools like Synopsis design compiler, RTL design was widely accepted in the 90s. The method received tremendous popularity, and widely employed in the industry. RT level requires the description of the data path & controller specified in a Hardware Description Language such as Verilog or VHDL, which was fed to the synthesis tool to seamlessly produce a combination logic network (logic level netlist). Over the years, RTL synthesis continued to grow. The success of RTL synthesis led the designers to think that raising the design abstraction to another level, i.e. the behaviour level, could enhance design productivity and reduce time to market. This was a perfectly viable idea and indeed bore fruitful results [1, 2, 3, 6].

To enhance design productivity and reduce time to market, researchers introduced high level synthesis, (also known as behavioural synthesis or architectural synthesis) that is a process of transforming a behavioural description (in high level language) into a functionally equivalent Register transfer level design. The high level synthesis provides a link between behavioural level and register transfer level. As a result, designer could specify behaviour of an application in a high level language such as C, C++, JAVA etc. that provides more flexibility for functional verification, estimation of system's performance and the capability to work at higher abstraction levels. Further, it provides chance to get benefits of high level transformation at higher level to produce more optimized design. On the other hand, in the absence of such a design process, a designer is forced to implement a RTL design, which indeed is a tedious task. Therefore, the current objective of research community is to provide facility to designer so that they can start the design process at behavioural (algorithmic) level and HLS tools generate a functionally equivalent optimized RTL design.

Therefore, the objective of HLS process is not only to transform a behavioural description into its equivalent RTL design, but also to rapidly assess various alternatives in order toproduce an optimal high performance low power design, that satisfies the user constraints such as power, delay and area. This is possible due to advances of design space exploration methodology in HLS, which maintains trade-off between conflicting parameters such as power and performance while searching for an optimal solution of its RTL design.This requirement of designing a high performance circuit with least power consumption is often common in the area of digital signal processing, multimedia, communication and network processing. For example, in many embedded systems, application specific hardware (or application specific processors) is highly utilized for simultaneously handling the need of low power, low area and high performance circuit. Smart phones are one of the best examples where designers used these elements such as DSP cores, which provide high processing speed with low battery consumption (i.e. low power). Therefore, during designing of these elements, the efficient exploration of an optimal solution plays an important role, which balance conflicting metrics such as power, area and execution time, to produce high-quality solutions in acceptable exploration time. This process is formally known as design space exploration (DSE).

A design space exploration problem consists of two orthogonal issues; first is to accurately evaluate design points, and second is the capability to explore all corners of the design space. An exhaustive search would eventually explore all corners of the design space and find an optimal solution for a smaller design at a cost of high exploration time, but for large and

complex designs exhaustive search is impractical. Therefore, the designer must balance the accuracy of evaluation by a more accurate evaluation model, and maximize the design space coverage by a better exploration algorithm. Moreover, another objective of DSE should be to reduce manual intervention with improving the automation process by taking intelligent decisions during exploration.

Use of multi-objective algorithms is an excellent way to handle such problem; the multi-objective algorithms have capability to handle conflicting objectives simultaneously and search an optimal solution which maintains trade-off amongst multiple objectives. The proposed method has an inclination towards yielding high quality solutions which fulfil multi parametric optimization requirements with its unique features for optimization. It also solves orthogonal issues such as exploring high-quality results in lesser exploration runtime.

1.2. Overview on the Abstraction Level of Optimization

Due to increasing complexity, the optimization process of the circuits starts from highest level of abstraction. In digital designs there are many abstraction levels viz. system level, behavioural level (algorithmic level or high level), register transfer level, logic level and layout (physical) level. Optimization at higher level always gives more opportunity to designer to handle complex decisions at early stage, that ease the process of optimization at lower level which is more complex than higher level. Moreover, higher level optimization also provides more flexibility, productivity, and design specification awareness than lower level of abstraction. More specifically, for current generation of high performance, power intensive designs, optimization at logic/transistor level is not sufficient for efficient optimization due to exponential complexity of the design. Therefore, optimization at higher level is very crucial for current complex system. Therefore, designers are expected consider user goals from higher abstraction levels i.e. algorithmic level to generate high quality design (at RT level) and also enhance opportunity of optimization at lower level of abstraction i.e. logic level/transistor level.

1.3. A Brief history of High Level Synthesis

The success of compiler for high level language in 1950s, automatically gave an inspiration to hardware designer for generating circuit implementation from high level behavioural specification, due to increasing complexity of digital circuits. To the authors' knowledge, the

first high level synthesis methodology CMU-DA was developed at Carnegie Mellon University at 1979 by Parker et.al [7, 8]. The innovative flow of CMU-DA quickly generated significant research interest. In subsequent years, between 1980s and 1990s, HLS has gained sufficient interest of researchers and several attempts such as HAL[9], Hercules [10], BSSC[11] introduced by researcher. Moreover, an approach presented in [34], describe a bottom up design technique in the synthesis of digital systems. Generally, these approaches break down the synthesis task into following major steps: a) code transformation b) operation scheduling, c) module selection and allocation d) datapath and controller generation. Several researchers addressed these individual problems of HLS. For example, to solve scheduling problem the researchers introduced list scheduling [13] and force-directed scheduling [14]. These early researches helped to form a base for high level synthesis. However, these efforts were not enough for wide acceptance of HLS among designer. In 90s, as a result of improvement in RTL synthesis methodologies and wide adoption of RTL based designs, focus on development of high level synthesis became more practical. In 1995, several industrial tools such as Behavioural Compiler [15, 16] from Synopsis, Monet from Mentor Graphics [17], were introduced. The main reasons behind failure of these tools was that, these tools had an exhaustive nature of DSE and used HDL such as VHDL or Verilog for behavioural description as input, which were not well suited for modelling behaviour at a high level. Further, since 2000, a new generation of high level synthesis methodologies has been developed. Although, most of these methodologies focused on using C based languages for behavioural description unlike previous approaches, but lacked in advance DSE methods.

1.4. Theoretical background on high level synthesis

The process of high level synthesis (HLS) accepts the behavioural description of a system along with a set of user constraints and goals, and generates a register transfer level design which satisfies the constraints [1-4]. HLS comprises of interdependent tasks such as design space exploration of architecture, scheduling, allocation and binding as shown in Figure 1.1. HLS performs sequence of operations to transform a behavioural description into RTL circuit. The final RTL circuit has two segments, one is datapath circuit and another is controller circuit. The datapath design comprises of functional units such as multipliers, arithmetic logical units, storage elements (such as registers) and interconnects. The controller can be described in the form of control state diagram. The behavioural (algorithmic) description is a set of operations and data transfer between storage elementsor can be simply

defined as mapping between input and output of the system. The first step of high level synthesis is the compilation of the formal language into an internal description using graphical representation that contains control and data flow graph. The data flow graph states the input/output relation of the application and the data dependency. The data flow graph is defined in terms of its vertex and edges, where the vertices denote the operations and the edges denoted the data dependency presented between the operations. The next step of high level synthesis is high level transformation, which includes compiler like transformations such as dead code elimination, common sub expression elimination, inline expansion of procedure, constant propagation, and loop transformation. Then next phase is design space exploration. Performing design space exploration at higher abstraction level provides more optimal results than at lower level of abstraction, i.e. logic level or transistor level. Therefore, DSE becomes very crucial segment of high level synthesis for an optimized circuit. Then finally, to realize the RTL design, high level synthesis performs scheduling, allocation and binding. Scheduling involves assigning the operations to control steps. Where, a control step is the fundamental sequencing unit in synchronous systems, corresponding to a clock cycle. Allocation and binding are responsible to assign operations and data into hardware units i.e. functional units (such as multiplier, ALUs), storage (such as registers), and interconnects

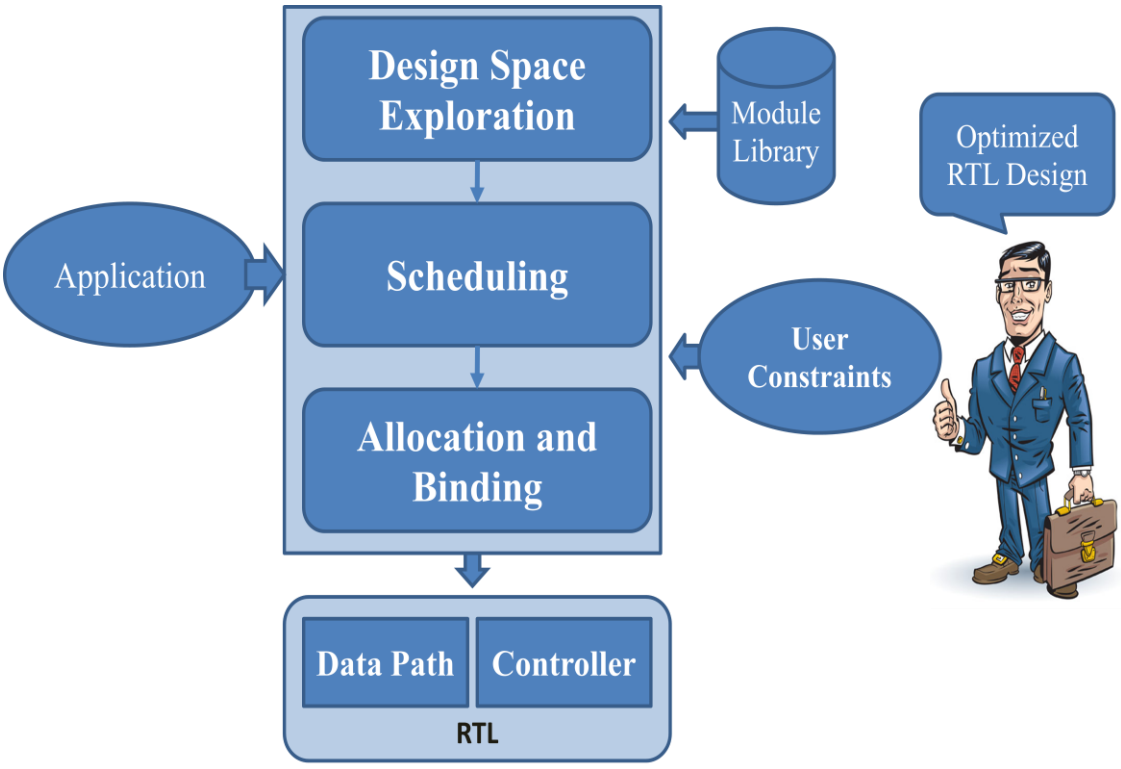


Figure 1.1 Basic block diagram of high level synthesis

(such as multiplexer and demultiplexer) and specifying their uses [2, 3, 12]. Further, the description of the HLS phases as follows:

- *Scheduling*

Scheduling is an important task in HLS which involves assigning the operations to control step. Scheduling algorithm can be broadly divided into two categories: constructive scheduling and iterative scheduling as given in Figure 1.2. In constructive scheduling scheduler starts with a node and construct a scheduling solution by assigning node (operation) to a control step and finally produce a scheduling solution. There are number of approaches under category of constructive scheduling such as, As soon as possible (ASAP), As late as possible (ALAP), List scheduling [86], Force direct scheduling [75], and Integer linear programming based scheduling[82, 83]. The ASAP scheduling process arranges the operations topologically according to their data or control flow. ASAP scheduling places the operations in the sorted order by stamping them in the earliest possible control step. The ALAP scheduling places the operations in the latest possible control step. These two algorithms are simple, yet needed in most of the advanced scheduling algorithms. The ALAP scheduling considers the number of steps resulting from the ASAP schedule as a latency constraint [5]. Moreover, the ILP-based scheduling uses the ASAP and ALAP. ILP minimizes cost functions in the form of power, area, and delay under resource, time or power constraints. The ILP-based algorithm provides an exact solution, but it is slow and has an exponential worst-time time complexity. It is difficult to use it for large and practical circuits as the formulation grows exponentially with the number of vertices [5, 82, 83]. However, List-based scheduling is a heuristic approach to solve the scheduling problem. The list-based algorithm takes a sequencing DFG and resource constraints as inputs and generates a scheduled sequencing DFG as output. In list scheduling, the operations available for scheduling are kept in a list for each control step. This list is ordered by some priority function such as mobility of the vertex or the length of path from the operation to the sink while ranking the vertices in decreasing order. An operation on the list is scheduled one by one if the resource needed by the operation is free; otherwise, it is deferred to the next clock cycle [5, 86]. Furthermore, the basic idea of force direct scheduling algorithm is to balance the concurrency of operations without increasing the total execution time to maximize the utilization of resources such that the number of required resources is minimal [5, 75].

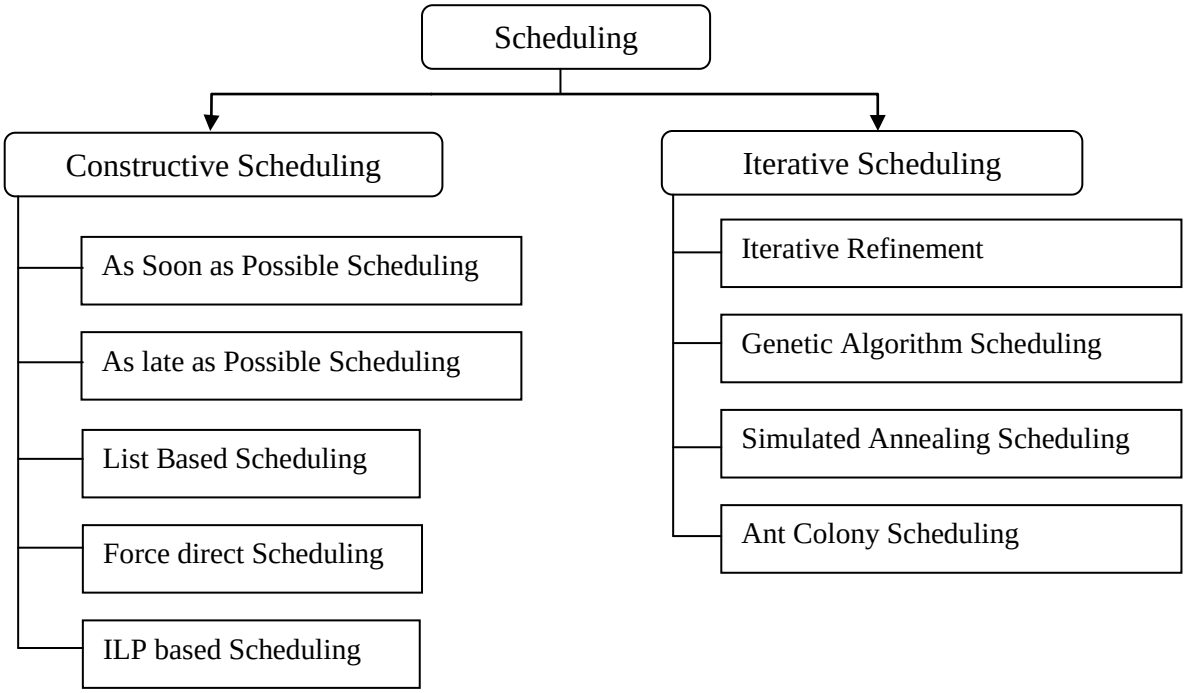


Figure 1.2 Taxonomy of widely used scheduling algorithm in VLSI

Another category of scheduling algorithm is iterative scheduling. Where designer starts with an initial (random) solution and iteratively updates the solution and finally produce an optimal scheduling solution which satisfied the user constraints such as power/area and latency. One of the benefit in iterative scheduling is that, designer have multiple scheduling solution, which are generated in intermediate steps. Mostly used iterative scheduling are iterative refinement [5, 86], genetic algorithm based scheduling [25, 26, 27, 29, 81], simulated annealing based scheduling [5, 90], and ant colony based scheduling algorithm [37, 79].

Simulated annealing based scheduling algorithm similar to the annealing process in Materials Science. In scheduling, the nodes of a DFG are analogous to the atoms, and temperature is analogous to the total number of available resources. The mobility of the nodes/vertices is dependent on the total number of available low-cost resources. Therefore, the simulated annealing scheduling approach explores the trade-offs among power, performance and area [5, 90].

Genetic algorithms are probabilistic search algorithms based on the principle of "survival of the fittest." Genetic algorithms create a collection of scheduling solutions that evolve according to a quality measure based on power and delay; the evolution works on a search space represented by a chromosome. The algorithm improves the average fitness of a

population (collection of chromosomes) by constructing a new population through selection and crossover and mutation [5, 29, 81, 86].

In iteration refinement scheduling, the task-levels used in scheduling iteration are the completion times of the tasks that result from the very previous scheduling iteration. For this, each scheduling iteration passes its output to the next iteration to use as task-levels. The objective of this iterative process is to search solutions with shorter finish times as a result of using a more refined estimate of the task-level throughout [86, 91].

- *Allocation and binding*

Allocation is the process of identifying required resources (i.e. functional unit, storage and interconnect) to realise the implementation of the application.Binding is the task to assign operation to particular resource such as computation to functional unit, storage to register and data transfer to interconnect. There are various algorithms present in the literature for solving binding process such as clique partitioning, circular-arc graph colouring or left edge algorithm. In clique partition approach, we analyse compatibility of two operations by "if operations need resources of the same type and are not scheduled in the same clock cycle thus operations are compatible and can use the same resources". To analyse compatibility of vertices/operations, a data structure called “compatibility graph” is used [5, 86, 89]. Moreover, in graph colouring approach, we analyse, that two operations have a conflict if they are not compatible. To analyse the conflicts of vertices/operations, a data structure called “conflict graph” is used [5, 87]. Furthermore, in the left edge algorithm, the birth time of a variable is mapped to the left edge, and the death time of a variable is mapped to the right edge. The variables are sorted in increasing order of their birth time. The first variable is then assigned to the first register. Then, the current register receives the next variable whose birth time is larger or equal to the death time of the previous variable [5, 86].

Therefore, high level synthesis can broadly be divided into high level transformation, architecture exploration, scheduling, allocation, and binding. Although all the phases are equally responsible for generation of optimal RTL design, but the design space exploration of architecture plays a key role in high level synthesis for constructing an optimal RTL circuit. This is because the DSE is responsible for finding optimal architecture and optimal high level transformation from the large design space while simultaneously maintaining trade-off among multiple conflicting parameters [2, 3, 4].

1.5. Theoretical Background on Design Space Exploration

The Design space exploration is a procedure for analysing the various points in the design space to obtain an optimum design point for given behavioural description based on predefined user constraints [3, 32]. With the increasing complexity of digital circuits,conducting exhaustive analysis on the design spaceof the current generation of very large scale integrated designs with multi objective nature is strictly prohibited. Thus, design space exploration is always considered a challenging task for researchers due to duality among multiple objectives and parameters involved in the process. Due to its non-trivial nature, researchers have attempted multiple techniques to resolve this issue. For example, researchers in [30, 35] tried to reduce the design space into a set of Pareto optimal points. However, this Pareto optimal set may itself become very large for analysis and selection of an optimal design point for system implementation. Moreover, distinct requirement of designs for different purposes makes design space exploration more complex. For example, in case of mobile devices, handheld devices require ASICs with low power and acceptable performance. On other hand, in real time systems, high performance systems require high speed ASICs with acceptable power consumption.

1.6. Reasons for Studying High Level Synthesis

With the increasing demand of low power high performance hardware accelerators used in embedded systems and time to market pressure, high level synthesis has gained attentions amongst designers. Therefore, dominance of high level synthesis has increased because of several reasons as discussed below [1, 2, 3, 5, 12]:

- *Continuous and reliable design flow*

The high-level synthesis process provides a continuous and reliable flow from high-level specifications in the form of C or SystemC to RTL description of the circuit in the form of VHDL or Verilog automatically with minimal manual intervention.

- *Shorter design cycle and fewer errors*

Due to automation of the design process, there has been reduction in the number of man-power used and time to market, resulting reduction in overall cost of the chip. Moreover, correct design decisions at the higher levels of circuit abstraction can ensure that the errors are not propagated to the lower levels, which are too complex and costly to correct.

- *Easy and flexible design space exploration*

Because a synthesis system can produce several designs in a short time, the designers have more flexibility to choose the better design considering different trade-offs of power, area and performance. Even power and performance optimization can be performed at any level of circuit abstraction, from system level to silicon. Thus, as the level of abstraction goes lower, the complexity of the circuit increases; and also reduces the degree of freedom, and thus opportunity of power reduction and high performance, decreases. Therefore, high level or behavioural level provides a better degree of freedom for design space exploration.

- *Availability of circuit technology to more people*

As design expertise is incorporated into the synthesis tools, it becomes easier for a non-expert to produce a chip that meets a given set of specifications. Hence, the designer can be hired at a lower price, which will reduce the non-recurring cost and overall design cost of the chip.

1.7. Thesis Organization

The rest of the thesis is organized as follows: Chapter 3 describes in details of the proposed framework for solving the design space exploration problem for data intensive application using particle swarm optimization during power performance trade-off. In chapter 4, describe proposed approach to solve the problem of automated design space exploration of datapath and loop unrolling factor for single loop based control and data intensive application during power-performance trade-off using swarm intelligence framework, while in chapter 5, solves the problem of simultaneous exploration of datapath and unrolling factor for single loop based control and data intensive application during area-performance trade-off. Moreover, chapter 6 describes the proposed approach for solving automated design space exploration of datapath and loop unrolling factors for nested loop based applications. The results of the proposed DSE approaches for various well known high level synthesis benchmarks indicating exploration time and quality improvements obtained when compared to the current existing DSE approach are provided in Chapter 7. Chapter 8 concludes the research work presented in the thesis and provides future scope of work in this area.

Chapter 2

Previous Works and Thesis Contribution

2.1. Related work

The problem of design space exploration in HLS is a NP-complete problem [22, 23, 24] and the heuristic algorithms has been proved their ability to solve NP-complete problem, this motivated researchers for utilizing these algorithms to solve DSE problem. Therefore, researchers employed heuristic algorithms such as Genetic algorithm for solving DSE of architectures as well as integrated exploration problem of scheduling, allocation and binding in HLS. For example, in [25] and [26], authors presented a work based on problem space genetic algorithm (PSGA), to solve DSE problem in HLS. The authors used binary encoding of the chromosomes for DSE in architectural synthesis for area-latency trade-off. Moreover, in [27], a framework based on node priority scheme has been suggested for DSE of data paths in high level synthesis which maintains the trade-off between latency and area. Though the results are promising, but exploration process is very computationally expensive. In addition, authors in [28], have applied GA to solve the problem of binding and allocation in HLS. The authors have introduced an unconventional crossover technique depending on a force directed data path binding algorithm. Although, the approaches presented in [26, 27, 28] optimized area and latency, but failed to consider power and execution time (function of latency as well as cycle time for pipelined dataset), which are critical issues for modern handheld, battery operated high speed devices. Recently, in [29], the DSE problem was addressed by proposing multi structure genetic algorithm (GA), which assists in deciding Pareto fronts amongst the different design variances. The approach is computationally complex in nature and considers only static power while calculating total power. Due to inconsideration of total power and computationally complex nature of approach, the approach lacks to produce high-quality optimized results within acceptable exploration time. Furthermore, in [30], the authors have proposed a discrete particle swarm optimization based design space exploration in high level synthesis. The work partially relates the PSO with DSE problem. However, the technique

suffers from some major drawbacks. The authors have not considered the concept of local best (cognitive factor) during exploration. Further, while updating the particles' velocity, the authors updated only direction (step length was kept constant). Therefore, if a particle is far away from an optimal solution, then algorithm required more iteration to reach near optimal solution. Moreover, the authors divided the swarm into sub-swarms and each objective was accomplished by one sub-swarm only. Hence, the technique required a large swarm size which may lead to heavy computation time per iteration. In the proposed approach every swarm explores the design space by considering all conflicting objective simultaneously. Moreover, authors in the [31] described an approach to solve DSE problem which is based on GA and weighted sum particle swarm optimization (WSPSO). The authors performed crossover between local/global best and current position (similar to GA) to update the position instead of using conventional way to update the position by velocity, which reduces the ability to clinically balance between exploration and exploitation. In addition, authors did not consider user constraints for power and execution time in the cost function, which are critical for hardware accelerators (computationally expensive application). In another research, a deterministic method was introduced in [32] based on Pareto optimal analysis. In this work, the design space was ordered in the form of an architecture vector design space for architecture variant analysis and optimization of performance parameters. But the approach was not completely capable to generate optimal points, due to high nonlinearity of DSE problem. In addition, authors in [33] have proposed machine learning method: random forest for DSE and introduced an experimental design which can wisely sample micro-architecture choices and used them for training in the learning model. However, the authors in [33] have not considered power and data pipelining. Furthermore in [35], authors have proposed a fuzzy-based DSE scheme using hierarchical criterion method. The approach is partially based on fuzzy logics and fuzzy sets which mimics the ordered design space for the performance parameters. The approach is very promising for area-delay tradeoffs as compared to other current approaches. However, it lacks the capability to mostly produce optimal solutions. Further, no promising results for power performance trade-off are presented. In another research, to explore the giant search space, an approach for synthesis of heterogeneous embedded systems by using Pareto Front Arithmetic (PFA) was proposed by the researchers in [36].Their approach exploited the hierarchical problem structure for searching the set of Pareto optimal set, but suffers from slow exploration speed and lacks concurrent consideration of state-of-art metrics such as power, execution time, and area. Moreover, the DSE problem was solved with ant colony optimization (ACO) algorithm in [37] where

authors handle resource and time constraints based scheduling during area latency trade-off. Furthermore, in [38], authors proposed a clustering based DSE algorithm which performs exploration of best knob settings based on high level transformation such as loop unrolling, function inline, array access but does not handle exploration of datapaths which is very crucial in HLS. Besides above the approaches presented in [25-38] are unable to handle loop based applications. In addition, the authors used exploration capability of evolutionary approach in [39] for DSE. In [39], during exploration process to asses a design point this approach determines circuit area by product of base circuit area and specified unrolling factor, in case of latency calculation, approach simply divide base latency by specified unrolling factor. The base area and base latency evaluated without any loop unrolling. This process of evaluation design point, when handle loop unrolling, is impractical for real application. Moreover, the selection of unrolling factor was user driven and considered only those unrolling factors which are multiples of loop iteration count. Although the works described in [27, 28, 30, 37, 38, 39] considered area and latency but failed to consider power and execution time (which are crucial for current power hungry hardware accelerators), due to which the approaches were unable to achieve high-quality optimized results.

Additionally, there are some tools presented in academia and industry for HLS such as an open-source high-level synthesis tool called LegUp was proposed in [21], which is used for FPGA-based processor/accelerator systems. LegUp is able to synthesize C language to hardware, thereby providing a nice platform to perform high level synthesis. Different FPGA architectures are supported by this tool, and allow new scheduling algorithms and parallel accelerators. Furthermore, tool such as ROCCC has also been proposed in [19], which is an open-source high-level synthesis tool for generating RTL structure from C. It is designed for kernels that perform computation intensive tasks such as DSP cores. Therefore, ROCCC applies to a specific class of applications (streaming-oriented applications), and is not a general C-to-hardware compiler, like LegUp [21]. In [18], the authors introduced SPARK tool for HLS. SPARK takes a behavioural description in ANSI-C as input and produces synthesizable register-transfer level VHDL. The shortcoming of this tool is that the unrolling factor for the loop is user-directed and the approach is unable to automatically determine the optimal combination of UF and datapath together. Further, a tool AutoPilot introduced in [40] addresses the problem of exploration in HLS. It performs C/C++/systemC-to-RTL synthesis. The scope of this tool is very limited and targeted only for FPGA's. Furthermore, in [41], authors introduced a tool 'SystemCoDesigner' for performing area-delay trade-off that offers rapid design space exploration with prototyping of behavioural systemC models. Moreover,

there also exist some other commercial tools for HLS in the market such as GAUT [20]. GAUT has also caught huge attention in the electronic design automation (EDA) community. It takes input in the form of a C/C++ description of the behaviour description, for automatically generating a RTL structure based on compulsory constraint of throughput (or initiation interval) and clock period. Furthermore, tools such as CatapultC from Mentor Graphics (now acquired by Calypto) [43], Cynthesizer from Forte[44], CyberWorkBenck from NEC[45], Vivado from Xilinx[46] use C/C++ to describe the functional intent and generating Register Transfer Level (RTL) structure. Tools described in [40-46] perform power-performance-area trade-off but, [41, 48] handle only area-performance trade-off. The shortcomings of all the above tools are that, automated exploration of loop unrolling factor and datapath is not performed and therefore, the tools are unable to automatically determine the optimal combination of UF and datapath based on the conflicting user constraints.

Most of the algorithm mentioned above, considered area and latency as parameter but did not consider power and execution time (which is function of latency as well as cycle time based on pipelining) during exploration, which are crucial for modern power efficient and high speed devices. Only few algorithms use static power as optimization metrics, instead of considering composition of dynamic and static power, due to which previous algorithm is unable to produce high-quality results for handheld, battery operated mobile devices. Also, the algorithm suffered by poor implementation runtime and there is no guarantee for always yielding superior design points. Moreover, only few approaches handle high level transformation such as loop unrolling during exploration. The approaches or tools which handle loop unrolling require manual intervention to decide the unrolling factor (UF) and some approaches consider only those UFs as potential candidates which evenly divide the iteration count. To the best of the author's knowledge no publicly available tool/approach exists in the literature so far which automatically explores datapaths and loop unrolling together. Moreover automated exploration of datapaths and loop unrolling factor for nested loop based applications has not been taken into account by any approach/tool. Therefore, the DSE methodologies proposed so far suffers from: a) higher computational complexity b) inability to handle essential parameters such as power (average dynamic power and leakage power) and execution time (cycle time, latency together)for modern devices such as portable devices c) inability to simultaneously explore datapath and loop unrolling for single/nested loop based applications. The deficiencies of the above motivated us for further research and propose novel solutions to the aforesaid problems. These deficiencies have been gradually resolved through multiple phases as highlighted in this thesis: first, the efforts are made to

handle parameters such as power and execution time for data intensive applications by proposing fast and efficient DSE methodology based on particle swarm optimization. Next, we handle single loop based CDFGs with power- performance and area- performance trade-off by proposing DSE approaches, which handle datapath and loop unrolling together. Finally, we handle nested loop based applications by proposing a DSE approach which handles datapath and nested loop unrolling simultaneously.

2.2. Selected bio inspired framework used for design space exploration

2.2.1. Genetic algorithm based DSE

Genetic algorithm is one the widely used bio inspired heuristic algorithm for solving various NP-Complete problems of different domains. GA is a model or abstraction of biological evolution inspired from Charles Darwin's theory of natural selection. In order to solve design space exploration problem, GA is used by the researchers in [25, 26, 27, 29]. The authors in [25, 26, 27, 29] solved integrated scheduling and datapath exploration problem. In these approaches, the chromosome has two parts: first part, which represents the scheduling information, while second represents the datapath information (i.e. number of functional units and operating frequency information). In [25, 26] authors encoded scheduling information in chromosomes as *'work remaining'*. Moreover, the scheduling information in [27], is encoded with '*node priority'* and this *node priority* is defined by location in chromosome (priority decreases from left to right). In [29], authors used the scheduling information in chromosome encoded by *"load factor"* and used a heuristic to decode the scheduling information from encoded chromosome. However, in [25, 26, 27, 29], the second part of chromosome is encoded with max number of FU's available during scheduling. Moreover, in [29], authors also considered multiple versions of the functional units during exploration process. Furthermore, in [25, 26, 27], the authors considered area and latency during cost calculation while, [29] considered power and execution time in cost calculation. In order to explore new solutions the approaches perform genetic operator (such as crossover and mutation) between two chromosomes.

2.2.2. Bacterial forging based DSE

Bacterial foraging optimization algorithm (BFOA) is a popular bio inspired optimization algorithm for global optimization problem [73]. BFOA is inspired by the social foraging

behaviour of Escherichia coli bacteria. The BFOA has been utilized by authors in [72], to solve design space exploration problem in high level synthesis. The framework presented in the [72] focused on solving design space exploration problem of datapath. The basic mechanisms viz. chemotaxis, replication and elimination-dispersal were imitated to explore new architectural solutions (resource combination) in BFOA. During the exploration process chemotaxis plays an important role, where process performs swim and tumble to determine new architectural solution. The algorithm also generates new solutions during dispersal process. The new solution is accepted only when new solution has better fitness than current fitness or has not been traversed before. Moreover, in order to determine the fitness of the solutions, authors used a penalty based cost function composed of area and execution time factors. With this cost function, authors maintained trade-off between area/power and execution time.

2.2.3. Hybrid genetic algorithm and particle swarm optimization based DSE

With the inspiration of two successful algorithms GA and PSO, the authors in [31] solved design space exploration problem with hybrid GA and PSO based algorithm. In this framework, to find new solution, crossover is performed between current position with global best position and local best position. Thus, to incorporate GA, crossover is performed, which is the basic operator of GA and to incorporate PSO, the crossover is performed between current position and global and local best position. In this approach, authors solved the integrated scheduling and datapath exploration where the encoding of the chromosome (combination of scheduling information and max available FUs) is adopted from [27] as described in the previous sections. In order to determine fitness of the solution authors' evaluated latency from scheduling solution, area from maximum FU available for scheduling and determined power with the help of compatibility graph to determine binding information. The authors use weighted combination of latency, area and power during fitness evaluation. New solutions were determined by performing crossover between current solution and local best/global best solution.

2.3. Background of particle swarm optimization

The design space exploration in high level synthesis is a NP complete problem [23, 24,59]. As discussed in literature, the NP-complete problems can be successfully solved by heuristic

approaches [63 - 66]. In literature, researchers gave a lot of efforts to solve DSE in HLS using GA and EA, but, PSO has not been explored enough for solving DSE in HLS. It has been proved in the literature that there are many applications where PSO outperforms GA or EA [67, 68]. Therefore, investigation on the exploration capability of PSO to solve DSE problem in HLS is essential.

Particle Swarm Optimization is a population based stochastic optimization technique. which was developed by Kennedy and Eberhart in 1995. PSO imitated the behaviour of flocks of birds [48, 49]. Similar to flocks of the birds, PSO searches the design space of an objective function by adjusting the trajectories of individual agents, called particles. The movement of a swarming particle consists of two major components: a stochastic component and a deterministic component. Each particle is attracted towards the position of the current global best x^{gb} and its own best location x^{lb} *in* history. When a particle finds a location that is better than any previously found locations, then it updates it as the new current best for particle *i*. There is a current best for all *n particles* at any time *t* during iterations. The aim is to find the global best among all the current best solutions until the objective no longer improves or after a certain number of iterations.

2.3.1. PSO algorithm

The essential steps of the particle swarm optimization can be summarized as the pseudo code shown in Figure 2.1.

Let x_i and v_i be the position and velocity vector for particle i^{th}, respectively. The position of i^{th} particle is changed by adding the velocity to the current position as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.1}$$

while the velocity is updated with the following rule:

$$v_i(t+1) = v_i(t) + b_1 r_1 (x_i^{lb} - x_i(t)) + b_2 r_2 (x^{gb} - x_i(t)) \tag{2.2}$$

where b_1 is the cognitive learning factor, b_2 is the social learning factor, r_1, r_2 are random numbers in the range [0, 1], is the best position of i^{th} particle with respect to the minimization problem, x^{gb} is the global best position found so far. As can be noted, the formulation of the problem leads to solutions which try to 'follow' the leader's x^{gb} position as well as attracting solutions versus the personal best solution of the particle .

Objective function $f\{x)$, $x = (x_1, ...,x_d)$
Initialize locations and velocity of n particles.
Find x^{gb} from min{f(x_1),f(x_2)..., $f(x_n)$} (at $t = 0$)
while (stopping criterion)
    $t = t + 1$ (pseudo time or iteration counter)
    for all particle
        Generate new velocity using equation
        Calculate new locations using equation
        Evaluate objective functions at new locations
        Find the current best for each particle x^{lb}
    end for
    Find the current global best *particle*
end while
Output the final results (x^{gb}) and cost

Figure 2.1 Basic particle swarm optimization algorithm

2.3.2. Inertia particle swarm optimization [93]

The inertia weight is introduced to control the exploration and exploitation ability of PSO, and provide a balance between the exploration and exploitation abilities during searching process. It has been observed that PSO produces better results when its *exploration* ability is *more* favoured in the *early* optimization stages to allow the exploration of as many promising areas of the search space as possible. Then, towards the end of the optimization process, the local *exploitation* ability of the algorithm should be promoted, instead, to allow for a more refined search around the best areas previously *roughly* detected [93]. This is possible by controlling the velocity of the particles in the later search stages. This means the effect of the previous velocity term, which is known as inertia factor, will gradually decrease over PSO iterations. Therefore, a linearly decreasing inertia weight, ω, was introduced by Shi and Eberhart [93], as shown in *equation* (2.3). In the linearly decreasing inertia weight, initially, ω is set to a high value, ω_{max}, around 0.9 in order to allow the particles to move freely, and quickly explore the global optimum neighbourhood [105]. In the later stages, the value of the inertia weight is decreased to a small amount, ω_{min}, around 0.4 in order to refine the search, and shift the optimization process from an exploratory mode to an exploitative mode [93] [106]. With inertia weight new velocity is updated by following eqn:

$$v_i(t+1) = \omega v_i(t) + b_1 r_1 (x_i^{lb} - x_i(t)) + b_2 r_2 (x^{gb} - x_i(t)) \quad 2.3$$

Where ω is called the inertia weight, which updates based on eqn. 2.4:

$$\omega(t) = \omega_{\max} - (\omega_{\max} - \omega_{\min}) \frac{t}{t_{\max}} \quad 2.4$$

2.4. Popular population based optimization

Other than PSO there are many population based optimization technique. Such as Genetic algorithm, simulated annealing, ant colony optimization, Tabu search are most popular []. The basic descriptions based on source of inspiration, important parameters, method to find new solution, control of exploration and exploitation, local minima problem, and application analysis are given in next sub sections.

2.4.1. Genetic algorithm

Genetic algorithm was proposed by Holland in 1975 [94, 95] that mimics the evolutionary processes in nature as explained in Darwin's theory. Genetic algorithm employs a population of possible solutions to an optimization problem. Specifically, they operate on encoded representations of the solutions, equivalent to the genetic material of individuals in nature, and not directly on the solutions themselves. As in natural evolution based on Charles Darwin's theory, selection provides the necessary driving mechanism for better solutions to survive. Each solution has a *fitness value (based on the fitness function of the problem)* that reflects how good solution is, compared with other solutions in the population. The better fitness value of an individual solution represents the higher chance of survival and reproduction. Recombination of genetic material in genetic algorithms is simulated through a crossover mechanism that exchanges portions between encoded solutions. Another very important operation is mutation, has a direct analogy from nature and plays the role of regenerating lost genetic material. Parameters which play a vital role in the success of GA are population size, number of iterations, crossover probability, and mutation probability. The mechanism for generation of new solution depend upon selection, crossover and mutation are defined below:

***Selection*-** Parent selection emulates the survival-of-the-fittest mechanism in nature. It is expected that a fitter chromosome receives a higher number of offspring and thus has a higher chance of surviving in the subsequent generation. There are many ways to achieve effective selection, including ranking, tournament, and proportionate schemes but the key assumption is to give preference to fitter individuals.

Crossover- Crossover is a recombination operator that combines subparts of two parent chromosomes to produce offspring that contain some parts of both parents' genetic material. A probability term, *pc,* is set to determine the operation rate.

Mutation- Mutation is an operator that introduces variations into the chromosome. This variation can be global or local. The operation occurs occasionally (usually with small probability p_m) but randomly alters the value of an encoded solution.

The drawbacks of the GA are there is no specific method to control exploration and exploitation in GA. The GA is computationally expensive and less efficient for design space exploration problem. On other hand GA performs better for combinatorial problem such as scheduling problem in HLS [107, 96].

2.4.2. Simulated annealing

Simulated annealing (SA) [99,100] is a random search population based technique for global optimization problems, which mimics the annealing process in material processing when a metal cools and freezes into a crystalline state with the minimum energy and larger crystal size so as to reduce the defects in metallic structures. The annealing process involves the careful control of temperature and cooling rate, often called annealing schedule. Simulated annealing algorithm uses Markov chain for identifying potential solutions, which converge under appropriate conditions concerning their transition probability.

The basic idea of the simulated annealing algorithm is to use random search in terms of a Markov chain, which not only accepts changes that improve the objective function, but also keep some changes that are not ideal with a probability [100]. For example, in case of minimization problem, any better moves or changes that decrease the value of the cost will be accepted; however, some changes that increase cost will also be accepted with a probability *p.* This probability *p,* also called the transition probability, is determined by *p*

$$p = e^{-\frac{\Delta E}{kbT}}$$

where *kb* is the Boltzmann's constant, *T* is the temperature for controlling the annealing process. ΔE is the change in energy levels.

During optimization process selection of initial temperature is very crucial parameter [101]. If T is too high, the system is at a high energy state on the topological landscape, and the minima are not easily reached. If T is too low, the system may be trapped in a local minimum, not necessarily the global minimum, and there is not enough energy for the system

to jump out the local minimum to explore other minima including the global minimum. So a proper initial temperature should be calculated.

The higher temperature indicates more exploration while lower temperature indicates the exploitation in optimization process. Thus, cooling rate controls the exploration and exploitation. Therefore selection of optimum cooling rate is very crucial for convergence. The drawback of the simulated annealing is that SA is a computationally expensive algorithm and more suitable for combinatorial optimization problem. Moreover, lack of randomness causes local optima solution [99].

2.4.3. Ant colony optimization

Ant Colony Optimization (ACO) was introduced by Dorigo *et al.* [102, 103, 104] in 1996, it is a cooperative heuristic searching algorithm inspired by the ethological study on the behaviour of ants. It was observed that ants could manage to establish the optimal path between their colony and the food source within a very short period of time without vision capability. This is done by an indirect communication known as *stigmergy* via *pheromone*, left by the ants on the paths. Though any single ant moves essentially at random, it will make a decision on its direction biased on the "strength" of the pheromone trails that lie before it, where higher amount of pheromone indicates a better path. As an ant traverses a path, it strengthens that path with its own pheromone. A collective behaviour emerges as more ants will choose the shortest trails, which in turn creates an even larger amount of pheromone on those short trails, which makes those short trails more likely to be chosen by future ants. The ACO algorithm is inspired by such observation. It is a population based approach where a collection of agents cooperate together to explore the search space. They communicate via a mechanism imitating the pheromone trails.

The central component of an ACO algorithm is a parametrized probabilistic model, which is called the *pheromone model*. The pheromone model consists of a vector of model parameters called *pheromone trail parameters*. The pheromone model is used to probabilistically generate solutions to the problem under consideration by assembling them from a finite set of solution components. At runtime, ACO algorithms update the pheromone values using previously generated solutions. The update aims to concentrate the search in regions of the search space containing high quality solutions. In particular, the strengthening of solution components depending on the solution quality is an important ingredient of ACO algorithms. It implicitly assumes that good solutions consist of good solution components. To learn which components contribute to good solutions can help assembling them into better

solutions. In general, the ACO approach attempts to solve an optimization problem by repeating the following two steps:

• Candidate solutions are constructed using a pheromone model, that is, a parametrized probability distribution over the solution space;

• The candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

ACO belongs to the class of meta-heuristics, which are approximate algorithms used to obtain good enough solutions to hard combinatorial optimization problems in a reasonable amount of computation time [102]. To the best of author's knowledge, ACO was used only to solve scheduling problem not for DSE in HLS.

2.5. Objective

The objective of this thesis is to develop fast and efficient DSE methodologies in HLS for application specific computing (or hardware accelerators for loop kernels) based on multi objective trade off. In order to realize the above aim, the following objectives have been set:

1. Develop a methodology for proficient DSE in HLS for data intensive applications during power- performance trade-off that produces high quality design solutions.
2. Develop an automated methodology for simultaneous exploration of datapath and unrolling factor for single loop based applications during power-delay trade-off in HLS.
3. Develop an automated methodology for simultaneous exploration of datapath and unrolling factor for single loop based applications during area-delay trade-off in HLS.
4. Develop an automated DSE framework for simultaneous exploration of datapath and unrolling factors for nested loop based applications during power- performance trade-off.
5. Design a power model which consists of dynamic and static power to analyse design solutions during DSE process.
6. Develop a delay prediction model for faster exploration process in case of single and nested loop based control data flow graph(CDFGs) without tediously unrolling CDFG loop completely.
7. Investigate the impact of algorithmic parameters regulating the DSE process on performance and quality of the final solution.

2.6. Summary of contribution

This thesis proposes fast and efficient design space exploration in high level synthesis based on multi-objective trade-off during designing of hardware accelerators for data intensive applications or loop kernels.

In order to resolve the issues present in the state-of-the-art approaches (related works), the following contributions have been made through this research:

- Solve the problem of design space exploration during power performance trade-off for data intensive applications.

(Publications: J3, C6, C8, C14)

a) Proposed a novel DSE methodology driven through particle swarm optimization (PSO) framework for multi-objective trade-off, capable of simultaneously improving Quality of Results (QoR) as well as reducing exploration time.
b) Introduced a novel model for power metrics used during evaluation of design points in design space exploration process.
c) Proposed a novel fitness function, used for design quality assessment in design space exploration process.
d) Proposed a novel mutation algorithm for improving DSE convergence and exploration time.
e) Propose a novel perturbation algorithm to handle boundary outreach problem during exploration.
f) A novel sensitivity analysis of different PSO parameters such as swarm size, inertia weight, acceleration coefficient, and termination condition and its impact on the DSE. This analysis is expected to assist the designer in pre-tuning the PSO parameters to an optimum value for achieving efficient exploration results within a quick runtime.

- Solve the problem of automated design space exploration during power-performance trade of single loop based control and data intensive applications.

(Publications: J2, C7)

a) Simultaneous exploration of data path and loop UF through an integrated multi-dimensional particle encoding process using swarm intelligence which maintains

trade-off between power-performance metrics as well as control states and execution delay during loop unrolling.

b) Proposed an estimation model for computation of execution delay of a loop unrolled CDFG (based on a resource configuration visited) without tediously requiring unrolling the entire CDFG for the specified loop value.

c) Presented an analysis of design metrics such as power, execution time and number of control steps of the global best particle found in every iteration with respect to increase/decrease in unrolling factor.

- Solve the problem of automated design space exploration during area-performance trade of for single loop based control and data intensive applications.

  (Publications: J4, C9)

  a) Simultaneous exploration of data path and loop UF using particle swarm optimization which balances the trade-off between area-performance metrics as well as control states and execution delay during loop unrolling.

  b) Presented an analysis of design metrics such as area, execution time and number of control steps of the global best particle found in every iteration with respect to increase/decrease in unrolling factor.

- Solve the problem of automated design space exploration during power-performance trade-off for nested loop based control intensive applications.

  (Publications: J1, C10)

  a) Proposed a novel automated exploration of architecture and UFs for nested loops using particle swarm intelligence that in parallel maintains trade-off between conflicting metrics of power–performance and balance orthogonal issues by improving QoR and reducing the exploration runtime.

  b) Proposed a novel execution time model which directly estimates the execution time of nested loop based on resource constraints and UFs without tediously requiring unrolling the entire CDFG for the specified UFs values in most cases.

Chapter 3

MO-PSE: Adaptive Multi Objective Particle Swarm Optimization Based Design Space Exploration in Architectural Synthesis for Application Specific Processor Design

This chapter presents a fast and efficient design space exploration framework termed as multi objective particle swarm exploration (MO-PSE), based on particle swarm optimization (PSO)[48, 49, 50] algorithm in high level synthesis for data intensive application. During exploration process, in the proposed MO-PSE, trade-off between conflicting parameters such as power consumption and execution time is maintained. In addition, MO-PSE is capable to resolve orthogonal issues such as enhancing quality of result as well as exploration speed, thereby being able to produce higher-quality results in lesser exploration time than existing approaches. To the best of the authors' knowledge this is the first framework that directly maps a complete PSO process for multi-objective DSE during power-performance trade-off for application specific computing in high level synthesis. Moreover, this chapter presents a novel power model (which considers static power as well as dynamic power) for assessment of a design point. Further, a novel cost model has also been presented in this chapter for evaluation of a design point. The detail description of the proposed process along with demonstration of the proposed framework has been given in subsequent sections.

3.1. Description of proposed methodology

3.1.1. Problem formulation

Given a data flow graph (DFG), explore the design space and find an optimal solution which satisfies the conflicting user constraints and minimize the overall cost. The problem can be formulated as:

For a given DFG, find a resource combination R_x:

$$R_x = \{N(R_1), N(R_2), ...N(R_d)...N(R_D)\};$$

with minimum hybrid cost: P_T and T_E;

and subjected to: $P_T <= P_{cons}$ and $T_E <= T_{cons}$.

where, *'N(R_d)'* is the number of instances of resource type *'R_d'*; *'D'* is the total number of resource types; *'R_x'* is a candidate resource combination for optimal solution; *'P_T'* and *'T_E'* are the power and execution time consumed by a candidate resource combination; *'P_{cons}'* and *'T_{cons}'* is power and execution time constraint specified by the user.

3.1.2. Generic overview of proposed MO-PSE

This section presents an overview of the proposed exploration algorithm. The flow chart shown in Figure 3.1 represents the proposed multi-objective particle swarm exploration (MO-PSE) algorithm and the pseudo code of proposed algorithm is given in Figure 3.2. Based on the flow chart provided in Figure 3.1, the description of the proposed algorithm is as follows:

The inputs to the proposed framework are behavioural description of application in the form of data flow graph (DFG) that describes data-path, user specified design constraints for power and execution time (with user specified weight factor), and module library. The module library contains four different information viz. Energy consumed by each resource in Pico joule (Pj), hardware area of each resource (#of transistor), latency of each resource in nanoseconds (ns) and user specified maximum available resources. The proposed framework checks the user constraints, if user constraints are not valid then show an error message and requests for valid user constraints values.

As mentioned before, in the proposed work, PSO algorithm has been directly mapped onto the DSE process of architectures. The proposed mapping of PSO on DSE is as follows:

PSO		DSE
Position of particle	→	Resource configuration
Velocity of particle	→	Exploration deviation/drift
Dimension	→	Number of Resource type

Further, the swarm population is mapped as set of initial design points in design space (considered as initial design solutions which will be subjected to improvement in each

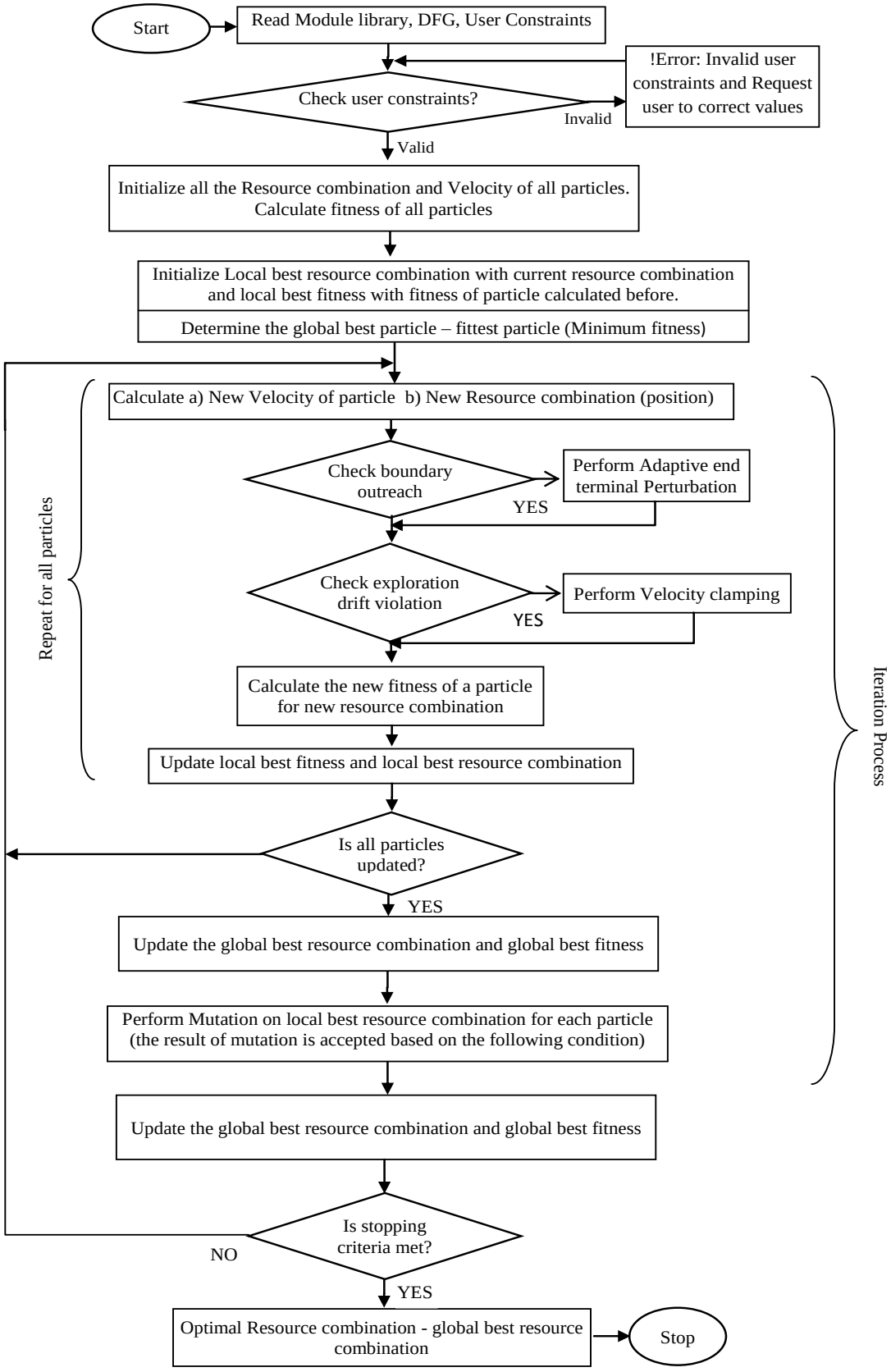


Figure 3.1 Flow chart of proposed MO-PSE

Algorithm: MO-PSO

Input- DFG, Module library, User Constraints

Output- Optimal resource configuration

{

Read Library ( )

Read DFG ( )

Determine boundary constraints for power and execution time

If (($P_{min} > P_c > P_{max} \parallel T_{min} > T_c > T_{max}$)) *//checking validity of user constraints*

{

!! Show error message and request for valid constraints

}

Initialization (resource configuration, velocity)

For i =1 to S //S = # of particles)

{

$C_f^{X_i} = f(Power, Execution\ time)$ *// calculate fitness of all particle*

}

//find best resource configuration that is the current global best resource configuration

$X_{gb} = X_i[Min(C_{f_{lb1}}^{X_1}, C_{f_{lb1}}^{X_2}, C_{f_{lb1}}^{X_3} ....C_{f_{lbn}}^{X_n})]$

While (stopping criterion)

{

For i=1 to S //S = (# of particle)

{

For d=1 to D

{

// determine new resource configuration and velocity for i^{th} particle and d^{th} dimension

$R_{d_i}^{+} = f(V_{d_i}^{+}, R_{d_i})$

IF ($-V_{d_i}^{max} > V_{d_i}^{+} > V_{d_i}^{max}$)

{

Perform Velocity Clamping ( )

}

//check boundary constraints outreach

IF ($\min(R_d) > R_{d_i}^{+} > \max(R_d)$)

{

Adaptive-end-terminal-perturbation ( )

}

}

// check for local best resource configuration

IF ($C_f^{X_i}(t) < C_{flb}^{X_i}$)

{

$C_{flb}^{X_i} = C_f^{X_i}(t)$

$X_{lbi} = X_i(t)$

}

}

// determine new global best resource configuration

$X_{gb} = X_i[Min(C_{f_{lb1}}^{X_1}, C_{f_{lb1}}^{X_2}, C_{f_{lb1}}^{X_3} ....C_{f_{lbn}}^{X_n})]$

Adaptive-Rotation-Mutation

$X_{gb} = X_i[Min(C_{f_{lb1}}^{X_1}, C_{f_{lb1}}^{X_2}, C_{f_{lb1}}^{X_3} ....C_{f_{lbn}}^{X_n})]$

} *// end of while loop*;

Output – Global best position (optimal resource configuration)

}

Figure 3.2 Pseudo code of MO-PSE algorithm

iteration) while the social and cognitive component of PSO are used as factors that supplement in exploration drift process during architecture optimization.

In the proposed approach, the initial swarm population has multiple particles. The first particle X_1: (first initial design point) is constructed by mapping the resource configuration with minimum value which represents the serial implementation, the second particle: X_2 (second initial design point) is constructed by mapping the resource configuration with maximum value which represents the maximum parallel implementation, the third particle: X_3 is constructed by mapping the average value of minimum and maximum resource configuration. The rest of the particles (X_4....X_n) are constructed as follows:

$$N(R_d) = (\min(R_d) + \max(R_d))/2 \pm \beta \tag{3.1}$$

where, '$\min(R_d)$' is minimum resource of d^{th} type, '$\max(R_d)$' is maximum resource of d^{th} type (obtained from module library) and 'β' is a random value between $\max(R_d)$ and $\min(R_d)$. The details of the initial swarm population are described in the upcoming sections. Additionally in the proposed algorithm, all the particles' initial velocity is assigned to zero. In fact, for any physical objects in the initial position, their initial velocities must be zero (as they are stationary). If particles are initialized with nonzero velocities, then physical analogy is violated [49]. Once the particles are initialized, determination of particles fitness is performed and the fittest particle (which has minimum fitness) is chosen. The fittest particle becomes the global best resource configuration and fitness of this particle act as best fitness for the next iteration. After this step, iteration process initiates. According to the algorithm, in each iteration, the new resource configuration (design points denoted by the position of the particle) of all particles based on the given function are upgraded and evaluated by:

$$R_{d_i}^{+} = f(V_{d_i}^{+}, R_{d_i}) \tag{3.2}$$

Eqn. (3.2) denotes the procedure for determining the next resource configuration (position) during PSO using the information of the current position or resource configuration (R_{d_i}) and new velocity or exploration drift ($V_{d_i}^{+}$). The detailed description of eqn. (3.2) will be given in section 3.2.6. The process of determining the new resource configuration ($R_{d_i}^{+}$) needs critical introspection. There may arise two cases where boundary constraints may not be met. The proposed approach handles it as follows:

a) As discussed during the resource up gradation process, there may be a possibility of boundary outreach (i.e. the new resource value jumps beyond the design space). In order to

combat this problem, we proposed an '*adaptive end terminal perturbation'* algorithm, which redeems any violations encountered during the DSE process.

b) The proposed approach also uses a concept of '*velocity clamping*' in the MO-PSE in order to control the excessive exploration drift thereby preserving unwarranted diversity control.

For example, during evaluation of exploration drift/velocity metric, large value of v^+ may result in the particle outreaching the boundaries of the given design space [49]. Therefore, *velocity clamping* is achieved to mitigate this issue based on:

$$V_{d_i}^{\max} = \pm(\min(R_d) + \max(R_d))/2 \tag{3.3}$$

After recalculation of the fitness of all particles with new resource configuration; updation of the local best positions (resource configuration) and global best position (resource configuration) is performed, when the new fitness is better than previous fitness. Once the global resource configuration is found, the proposed adaptive rotation mutation scheme on all local best resource configurations is executed with the hope of exploring a better global best resource configuration than the existing one. For this process, fitness of mutated local best configurations are evaluated and the global best solution is updated, if found fitter.

Similarly, the steps above are repeated (iteration process) until the stopping criterion is reached (the stopping criterions are described in the section 3.2.10). Hence, after completing the process the algorithm yields the optimal architecture which is the global best resource solution for the given application and user constraints.

The proposed algorithm however has some scope of improvement in terms of deciding an ideal stopping criterion. This is because; an ideal value will help in obtaining faster convergence time for a solution. As shown in results (chapter 7) later in the thesis, several experiments were carried out to achieve the best possible stopping criterion. We however still believe that this topic requires further investigation. Further, regarding the usability of the approach, it is limited to handle multi-criterion optimization problems during design of application specific data intensive (and control) processors. Its applicability for handling optimization problems during designing general purpose counterparts needs further investigation. Nevertheless, the greatest advantage of the approach is its adaptation to changing technology. With rapid change in technology, there is a change in the values of static power per transistor (owing to voltage scaling and device geometry shrinkage) as well

as in frequency. These parameters directly affect the dynamic energy and static power dissipation of the circuit. In our approach, these variables (V: voltage supply and pc: static power per transistor as shown in eqn. 3.12 and 3.8 respectively) are user controlled (through module library), therefore it provides flexibility to our DSE algorithm to handle the changing technology demands.

3.1.3. Proposed Models for Evaluation of Particles (Design Points) during MO-PSE

3.1.3.1. Proposed Power Model

Let the total power consumed by the functional resources is given as 'P_T' where, P_T is a composition of dynamic power (P_D) and static power (P_S) given by eqn. (3.4):

$$P_T = P_D + P_S \tag{3.4}$$

The average dynamic power consumed by the functional resources is a function of dynamic activity of the resources and can be described as:

$$P_D = \frac{N \cdot E_{FU}}{L + (N-1) \cdot T_c} \tag{3.5}$$

Where, 'E_{FU}' represents the total energy consumption of the resources obtained from [61], 'N' represents the data elements to be processed (during data pipelining), 'L' represents the latency of a scheduling solution and 'T_c' represents the initiation interval or cycle time of a scheduling solution. Equation (3.5) can be further written as in eqn. (3.6):

$$P_D = \frac{N \cdot (E_{Res} + E_{mux} + E_{demux})}{L + (N-1) \cdot T_c} \tag{3.6}$$

Where, 'E_{Res}' is the energy consumed by the major functional units such as adders, subtractors and multipliers.

On the other hand, static power is majorly a function of area of resources and is independent of dynamic activity of a module. Therefore static power can be written as in equation (3.7 & 3.8):

$$P_S = \sum_{i=1}^{v} (N_{Ri} \cdot K_{Ri}) \cdot p_c \tag{3.7}$$

$$P_S = (N_{R1} \cdot K_{R1} + N_{R2} \cdot K_{R2} + .. + N_{Rv} \cdot K_{Rv}).p_c \tag{3.8}$$

Where 'N_{Ri}' represents the number of resource R_i. 'K_{Ri}' represents the area occupied by resource R_i, 'v' is the number of resources (FU's) and 'p_c' denotes the power dissipated per area unit (e.g. transistors).

Substituting eqn. (3.6) and (3.7) in eqn. (3.4):

$$P_T = \frac{N \cdot (E_{Res} + E_{mux} + E_{demux})}{L + (N-1) \cdot T_c} + \sum_{i=1}^{v} (N_{Ri} \cdot K_{Ri}) \cdot p_c \tag{3.9}$$

3.1.3.2. Model for execution time parameter

For a system with 'v' functional resources the time of execution (T_E) can be represented as [32] [35]:

$$T_E = [L + (N-1) \cdot T_c] \tag{3.10}$$

Where, the variables L, N and T_c have been defined in section 3.1.3.1.

3.1.3.3. Proposed model for fitness function

The fitness function (C_f) developed which considers total execution time and total power consumptions shown in eqn. (3.11).

$$C_f^{X_i} = \varphi_1 \frac{P_T - P_{cons}}{P_{\max}} + \varphi_2 \frac{T_E - T_{cons}}{T_{\max}} \tag{3.11}$$

Where $C_f^{X_i}$ = Fitness of particle Xi; φ_1, φ_2= User weight for power and execution time parameters. The function to calculate P_T and T_E are stated in eqn. (3.9) and (3.10) respectively.

3.2. Demonstration with detail description of the proposed methodology

3.2.1. Resource Library Information and Operating Constraints

The values of E_{FU} assumed in the library of the proposed approach have been adopted from [61, 54, 74] (portion of library information are shown in Table 3.1 and Table 3.2). The estimation of E_{FU} has been done in [61] through the standard function for dynamic energy as shown in eqn. (3.12):

$$E_{FU} = 1/2 \cdot C \cdot V^2 \cdot a \tag{3.12}$$

Where '*V'* is the supply voltage of functional unit FU; *'C'* is the physical capacitance of the functional unit FU and '*a'* is the average switching activity at the inputs of FU.

Table 3.1. E_{FU} at 5V and α = 0.5 [61]

Major FU`s	Add16	Mul16	Sub16
Energy (pJ)	106.76	2310.6	106.76
Area(#transistor)	2032	2464	2032
Latency (ns)	20	100	20

Table 3.2. E_{FU} at 5V and α = 0.5 [61]

Minor FU`s	Mux16:2/1	Mux16:4/1	Mux32:4/1
Energy(pj)	24.05	68.032	136.06

The DFG of 2nd order digital IIR filter shown in Figure 3.3 will be used as a simple example in the upcoming sections to demonstrate the proposed approach. Additionally, for the sake of demonstration of the proposed algorithm using this example, we will assume some real constraint values for Power (P_{cons}) and Execution time (T_{cons}) as well as user defined specifications as follows:

P_{cons} = 8W; T_{cons} = 310 us;

Maximum available multiplier FU's: 6, and adder FU's: 4, and subtractor FU's: 4; N = 1000 (Number of sets of data) while 'p_c' is assumed to be 1mW; additionally, the number/type of mux and demux is directly extracted from the scheduling solution. (*Note*: The values chosen for constraints and filter DFG as application are purely arbitrary. The efficiency of the method by any means is not restricted to the values/example assumed. This has been verified by our implementation results achieved for various benchmarks). Therefore, the goal of exploration problem is to simultaneously meet the provided constraints for power and execution time.

3.2.2. Max-Min analysis for user threshold

In order to determine the extreme bounds values for the user constraints, the Max-Min analysis is performed for the applications. In this analysis boundary values (maximum and minimum values) of power and execution time are identified based on the given resource constraints for the application. The minimum value of execution time and maximum value of power is determined by utilizing maximum available resources to execute operations. This indicates maximum possible parallelism of the target application. On the other hand, maximum value of execution time and minimum value of power is determined by using minimum resources (i.e. single instance of all resources). This indicates complete serial execution of the target application. This minimum and maximum value of power and

execution time are used to set the upper and lower threshold (bounds) for user constraints. Finally, conclusion of this analysis is given in the table 3.3.

Table 3.3 Min Max analysis of user constraints

Resource Utilization	Power	Execution time	Execution Behaviour
maximum	Maximum	Minimum	Maximum parallel
Minimum	Minimum	maximum	Completely serial

3.2.3. Boundary Constraints Check Module

This module checks whether the specified user constraint falls in the valid range of boundary limits. The following condition is checked for each parametric constraint specified:

1. Check: $P_{\min} > P_{cons} > P_{\max} \;||\; T_{\min} > T_{cons} > T_{\max}$

2. If the above condition is true then stop and correct the constraints.

   Else the above condition fails and goes to step 3.

3. Execute the initialization process of Module.

3.2.4. Initialization of particles

a) Position

This section will formally define the positions of particles used in the proposed algorithm. For a DFG, the particle position 'X_i' of an 'i^{th}' particle is given as:

$$X_i = R_x = (N(R_1), (N(R_2),..(N(R_d).. (N(R_D))$$

Initialization plays very important role in the algorithm. Usually, the positions (resource configuration) of particles are initialized to uniformly cover the design space. It is important to note that the efficiency of the MO-PSE is biased by the initial diversity of the population, i.e. how much of the design space is covered and how well particles are distributed over the design space. Therefore, the algorithm initializes position as follows (keeping in mind that an optimal design solution to a multi-objective exploration problem will always lies between the maximum parallel and serial implementation of the application):

- The first particle's position is initialized by minimum resources (serial implementation):
  X_1= (min(R1), min(R2),.. min(RD))
  X_1= (1, 1, 1)
- The second particle's position is initialized by maximum resources (maximum parallel implementation):
  X_2 = (max(R1), max(R2),.. max(RD))

Therefore based on the user defined resources assumed in section 3.2.1, X_2 can be customized as follows:

X_2= (6, 4, 4)

- The third particle's position is initialized by average of maximum and minimum values.

  X_3=((min(R1)+max(R1))/2, (min(R2)+max(R2))/2,.....,((min(RD)+max(RD))/2

  Therefore, X_3 can be customized as:

  X_3= (3, 2, 2)

- The rest of the particles (X_4....X_n) are initialized by eqn. 3.1. This function has been proposed to introduce an element of stochasticity (as well as diversity) into the initialization process.

MO-PSE, like other population based exploration process, relies on the initial population. If the proposed MO-PSE is unable to efficiently initialize the population then this will affect the following:

- Exploration time- An inefficient initialization causes higher exploration time. For example if initial solution are too far from the global best solution then exploration process requires more time to reach the optimum solution.
- Local minima – If initialization process is not able to cover the entire design space then there is a possibility that the MO-PSE may suffer with the local minima problem.

Therefore, the proposed MO-PSE utilizes efficient initialization scheme as discussed in this thesis to avoid the above aforesaid problems.

b) Velocity

Furthermore, velocities of all particles are initialized to zero (see Table 3.4); the motivation for such an initialization process has been discussed in section 3.1.2.

c) Acceleration Coefficients

In order for the algorithm to achieve convergence, it has been theoretically established before in [50, 69] that the cognitive learning factor (b_1) and the social learning factor (b_2) can be initialized to any value between [1, 4] *(Note: The detail of this proof has been given by authors in [50]).* The impact of convergence and exploration time of MO-PSE for various benchmarks based on the variation of 'b' between [1, 4] has been provided in result chapter in the thesis. However, we have been able to empirically prove, that for tested benchmarks tuning the value of 'b' between [2, 3] yields the best results during DSE.

Table 3.4 Initial velocity of particle

Xn :Vn	V_1	V_2	V_3
X_1	0	0	0
X_2	0	0	0
X_3	0	0	0

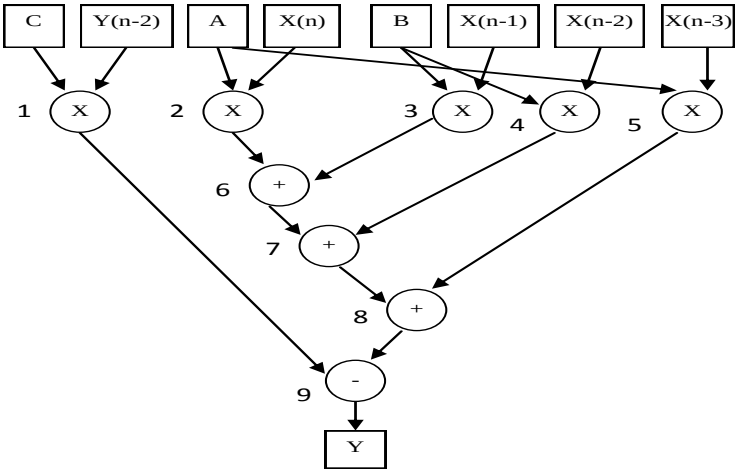


Figure 3.3 Data Flow Graph of 2nd order IIR

3.2.5. Calculation of fitness of a particle

Based on the initialization of particles performed in section 3.2.3, the initial fitness of the particles is calculated using eqn. (3.11). But, before eqn. (3.11) can be applied, the individual values of power and execution time for all particles needs to be calculated. For example, the calculation of total power (P_T) of X_1= (1, 1, 1) using eqn. and (3.9) is as follows. (*Note: Values obtained for 'L' and 'T_c' used for power and execution time determination are calculated using functional pipelining shown in Figure 3.4 to incorporate the transformation technique of algorithmic concurrency*).

$$P_T = \frac{N \cdot (E_{Res} + E_{mux} + E_{dmux})}{L + (N-1) \cdot T_c} + (N_{R1} \cdot K_{R1} + N_{R2} \cdot K_{R2} + .. + N_{Rn} \cdot K_{Rn}).p_c$$

$$\mathrm{P}_T = \frac{1000\left(\begin{matrix} 5*2310.6 + 3*106.76 + 1*106.76 + \ 2*192.49 \\ +\ 1*192.49 + 2*68.032 + 1*68.032 \end{matrix}\right)}{520 + (1000-1)500} + (1*2464 + 1*2032 + 1*2032)*1mw$$

$$= 6.553\mathrm{W} \qquad (3.13)$$

Subsequently, the execution time is calculating using eqn. (3.10):

$$T_E = [L + (N-1) \cdot T_c]$$

$$T_E = 520 + (1000-1)500 \quad = 500.020\mu s \tag{3.14}$$

While, P_{max} = 31. 15W and T_{max} = 500.02us are calculated by considering the maximum resources during scheduling based on user provided specification and minimum resources (one instance of each resource type) respectively.

Finally, substituting eqn. (3.13), (3.14) in eqn. (3.11), the fitness of the particle X_1 is calculated as:

$$C_f^{X_1} = 0.5\left(\frac{6.553-8}{31.159}\right) + 0.5\left(\frac{500.020-310}{500.020}\right)$$

$$C_f^{X_1} = 0.1667 \tag{3.15}$$

(Note- φ_1 = φ_1 = 0.5 is chosen for providing equal weightage to power and execution time factor during calculating fitness of particle). Similarly, the fitness of all other particles is calculated using eqn. (3.11).

$C_f^{X_1}$ = 0.1667 fitness of X_1 (1, 1, 1)

$C_f^{X_2}$ = 0.1691 fitness of X_2 (6, 4, 4)

$C_f^{X_3}$ = 0.0116 fitness of X_3 (3, 2, 2)

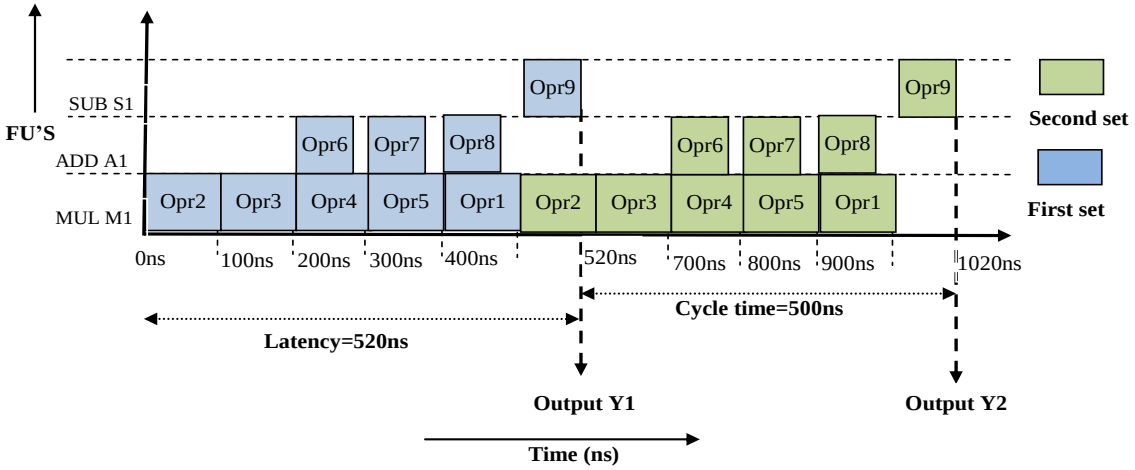


Figure 3.4 Determination of initiation interval / cycle time during functional pipelining of data path for particle X_1 (1(+), 1(*), 1(-))

3.2.6. Determination of local & global best positions/resource configurations

After calculating the fitness of each particle, the next step of the algorithm (as shown in Figure 3.1) is to determine the local best position X_{lbi} of each particle and finally determine the global best particle (X_{gb}). For example, let us assume that the new fitness of a particle 'X_i' is given as '$C_f^{X_i}$' and the fitness of the previous local best particle 'X_{lbi}' is given as '$C_{flb}^{X_i}$'. Now, if the new fitness of particle is less than the current local best fitness then new X_i becomes new X_{lbi} of the particle. The process of determining upgraded X_{lbi} is shown in Figure 3.5. Since in iteration 1, there was no previous local best position ('X_{lbi}') therefore the current position (X_i) assumes the value of X_{lbi}. Next, the X_{gb} of the population is determined using eqn. (3.16) as follows:

$$X_{gb} = X_i[Min(C_{f_{lb1}}^{X_1}, C_{f_{lb1}}^{X_2}, C_{f_{lb1}}^{X_3} ....C_{f_{lbn}}^{X_n})] \tag{3.16}$$

Therefore, substituting the values of fitness cost for each particle in eqn. (3.16) yields:

$$\mathrm{X_{gb}} = X_i\left[\mathrm{Min}\left(0.1667, 0.1691, 0.0116\right)\right]$$

$$\mathrm{X_{gb}} = X_3 \quad (3,2,2) \tag{3.17}$$

Thus, the third particle's 'X_i' is the 'X_{gb}' for next iteration.

3.2.7. Determination of new configuration of the particle

After completing the initialization phase, the iteration process is started; the first task of the iteration is calculation of $R_{d_i}^{+}$ (new resource configuration). The $R_{d_i}^{+}$ is calculated using eqn.

Procedure to upgrade local best resource combination from 2nd iteration onwards

$C_{f_{lbi}}^{X_i}$ - Local best cost of particle 'X_i'
$C_f^{X_i}$ - cost of particle 'X_i'
X_{lbi} – Local best resource combination of particle 'X_i'
X_i – Position (resource combination) of i^{th} particle.

If ($C_f^{X_i} < C_{f_{lbi}}^{X_i}$) then
$C_{f_{lbi}}^{X_i} = C_f^{X_i}$
$X_{lbi} = X_i$

Figure 3.5 Procedure for local best up-gradation

(3.2):

$$R_{d_i}^{+} = f(V_{d_i}^{+}, R_{d_i})$$

The function of $R_{d_i}^{+}$ consist of two parts: first part is new velocity ($V_{d_i}^{+}$) and second part is previous resource configuration (R_{d_i}). Further, $V_{d_i}^{+}$ is calculated using eqn. (3.18):

$$V_{d_i}^{+} = \omega V_{d_i} + b_1 r_1 \left[R_{d_{lbi}} - R_{d_i} \right] + b_2 r_2 \left[R_{d_{gb}} - R_{d_i} \right] \quad (3.18)$$

In eqn. (3.18), three components contribute to determination of new velocity (exploration drift).

- First, ωV_{d_i} is called inertia component. This component represents the momentum which prevents drastic change in the direction of a particle.
- Second, $b_1 r_1 [R_{d_{lbi}} - R_d]$ is called the cognitive component. The effect of the cognitive component represents the tendency of a particle to return to its individual best resource configuration from the past.
- Third, $b_2 r_2 \left[R_{d_{gb}} - R_{d_i} \right]$ is called the social component. The social component's effect is that, each particle moves towards the best resource configuration found by all its neighbours (including itself).

Where ω is inertia weight, b_1 and b_2 are acceleration coefficients and r_1 and r_2 are random numbers between [0-1]. Therefore, eqn. (3.2) can be re-written as:

$$R_{d_i}^{+} = R_{d_i} + V_{d_i}^{+} \quad (3.19)$$

Where, $R_{d_i}^{+}$, R_d , and $V_{d_i}^{+}$ have been defined in section 3.1.2. The visualization of determination of new position is shown in Figure 3.6 where, X_i represents D dimension vector of resource combination. In Figure 3.6, green arrows show contribution of previous velocity, cognitive component, and social component to determine new position (represented by yellow arrow). Based on the above explanation the $R_{d_i}^{+}$ in D-dimensions (for demonstration using the example, three dimensions 1, 2, 3 have been used to represent each particle/design point) are calculated as follows:

i) Particle X_1:

In '1st' dimension:

Using eqn. (3.18) new velocity is calculated as:

$$V_{1_1}^{+} = 1*0 + 2*0.5(1-1) + 2*0.5(3-1) = 2 \quad (3.20)$$

Using eqn. (3.19) and (3.20) the upgraded resource configuration for 1st dimension is:

$$R_{1_1}^{+} = 1+2 = 3 \quad (3.21)$$

In '2nd' dimension:

Using eqn. (3.18) new velocity is calculated:

$$V_{2_1}^{+} = 1*0 + 2*0.5(1-1) + 2*0.5(2-1) = 1 \quad (3.22)$$

Using eqn. (3.19) and (3.22) the upgraded resource configuration for 2nd dimension is:

$$R_{2_1}^{+} = 1+1= 2 \quad (2.23)$$

In '3rd' dimension:

Using eqn. (3.18) new velocity is calculated:

$$V_{3_1}^{+} = 1*0 + 2*0.5(1-1) + 2*0.5(2-1) = 1 \quad (3.24)$$

Using eqn. (3.19) and (3.24) the upgraded resource configuration for 3rd dimension is:

$$R_{3_1}^{+} = 1+1 = 2 \quad (3.25)$$

X_1^{+} = (3, 2, 2) is new resource configuration of 1stparticle.

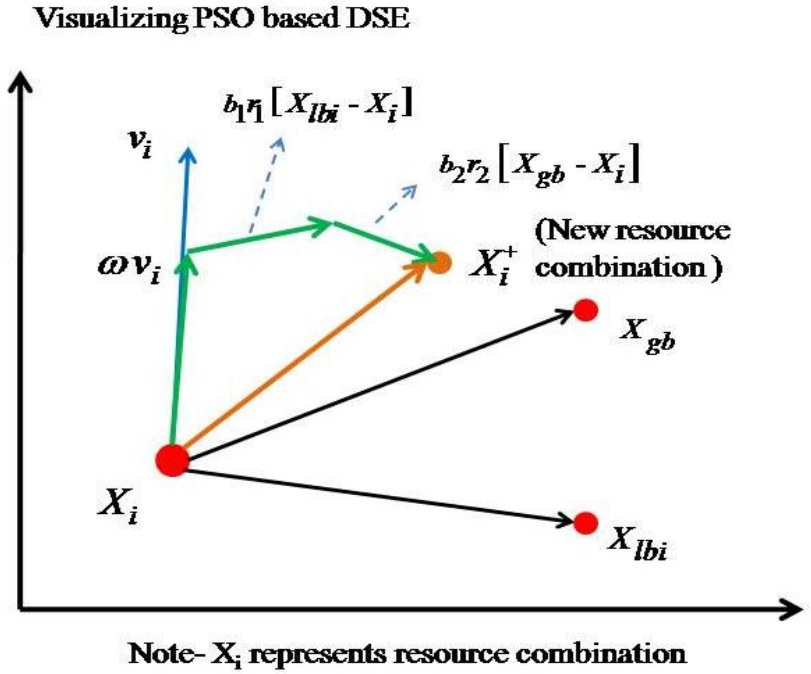


Figure 3.6 Determination of New Position [48, 49]

ii) Particle X_2:

Similarly, the value of x_2^+ for 2nd particle is found as:

x_2^+ = (3, 2, 2)

iii) Particle X_3:

Similarly, the value of x_3^+ for 3rd particle is found as:

x_3^+ = (3, 2, 2).

3.2.8. Adaptive End Terminal Perturbation Algorithm: Particle Outreach Verification Module

DSE being a complicated optimization process, MO-PSE methodology requires embedded algorithms to handle critical issues such as boundary outreach violation. This happens during the resource up gradation phase, where there is a possibility of particle outreaching the boundary of the design space intervals. The proposed MO-PSE always handles such a case by applying an algorithm called '*adaptive end terminal perturbation*' after the resource up-gradation process (determination of new particle position) to assure that every particle must be within the design space boundary interval (max and min). The proposed '*adaptive end terminal perturbation*' algorithm to handle the violation of boundary constraint limit during resource up-gradation is shown in Figure 3.7.

Adaptive end terminal perturbation

Input- Resource value (R_{d_i}) which crosses the design space

Output- New valid value of resource lying in the design space

//When R_{d_i} crosses the design space boundary

While (R_{d_i} < min (R_d))

$$R_{d_i} = R_{d_i} + Y$$

End While

While (R_{d_i} > max (R_d))

$$R_{d_i} = R_{d_i} - Y$$

End While

/* where 'Y' is a random value between min (R_d) and max (R_d).

Figure 3.7 Adaptive end terminal perturbation algorithm

3.2.9. Velocity clamping

As mentioned in section 3.1.2, '*velocity clamping*' in the MO-PSE is used in order to control the excessive exploration drift thereby preserving unwarranted diversity control. The velocity clamping is performed, when any particle's exploration drift (velocity) crosses the $\pm V_{d_i}^{\max}$. It helps particles to stay within the design space and to take sensibly step size in order to explore through the design space. Without this velocity clamping in the searching space the process will be prone to explode and particles' resource configuration shall change hastily. The exploration drift is proposed by the following eqn:

$$V_{d_i}^{+} = \begin{cases} +V_{d_i}^{\max} & if\ V_{d_i}^{+} > +V_{d_i}^{\max} \\ -V_{d_i}^{\max} & if\ V_{d_i}^{+} < -V_{d_i}^{\max} \end{cases} \tag{3.26}$$

where, the value of $\pm V_{d_i}^{\max}$ is determining using $V_{d_i}^{\max} = \pm \frac{(\max(R_d) - \min(R_d))}{2}$ (as described in eqn. (3.3));

3.2.10. Mutation operation

In the proposed work, the balanced contribution of local best and global best solutions during searching a new solution makes PSO less vulnerable to premature convergence as compared to genetic algorithm and other population based DSE process. The local best component (cognitive component) enhances the exploration capability to search locally, which introduces diversity in the solution. On the other hand, global best component (social component) attracts solution towards current global best solution, which introduces convergence of the population. But, over the time when momentum of the exploration becomes small then this can cause premature convergence. In order to overcome this premature convergence problem we introduced adaptive rotation mutation. The mutation process introduces diversity in the solution as well as may produce better quality solution than current solution.

Mutation operation is performed on all local best resource configuration with probability Mp=1.0. The proposed *adaptive rotation mutation* algorithm is shown in Figure 3.8. Adaptive rotation mutation algorithm is a novel algorithm for mutation operation where algorithm uses two basic operations for mutation. First is rotation operation, and second is increment or decrement operation. To perform these operations, the total population is

Adaptive rotation mutation

Input – Local best resource combination X_{lbi}

Output – New mutated local best resource combination X_{lbi}

For i=1 to S **Do**
  If (i%2==0) // Left Rotation
    For d=1 to D **Do**
      $Temp = R_{d_i}$
      $R_{d_i} = R_{(d+1)_i}$
      $R_{(d+1)_i} = Temp$
      $d++$
    End For
  End If
  If (i%2==1)
    For d=1 to D **Do**
      $R_{d_i} = R_{d_i} \pm Q$
      // Q is a random number between [1-3]
      d++;
    End For
  End If
  i++;
End For

Figure 3.8 Adaptive rotation mutation algorithm

divided into two groups: one is the even group in which algorithm performs left rotation operation and second group is the odd group in which algorithm performs increment or decrement with a random number. After mutation algorithm calculates new fitness value of local best resource configuration and if found better fitness then new value will become local best fitness otherwise does not change older value as shown in eqn. 3.27. For the sake of clarity, an example of mutation operation is given below:

Assume before mutation the local best resource configurations are:

x_{lb1} = (3, 2, 2) - X_1 particle

x_{lb2} = (2, 1, 1) - X_2 particle

X_{lb3} = (4, 3, 3) - X_3 particle

After mutation the mutated local best resource configurations are:

x_{lb1} = (4, 2, 3) after increment/ decrement with value one.

x_{lb2} = (1, 1, 2) after rotation (left) operation

x_{lb3} = (3, 4, 2) after increment/ decrement with value one.

$$C_{f_{lbi}}^{X_i}(new), \mathrm{X}_{lbi}(new) = \begin{Bmatrix} if C_{f_{lbi}}^{X_i}(new) < C_{f_{lbi}}^{X_i}(old) then C_{f_{lbi}}^{X_i}(new), \mathrm{X}_{lbi}(new) \\ else C_{f_{lbi}}^{X_i}(old), \mathrm{X}_{lbi}(old) \end{Bmatrix} \quad (3.27)$$

Once mutation operation is completed then MO-PSE calculates new fitness of mutated local best resource configurations using eqn. (3.11). If any fitter particle is found then, the value of R_{gb} will be updated with new resource configuration.

3.2.11. Termination criteria (Z)

Termination criterion is a very important aspect of an iterative algorithm. Therefore, while deciding the termination criterion, two important aspects have been considered in our approach:

- The proposed algorithm should not prematurely converge because of the terminating criterion.
- The algorithm must not trap inside an infinite loop.

With the consideration of above aspects, the proposed algorithm proposed the following terminating criteria. If one of them is true then algorithm will terminate. Two conditions are as follows:

a) Terminates when the maximum number of iteration have been exceeded (M = 100) or,

b) Terminates under the following two stopping criteria :

I. S^1: When no improvement is seen in R_{gb} over '£' number of iteration. (£=10)

II. S^2: If the population reaches to equilibrium state i.e. all particles velocity become zero ($V^+ = 0$).

3.3. Handling control flow graphs through proposed approach

Handling of the CDFG is not a trivial task. The proposed algorithm is capable to handle conditional CDFG. To handle the CDFG, first algorithm finds the worst case path (which is computationally largest). After that, proposed algorithm performs exploration process and find an optimal result for worst case delay of the given CDFG which satisfies the given user constraints for power and execution time. This process enables the methodology to extract the latency information for a given resource configuration during fitness evaluation. Initially, we emphasize on handling three different types of CDFG problems. Firstly, CDFG 1: multiple conditional operators followed by different operation as child node is shown in Figure 3.9(a). Secondly, CDFG 2: single conditional operator followed by two similar operator types as

child node is shown in Figure 3.9(c). Thirdly, CDFG 3: single conditional operator followed by multiple different operator types as child node is shown in Figure 3.9(e). For example, in case of Figure 3.9(a), the proposed algorithm extracts the worst case DFG as shown in Figure 3.9(b) from the given CDFG problem. Then, the latency and cycle time information are extracted from worst case DFG for a given resource configuration for final fitness computation. Based on the fitness, the proposed algorithm explores the optimal resource configuration. *(Note-The datapath circuit and schematic diagram of CDFG1 are given in Appendix A.)*

Note – Results of the proposed method are given in chapter 7 section 7.1

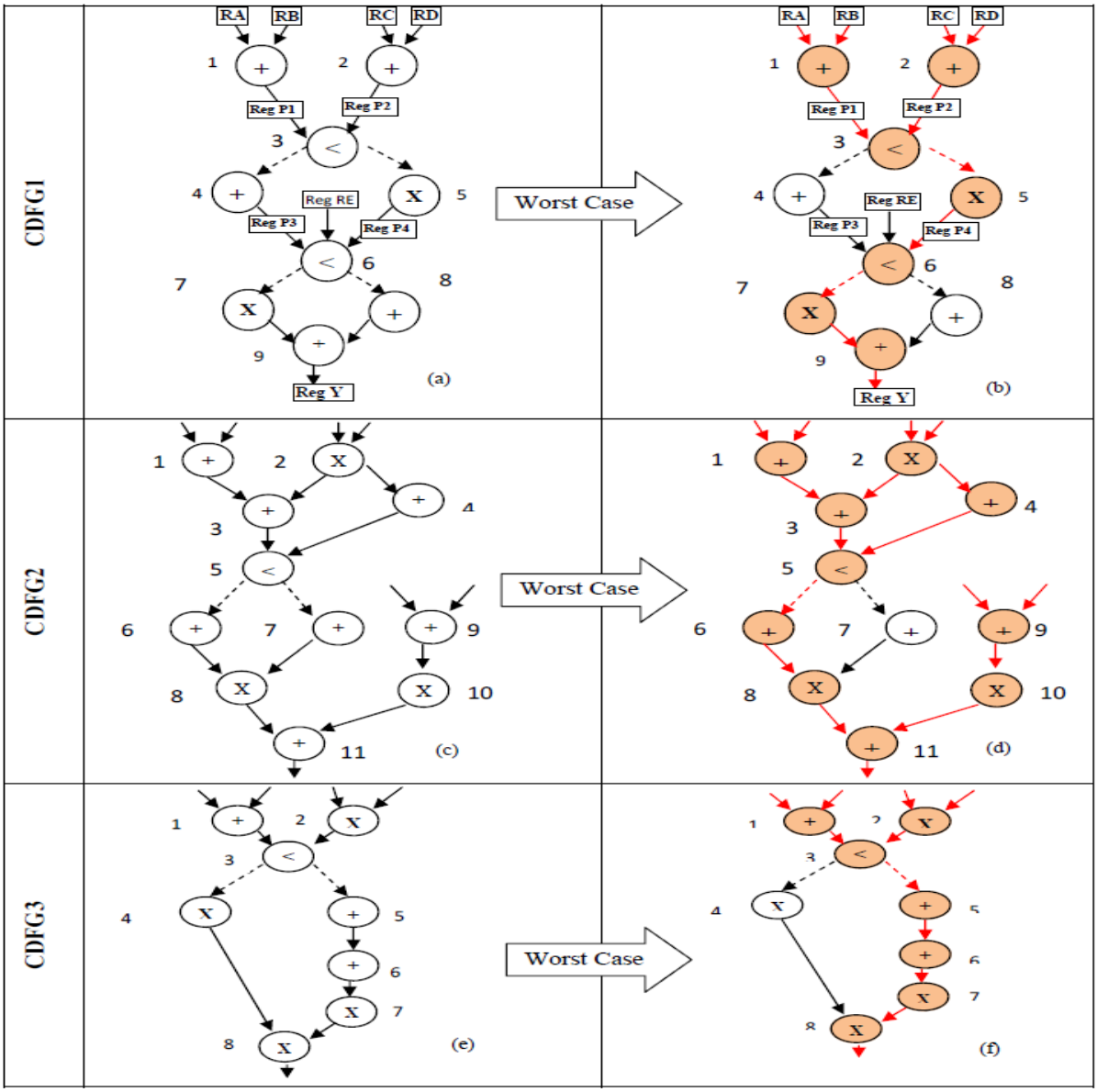


Figure 3.9 Control and data flow graph [56] (a) CDFG1 (b) computationally worst case DFG of CDFG1 (c) CDFG2 (d) computationally worst case DFG of CDFG2 (e) CDFG3 (d) computationally worst case DFG of CDFG3 (only colored node of (b), (d), (e) are taken into account in while extracting the latency information)

Chapter 4

Automated Exploration of Datapath and Unrolling Factor during Power-Performance Trade-off in Architectural Synthesis Using Multi-Dimensional PSO Algorithm

After achieving the first milestone of this research, i.e. solving the design space exploration problem for data intensive applications through proposed MO-PSE. The second milestone is to solve the design space exploration problem for control and data intensive applications (single loop based). In the loop based applications designers have opportunity to optimize the design by loop transformation. Loop unrolling is a widely used technique for loop transformation by the designers, to exploit the parallelism, for better throughput in control data flow applications. In loop unrolling, designers replicate the loop body to achieve parallelism. But at the same time, to perform parallel operations; more resources, interconnects and storage are required which further increase the power consumption of the design. So, it becomes critical for designers to search an optimal combination of unrolling factor and datapath for high quality design [3, 18, 38].

This chapter presents an automated design space exploration methodology for simultaneous exploration of datapath and unrolling factor in high level synthesis. Proposed approach utilizes the exploration capability of particle swarm optimization to solve design space exploration problem during multi parametric optimization for control and data intensive applications (based on single loop). To the best of the authors' knowledge, this is the first approach which automatically handles both datapath and loop unrolling factor simultaneously by swarm based optimization for power performance trade-off in high level synthesis for application specific computing. Moreover, a novel prediction (estimation) model for execution time is proposed in this chapter which is a non-linear function of UF. Using this model, the execution time of the complete application can be estimated based on resource combination found without completely unrolling CDFGs in most cases. The detailed description of the proposed methodology is given in subsequent sections.

4.1. Problem Formulation

Given a CDFG, explore the design space and find out an optimal solution which satisfies the user constraints and minimizes the overall cost. The problem can be formulated as:

Find: *Optimal* $(X_i) = (R_x, UF_N)$

with minimum hybrid cost: P_T and T_E;

Subjected to: $P_T <= P_{cons}$ and $T_E <= T_{cons}$

where, 'X_i' is a set comprising of resources combination and UF formally represented as:

$$X_i = (R_x, UF_N) = \{N(R_1), N(R_2), ...N(R_d)...N(R_{D-1}), UF_N\}$$

where, $N(R_d)$ is the number of instances of resource type *'R_d'*, 'UF_N' is N^{th} unrolling factor of loop, 'R_x' is a candidate resource combination for optimal solution; *'P_T'* and *'T_E'* are the power and execution time consumed by a candidate solution; *'P_{cons}'* and *'T_{cons}'* is power and execution time constraint specified by the user. *'D-1'* is the total number of resource types.

4.2. The Proposed Framework and Mapping Process

The block diagram of the proposed approach is shown in Figure 4.2 and the pseudo code of proposed approach is presented in Figure 4.3 while the proposed mapping is given in Figure 4.1. To transform the PSO into multi-objective DSE problem the position of a particle is represented by a set comprising of resource combination and UF; total number of dimensions is represented by sum of the number of resource types and UF. Finally, the velocity of the particle in d^{th} dimension acts as a parameter that provides the drift during DSE. Based on the pseudo code shown in Figure 4.3, the description of the algorithm with demonstration is given in subsequent sections.

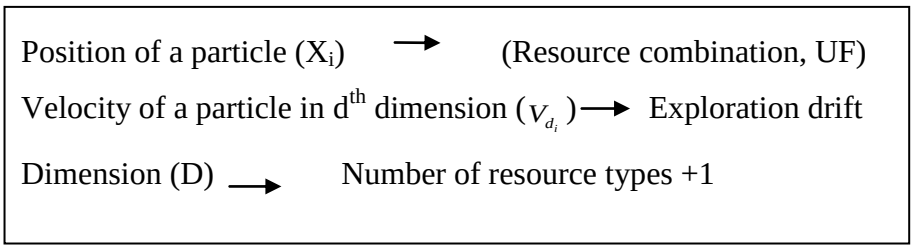


Figure 4.1 Proposed mapping of the DSE problem with PSO

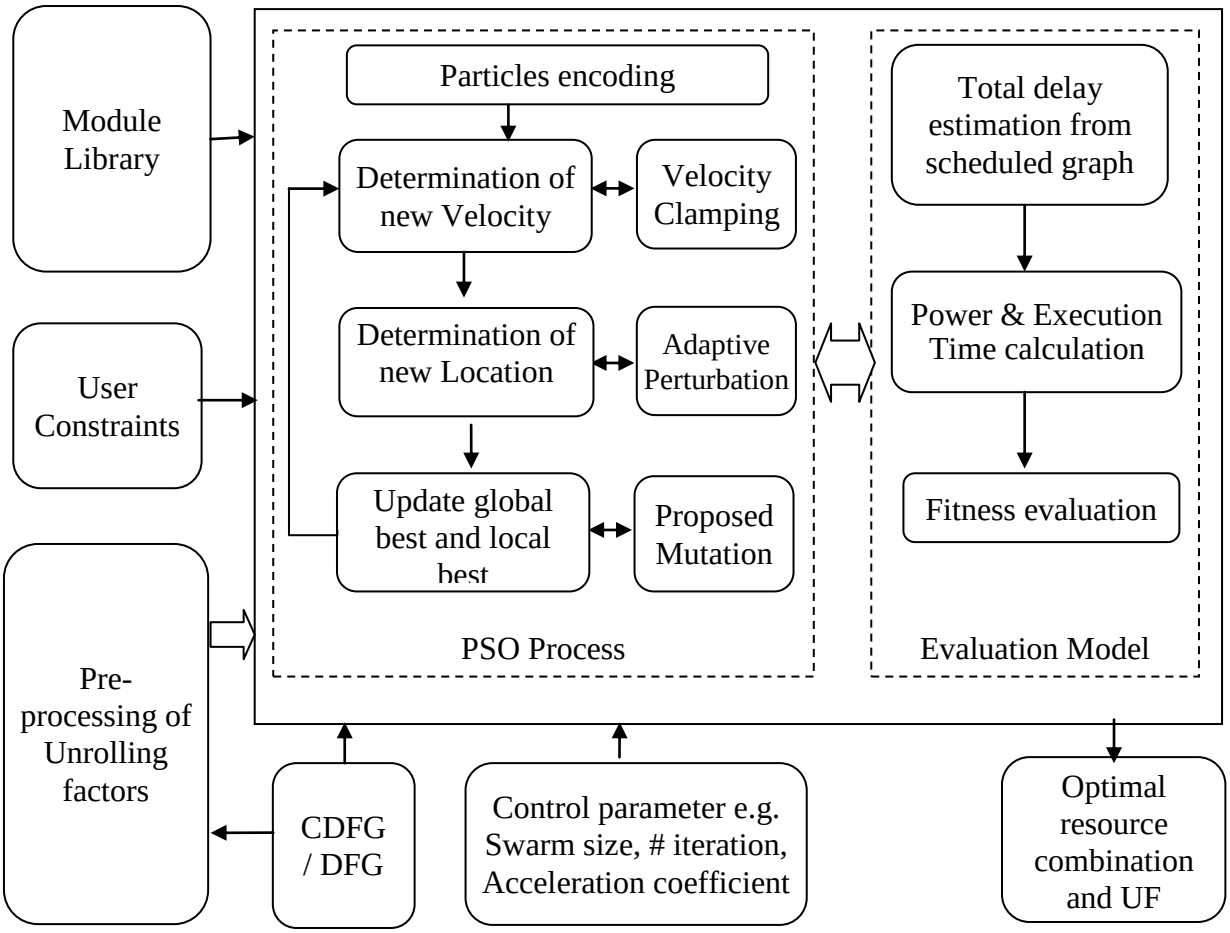


Figure 4.2: Block diagram of proposed approach

4.3. Proposed Evaluation Models

For evaluation of a particle (or design point), the following models have been proposed.

4.3.1. Proposed model for execution time

In order to describe the formulation of proposed execution time (T_E) (function of loop unrolling factor) for a CDFG, an example of loop unrolling is used, shown in Figure 4.4. Figure 4.4(a) shows 'C' code of the original loop and Figure 4.4(b) shows an As Soon As Possible (ASAP) scheduled CDFG unrolled once with resource constraints of 2(*), 2(+) and 1(<). The execution delay model for a loop unrolled CDFG is derived considering the following three possible cases:

Algorithm: *H-SI* based Exploration

Input: CDFG, module library, user constraints

Output: Optimal combination resources (FU's) and unrolling factor (UF)

Begin

Read library information and CDFG/DFG

Check for boundary constraint ( )

//perform pre-processing of UF for screening the UF //

Pre-processing UF ( )

//Initialize particle location and velocity//

Initialization ( )

//Calculate fitness of all particles//

For i=1 to S Do *// where S is the swarm size//*

$C_f^{X_i} = calculatefitness(X_i)$

End Loop *//for i*

// Update local best solution and fitness of the particle//

Update global best solution ( )

//Iteration process starts here//

While (!stopping condition)

For i=1 to S **Do**

For d=1 to D **Do**

//Determination of new velocity//

$V_{d_i}^{+} = \omega V_{d_i} + b_1 r_1 \left[R_{d_{lbi}} - R_{d_i} \right] + b_2 r_2 \left[R_{d_{gb}} - R_{d_i} \right]$

Check for **velocity clamping ( )**

//Determination of new solution//

$R_{d_i}^{+} = R_{d_i} + V_{d_i}^{+}$

Check for **adaptive end terminal perturbation ( )**

End Loop *// for d*

$C_f^{X_i} = calculatefitness(X_i)$

//Update local best position of the particle //

if ($C_f^{X_i} < C_{f_{lbi}}^{X_i}$) *then*

$C_{f_{lbi}}^{X_i} = C_f^{X_i}$

$X_{lbi} = X_i$

End *if*

End Loop *// for i*

Update global best solution ( )

//perform mutation //

Adaptive rotation mutation ( )

Update global best solution ( )

End while loop

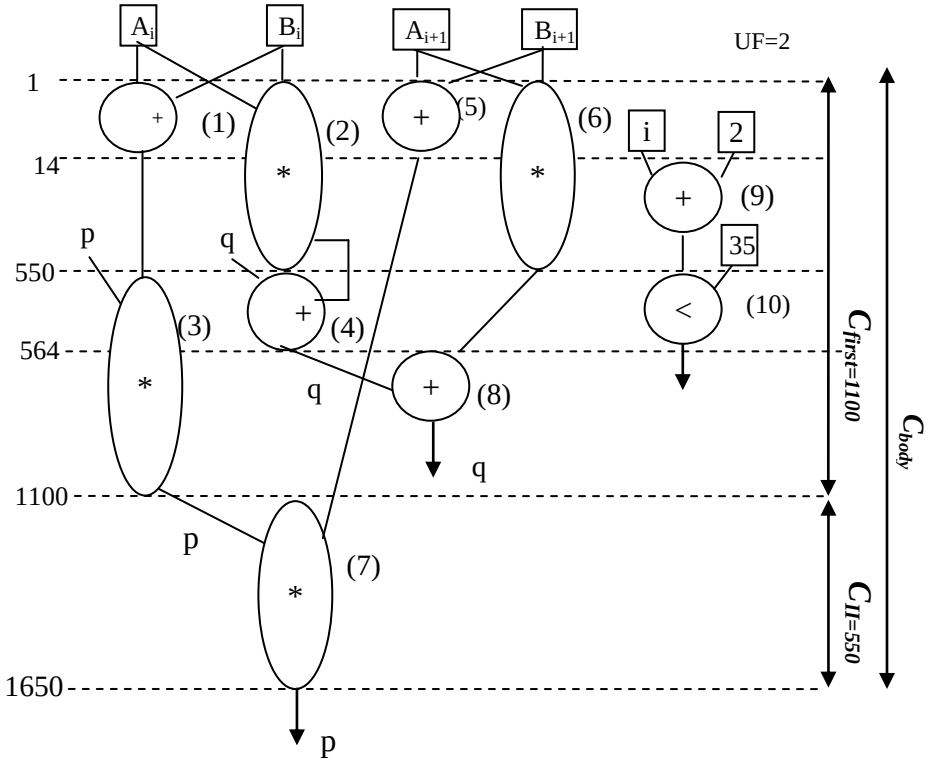
Output: global best solution

END

Figure 4.3 Pseudo code of proposed exploration

```
For(i=0; i<36; i++) {
        P = P * (A_i + B_i)
        q = q + (A_i * B_i)
}
```

(a) 'C' code of original loop



(b) Loop unrolled once (UF = 2)

Figure 4.4 Demonstration of loop unrolling based on 2(*), 2(+), 1(<) constraints using As Soon As Possible (ASAP) scheduling for test case

Case 1: When the unrolling factor (UF) is equal to one (indicates no unrolling) then,

*Total # of control steps (CSs) = # of CSs required to execute loop body once * # of duplicate iterations of loop body*

$$C_T = (C_{body} * \alpha) \tag{4.1}$$

$$= (C_{body} * \left\lceil \frac{I}{UF} \right\rceil^{quotient}) \; where, \; C_{body} = C_{first} \; and \; UF = 1$$

$$C_T = (C_{first} * I) \tag{4.2}$$

where, 'C_T' is total CSs required to execute the loop completely, 'C_{body}' is the number of CSs required to execute loop body once, 'C_{first}' is number of CSs required to execute first iteration, 'I' is the maximum number of iteration (loop count), α is $\left(\frac{I}{UF}\right)^{quotient}$.

Case 2: When UF evenly divides the loop count (I), then the total number of CSs is: $C_{body} = (C_{first} + (UF - 1) * C_{II})$ (as evident from Figure 4.4(b)). Now, substituting C_{body} in eqn. (4.1) yields:

$$C_T = (C_{first} + (UF - 1) * C_{II}) * \alpha \tag{4.3}$$

where, 'C_{II}' is the number of CSs required between initiations of consecutive iterations.

Case 3: When UF unevenly divides I: In such case, *ImodUF* iterations will be executed sequentially, therefore, the total number of CSs is:

$$C_T = (C_{first} + (UF - 1) * C_{II}) * \alpha + (I \bmod UF) * C_{first} \tag{4.4}$$

{Total CSs for unrolled loop} {Total CSs for sequential loop}

Furthermore, execution time for the system is calculated as:

$$T_E = \Delta * C_T \tag{4.5}$$

where, 'Δ' is the delay of one CS in nanoseconds.

Finally, T_E can be formulated as:

$$T_E = \Delta * ((C_{first} + (UF - 1) * C_{II}) * \alpha + (I \bmod UF) * C_{first}) \tag{4.6}$$

Eqn. (4.6) is an estimation model for T_E, where there is no need to tediously unroll the CDFG to calculate T_E, unless # of independent operations required to be performed in parallel due to unrolling exceeds the available resource units (specified in a solution).

While for the DFGs, the estimation of execution time is [32, 35]:

$$T_E = [L + (N - 1) \cdot T_c] \tag{4.7}$$

where, '*N'* is the number of input samples to be processed by a functionally pipelined data-path, '*L'* represents the latency of a scheduling solution and 'T_c' represents the initiation interval or cycle time of a scheduling solution

4.3.2. Proposed Power Model

The total power consumed by a resource combination is denoted by 'P_T'. 'P_T' is composed of dynamic power (P_D) and static power (P_S) given by eqn. (4.8) below:

$$P_T = P_D + P_S \tag{4.8}$$

The average dynamic power consumed by is source combination is a function of dynamic activity of the resources for CDFG. It is formulated as:

$$Avg\ dynamic\ power(P_D) = \frac{Total\ energy\ consumption}{Total\ execution\ time}$$

$$P_D = \frac{\alpha * (E_{FU} + E_{MUX/DMUX})}{\Delta * ((C_{first} + (UF - 1) * C_{II}) * \alpha + (I \bmod UF) * C_{first})} \quad (4.9)$$

For a DFG, the average dynamic power can be described as:

$$P_D = \frac{N \cdot (E_{FU} + E_{MUX/DMUX})}{L + (N - 1) \cdot T_c} \quad (4.10)$$

where, 'E_{FU}' represents the total energy consumption of the resources, '$E_{MUX/DMUX}$' represents the total energy consumed by multiplexer and demultiplexer. The variables C_{first}, C_{II}, I, UF, α, L, N and T_c have already been defined in previous sections.

On the other hand, static power is a function of area of the resources, multiplexer and the leakage power per transistor. Therefore static power can be defined as:

$$P_S = \left[\sum_{i=1}^{v} (N_{R_i} \cdot K_{R_i}) + N_{MUX/DMUX} \cdot K_{MUX/DMUX} \right] * P_c \quad (4.11)$$

where, 'N_{Ri}' represents the number of instance of resource R_i; 'K_{Ri}' represents the area occupied by resource R_i, 'v' is the number of resource types, '$N_{MUX/DMUX}$' is number of the multiplexer or demultiplexer, '$K_{MUX/DMUX}$' is area occupied by the multiplexer or demultiplexer and 'p_c' denotes the power dissipated per area unit (e.g. transistors).

4.3.3. Proposed model for fitness function

The proposed fitness function (considering execution time and power consumption of a solution) is defined as:

$$C_f^{X_i} = \varphi_1 \frac{P_T - P_{cons}}{P_{\max}} + \varphi_2 \frac{T_E - T_{cons}}{T_{\max}} \quad (4.12)$$

Where, $C_f^{X_i}$ = Fitness of particle X_i; φ_1, φ_2 = User defined weights for power and execution time.

4.4. Demonstration of proposed methodology

This section describes the proposed approach (based on particle swarm optimization [48, 49]) with demonstration of a sample test case.

4.4.1. User specification

The CDFG used for demonstration (shown in Figure 4.4) along with the user specified design constraints for power and execution time as well as module library [52, 54, 74, 92] are taken as inputs to the proposed framework. For the sake of explanation, we are assuming some real values for power constraint ($P_{cons=}$1.5mW) and execution time constraint (T_{cons}=500us); Maximum available multiplier FU's: 4, adder FU's: 4, comparator FU's: 2 and total user specified loop iteration I = 36; additionally power dissipated per transistor (p_c) is assumed to be 29.33nW; also, number/type of mux/demux is directly extracted from the scheduling solution.

4.4.2. Boundary constraint check module

After specifying constraints, the proposed framework checks for valid user constraints. If user constraints are not valid then an error is shown and again requests for valid values:

1. Check: $P_{\min} > P_{cons} > P_{\max} \parallel T_{\min} > T_{cons} > T_{\max}$

2. If above condition is true then stop and correct constraints.

Else execute the initialization process of module.

In order to check the constraints, maximum and minimum value of power and execution time are determined. Minimum power (P_{min}) and maximum execution time (T_{max}) are calculated with minimum resource (single instance of resource) and min(UF)=1, while, maximum power (P_{max}) and minimum execution time (T_{min}) is calculated with maximum resource and max(UF)=I (maximum # of loop iterations).

4.4.3. Pre-processing of unrolling factor

In order to prune the design space, the proposed methodology performs pre-processing of the unrolling factor. The proposed pre-processing algorithm, shown in Figure 4.5, filters unfit UFs to create a list of viable solutions. The pre-processing algorithm filters those UFs which produce higher sequential loops and also filter higher value of UF because higher UFs gives minor improvement in execution time with high power consumption. Therefore, overall cost is high of such unrolling factors. Moreover, to ensure the inclusion of good candidates, some special UFs have been added which may have been initially screened out in pre-processing phase, using the algorithm shown in Figure 4.6.

An example of pre-processing for test case is given in Table 4.1 for I=36.

Pre-processing of unrolling factor

Input – value of 'I' (Total no. of loop iteration)
Output – screened set of unrolling factor (UF)

1 **Begin**
// Screening of UF//
2 **For** UF =2 to I **Do**
2.1 **IF** ((I mod UF < $\frac{UF}{2}$) &&
(UF <= I/2)) **Then**
//Add UF into the accepted UF list//
2.2 Accepted UF[k] = UF
2.3 k++
2.4 **End IF**
2.5 **End For**
3 **End**

Figure 4.5 Pre-processing of UF

Algorithm

1 Begin
2 For UF =2 to I do
//All U F are added into the accepted list until (I mod UF) < $\frac{UF}{2}$ *//*
2.1 **IF** (*(I mod UF)<* $\frac{UF}{2}$) **Then**
2.2 Terminate adding process jump to the end of the function
2.3 **End IF**
2.4 Accepted UF[k] =UF
2.5 k++
2.6 **End For**
3 **End**

Figure 4.6 Algorithm for inclusion of some special UFs

4.4.4. Proposed initialization process of particles

After preprocessing step, initialization of particles is done. During initialization process particles position, velocity and acceleration coefficient are initialized as follows:

a) *Position*: For a CDFG, a particle position X_i is given as:

$X_i = (N(R_1), (N(R_2),..(N(R_d).. (N(R_{D-1}),UF)$

In the proposed approach, the initialization of particles is such that it uniformly covers the entire design space as follows:

$$X_1 = (min(R_1), min(R_2),.. min(R_{D-1}),min(UF)) \quad (4.13)$$

$$X_2 = (max(R_1), max(R_2),.. max(R_{D-1}),max(UF) \quad (4.14)$$

$$X_3=(((min(R_1)+max(R_1))/2..,((min(R_{D-1})+max(R_{D-1}))/2,max(UF)/2) \quad (4.15)$$

Rest of the particles positions ($X_4 \ldots X_n$) are initialized with random values between minimum and maximum values of resources and UF as explain in section 3.2.2. Since, an optimal design solution to a multi-objective exploration problem always lies between the maximum parallel and serial implementation of the application. Therefore, keeping in mind the above, X_1 is represented by the serial implementation, X_2 by parallel implementation, X_3 with the mid value between serial and parallel implementation and X_4-X_n as scattered positions between serial and parallel implementations. Hence, using eqn. (4.13) – (4.15),

Table 4.1: An example of pre-processing of unrolling factors for test case

I=36			
UF	Sequential Loop (I mod UF)	Pipelined loop *(I- I mod UF)*	Accepted (1)
2	0	36	1
3	0	36	1
4	0	36	1
5	1	35	1
6	0	36	1
7	1	35	1
8	4	32	0
9	0	36	1
10	6	30	0
11	3	33	1
12	0	36	1
13	10	26	0
14	8	28	0
15	6	30	1
16	4	32	1
17	2	34	1
18	0	36	1
19	17	19	0
20	16	20	0
21	15	21	0
22	14	22	0
23	13	23	0
24	12	24	0
25	11	25	0
26	10	26	0
27	9	27	0
28	8	28	0
29	7	29	0
30	6	30	0
31	5	31	0
32	4	32	0
33	3	33	0
34	2	34	0
35	1	35	0
36	0	36	0

X_1 = (1, 1, 1, 1), X_2 = (4, 4, 2, 36); X_3= (2, 2, 1, 18) and X_4= (2, 2, 1, 2).

b) *Initialization of velocity, acceleration coefficient*

Velocities of all particles are initialized to zero in the proposed approach [49] while acceleration coefficient is initialized to any value between 1 and 4 in order to attain convergence as proved mathematically in [50].

4.4.5. Determination fitness and update local and global best position

Once the all particles are initialized, fitness of the particles is determined. First power is calculated according to eqn. (4.8), (4.9), and (4.11).

Example, for particle X_4= (2, 2, 1, 2):

$$P_S = (2464 * 2 + 2030 * 2 + 2030 + 3 * 3 * 126 + 3 * 3 * 126) * 2.933 * 10^{-6} = 0.389mW$$

$$P_D = \frac{18 * (4 * 9.8 + 5 * 0.739 + 0.739 + 6 * 3 * 0.1 + 6 * 3 * 0.1)\ \text{pj}}{20 * ((1100 + (2-1) * 550) * 18 + 0 * 1100)} = 0.0006mW$$

$$P_T = 0.39mW \tag{4.16}$$

Next, execution time metric is calculated as using eqn. (4.6):

$$T_E = 20 * ((1100 + (2-1) * 550) * 18 + 0 * 1100) = 594000ns \tag{4.17}$$

Finally the fitness of the particle is calculated by eqn. (4.12). Therefore, the fitness of X_4 is:

$$C_f^{X_4} = -0.101 \tag{4.18}$$

Similarly, fitness of all the particles is determined by providing equal weightage to power (φ_1) and execution time (φ_2). Further, local best position of the particles and global best position are found out as describe in chapter 3 section 3.2.5.

4.4.6. Determine global best position

The global best position of the population is determined using eqn. (4.19) as follows:

$$X_{gb} = X_i[Min(C_{f_{lb1}}^{X_1}, C_{f_{lb1}}^{X_2}, C_{f_{lb1}}^{X_3} ... C_{f_{lbn}}^{X_n})] \tag{4.19}$$

The global best particle position has minimum cost among all local best positions (X_{lb1} …..X_{lbn}).

4.4.7. Determination of new position of each particle

Iteration process initiates at this step. According to the algorithm, in each iteration, the new position of a particle X_i in d^{th} dimension can be given by:

$$R_{d_i}^{+} = R_{d_i} + V_{d_i}^{+} \tag{4.20}$$

where, $R_{d_i}^{+}$ = new resource value or UF value of particle X_i in d^{th} dimension and R_{d_i} = previous resource value or UF value of particle X_i in d^{th} dimension; $V_{d_i}^{+}$ is the new velocity of particle X_i in d^{th} dimension which is updated by eqn. (4.21):

$$V_{d_i}^{+} = \omega V_{d_i} + b_1 r_1 \left[R_{d_{lbi}} - R_{d_i} \right] + b_2 r_2 \left[R_{d_{gb}} - R_{d_i} \right] \tag{4.21}$$

where, ' $R_{d_{lbi}}$ ' is the resource value of X_{lbi} in d^{th} dimension, ωV_{id} is the inertia component, ' $R_{d_{gb}}$ ' is the resource value of X_{gb} in d^{th} dimension, b_1, b_2 are acceleration coefficients and r_1 and r_2 are random numbers between [0-1]

Where, $X_{lbi} = \{R_{1_{lbi}}, ...R_{D-1_{lbi}}, \mathrm{UF}\}$ *and* $X_{gb} = \{R_{1_{gb}}, ...R_{D-1_{gb}}, \mathrm{UF}\}$

4.4.8. Adaptive end terminal perturbation & adaptive rotation mutation

To handle boundary outreach problem during exploration process we propose adaptive end terminal perturbation, described in section 3.2.7.

In order to increase variation and diversity, mutation is performed on all the local best positions of each particle with probability M_p=1.0 using *Adaptive rotation mutation* described in section 3.2.9.

4.4.9. Stopping condition

The proposed algorithm terminates when the maximum number of iterations exceeds by 100 count, or when no improvement is visible in X_{gb} over '£' number of iteration. (£=10). The detailed description of the stopping criterion is given in section 3.2.10.

Note – Results of the proposed method are given in chapter 7 section 7.2

Chapter 5

Simultaneous Exploration of Optimal Datapath and Loop Based High level Transformation during Area-Delay Trade-off in Architectural Synthesis Using Swarm Intelligence

With the shrinking of device geometry with the evolution of technology, area optimization has become an important aspect for the designers. The area optimization is highly dependent on unrolling factor in case of control and data intensive application. Thus, the impact of unrolling factor (with datapath) in the circuit area during designing of ASPs or hardware accelerators requires significant attention. Therefore, it becomes very essential to solve the problem of simultaneous exploration of optimal datapath and UF during area- delay trade-off in HLS.

This chapter presents an automated design space exploration methodology based on hyper dimensional swarm encoding for simultaneous searching of datapath and unrolling factor (for single loop based application) during area-performance trade-off in high level synthesis for application specific computing. During exploration process the proposed methodology not only maintains trade-off between conflicting parameters such as area and execution time but also resolves orthogonal issues such as quality of results and exploration speed. The detail description of the proposed methodology is given in subsequent sections of this chapter.

5.1. Problem Formulation

Given a CDFG, explore the design space and find an optimal solution which satisfies the conflicting user constraints and minimize the overall cost. The problem solved is formulated as:

Find: *Optimal* $(X_i) = (R_x, UF_N)$

with minimum hybrid cost: A_T and T_E;

Subjected to: $A_T <= A_{cons}$ and $T_E <= T_{cons}$

where, 'X_i' is a set comprising of resources combination and UF formally represented as:

$$X_i = (R_x, UF_N) = \{N(R_1), N(R_2), \ldots N(R_d) \ldots N(R_{D-1}), UF_N\}$$

where, '$N(R_d)$' is the number of instances of resource type 'R_d'; 'D-1' is the total number of resource types; UF_N is N^{th} unrolling factor; 'R_x' is an candidate resource combination; 'A_T' and 'T_E' are the area and execution time consumed by a candidate solution; 'A_{cons}' and 'T_{cons}' is area and execution time constraint specified by the user.

5.2. The Proposed Framework

The block diagram of proposed approach is shown in Figure 5.1 and the flow diagram of proposed approach is shown in Figure 5.2. Based on the flow diagram, the description of the algorithm with demonstration is given in subsequent sections.

5.3. Evaluation Models

During exploration process the design points are need to be evaluate. To evaluate a design point model for execution time, model for area evaluation and model for cost (fitness) have been presented in the following subsection:

5.3.1. Execution time model

To evaluate the execution time of CDFG based application without tediously complete unrolling the CDFG during exploration process, the execution time model formulated as:

$$T_E = \Delta * ((C_{first} + (UF - 1) * C_{II}) * \alpha + (I \bmod UF) * C_{first}) \quad (5.1)$$

Where, 'T_E' is the total execution time; 'C_{first}' is number of CSs required to execute first iteration, 'C_{II}' is the number of CSs required between initiations of consecutive iterations 'I' is the maximum number of iteration (loop count), UF is the specified loop unrolling factor;

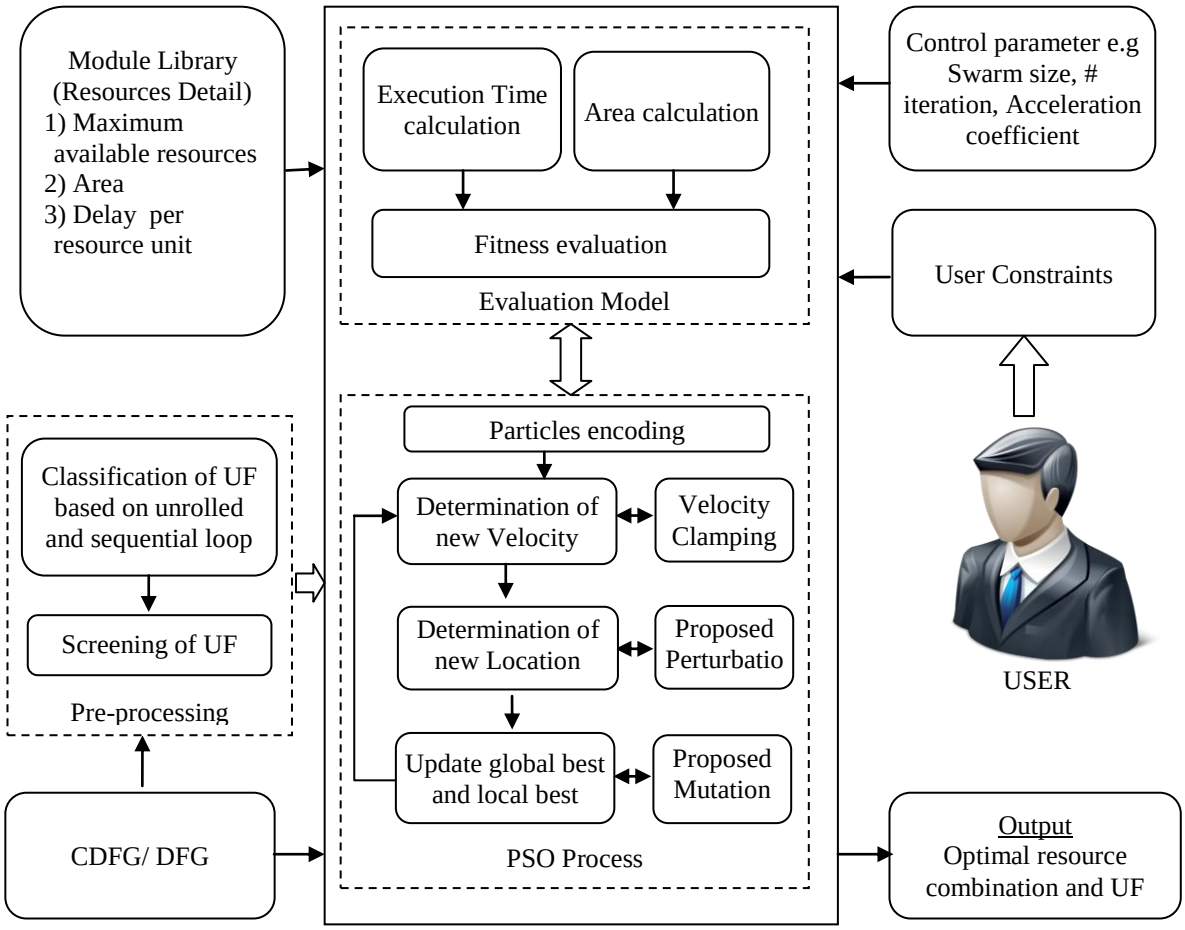


Figure 5.1 Block diagram of proposed approach

'Δ' is the delay of one CS in nanoseconds; α is $floor\ value\left(\frac{I}{UF}\right)$. *(Note - The detail description of the execution time model has been given in chapter 4 section 4.3.)*

Eqn (5.1) is an estimation model for T_E, where the necessity of tediously unrolling the CDFG is not required to calculate T_E, unless # of independent operations required to be performed in parallel due to unrolling exceeds the available resource units (specified in a solution).

5.3.1.1. Area model

The total area consumed by a resource combination and multiplexer is denoted by 'A_T'.

$$A_T = \left[\sum_{i=1}^{v} (N_{R_i} \cdot K_{R_i}) + N_{MUX/DMUX} \cdot K_{MUX/DMUX} \right] \tag{5.2}$$

where, 'N_{Ri}' represents the number of instance of resource R_i; 'K_{Ri}' represents the area occupied by resource R_i (#transistor), 'v' is the number of resource types, '$N_{MUX/DMUX}$' is number of the multiplexer or demultiplexer, and '$K_{MUX/DMUX}$' is area occupied by the

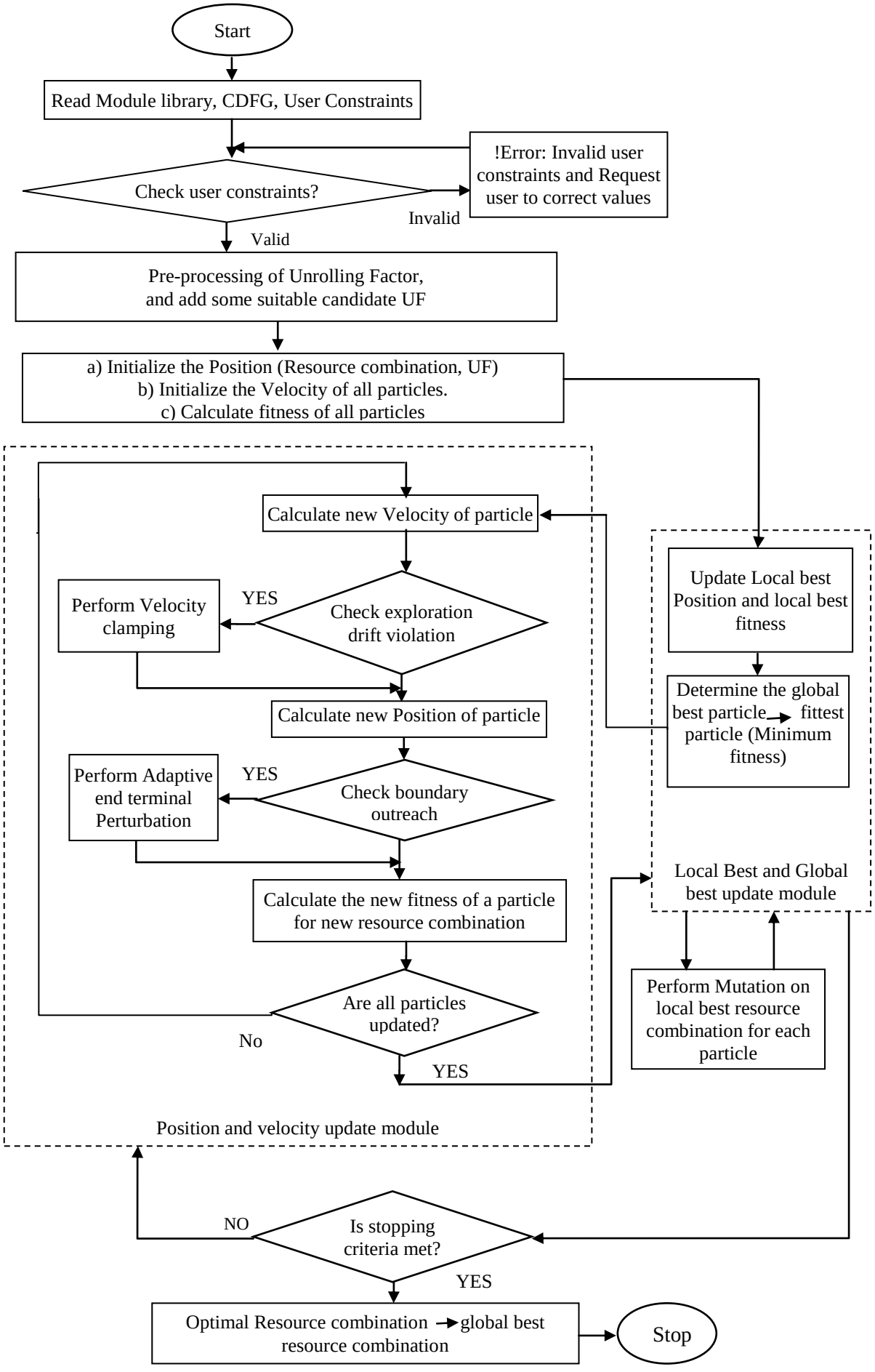


Figure 5.2 Flow diagram of proposed algorithm

multiplexer or demultiplexer *(Note:- area is calculated in terms of number of au) (1 au = 1 transistor).*

5.3.1.2. Model for fitness function

The fitness function (considering execution time and area of a solution) is defined as:

$$C_f^{X_i} = \varphi_1 \frac{A_T - A_{cons}}{A_{\max}} + \varphi_2 \frac{T_E - T_{cons}}{T_{\max}} \quad (5.3)$$

Where, $C_f^{X_i}$ = Fitness of particle X_i; φ_1, φ_2 = User defined weights for area and execution time; T_{Max} is maximum execution time calculated with minimum resource (single instance of resource) and min (UF)=1 while A_{max} is calculated with maximum resource and max(UF)=I (maximum # of iterations).

5.4. Demonstration of Proposed Methodology

5.4.1. User specification

The CDFG used for demonstration (as shown in Figure 5.3) along with the user specified design constraints for area and execution time as well as the module library[52, 54, 74, 92] are taken as inputs to the proposed framework (area of adder= 2030au, multiplier= 2464au, comparator= 2030au multiplexer (2:1) = 126au ; delay of adder= 270ns, multiplier = 11000ns, comparator= 270ns multiplexer= 20ns; where one (4:1) multiplexer needs three (2:1) multiplexer and one (8:1) multiplexer needs seven (2:1) multiplexer; we assume (1 au = 1 transistor). For the sake of explanation, we are assuming some real values for area constraint ($A_{cons=}$18000 au and execution time constraint (T_{cons}=60us); Maximum available multiplier FU's: 8, adder FU's: 4, comparator FU's: 2 and total user specified loop iteration I = 8; number/type of mux/demux is directly extracted from the scheduling.

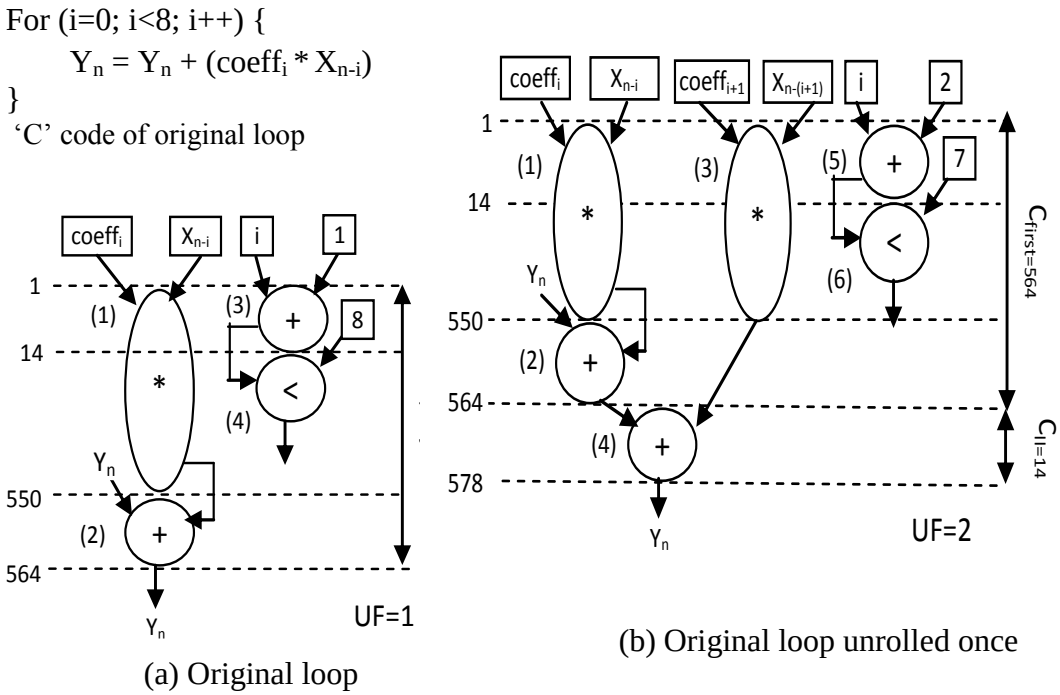
5.4.2. Boundary constraint check module

After specifying constraints, the proposed framework checks for valid user constraints. If user constraints are not valid then an error is shown and requests for valid values:

1. Check: $A_{\min} > A_{cons} > A_{\max} \,||\, T_{\min} > T_{cons} > T_{\max}$

2. If above condition is true then stop and correct constraints.

Else execute the initialization process of module.



(a) Original loop

(b) Original loop unrolled once

Figure 5.3 Demonstration of loop unrolling based on a resource constraint of 2(*), 2(+), 1(<) for FIR

In order to check the constraints, maximum and minimum value of area and execution time are determined. Minimum are (A_{min}) and maximum execution time (T_{max}) are calculated with minimum resource (single instance of resource) and min (UF)=1, while, maximum area (A_{max}) and minimum execution time (T_{min}) is calculated with maximum resource and max(UF)=I (maximum # of loop iterations).

5.4.3. Pre-processing of unrolling factor

In order to reduce the design space and enhance the exploration speed, pre-processing of the loop unrolling factor is performed by the proposed algorithm. The pre-processing algorithm is given in chapter 4 (Figure 4.5 and Figure 4.6). An example of pre-processing shown in Table 5.1 for FIR shown in Figure 5.3 (used for demonstration).

5.4.4. Initialization process of particles

After pre-processing step, initialization of the particles takes place as describe in chapter 4 section 4.4.

a) Position

Table 5.1: An example of pre-processing of unrolling factors for FIR

I=8			
UF	Sequential Loop (I mod UF)	Pipelined loop *(I- I mod UF)*	Accepted (1)
2	8	0	1
3	6	2	0
4	8	0	1
5	5	3	0
6	6	2	0
7	7	1	0
8	8	0	0

Hence, the initial solution are:

$X_1 = (1, 1, 1, 1)$

$X_2 = (8, 4, 2, 8)$

$X_3 = (4, 2, 1, 4)$

b) Initialization of velocity and acceleration coefficient

Velocities of all particles are initialized to zero in the proposed approach and acceleration coefficient can be initialized to any value between 1 and 4 [50].

5.4.5. Determination of local and global best position

Once the all particles are initialized, fitness of the particles is determined. First area is calculated according to eqn. (5.2).

Example, for particle $X_3 = (4, 2, 1, 4)$:

$$A_T = (2464 * 4 + 2030 * 2 + 2030 * 1 + 12 * 126 + 3 * 7 * 126 + 3 * 3 * 126)$$

$$A_T = 21238 \text{ au} \tag{5.4}$$

Where the module values (library) are assumed, discussed in section 5.3.1.

Next, execution time metric is calculated as using eqn. (5.1):

$$T_E = 20 * ((564 + (4 - 1) * 14) * 2 + 0 * 564) = 24240 ns \tag{5.5}$$

(Note- the values of C_{first}, C_{II}, are derived from the ASAP scheduled CDFG with resource combination: 4 (*), 2 (+), 1(<), UF=2) shown in figure 5.3(b).

Finally the fitness of the particles is calculated by eqn. (5.3). Therefore, the fitness of X_3 is:

$$C_f^{X_3} = -0.1507 \tag{5.6}$$

Similarly, fitness of all the particles is determined and local best position of the particle and global best particle are found out as describe in chapter 3 section 3.2.5. (Note- $\varphi_1 = \varphi_1 = 0.5$ is chosen for providing equal weightage to area and execution time).

5.4.6. Determination of new position of each particle

Iteration process initiates at this step. According to the algorithm, in each iteration, the new position of a particle X_i in d^{th} dimension can be given by:

$$R_{d_i}^{+} = R_{d_i} + V_{d_i}^{+} \tag{5.7}$$

where, $R_{d_i}^{+}$ = new resource value or UF value of particle X_i in d^{th} dimension and R_{d_i} = previous resource value or UF value of particle X_i in d^{th} dimension; $V_{d_i}^{+}$ is the new velocity of particle X_i in d^{th} dimension which is updated by eqn. (5.8):

$$V_{d_i}^{+} = \omega V_{d_i} + b_1 r_1 \left[R_{d_{lbi}} - R_{d_i} \right] + b_2 r_2 \left[R_{d_{gb}} - R_{d_i} \right] \tag{5.8}$$

where, ' $R_{d_{lbi}}$ ' is the resource value of X_{lbi} in d^{th} dimension, ωV_{id} is the inertia component, ' $R_{d_{gb}}$ ' is the resource value of X_{gb} in d^{th} dimension, b_1, b_2 are acceleration coefficients and r_1 and r_2 are random numbers between [0-1].

Note- $X_{lbi} = \{R_{1_{lbi}}, \ldots R_{D-1_{lbi}}, \mathrm{UF}\}$ *and* $X_{gb} = \{R_{1_{gb}}, \ldots R_{D-1_{gb}}, \mathrm{UF}\}$

For example, particle position X_3, $V_{d_i}^{+}$ and $R_{d_i}^{+}$ are calculated for 1^{st} dimension using eqn. (5.8) and (5.7) as:

$V_{1_3}^{+} = 0.5*0 + 2*0.5(4-4) + 2*0.5(4-4) = 0;$ $\quad R_{1_3}^{+} = 4+0 = 2$

(Note- $R_{d_{gb}}$ *= 4 for 1st dimension is used in eqn. (5.7) and* V_{d_i} *=0 has been assumed initially (as explained in section 5.3.4))*

Similarly, the $V_{d_i}^{+}$ and $R_{d_i}^{+}$ for all dimensions of all particles can be calculated. To handle boundary outreach and excessive drift adaptive end terminal perturbation and velocity

clamping is performed. *(Note: The details of adaptive end terminal perturbation and velocity clamping are given in chapter 3 section 3.2.7.and section 3.2.8 respectively.*

5.4.7. Mutation operation

To increase variation and diversity, mutation is performed on all the local best position of each particles with probability M_p=1.0 using *Adaptive rotation mutation.* The detail description of the mutation algorithm is given in chapter 3 section 3.2.9.

5.4.8. Stopping condition

From chapter 3 section 3.2.10, the proposed algorithm can terminate either:

a) When the maximum number of iterations exceeds 100, or,

b) When no improvement is visible in X_{gb} over '£' number of iteration (£=10).

Note – Results of the proposed method are given in chapter 7 section 7.3

Chapter 6

Swarm Inspired Exploration of Architecture and Unrolling Factors for Nested Loop Based Application in Architectural Synthesis

Loop Unrolling is very popular technique to exploit parallelism and when loop unrolling combine with data path during design space exploration process, then produces better quality design, as we found in our previous investigation on single loop based application (as explain in chapter 4 and 5). Further, this investigation required more efforts for nested loop based application because in nested loops based application designer have more opportunity for unrolling, but, in other hand, the complexity of the problem also increases because of design space increases exponentially. Moreover, the direction of unrolling (i.e., which loops should be unrolled) and the unrolling factor have an extremely strong effect on the efficiency of the execution of the unrolled loop. As we know, in the domain of multimedia, digital signal processing, medical etc, there exist many applications, which are nested loop in nature. Therefore, considering nested loops as part of optimization process during DSE in high level synthesis is crucial for designers.

In order to handle such applications, a novel framework for automated design space exploration of architectures and unrolling factors for perfectly nested loops in architectural synthesis (AS) has been presented in this chapter, which maintains trade-off between power and performance during exploration and also resolve orthogonal issues such as exploration speed and quality of result (QoR). Moreover, a novel model for determining (predicting) execution time based on architecture and unrolling factors (UFs) for nested loop without tiresomely unrolling completely is proposed. The detail description of the proposed approach is given in subsequent sections in this chapter.

6.1. Problem Formulation

Given a CDFG, find a minimal cost solution ($\vec{S}$) which satisfies the conflicting user constraints. The formulation is as follows:

Minimize $f(\vec{S})$ (Hybrid cost of P_T and T_E)

Subjected to: $P_T \le P_{cons}$ *and* $T_E \le T_{cons}$

$$\vec{S} = (\vec{R}, \overrightarrow{UF})$$
$$where,\ \vec{R} = (R_1, R_2, \cdots, R_n)$$
$$L_i^R \le R_i \le U_i^R \ where\ i = 1\ to\ n$$
$$and,\ \overrightarrow{UF} = (UF_1, UF_2, \cdots, UF_m)$$
$$L_j^{UF} \le UF_j \le U_j^{UF}\ where\ j = 1\ to\ m$$

$$\vec{S} = \{R_1, R_2, \cdots, R_n, UF_1, UF_2, \cdots, UF_m\} \quad (6.1)$$

and, D= n+m

where, 'R_n' is the number of instances of n^{th} resource type; 'n' is the total number of resource types; UF_m is unrolling factor of m^{th} loop; 'm' is the total depth of nested loop; '$\vec{R}$' is an candidate resource combination, '$\overrightarrow{UF}$' is a set of candidate UFs; 'L_i^R' and 'U_i^R' are minimum and maximum available i^{th} resource (R_i) type; 'L_j^{UF}' and 'U_j^{UF}' are minimum (no unrolling) and maximum (equal to loop count I) unrolling factor of j^{th} loop; 'P_T' and 'T_E' are the power and execution time consumed by a candidate solution; 'P_{cons}' and 'T_{cons}' are power and execution time constraint specified by the user;'D' is number of dimension in a solution;'m' is number of resource types; 'n' depth of the loop (in nested loop). Unrolling factor indicates the number of times a loop body (single or nested) is unfolded to exploit parallelism. For example, in Figure 6.1 the original loop body (shown in Figure 6.1 (b)) has been typically unrolled (UF=2, 3) with respect to both inner and outer loop (shown in Figure 6.1 (c)). *Note-nested loop cannottransform into single loop when loop body dependent upon both inner and outer indices.*

6.2. The Proposed Framework and Mapping Process

The proposed mapping between Design space exploration problem for nested loop based application and particle swarm optimization is shown in Figure 6.2 and the block diagram of the methodology given in Figure 6.3 while the algorithm for exploration process is explained

```
For(i=0;i<8;i++){
  For(j=0;j<8;j++){
     Ai +=Bj*Bj-i
 }
}
```

Figure 6.1 (a) 'C' code for original loop of Autocorrelation benchmark

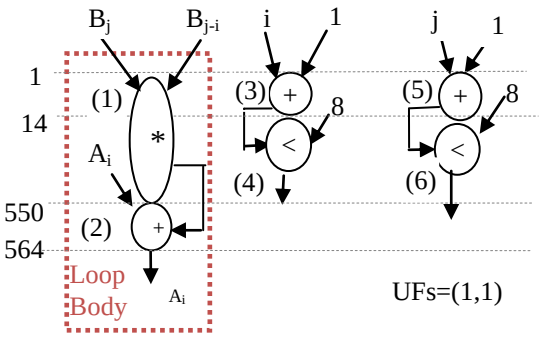


Figure 6.1 (b) CDFG of Original loop

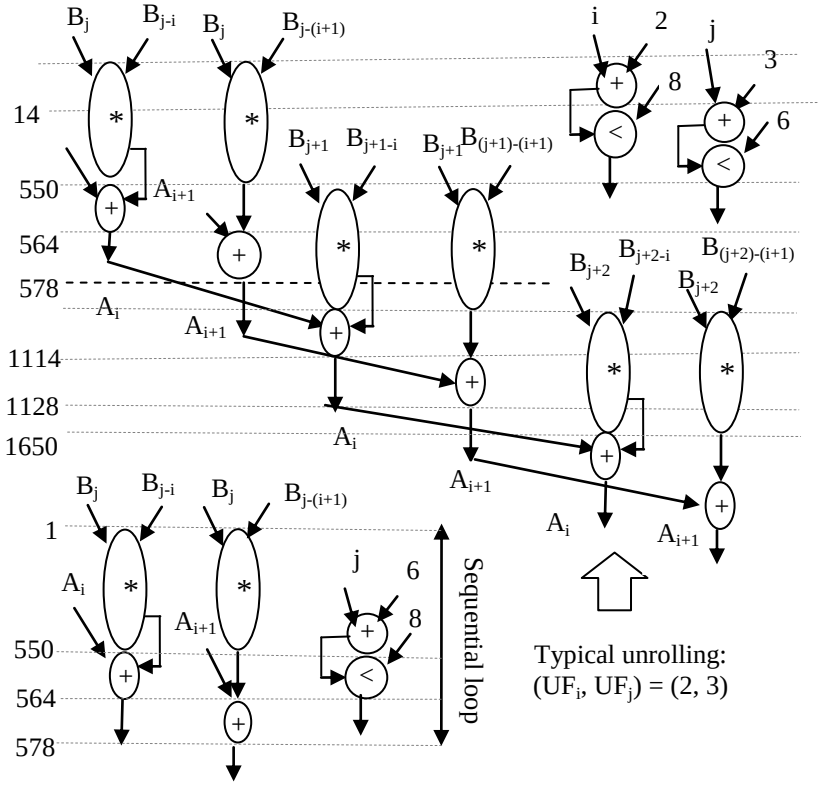


Figure 6.1(c) figure shows typical physically unrolled loop for UF= (2, 3) and schedule with ASAP with 2(*), 1(+), 1(<).

Position of a particle	→	Solution (S_i) of problem
Velocity of a particle (V_{d_i})	→	Exploration drift
Dimension (D)	→	#Resource types + # UFs

Figure 6.2 Proposed Mapping of DSE problem with PSO

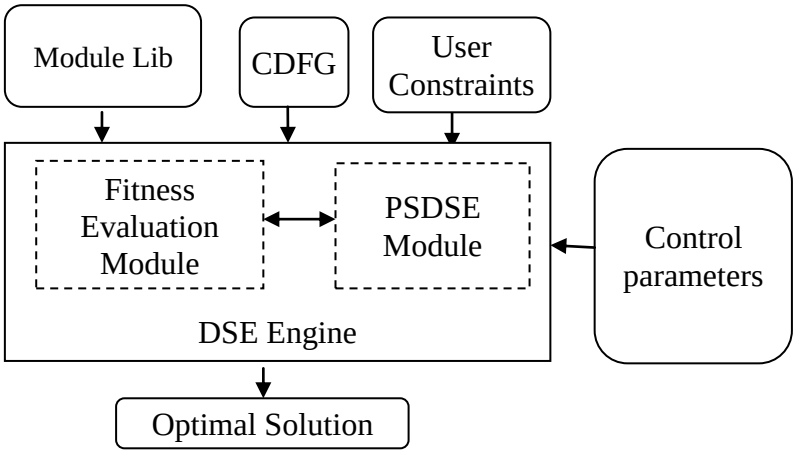


Figure 6.3: Block diagram of proposed DSE Engine

Algorithm: *PSDSE*

Input: CDFG, module library, user constraints

Output: Optimal combination resources (FU's) and unrolling factors (UFs)

Begin

Read library information and CDFG

Constraint validity check()

Pre-processing UF ()

Initialization()

Cost evaluation Process()

Update global best and local best solution

//Iteration process starts here//

While (!stopping condition)

For i=1 to A **Do** *//where 'A' is swarm size*

Determine new solution

Cost evaluation Process()

Update local best solution of the particle

End Loop

Update global best solution

Adaptive rotation mutation ()

Update global best solution

End while loop

Output: global best solution which is optimal combination of resources (FU's) and UFs

End

Figure 6.4 Exploration algorithm

in Figure 6.4. Based on the algorithm, the detail description with demonstration is given in section 6.2.

6.3. Proposed Evaluation Models

For evaluation of a solution (or design point), the following models have been proposed.

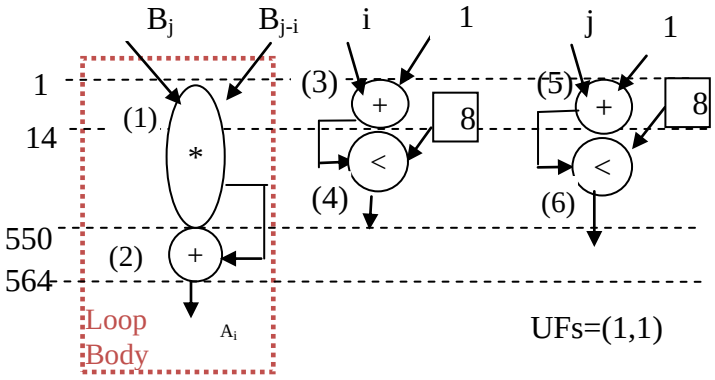
6.3.1. Proposed model for execution time

In order to describe the formulation of proposed execution time (T_E) (function of loop unrolling factor) for a CDFG, an example of loop unrolling is used shown in Fig. 6.5. Fig. 6.5(a) shows the 'c' code of original loop, Fig 6.5(b) shows CDFG for original loop and Fig. 6.5(c) shows an as soon as possible (ASAP) scheduled CDFG unrolled 2*λ times (In this example λ =2) with resource constraint of 2(*), 1(+) and 1(<).

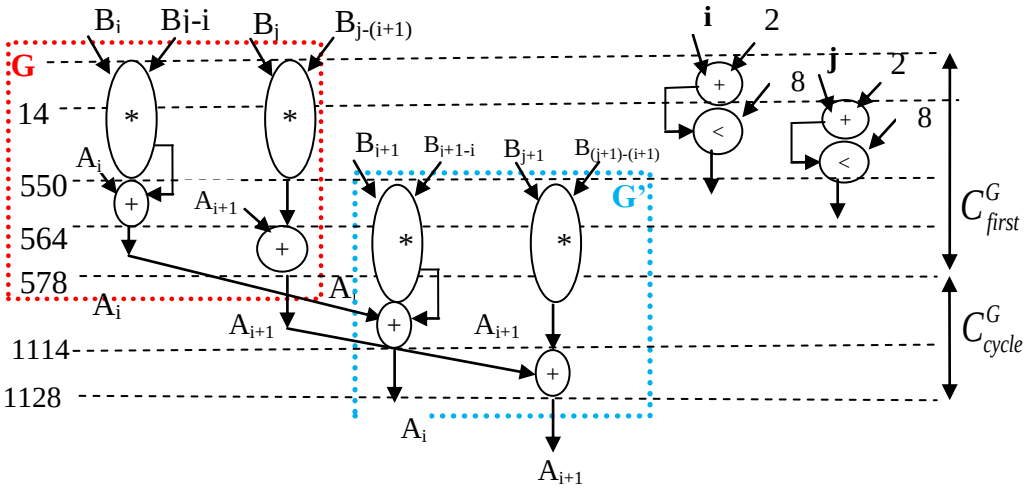
The Proposed Procedure to estimate execution time for nested loop and single loop application is given in Figure 6.6.

```
For(i=0;i<8;i++){
  For(j=0;j<8;j++){
     Ai +=Bj*Bj-i
 }
}
```

(a)'C' code of original loop



(b) CDFG of Original loop



(c) Determine C^G_{first} and C^G_{cycle} for G where G is formed by scheduling twice 'λ' time unrolled loop body with ASAP with 2(*), 1(+), 1(<), here 'λ' = 2

Figure 6.5 CDFG of Autocorrelation for demonstration

Procedure to estimate execution time

Input-Resource combination (RC), UFs, CDFG

Output- Execution time of CDFG based on RC and UFs

Step 1- Determine max number parallel independent operations in original loop body (denoted by 'γ').

Step 2- Determine effective unrolling factor 'UFe' and α_e defined as:

$$UF_e = \prod_{z=1}^{m} UF_z \tag{6.2}$$

$$\alpha_e = \prod_{z=1}^{m} floor\ value\ of\ \left(\frac{I_z}{UF_z}\right) \tag{6.3}$$

λ = number of resources corresponding to 'γ'

Where 'I_z' is z^{th} loop of I; UF_z is the z^{th} unrolling factor of UF.

Step 3- if ((Total number of independent operations due to effective unrolling > λ) && (UF_e mod λ =0))

Go to step 4.

Else

Unrolled completely and determine C_{Total} and use eqn. (6.5)

Step 4- Unroll loop body λ times which to group (denoted by 'G').

Step 5- Schedule G with as soon as possible (ASAP) to determine C^G_{first} and C^G_{cycle} as describe in Fig 6.5(c).

Step 6- Determine total control steps (CS)

$$C_{Total} = \left[C^G_{first} + (\frac{UF_e}{\lambda} - 1) * C^G_{cycle} \right] * \alpha_e \tag{6.4}$$

Step 7- Repeat step 2 and 6 for sequential loop (if any). Finally

$$C_{Total} = C_{Total}(pipelined) + C_{Total}\ (sequential)$$

$$T_E = \Delta * C_{Total} \tag{6.5}$$

where, 'T_E' is total execution time; 'Δ' is the delay of one control step in nanoseconds; 'C^G_{first}' is required number of CS to execute G once; 'C^G_{cycle}' is number of CS between two consecutive iteration of G, 'C_{Total}' is total number of CS required to execute the loop completely.

Figure 6.6 Procedure to estimate execution time for loop based application

On the other hand, for single loop application, T_E formulated as:

$$T_E = \Delta * ((C_{first} + (UF - 1) * C_{II}) * \alpha + (I \bmod UF) * C_{first}) \quad (6.6)$$

Where 'C_{first}' is number of CSs required to execute first iteration, 'C_{II}' is the number of CSs required between initiations of consecutive iterations, 'I' is the maximum number of iteration (loop count), and 'α' is *floor value of (I/UF).(Note- detail description of the model given in chapter 4 section 4.3.)*

The model given in eqn. (6.6) is not used when #of independent operations required to be performed in parallel due to unrolling exceeds the available resource units specified in a solution. In such cases loop unrolled tediously. To overcome above limitation proposed model given in (6.5) is used with UF_e = UF.

Note- The utility of the proposed model is wide spectrum. The execution time model proposed in this paper is useful for any DSE process which handles single or perfectly nested loop.

6.3.2. Power Model

The total power consumed by a resource combination is denoted by 'P_T'. 'P_T' is composed of dynamic power ('P_D') and static power (P_S) given by (6.7) below:

$$P_T = P_D + P_S \quad (6.7)$$

P_T is formulated as:

$$P_T = \frac{(E_{FU} + E_{MUX/DMUX})}{T_E} + \left(\sum_{i=1}^{v} (N_{R_i} \cdot K_{R_i}) + N_{MUX/DMUX} \cdot K_{MUX/DMUX} \right) * P_c \quad (6.8)$$

where, 'E_{FU}' represents the total energy consumption of the resources , '$E_{MUX/DMUX}$' represents the total energy consumed by multiplexer and demultiplexer. 'N_{Ri}' represents the number of instance of resource R_i; 'K_{Ri}' represents the area occupied by resource R_i, 'v' is the number of resource types, '$N_{MUX/DMUX}$'is number of the multiplexer or demultiplexer, '$K_{MUX/DMUX}$' is area occupied by the multiplexer or demultiplexer and 'P_c' denotes the power dissipated per area unit (e.g. transistors).

6.3.3. Cost model

The fitness function (considering execution time and power consumption of a solution) is defined as:

$$f(\mathrm{S}) = \varphi_1 \frac{P_T - P_{cons}}{P_{\max}} + \varphi_2 \frac{T_E - T_{cons}}{T_{\max}} \tag{6.9}$$

Where, $f(\mathrm{S})$ =fitness (cost) of solution S; φ_1 , φ_2 = User defined weights for power and execution time.

6.3.4. **Demonstration of proposed execution time estimation procedure**

To demonstrate the procedure for estimation of execution time (presented in Figure 6.6) for nested loops, the CDFG presented in Figure 6.1 and Figure 6.5 is used. The Figure 6.5(c) shows the unrolled loop with UF(2, 2) which has been scheduled by as soon as possible (ASAP) scheduling algorithm with resource constraint of 2(*), 1(+) and 1(<).The demonstration of the procedure based on Figure 6.5 is as follows:

Resource combination = 2(*), 1(+), 1(<)

Unrolling Factor = UF_1=4, UF_2= 4

Step 1- Determine 'γ'

From the Figure 6.5(b) only one multiplication operation is independent in the loop body so

γ = 1

Step 2- Using eqn.6.2 and eqn.6.3

$$UF_e = \prod_{z=1}^{m} UF_z$$

UFe = UF1 * UF_2 = 4 * 4 =16

$$\alpha_e = \prod_{z=1}^{m} floor\ value\ of \left(\frac{I_z}{UF_z} \right)$$

$$\alpha_e = floor\ value\ of \left(\frac{I_1}{UF1} \right) * floor\ value\ of \left(\frac{I_1}{UF1} \right)$$

$$= floor\ value\ of \left(\frac{8}{4} \right) * floor\ value\ of \left(\frac{8}{4} \right) = 4$$

λ = 2 i.e. multiplier (Number of resources of γ resource type)

Step 3- Number of operations due to effective unrolling = UFe * γ = 16 * 1

Check condition explained in step 3 of Figure 6.6

Accordingly, ((16 > 2) and (16 mod 2 = 0)) = True

Thus, move to step 4

Step 4- Schedule the 2 * λ times unrolled loop body, using ASAP to form G and G'.

Step 5- Determine C^G_{first} and C^G_{cycle} as shown in Figure 6.5 (c).

(Note- the values of C^G_{first} and C^G_{cycle} *are derived from the ASAP scheduled CDFG with resource combination: 2 (*), 1 (+),1(<), UFs= (2,2)) shown in Figure 6.5(c).*

Step 6 and Step 7- Determine execution time metric using eqn. (6.4) and (6.5):

$$T_E = ((578 + (16/2 - 1) * 550) * 4) * 20 = 354240ns \quad (6.10)$$

In case of Figure 6.1, the Figure 6.1(c) shows the physically unrolled loop with UF(2, 3) which is scheduled by as soon as possible (ASAP) algorithm with resource constraint of 2(*), 1(+) and 1(<). Moreover, the UF for j^{th} loop (i.e. 3) is not a factor of (divisible) j^{th} loop index (i.e. 8) which leads two remainder loops for every 'i' index resulting in total sixteen remainder loops for i = 8 as shown in Figure 6.1(c) i.e.

Total loop iterations (64) = unrolled loop (48) + sequential loop (16);

The demonstration of execution time evaluation for CDFG (given in Figure 6.1) is as follows:

Step 1- Determine 'γ'

From the Figure 6.1(b) only one multiplication operation is independent in the loop body so

γ= 1

Step 2- Using eqn.6.2 and eqn.6.3

$$UF_e = \prod_{z=1}^{m} UF_z$$

UFe = UF1 * UF_2 = 2 * 3 = 6

$$\alpha_e = \prod_{z=1}^{m} floor\ value\ of\ \left(\frac{I_z}{UF_z}\right)$$

$$\alpha_e = floor\ value\ of\ \left(\frac{I_1}{UF1}\right) * floor\ value\ of\ \left(\frac{I_1}{UF1}\right)$$

$$= floor\ value\ of\ \left(\frac{8}{2}\right) * floor\ value\ of\ \left(\frac{8}{3}\right) = 8$$

$\lambda = 2$ i.e. multiplier (Number of resources of γ resource type)

Step 3- Number of operation due to effective unrolling = UFe * $\gamma = 6 * 1$

Check condition explain in step 3 of Figure 6.6

Accordingly, ((6 > 2) and (6 mod 2 = 0)) = True

Thus, move to step 4

Step 4- Schedule the 2* λ times unroll loop body, using ASAP to form G and G'

Step 5- Determine C_{first}^{G} and C_{cycle}^{G} for 'λ' =2 (as shown in Figure 6.5 (c))

Step 6 and Step 7- Determine execution time metric using eqn. (6.4) and (6.5):

$$C_{Total}(pipelined) = ((578 + (6/2-1)*550)*8) = 13424\,cs$$

Similarly for sequential loop (as shown in Figure 6.1)

C_{Total} (sequential) = 6936 cs

C_{Total} = 13424 + 6936 = 203605

Using eqn. (6.5)

T_E = 20360 * 20 = 407200 ns

6.4. Demonstration of proposed methodology

6.4.1. Input module library, user constraints and CDFG

First of all, the PSDSE framework takes module library [52, 54], CDFG, and user specified design constraints for power and execution time as inputs. For the sake of demonstration, power constraint (P_{cons}=1.25mW) and execution time constraint (T_{cons}=250us) are taken as input. Further, the module library consist area (#transistor), energy consumption, delay and maximum availability of functional units. For demonstration maximum available multiplier = 8, adder = 8, comparator = 2 and total user specified loop iteration I_1 = 8 and I_2 = 8; power dissipated per transistor (P_c) is assumed to be 29.33nW; furthermore, number/type of mux/demux is directly extracted from the scheduling and binding solution.

6.4.2. Constraint validity check

After taking the user constraints for power and execution time the algorithm checks validity of user constraints, In order to check the constraints, maximum and minimum value of power and execution time need to be calculated. Maximum power (P_{max}) and minimum

execution time (T_{min}) is calculated with maximum resource and max(UF_i)=I_i (maximum # of loop iterations) while minimum power (P_{min}) and maximum execution time (T_{max}) are calculated with minimum resource (single instance of resource) and min (UF_i)=1.

After determining maximum and minimum value of power and execution time the algorithm checks for valid user constraints. If user constraints are not valid then an error is shown and requests for valid values.

IF ($(P_{\min} \le P_{cons}\ AND\ P_{\max} \ge P_{cons})\ OR\ (T_{\min} \le T_{cons}\ AND\ T_{\max} \ge T_{cons})$)

Proceed for initialization process.

ELSE

Requests to user for input correct constraints.

6.4.3. Pre-processing of unrolling factor

The pre-processing step filters unfit UFs to create a list of feasible solutions. Moreover, to ensure the inclusion of good candidates, some special UFs have been added which may have been initially screened out in pre-processing phase. The detail description of the pre-process is given in chapter 4 section 4.4.

6.4.4. Initialization process of the particle

After preprocessing step initialization of the particle take place. During initialization process solutions (particles position), velocity and acceleration coefficient are initialized as follows:

a) *Solution*: For a CDFG, a particle position S_i representing a candidate solution (as described in section 6.1) is given in eqn. (6.1):

$$\vec{S} = \{R_1, R_2, \cdots, R_n, UF_1, UF_2, \cdots, UF_m\}$$

In PSDSE, the initialization of particles is such that it comprehensively covers the entire design space as follows:

$$\vec{S}_1 = \{R_1^{\min}, R_2^{\min}, \cdots, R_n^{\min}, UF_1^{\min}, UF_2^{\min}, \cdots, UF_m^{\min}\} \quad (6.11)$$

$$\vec{S}_2 = \{R_1^{\max}, R_2^{\max}, \cdots, R_n^{\max}, UF_1^{\max}, UF_2^{\max}, \cdots, UF_m^{\max}\} \quad (6.12)$$

$$\vec{S}_3 = \{R_1^{avg}, R_2^{avg}, \cdots, R_n^{avg}, UF_1^{avg}, UF_2^{avg}, \cdots, UF_m^{avg}\} \quad (6.13)$$

$$\text{avg} = (\text{max} + \text{min})/2$$

Where ' R_n^{min} ' is minimum the number of instances of n^{th} resource type, ' R_n^{max} ' is maximum the number of instances of n^{th} resource type, UF_m^{min} is minimum unrolling factor (equal to 1) of m^{th} loop, UF_m^{max} is maximum unrolling factor (equal to I) of m^{th} loop.

Rest of the solutions (S4…Sn) is initialized with random values between minimum and maximum values of resources and UFs as describe in chapter 3 section 3.2.3. Hence, using eqn. (6.11) – (6.13),

S_1 = (1, 1, 1, 1, 1),

S_2 =(8, 8, 2, 8, 8);

S_3= (4, 4, 1, 4, 4);

S_4= (2, 1, 1, 4, 4).

b) ***Initialization of velocity and acceleration coefficient***

Velocities of all particles are initialized to zero in the proposed approach[49] and acceleration coefficient can be initialized to any value between 1 and 4 [50].

6.4.5. Cost evaluation and local, global best update

Once the all particles are initialized, fitness of the particles is determined. First power is calculated according to (6.8). Example, for particle S_4= (2, 1, 1, 4, 4):

$$P_T = \frac{4*(16*9.8+17*0.739+1*0.739+3*6*0.1+3*15*0.1)\,\text{pj}}{((578+(16/2-1)*550)*4)*20ns} +$$

$$(2464*1+2030*2+2030*1+3*7*126+3*15*126)*2.933*10^{-6}$$

$$P_T = 0.495 \quad mW \tag{6.14}$$

Next, determination the execution time for particle S_4= (2, 1, 1, 4, 4) where 2(*),1(+),1(<) are resources and 4,4 are UF_S . The execution time of the solution S_4 is:

T_E = 354240ns by eqn. 6.10 as describe in section 6.3.4

Finally the fitness of the particles is calculated by (6.9). Therefore, the fitness of S_4 is:

$$f(S_4) = -0.0654 \tag{6.15}$$

Similarly, fitness of all the particles is determined. *(Note- φ_1 = φ_1 = 0.5 is chosen for providing equal weightage to power and execution time).*

After cost evaluation next step is local best solution (position) update. If cost of new solution is lesser than the local best cost of the particle then the new solution become local best solution and new solution cost become local best cost. Further, the global best solution is updated as follows:

$$S_{gb} = S_i[Min(f(S_{lb1}), f(S_{lb2}), \ldots . f(S_{lbA}))] \tag{6.16}$$

6.4.6. Determination of new solution

To determine new solution firstly, determine new velocity of the particle. The velocity of the particle is determined as (6.17) is given below:

$$V_{d_i}^{+} = \omega V_{d_i} + b_1 r_1 \left[S_{d_{lbi}} - S_{d_i} \right] + b_2 r_2 \left[S_{d_{gb}} - S_{d_i} \right] \tag{6.17}$$

where, $V_{d_i}^{+}$ is the new velocity of particle ith solution in dth dimension, ' $S_{d_{lbi}}$ ' is the value of ith solution in dth dimension, ωV_{d_i} is the inertia component, ' $S_{d_{gb}}$ 'is the value of S_{gb} in dth dimension, b_1, b_2 are acceleration coefficients and r_1 and r_2 are random numbers between [0-1].

Secondly, new solution (position) of the particle is updated with new velocity defined in (6.18) as follows:

$$S_{d_i}^{+} = S_{d_i} + V_{d_i}^{+} \tag{6.18}$$

where, $S_{d_i}^{+}$ = new resource value or UF value of ith particle in dth dimension and S_{d_i} = previous resource value or UF value of ith particle in dth dimension;

Note- $S_{lbi} = \{R_{1_{lbi}}, \ldots R_{n_{lbi}}, UF_{1_{lbi}}, \cdots, UF_{m_{lbi}}\}$ *and* $S_{gbi} = \{R_{1_{gb}}, \ldots R_{n_{gb}}, UF_{1_{gb}}, \cdots, UF_{m_{gb}}\}$

(Note: To control the excessive exploration drift velocity clamping and to handle boundary outreach violation adaptive end terminal perturbation are described in chapter 3).

6.4.7. Mutation operation and Stopping criterion

With the hope of better solution and improving local optima problem the adaptive rotation mutation is performed on all the local best position of each particle with probability equal one. The description of mutation operation is given in chapter 3.

The proposed algorithm terminates when no improvement observe in S_{gb} over '£' number of iteration. (£=10) or when the iteration count exceeds maximum number of iterations equal to 100.

6.5. Process of transforming non-perfect nested loop into perfect nested loop

The proposed approach only handlesthe perfectly nested loop based applications. A perfect nested loop is a loop where there are no statements between different loop-levels. For example, the CDFG (autocorrelation) of nested loop in Figure 6.7(a) is not a perfect loop (called non-perfect nested loop) since there is code between the innermost level and the outer level (e.g. sum=0; and r[i]= sum>>15).To handle such application, transformation of non-perfect nested loop to perfect nested loop is performed. In order to make a non-perfect nested loop to perfectly nested loop, a transformation method is used, which has been adopted from [70]. This section presents an overview of the transformations. The transformation is divided into two parts viz. renaming and loop distribution. The procedure is as follows:

6.5.1. Renaming.

In this step all reused variables are renamed, to make loop distribution feasible. For this,an array of same dimension as number of iterations of the outer loop isintroduced (if this number is not available at compile time, then take maximum value in the worst case) and then use the array elements to replace the variables in the code. For example, in Figure 6.7(a) '*sum'* is a variable which is initialized to the loop body of the innermost loop. Its value will be stored after innermost loop is finished. For each iteration of the outer loop, *sum*is reused. So an array for *sum* is created with 'nr' dimension (loop count of outer loop) (i.e. *sum[nr]*). The nested loop given in Figure 6.7(a) changes to Figure 6.7 (b) after renaming is applied.

6.5.2. Loop distribution

After renaming has been done, the next step is loop distribution. In loop distribution, the loop body of the outer loop is divided into three parts: initialization part, the innermost loop, and the result stored part. This distribution converts the original loop into three different loops. The first loop is initialization loop, which is a single level loop. The second part is innermost loop, which is the main part of the code and form a perfectly nested loop. The third part is result store part, which is also single level loop. For example, the renamed loop shown in Figure 6.7(b) converted to the perfectly nested loop after distribution shown in Figure 6.7 (c).

Note – Results of the proposed method are given in chapter 7 section 7.4

```
inti, k;
int sum;
for (i = 0; i < nr; i++)
    {
        sum = 0;
        for (k = nr; k <nx + nr; k++)
       {
                sum += x[k] * x[k-i];
       }
        r[i] = sum  >> 15;
 }
```

Initialization Statement

Main loop body

Result Store Statement

Figure 6.7 (a) Original nested loop

```
inti, k;
int Sum[nr];
for (i = 0; i < nr; i++)
    {
       Sum[i] = 0;
        for (k = nr; k <nx + nr; k++)
       {
                Sum[i] += x[k] * x[k-i];
       }
       r[i] = Sum[i]  >> 15;
  }
```

Renaming of 'Sum; to 'Sum[]'

Figure 6.7 (b) Renaming

```
inti, k;
int Sum[nr];
for (i = 0; i < nr; i++)
    {
       Sum [i]= 0;
     }
for (i = 0; i < nr; i++) {
     for (k = nr; k <nx + nr; k++)
       {
          Sum[i] += x[k] * x[k-i];
       }
}
for (i = 0; i < nr; i++)  {
         r[i] = Sum[i]  >> 15;
 }
```

Initialization loop

Main perfectly Nested Loop

Result Store loop

Figure 6.7 (c) Final after renaming and distribution

Figure 6.7 An example of loop transformation from non-perfect to perfect nest loop

Chapter 7
Result and Analysis

This chapter describes the complete experimental results of the proposed methodologies for design space exploration described in previous chapters. This chapter divided into four sections where each section present results of the respective methodology. Four sections are as follows:

a) MO-PSE: Adaptive Multi Objective Particle Swarm Optimization Based Design Space Exploration in Architectural Synthesis for Application Specific Processor Design
b) Automated Exploration of Datapath and Unrolling Factor during Power-Performance Trade-off in Architectural Synthesis Using Multi-Dimensional PSO Algorithm
c) Simultaneous Exploration of Optimal Datapath and Loop Based High level Transformation during Area-Delay Trade-off in Architectural Synthesis Using Swarm Intelligence
d) Simultaneous Exploration of Optimal Datapath and Loop Based High level Transformation during Area-Delay Trade-off in Architectural Synthesis Using Swarm Intelligence

The results included implementation details, library details, sensitivity analysis of control parameter and improvements achieved compared to the state of art approaches.

7.1 Experimental results: the proposed approach 'MO-PSE: Adaptive Multi Objective Particle Swarm Optimization Based Design Space Exploration in Architectural Synthesis for Application Specific Processor Design'

This section describes the experimental results of the proposed approach explained in Chapter 3 and the improvements obtained compared to recent approach [27, 29]. The proposed MO-

PSE approach has been implemented in java language and run on Intel core i5-2450M processor 2.5 GHz with 3MB L3 cache memory and 4GB DDR3 primary memory. Various high level synthesis DSP benchmarks were chosen for testing such as autoregressive filter (ARF) [53,57], band-pass filter (BPF) [5], elliptic wave filter (EWF) [53], finite impulse response (FIR) filter [5,57], IIR Butterworth [53], MESA-Horner Bezier[53], MPEG motion vectors (MMV) [5, 53], JPEG Downsample [5]. The library is given in chapter 3 Table 3.1, 3.2. The proposed MO-PSE approach is experimented with two aspects given below:

- Analysis of variation in multiple PSO parameters and their impact on the MO-PSE performance (or efficiency).
- Comparison of MO-PSE with previous DSE approaches [29], [27] in terms of exploration time (or speed) and Quality of Results (QoR) achieved.

7.1.1. Impact of Proposed MO-PSE with variation in PSO parameters: Investigation and Analysis

This section will investigate the impact of variation in multiple internal PSO parameters on proposed design space exploration approach for selected benchmarks. However, it should be noted that this subsection will not discuss the quality of results achieved through the proposed DSE approach. The quality of solution found and its comparison with other DSE approaches will be discussed in the next subsection. Further in this subsection, post-experimental analysis will also assist in pre-tuning PSO parameters to an appropriate value (for MO-PSE) while performing DSE comparison.

7.1.1.1. Inertia weight (ω)

In the proposed approach (MO-PSE), inertia weight controls to the exploration drift process of the particle by weighing the involvement of the previous exploration drift. During the experiment, the following three variations of '**ω'** have been analysed and its impact on the performance of MO-PSE has been reported:

a) Linearly decreasing 'ω' in every iteration between [0.9- 0.1] throughout the exploration process.

b) A constant value of ω = 1 throughout the exploration process.

c) A constant value of ω = 0.5 throughout the exploration process.

As evident from Table 7.1 and Table 7.2, for all benchmarks the MO-PSE's convergence time and exploration time is generally better with linearly decreasing value of 'ω'. For example in Table 7.1, in case of MPGE Motion Vectors, the convergence time is 100.5ms for linearly decreasing 'ω', while it is 111.33ms and 102.6ms at 'ω' = 0.5 and 'ω' = 1 respectively. Therefore, it is clear from the empirical evidence that MO-PSE during exploration of variances converges to optimal solution faster when the 'ω' is decreased linearly in every iteration until the magnitude becomes 0.1. Further, in case of MESA, the exploration time for finding the optimal solution is 78.5ms when 'ω' is linearly decreased between [0.9 – 0.1] compared to 86ms and 109.33ms when 'ω' = 0.5 and 'ω' = 1 respectively as shown in Table 7.2.

Table 7.1. Comparison of convergence time (ms)
with respect to parameter " ω "

Benchmark	**Linearly decreasing**	**ω (0.5)**	**ω (1)**
IIR Butterworth	22.5	27.33	27.5
BPF	121.66	118.33	116
EWF	116.16	121	168
ARF	127	134	137
JPEG SAMPLE	80.5	81	108
MESA	39	45.16	50.5
FIR	73.6	82.5	62.66
MPEG MMV	100.5	111.33	102.66

7.1.1.2. Acceleration Coefficients (b)

a) Comparison amongst constant acceleration coefficient

The Acceleration Coefficients b_1 and b_2 control the influence of cognitive and social component in exploration drift. Where b_1 expresses how much confidence a particle has in itself, while b_2 expresses how much confidence a particle has on other particles [49]. During experimentation 'b_1' and 'b_2' were kept equal to 'b'. Four different values of b were taken

Table 7.2 Comparison of exploration time (ms)
with respect to parameter " ω "

Benchmark	**Linearly decreasing**	**ω (0.5)**	**ω (1)**
IIR Butterworth	56.5	69.5	71.33
BPF	261	244	238
EWF	323	320	368
ARF	252.16	263	269.33
JPEG SAMPLE	205	206	252.16
MESA	78.5	86	109.33
FIR	136	147	122.66
MPEG MMV	212	227	213.66

for experimental analysis viz. b= 1,2,3,4. (Note: The reason for setting 'b' in the range [1-4] for PSO has been shown in [50]). As indicated in Table 7.3 and Table 7.4, in terms of convergence and exploration time both, pre-opting an exact choice of 'b' before performing design space exploration for ASP's is a non-trivial issue. This is because there is no general trend that can be observed for the selected benchmarks for making a pre-defined choice as shown in Figure 7.1 and Figure 7.2. As seen in Table 7.3 and 7.4 in order to achieve minimal convergence and exploration time, for IIR and MESA value of b = 1 is suitable, for FIR value of b = 2 is suitable, for ARF, BPF, MPEG value of b=3 is suitable and for EWF and JPEG Downsample value of b=4 is suitable. However, it can assumed that the best quality solutions for attaining faster convergence and exploration speed are mostly clustered at b = 2 or b =3 in most tested cases.

b) comparison between time varying and constant acceleration coefficient

By changing the acceleration coefficient b_1 and b_2 with time, the effect of cognitive component on new velocity decreases, simultaneously the effect of social component on new velocity increases is tested in this section. With a large b_1 and small b_2 at the beginning, agents are allowed to move around the design space, indicating higher exploration capability. On the other hand, a small value of b_1 and a large value of b_2 signify the convergence to the global best. Therefore, during exploration process the value of b_1 is

Table 7.3 Comparison of exploration time (ms) with respect to parameter "b"

Benchmark	b(1)	b(2)	b(3)	b(4)
IIR Butterworth	55.5	56.5	60.166	56.833
BPF	200	261	195.166	205.5
EWF	328.66	323	316.66	313.66
ARF	274	252	251.83	287
JPEG SAMPLE	207.33	205.16	228.33	205.15
MESA	76.833	78.5	79.66	82.83
FIR	149.33	136	147.16	153
MPEG MMV	275.66	212.66	199	288

Table 7.4 Comparison of convergence time (ms) with respect to parameter "b"

Benchmark	b(1)	b(2)	b(3)	b(4)
IIR Butterworth	26.166	23.5	26	28.5
BPF	78.33	121.66	78	84.166
EWF	115.5	116.166	117	121.66
ARF	129.5	127	124.33	157.33
JPEG SAMPLE	80.166	80.5	86.5	73.166
MESA	38.166	39	41	42
FIR	80.166	73.66	86.5	90.33
MPEG MMV	162.5	100.5	89	176

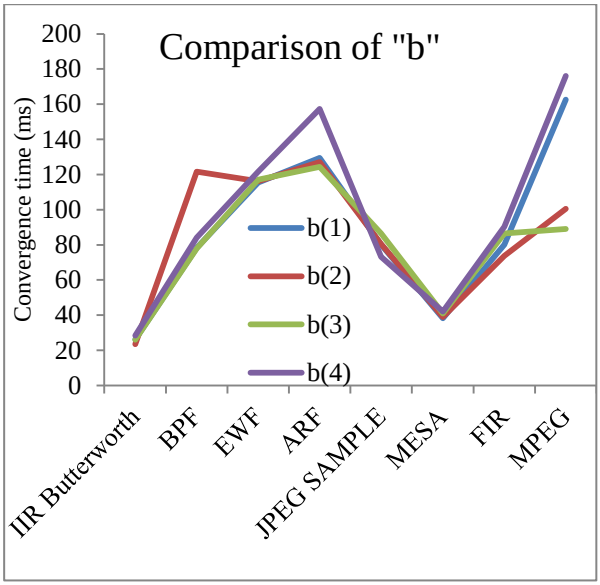


Figure 7.1 Comparison of convergence runtime w.r.t 'b'

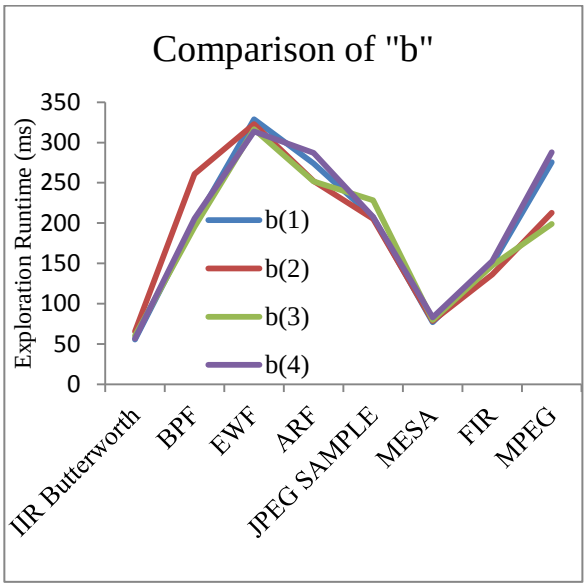


Figure 7.2 Comparison of exploration runtime w.r.t 'b'

decreased from 2.5 to 0.5 and value of b_2 is increased from 0.5 to 2.5 with time. Formally, value of b_1 and b_2 are evaluated as:

$$b_1 = (b_{1f} - b_{1i})\frac{t}{T} + b_{1i}$$

$$b_2 = (b_{2f} - b_{2i})\frac{t}{T} + b_{2i}$$

Where b_1 is current acceleration coefficient, b_2 is current social acceleration coefficient, b_{1i} is initial cognitive acceleration coefficient = 2.5, b_{1f} final cognitive acceleration coefficient = 0.5, b_{2i} is initial social acceleration coefficient =0.5, b_{2f} is final social acceleration coefficient =2.5,'t' is current iteration, and 'T' is total no of iterations.

In case of constant value, the value of cognitive coefficient and social coefficient are set as a constant value equal to 2 (b_1=2, b_2=2) during the exploration process (as concluded from previous section). Reason behind this setting is to investigate the effect of a constant and same value of acceleration coefficients which aims to maintain equal balance between cognitive effect and social effect during the entire design space exploration process.

As evident from Table 7.5 and Table 7.6, for all benchmarks the PSO based DSE's convergence time and exploration time is better with constant value of acceleration coefficient (equal to 2) as compared to time varying acceleration coefficient. Reason behind this is the constant and equal value of acceleration coefficients during the entire exploration

process gives equal weightage to the social component and cognitive component (from the beginning of exploration) thereby reducing the convergence and exploration time. Further, as it has been observed during experiment that particles tend to converge to optimal early (owing to lesser diversity in candidate population), therefore the effect of time varying acceleration coefficient (where the weightage of social component is slowly increased from 0.5 onwards to a high value) is not dominant in context of DSE. *(Note: - if a range between 2.5 to 2 is kept for time varying acceleration coefficient, then this setting of acceleration coefficients is likely to produce better exploration speed in context of DSE in HLS).* For example, in Table 7.5, in case of ARF, the convergence time is 579.8ms with constant b_1 and b_2 while the convergence time is 654.57ms with time varying acceleration coefficient. Further, in Table 7.6, in case of DCT the exploration time is 690.57ms and 773.14ms for constant b_1, b_2 and time varying b_1, b_2 respectively. It is clearly evident that for multi objective DSE problem in HLS the constant value of acceleration coefficient (PSO parameter) provides better result as compared to time varying acceleration coefficient for all the tested benchmark. It is important to note that in both the cases analyzed viz. a) time varying acceleration coefficient; b) fixed acceleration coefficient, the PSO-DSE explores the same solution set.

Table 7.5 Comparison of convergence time (ms) w.r.t. constant acceleration coefficient and time varying acceleration coefficient

Benchmarks	**Convergence time (ms) b_1=2, b_2=2 (Constant)**	**Convergence time (ms) b_1(Decreasing), b_2(Increasing)**
IIR Butterworth	20.0	26.6
ARF	579.8	654.57
DCT	522	566.8
MESA	59.28	75.71
EWF	216.14	229.57
MPEG	447.42	525.0
IDCT	768.57	882.57
BPF	170.57	172.0
JPEG	42.57	55.57
WDF	264.28	288.14

Table 7.6 Comparison of Exploration Time (ms) w.r.t. constant acceleration coefficient and time varying acceleration coefficient

Benchmarks	**Exploration time(ms) b_1=2, b_2=2 (Constant)**	**Exploration time(ms) b_1(Decreasing), b_2(Increasing)**
IIR Butterworth	29.6	33.6
ARF	685.85	757.85
DCT	690.57	773.14
MESA	67.57	85.28
EWF	226.14	242.28
MPEG	505.85	572.71
IDCT	901.85	977.57
BPF	216.14	218.71
JPEG	81.42	103
WDF	273.85	301.28

7.1.1.3. Swarm size (S)

A large 'S' value covers larger number of design points in the design space per iteration, but a larger number of particles increase the computational complexity per iteration. However, it should also be noted that using a few number of particles will require a large number of iterations during exploration of variances and hence will deteriorate the success rate [49]. During the experimental analysis of selected benchmarks, it was found that the best size of swarm for proposed MO-PSE is three (i.e. analogues to selecting three diverse initial parents (or design variances) in evolutionary algorithms) for most of the benchmarks. The results are shown in Table 7.7 and Table 7.8. As evident from the results, the best balance between achieving fast exploration and searching optimal solution can be obtained by setting S = 3 for the tested benchmarks.

7.1.1.4. Stopping criterion (Z)

During experiment two stopping criterion have been tested, described in section 3.2.10. The first stopping criterion, maximum number of iteration controls the endless process while second controls the convergence time and exploration time. In worst case MO-PSE reaches

Table 7.7. Comparison of convergence time (milliseconds) with respect to swarm size (S)

Benchmark	S(3)	S(5)	S(7)
IIR Butterworth	22.5	30.3	34.16
BPF	121.66	104.5	135.83
EWF	116.166	171.66	233.5
ARF	127	177.83	203.33
JPEG SAMPLE	80.5	106.833	132.833
MESA	39	56.16	59
FIR	73.66	84	124
MPEG MMV	100.5	138	220.33

Table 7.8 Comparison of exploration time (milliseconds) with respect to swarm size (S)

Benchmark	S(3)	S(5)	S(7)
IIR Butterworth	56.5	67.33	81.66
BPF	261	298.5	404.66
EWF	323	502.16	692.66
ARF	252	387.16	504.5
JPEG SAMPLE	205.16	314	428
MESA	78.5	112.83	130.16
FIR	136	183	267
MPEG MMV	212.66	319.166	480.33

maximum number of iteration. Both option of second condition have been tested and result shown in Table 7.9. As apparent from result, the MO-PSE produces the better result in terms of faster convergence with first stopping criterion (S^1). For example in Table 7.9, in case of JPEG Downsample, the convergence time for finding optimal solution is 80.5ms with S^1, while 219ms with S^2. Moreover, in case of MPEG Motion Vector, the convergence time for finding optimal solution is 100.5ms with S^1, while 390ms with S^2.

Table 7.9 Comparison of convergence time (ms)
with respect to stopping criterion (S^1 S^2)

Benchmarks	S^1	S^2
IIR Butterworth	23.5	42.66
BPF	121.66	171.166
EWF	116.166	233.833
ARF	127	984
JPEG SAMPLE	80.5	219
MESA	39	78.5
FIR	73.66	486.5
MPEG MMV	100.5	390

7.1.2. Comparison of MO-PSE with previous approaches

This subsection will describe comparison of proposed MO-PSE with various previous approaches [29] and [27]. The MO-PSE is compared with [29],[27] approaches in terms of following parameters: a) Implementation runtime b) Resource configuration c) Execution time d) Power and e) Quality of Results (QoR). The QoR is evaluated as:

$$QoR = w_1 \frac{P_T}{P_{\max}} + w_2 \frac{T_E}{T_{\max}} \tag{7.1}$$

Where P_T, T_E, P_{max}, T_{max} defined in chapter 3. W_1 and W_2 are equal to 0.5 for giving equal weightage to power and execution time.

As describe in eqn. (3.11), φ1 and φ2 are the user specified weightage for power consumption and execution time. During our experiments, for the proposed MO-PSE, the following settings were maintained based on the inferences drawn from the obtained results in section 7.1.1: φ1 and φ2 equal to 0.5, the value of ω will be linearly decreased between [0.9-0.1], value of b = 2, swarm size = 3 (indicating three initial solutions), stopping criterion = S^1 and M = 100.

7.1.2.1. Comparison with [29]

Table 7.10 and Table 7.11 show the detailed comparison with [29]. As obvious from Table 7.10 and Table 7.11, without compromising the QoR, the exploration speed of proposed MO-PSE is multiple times higher than [29]. Moreover, the approach presented in [29], fails (underlined values in Table 7.10) to meet the specified constraint for power with respect to benchmarks such as IIR Butterworth, MESA, FIR, EWF and MPEG. Besides, the MO-PSE achieved better QoR (normalized cost function of power and execution time) as compared to [29] for most of the benchmarks. The average improvement in QoR is more than 10% and average reduction in exploration runtime is 90% as shown in Table 7.11. For quick observation, graphical representation of the comparison is given in Fig 7.3 and Fig 7.4 respectively.

7.1.2.2. Comparison with [27]

The proposed approach when compared to [27] achieved increased acceleration in exploration process as shown in Table 7.12. Besides above, the proposed MO-PSE approach considers cycle time resulting from initiation interval and latency to create a genuinely pipelined functional data-path during performance calculation. Therefore, the execution time (in µs) for the proposed MO-PSE is determined from eqn. (3.10). On the other hand, the approach presented in [27], does not able to optimize the execution time considerably due to its inability to create a genuinely pipelined functional data-path. Therefore the total execution time (function of latency, cycle time, and pipelined data as shown in eqn. 3.10.) does not get sufficiently optimized for [27]. Thus, for determining of execution time in [27], "*N*" set of processing data is multiplied directly with the latency as per: $T_E^{[27]} = N * L$. Further, the proposed MO-PSE directly considers the total power consumption during exploration as shown in eqn. (3.4). In contrast to MO-PSE, [27] only considers area minimization while

Table 7.10 Experimental Result of comparison with [29] for the tested benchmarks

Benchmark	**Parameters of comparison**					
	Resource Configuration		**Execution Time (us)**		**Power (W)**	
	MO-PSE	**[29]**	**MO-PSE**	**[29]**	**MO-PSE**	**[29]**
IIR Butterworth	2(*), 1(+)	4(*), 1(+)	220.01us	120us	7.01W	11.9W
			Constraint =300us		*Constraint =8W*	
MESA Horner Bezier	3(*), 1(+)	3(*), 1(+)	100.1us	100us	9.61W	9.65W
			Constraint = 400us		Constraint = 8W	
ARF	3(*), 1(+)	4(*), 1(+)	520.2us	341us	9.49W	11.9W
			Constraint = 600us		*Constraint = 10W*	
EWF	2(*), 1(+)	2(*), 4(+)	320us	200us	7.02W	13.1W
			Constraint =500us		*Constraint =8W*	
FIR	3(*), 1(+)	4(*), 2(+)	220.02us	201us	9.51W	14.01W
			Constraint =500us		*Constraint =11W*	
MPEG MMV	4(*), 1(+)	5(*), 1(+)	340us	281us	11.9 W	14.4W
			Constraint =600us		*Constraint =12W*	
BPF	2(*), 1(+)	4(*), 2(+)	500.04us	140us	7.01W	14.0W
			Constraint =600us		*Constraint =12W*	
JPEG Downsample	2(*), 4(+)	1(*), 2(+)	140.32us	300us	13.13W	6.57W
			Constraint =400us		*Constraint =10*W	

optimizing circuit latency (and does not report power values in the paper). Therefore, for the sake of comparison, we have evaluated the estimated power using [27] for all selected benchmarks.

With respect to achieved QoR, the MO-PSE produces better solutions in terms of execution time compared to [27] for all the benchmarks as clearly shown in Table 7.12. For example in case of IIR Butterworth filter benchmark the optimal resource configuration found 2 (*) and 1(+), the execution time of solution is 220.01 µs and power consumed by optimal solution is 7.01W. On the other hand [27], based on same constraints, yields an optimal resource configuration which is 4(*), 1(+) with 260 µs execution time and 9.42w power consumption. Moreover, for MPEG MMV the optimal resource configuration found is 4(*), 1(+) with

340µs execution time and the power consumption is 11.9w; while [27] yields 4(*), 2(+) as an optimal architecture with 480µs execution time and 13.91w power consumption. Additionally, during experiments it was found out that [27] suffers from power constraint violation (after estimation) for some selected benchmarks such as IIR, ARF and MPEG MMV.

Therefore, it can be summarized from Table 7.13 that proposed approach achieved better QoR more than 34%; the average reduction in exploration time is more than 40% compared to [27] for the tested benchmarks. For quick observation, graphical representation of the comparison is shown in Figure 7.3 and Figure 7.4 respectively.

Table 7.11 Comparison of proposed approach with [29] in terms of exploration time and QoR

	Exploration Time		**Quality of Result (QoR)**	
Benchmark	**MO-PSE**	**[29]**	**MO-PSE**	**[29]**
IIR Butterworth	0.065 sec	20secs	0.43	0.48
MESA Horner Bezier	0.078 sec	2.30 min	0.361	0.367
ARF	0.252 sec	10.5 min	0.34	0.33
EWF	0.323 sec	10.1 min	0.47	0.63
FIR	0.136 sec	3.20 min	0.27	0.32
MPEG MMV	0.12 sec	6.43 min	0.25	0.26
BPF	0.261 sec	4.33 min	0.46	0.50
JPEG Downsample	0.205 sec	7.59 min	0.36	0.37
	Average Reduction = 99%		Average Improvement= 10%	

Table 7.12 Experimental Result of comparison with [27] for the tested benchmarks

Benchmark	**Parameters of comparison**					
	Resource Configuration		**Execution Time (us)**		**Power (W)**	
	MO-PSE	**[27]**	**MO-PSE**	**[27]**	**MO-PSE**	**[27]**
IIR Butterworth	2(*), 1(+)	3(*), 1(+)	220.01us	260us	7.01W	9.42W
			Constraint =300us		*Constraint =8W*	
MESA Horner Bezier	3(*), 1(+)	2(*), 1(+)	100.1us	620us	9.61W	6.95W
			Constraint = 300us		Constraint = 8W	
ARF	3(*), 1(+)	4(*), 1(+)	520.2us	580us	9.49W	11.8W
			Constraint = 600us		*Constraint = 10W*	
EWF	2(*), 1(+)	1(*), 1(+)	320us	1180us	7.02W	4.49W
			Constraint =500us		*Constraint =8W*	
FIR	3(*), 1(+)	2(*), 2(+)	220.02us	540us	9.51W	8.98W
			Constraint =500us		*Constraint =11W*	
MPEG MMV	4(*), 1(+)	4(*), 2(+)	340us	480us	11.9 W	13.91W
			Constraint =600us		*Constraint =12W*	
BPF	2(*), 1(+)	2(*), 2(+)	500.04us	600us	7.01W	8.98W
			Constraint =600us		*Constraint =12W*	
JPEG Downsample	2(*), 4(+)	2(*), 1(+)	140.32us	600us	13.13 W	6.524W
			Constraint =400us		*Constraint =10*W	

Table 7.13 Comparison of proposed approach with [27] in terms of exploration time and QoR

	Exploration Time		**Quality of Result (QoR)**	
Benchmark	**MO-PSE**	**[27]**	**MO-PSE**	**[27]**
IIR Butterworth	0.065 sec	0.099 sec	0.43	0.55
MESA Horner Bezier	0.078 sec	0.162 sec	0.36	0.60
ARF	0.252 sec	0.343 sec	0.34	0.40
EWF	0.323 sec	0.670 sec	0.47	0.94
FIR	0.136 sec	0.296 sec	0.27	0.48
MPEG MMV	0.12 sec	0.312 sec	0.25	0.32
BPF	0.261 sec	0.421 sec	0.46	0.57
JPEG Downsample	0.205 sec	0.546 sec	0.36	0.62
	Average Reduction = 49.45%		Average Improvement= 34.37%	

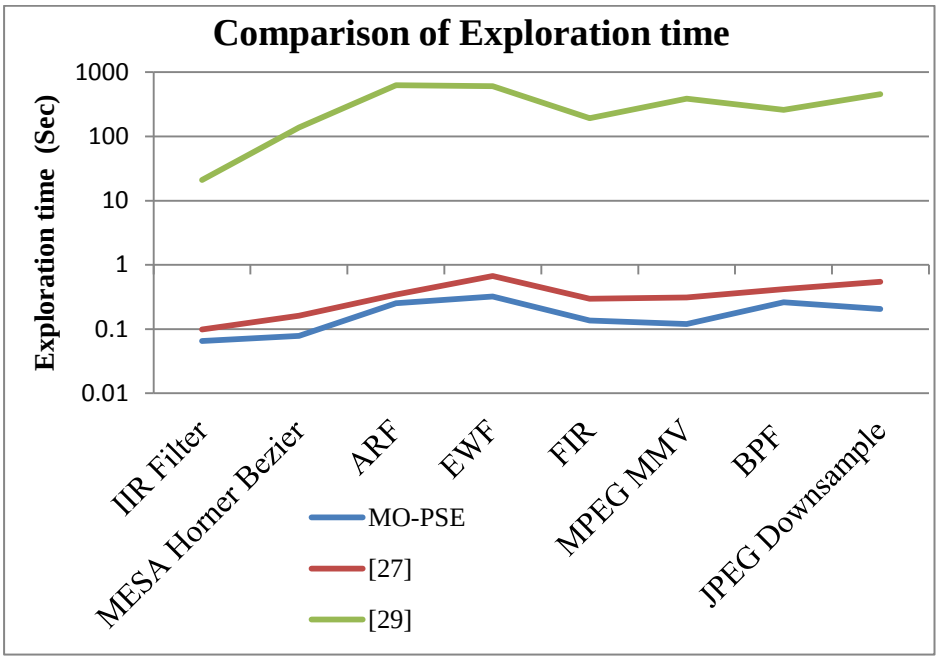


Figure 7.3 Comparison of MO-PSE and [27], [29] in terms of exploration time

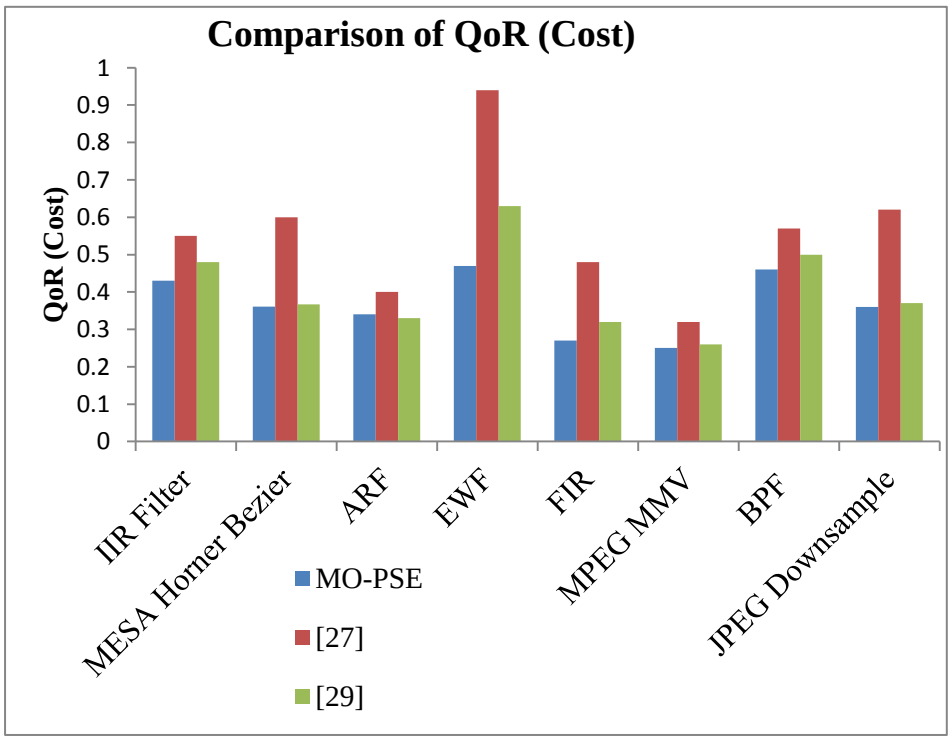


Figure 7.4 Comparison of MO-PSE and [27], [29] in terms of QoR

7.1.2.3. Results obtained through proposed approach for conditional CDFGs

As evident from the Table 7.14, based on the user constraints specified for execution time and power, the proposed approach has been comprehensively able to meet the specified constraints and find an optimal result. For example, in case of CDFG1, the explored solution is 2(*),1(+),1(<), 12(mux),3(demux). Additionally, the explored solution for CDFG1 has execution time and power of 10.97ms and 0.26mW based on the specified constraint values of 12ms and 0.30 mW respect

Table 7.14 Results of estimated power and execution time using proposed approach for the CDFG benchmarks

Note: for proposed approach settings : φ = 0.5, the value of ω will be linearly decreased between [0.9-0.1],value of b = 2, swarm size = 3, M = 100, N=1000, £=10 and stopping Criterion: S^1					
Bench-marks		**Execution Time**		**Power**	
	Resources found	**Cons-traint**	**Proposed solution**	**Cons-traint**	**Proposed solution**
CDFG1	2(*), 1(+),1(<)	12ms	10.9ms	0.30mW	0.26mW
CDFG2	3(*), 1(+),1(<)	12ms	10.67ms	0.40mW	0.33mW
CDFG3	2(*), 1(+),1(<)	13ms	11.0ms	0.30mW	0.26mW

7.2 Experimental results: the proposed approach 'Automated Exploration of Datapath and Unrolling Factor during Power-Performance Trade-off in Architectural Synthesis Using Multi-Dimensional PSO Algorithm'

This section describes the experimental results of the proposed approach explained in Chapter 4 and the improvements obtained compared to recent approach [27,42]. The proposed approach has been implemented in Java and run on Intel core i5-2450M processor,2.5 GHz with 3MB L3 cache memory and 4GB DDR3 RAM. Multiple benchmarks were chosen for testing that include DFGs and CDFGs such as autoregressive filter (ARF) [53,57], band-pass filter (BPF) [5], discrete cosine transformation (DCT) [5, 58], discrete wavelet transformation (DWT) [5], elliptic wave filter (EWF) [53], fast Fourier transformation (FFT) [56], finite impulse response (FIR) filter [5,57], IIR Butterworth [53], MESA-Horner Bezier[53], inverse discrete cosine transformation (IDCT) [5, 58], MPEG motion vectors (MMV) [5, 53], wave digital filter (WDF) [5], differential equation [56] and test case. Module library describe in chapter 4 section 4.4.1. The results are divided into three phases.

- Sensitivity analysis of PSO swarm size (S).

  (*Note*: In the proposed algorithm, the convergence time is evaluated @ I' (the starting iteration from which onwards the global best particle position remains constant for 10 consecutive iterations) while the exploration time is evaluated @ I = I' + 10.
- The results obtained through proposed approach.
- Comparison of proposed approach with previous DSE approaches [42][27] in terms of quality of result (QoR) and exploration run time.

7.2.1. Sensitivity analysis

The impact of various PSO parameters on proposed design space exploration is analysed and presented in this section. This experimental analysis assists the designer in pre-tuning the PSO parameters to an optimum value before performing DSE. In this subsection analysis of swarm size is presented and inertia weight, acceleration coefficient and termination criterion not presented because the behaviour of these PSO parameter are

similar with section 7.1. Thus, we present only swarm size analysis and other PSO parameters values taken from section 7.1.

- *Swarm Size (S)*

Significant swarm size maintains tradeoff between exploration time and quality of result. A larger swarm size covers larger design space during one iteration step (with a chance to get a better result) but simultaneously subjected to increase in exploration time because of larger number of particles as well as greater computational complexity per iteration. On the contrary, a smaller swarm size needs more iteration to explore a better result for larger problem size. Therefore, three different swarm sizes have been analyzed and their impacts on cost and exploration time are reported (as shown in Table 7.15). At the time of selection, first priority is given to lower cost solution (higher quality result), followed by the second priority given to the exploration time. Based on this analysis, the selected swarm sizes for benchmarks, used as our base line parameter are underlined in Table 7.15. As evident from Table 7.15, the best tradeoff between fast exploration and searching optimal solution can be obtained by setting S=3 in case of small benchmarks (with smaller design space) and S=5 or 7 in case of larger benchmark (with larger design space). For example, in case of ARF, S=3 give an optimal solution (cost = -0.159) in minimum exploration run time of 3183 ms. While, in case of FFT, S=5 gives an optimal solution (cost = -0.232) in minimum exploration time of 85656 ms.

7.2.2. Results obtained through proposed algorithm

As evident from Table 7.16 and 7.17 the solution explored by the proposed approach comprehensively meets the user defined constraints for power and execution time as well as minimizes the hybrid cost. For example, in case of EWF, the explored solution 3(*),1(+) has execution time 39.9 ms and power of 0.61 mW based on the user constraints of 50 ms and 0.7 mW respectively. Moreover, in case of FFT, the explored solution 4(*), 3(+), 2(-), 1(<) and UF=4 has execution time of 348.48us and power of 1.44mW which satisfies (as well as minimizes) the given user constraints. It is worthy to mention that the proposed approach also has the capability to explore multiple optimal solutions as reported in Table 7.16 and 7.17.

The Figure 7.5 shows the gradual improvement in the global cost, per iteration, over the lifetime of the algorithm for all the tested benchmarks. The straight line in the curve denotes

Table 7.15 Comparison of cost and exploration time with respect to swarm size (S) for the proposed approach

Benchmark	Swarm Size	Cost	Exploration run time (milliseconds)
IIR Butterworth	3	-0.135	90
	5	-0.135	130
	7	-0.135	120
DCT	3	-0.162	2468
	5	-0.162	3001
	7	-0.162	3006
MESA Horner	3	-0.152	352
	5	-0.152	409
	7	-0.152	467
ARF	3	-0.159	3183
	5	-0.159	3890
	7	-0.159	4061
EWF	3	-0.109	508
	5	-0.109	528
	7	-0.109	580
DWT	3	-0.116	357
	5	-0.129	490
	7	-0.129	457
MPEG Motion Vector	3	-0.127	2373
	5	-0.127	3706
	7	-0.127	3644
IDCT	3	-0.173	2839
	5	-0.173	2779
	7	-0.173	3184
BPF	3	-0.98	825
	5	-0.98	838
	7	-0.98	830
JPEG Downsample	3	-0.153	176
	5	-0.153	215
	7	-0.153	232
WDF	3	-0.153	788
	5	-0.153	916
	7	-0.153	1009
FIR	3	-0.213	3104
	5	-0.213	5100
	7	-0.213	5420
FFT	3	-0.225	93990
	5	-0.232	85656
	7	-0.232	97488
Differential Equation	3	-0.219	21045
	5	-0.226	26770
	7	-0.226	79279
Testcase	3	-0.241	7831
	5	-0.241	11013
	7	-0.241	15623

Table 7.16 Results of Estimated Power and Execution Time Using Proposed Approach for DFGs

Note: for proposed approached baseline parameters: $\varphi_1 = \varphi_2 = 0.5$, the value of ω is linearly decreased between [0.9-0.4],b = 2, swarm size (S) = 3,5 M = 100, λ=1000, £=10					
Benchmark		**Execution Time**		**Power**	
	Resources found	**Constraint**	**Proposed solution**	**Constraint**	**Proposed solution**
ARF	4(*), 1(+)	75ms	43.74ms	0.8mW	0.65mW
IDCT	3(*), 1(+)	70ms	33.37ms	0.9mW	0.79mW
BPF	4(*), 2(+)	30ms	11.50ms	0.75mW	0.72mW
IIR Filter	2(*), 1(+)	30ms	22.1ms	0.35mW	0.28mW
DCT	4(*), 1(+)	60ms	33.35ms	1.0mW	0.83mW
EWF	3(*), 1(+)	50ms	39.9ms	0.7mW	0.61mW
DWT	4(*), 1(+)	30ms	10.98ms	0.6mW	0.58mW
	4(*), 2(+)	30ms	10.97ms	0.6mW	0.57mW
JPEG Down-sample	2(*), 1(+)	20ms	10.95ms	0.6mW	0.56mW
MPEG MMV	4(*), 1(+)	36ms	33.35ms	1.0mW	0.65mW
	6(*), 1(+)	36ms	22.42ms	1.0mW	0.77mW
MESA	3(*), 1(+)	30ms	10.90ms	0.5mW	0.45mW
WDF	4(*), 1(+)	35ms	11.51ms	0.8mW	0.76mW

no change in last 10 iterations which indicates that the algorithm has converged to its optimal point.

The Figure 7.6 shows the variation in power, execution time and number of control steps (number of control steps needed to execute loop body C_{body}) with respect to the unrolling factor as explored by the global best particle. We analyzed the following three aspects of our solution found by the proposed algorithm:

a) *When a solution with higher UF is explored at unchanged resource combination during exploration process*: In this case, a steady rise in power consumption and # of control steps are noted with a simultaneous decline in the execution time value. This is due to the fact that a solution with higher UF (but with same resource combination) will have more code density (resources) as well as greater number of multiplexers for switching operation resulting in higher value of power and control steps. However, higher UF will provide a better performance than its counterpart with lower UF (under same resource constraint). This trend can be observed in case of differential equation benchmark as shown in Fig 7.6 (a), (b) and

(c) where as the value of UF increases from 2 to 4 under same resource combination of (1(+), 1(-), 6(*), 1(<)), the rise in power and # of CS as well as drop in execution time value are observed (the change in power, # of CS and execution time are highlighted with labels). However, for some applications an uncommon situation can also be observed as explained below:

A solution with significant increase in UF (such as UF = 18) found compared to its previous solution (UF = 2) may result in a sudden quantum jump in the value of execution time (which is contrary to the belief) during the exploration process (as shown in Figure 7.6 (f)). The increase in execution time is due the fact that Testcase is highly sequential in nature whereby the output of the previous loop is used as an input in the next loop quite early (thereby lacking the option for parallelization). Besides the logic above, the decrease in resources also

Table 7.17 Results of Estimated Power and Execution Time Using Proposed Approach for CDFGs

Note: For proposed approached baseline parameters: $\varphi_1 = \varphi_2 = 0.5$, the value of ω is linearly decreased between [0.9-0.4], b = 2, swarm size (S) = 3,5 M = 100, £=10

Bench-mark	**Resources found**	**Execution Time**		**Power**	
		Constraint	**Proposed solution**	**Constraint**	**Proposed solution**
FIR (I=8)	4(*), 1(+),1(<), UF=4	60us	24.24us	0.5mW	0.47mW
	3(*), 1(+),1(<), UF=2	60us	46.24us	0.5mW	0.34mW
FFT (I=16)	4(*), 3(+), 2(-), 1(<), UF=4	800us	348.48us	2.0mW	1.44mW
	5(*), 1(+), 1(-), 1(<), UF=4	800us	358.5us	2.0mW	1.51mW
Differential equation (I=16)	6(*), 1(+), 1(-), 1(<), UF=2	600us	277.4us	1.2mW	0.73mW
	6(*), 1(+), 1(-), 1(<), UF=4	600us	225.6us	1.2mW	0.95mW
Test case (I=36)	2(*), 1(+),1(<), UF=1	500us	406us	1.5mW	0.26mW
	3(*), 1(+),1(<), UF=2	500us	401us	1.5mW	0.39mW

contributes to the radical increase in execution time. Therefore this condition is highly application dependent. Further in case of # of CS required, besides the explanation provided above, the reduction in resources (from 2(+), 4(*), 1(<)) to 2(+), 2(*), 1(<)) also contributes to its increase (as seen in Fig. 7.6 (i)). However, the increase in power (even with reduced resources from 2(+), 4(*), 1(<)) to 2(+), 2(*), 1(<)) is anticipated due to heavy sharing of resources (courtesy of large UF value) resulting in a large multiplexer size being required.

b) *When a solution with higher resource combination is found at unchanged value of UF*: in

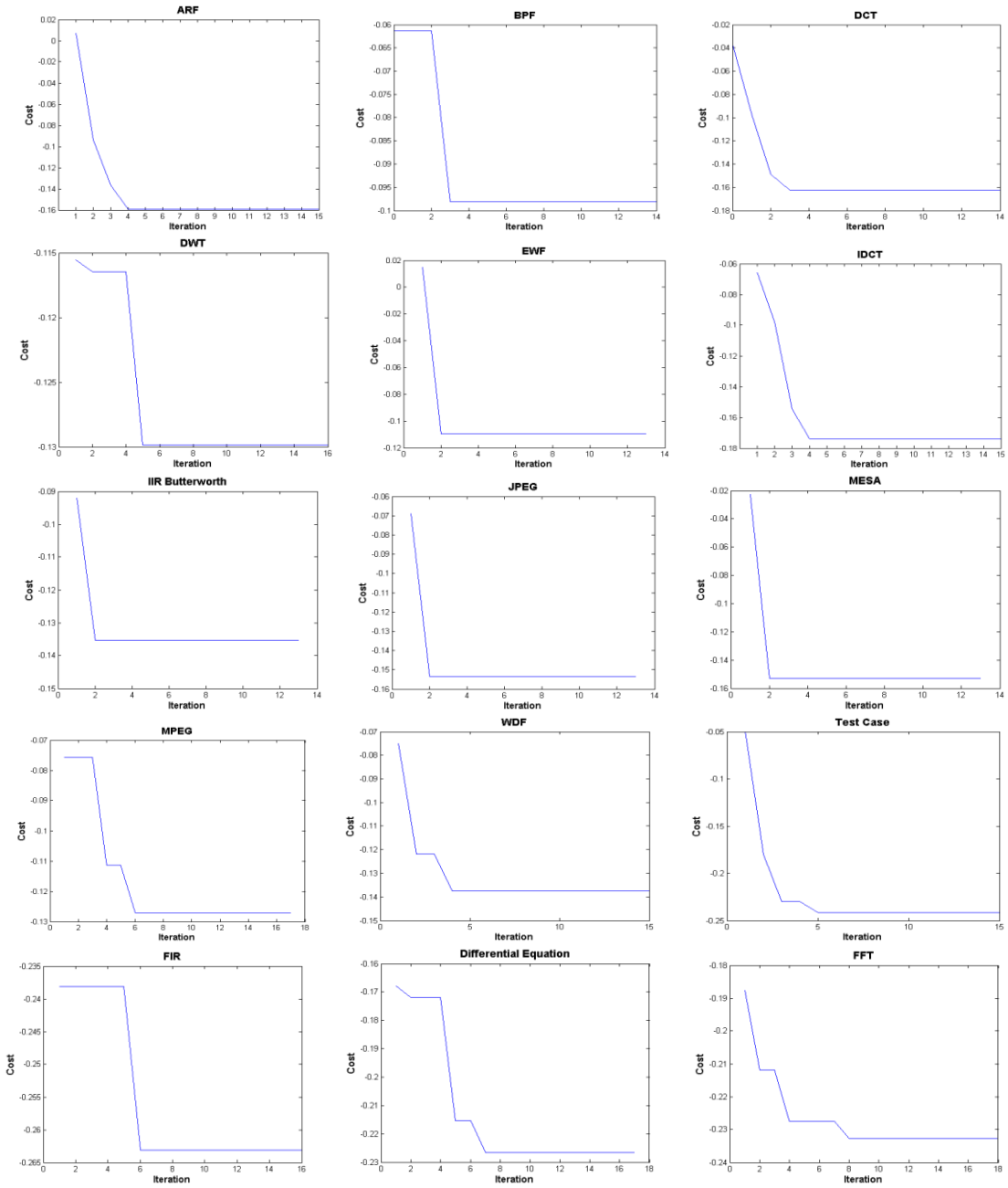


Figure 7.5 Change in cost of global best particle for various benchmarks
Note: Baseline parameter: $\varphi_1$and φ_2=0.5; ω= linearly decreasing [0.9-0.4]; b = 2; S=3, 5; M=100; λ=1000; £=10

such a case, power increases because of augmentation in number of resources. On other hand, # of control steps required and final execution time decreases (i.e. performance improves). This trend can be observed in case of differential equation, as shown in figure 7.6 (a), (b), (c), where a new solution with resource combination of(1(+),1(-), 6(*), 1(<)) is explored by the algorithm compared to previous solution of (1(+), 1(-), 4(*), 1(<)); whereby in both cases the UF value = 4 did not change while exploration. The power value increases and at the same time execution time and control steps decreases (highlighted with data labels).

c) *In the third case, when solution explored has redundant resources for a problem*: In such a case, decrement in a type of resource has no impact on execution time and control steps but static power will be steadily decreasing resulting in decrease of total power. For example in case of test case reported, as shown in fig 7.6 (g), (h), (i), UF = 1 remains constant, but the resources changes from 1(+),3(*),1(-) to 1(+),2(*),1(-) during exploration. Now for the test case CDFG (in Fig. 4.4), there are only two multiplication operations which can be performed in parallel @ UF = 1 (i.e. in the non-unrolled version), hence, one extra multiplier in the solution becomes redundant when it is not unrolled. Therefore, for some applications at specific UF values there is no impact of higher resource quantity in the improvement of performance and control step.

Thus from the empirical evidences obtained it is clearly shown that proposed algorithm responds as per our anticipated reckoning and is able to perform a simultaneous exploration of optimal datapath and UF under multi-objective user constraints at minimum exploration time.

7.2.3. Comparison of proposed approach with [42] and [27]

Table 7.18 and Table 7.19 show the comparative qualitative analysis with [42] and [27] for DFGs and CDFG benchmarks. For the sake of reporting comparative results, completely unrolled CDFG (flattened version of the application), is taken as an input for [42] and [27] as it is not directly handled by them. Therefore, as evident from Table 7.18 and Table 7.19, the QoR as well as the exploration speed of proposed approach is significantly better than [42] and [27] respectively. The QoR is determined using eqn. (7.1).

Therefore, simple calculation revels that proposed approach is simultaneously able to achieve average improvement in QoR of greater than 28% and 36% as well as average

reduction in exploration time of greater than 99% and 94% when compared with [42] and [27] respectively as shown in Table 7.18 and Table 7.19.

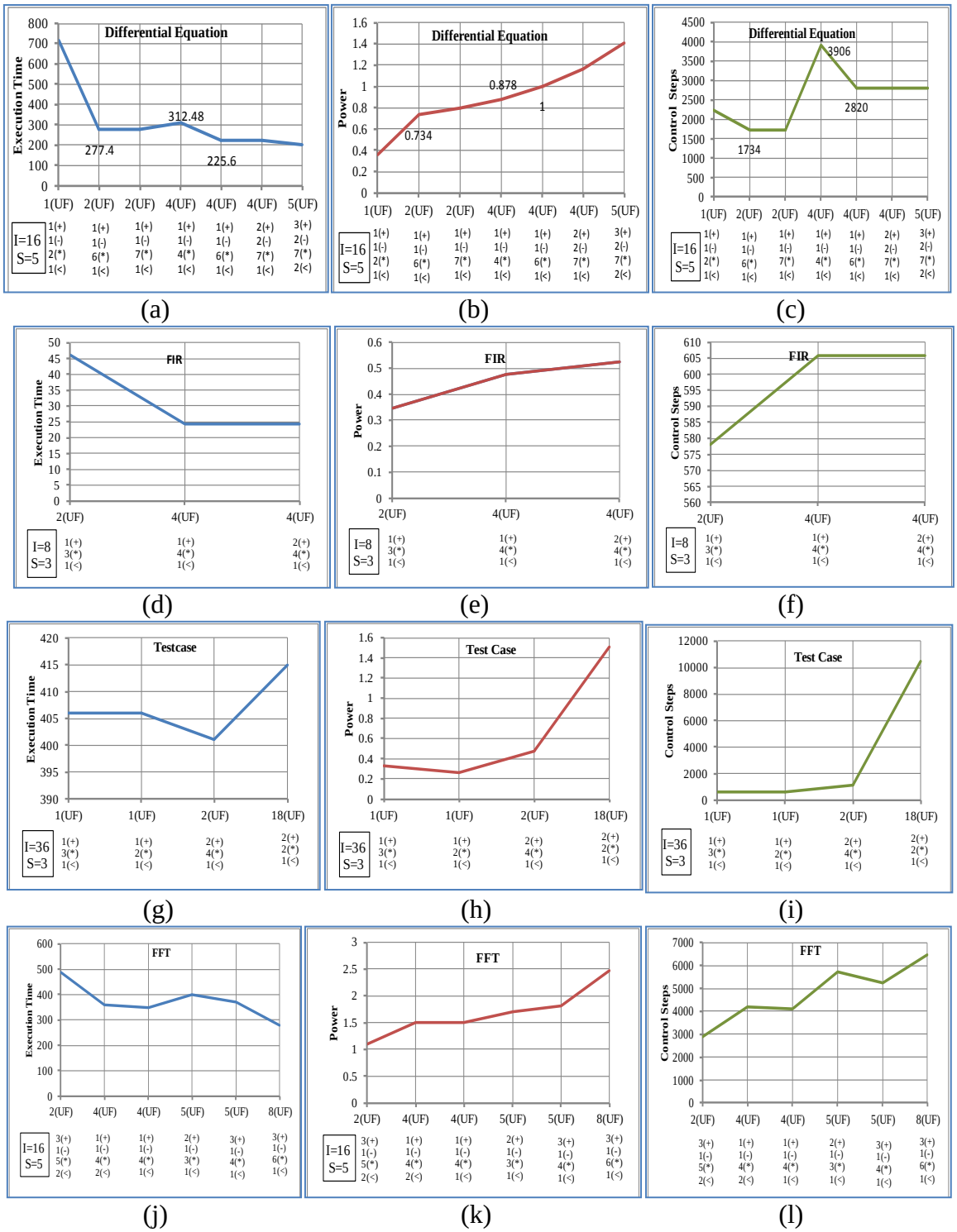


Figure 7.6 Analysis of power, execution time, control steps of global best particle w.r.t unrolling factor

Table 7.18 Comparison of proposed approach with [42] in terms of exploration run time and QoR

Benchmark	Resource combination		Exploration Runtime		QoR (cost)	
	Proposed approach	**[42]**	**Proposed approach (sec)**	**[42] (Min)**	**Proposed approach**	**[42]**
IIR Filter	2(*),1(+)	1(*),1(+)	0.09	1.30	0.46	0.72
DCT	4(*),1(+)	2(*),2(+)	2.46	7.59	0.44	0.52
MESA Horner	3(*),1(+)	2(*),1(+)	0.352	2.12	0.46	0.50
EWF	3(*),1(+)	1(*),1(+)	0.508	13.09	0.59	0.81
DWT	4(*),2(+)	2(*),1(+)	0.457	3.02	0.48	0.56
ARF	4(*),1(+)	2(*),1(+)	3.183	5.16	0.43	0.49
MPEG MMV	4(*),1(+)	3(*),1(+)	2.373	5.45	0.32	0.39
IDCT	3(*),1(+)	1(*),1(+)	2.779	14.30	0.44	0.75
BPF	4(*),2(+)	3(*),2(+)	0.825	5.4	0.51	0.63
JPEG Downsample	2(*),1(+)	1(*),2(+)	0.176	2.46	0.43	0.56
WDF	4(*),1(+)	2(*),1(+)	0.788	7.50	0.48	0.57
FIR	4(*), 1(+),1(<), UF=4	3(*), 1(+),1(<), UF=8	3.10	4.31	0.35	0.41
FFT	4(*), 3(+), 2(-), 1(<), UF=4	3(*), 2(+), 1(-), 1(<), UF=16	85.656	>1hr	0.28	0.60
Differential equation	6(*), 1(+), 1(-), 1(<), UF=2	4(*), 1(+), 2(-), 1(<), UF=16	26.77	>1hr	0.24	0.52
Test case	2(*), 1(+),1(<), UF=1	2(*), 4(+),1(<), UF=36	7.831	>1hr	0.29	0.78
Average decrement in exploration run time = 99.25%			**Average reduction in cost = 28.88%**			

Table 7.19 Comparison of proposed approach with [27] in terms of exploration run time and QoR

Benchmark	Resource combination		Exploration Runtime		QoR (cost)	
	Proposed approach	**[27]**	**Proposed approach (sec)**	**[27] (sec)**	**Proposed approach**	**[27]**
IIR Filter	2(*),1(+)	3(*),1(+)	0.09	2.13	0.46	0.52
DCT	4(*),1(+)	1(*),1(+)	2.46	13.3	0.44	0.80
MESA Horner	3(*),1(+)	1(*),1(+)	0.352	3.41	0.46	0.75
EWF	3(*),1(+)	1(*),1(+)	0.508	10.3	0.59	0.84
DWT	4(*),2(+)	1(*),1(+)	0.457	4.48	0.48	0.77
ARF	4(*),1(+)	2(*),1(+)	3.183	7.22	0.43	0.50
MPEG MMV	4(*),1(+)	5(*),1(+)	2.373	6.63	0.32	0.35
IDCT	3(*),1(+)	2(*),2(+)	2.779	11.29	0.44	0.55
BPF	4(*),2(+)	2(*),1(+)	0.825	7.7	0.51	0.75
JPEG Downsample	2(*),1(+)	1(*),1(+)	0.176	8.21	0.43	0.72
WDF	4(*),1(+)	1(*),1(+)	0.788	10.77	0.48	0.85
FIR	4(*), 1(+),1(<), UF=4	4(*), 1(+),1(<), UF=8	3.10	5.03	0.35	0.38
FFT	4(*), 3(+), 2(-), 1(<), UF=4	2(*), 1(+), 1(-), 1(<), UF=16	85.656	1415	0.28	0.70
Differential equation	6(*), 1(+), 1(-), 1(<), UF=2	3(*), 1(+), 1(-), 1(<), UF=16	26.77	436	0.24	0.51
Test case	2(*), 1(+),1(<), UF=1	2(*), 1(+),1(<), UF=36	7.831	351	0.29	0.87
Average decrement in exploration run time = 94.50%			**Average reduction in cost = 36.58%**			

7.3 Experimental results: the proposed approach 'Simultaneous Exploration of Optimal Datapath and Loop Based High level Transformation during Area-Delay Trade-off in Architectural Synthesis Using Swarm Intelligence'

This section describes the experimental results of the proposed approach explained in Chapter 5 and the improvements obtained compared to recent approach [27,59]. The proposed approach has been implemented in Java and run on Intel core i5-2450M processor (2.5 GHz) with 3MB L3 cache memory and 4GB DDR3 RAM. Multiple benchmarks were chosen for testing that include DFGs and CDFGs. such as ARF, BPF, DCT, DWT, EWF, FFT, FIR filter, IIR Butterworth, MESA-Horner Bezier, IDCT, MPEG motion vectors (MMV), WDF, differential equation and test case. The module library is provided in chapter 5 section 5.3.1. The results are divided into three phases.

- Sensitivity analysis of various PSO parameters such as swarm size(S) and its impact on the proposed DSE methodology in terms of cost, exploration time.
- The results obtained through proposed.
- Comparison of proposed approach with previous DSE approaches [27] [59] in term of quality of result (QoR) and exploration run time achieved.

7.3.1. Sensitivity analysis

The impact of various PSO parameters on proposed design space exploration is analysed and presented in this section. This experimental analysis assists the designer in pre-tuning the PSO parameters to an optimum value before performing DSE.

7.3.1.1. Swarm Size (S)

Significant swarm size maintains tradeoff between exploration time and quality of result. A larger swarm size covers larger design space during one iteration step (with a chance to get a better result) but simultaneously subjected to increase in exploration time because of larger number of particles as well as greater computational complexity per iteration. On the contrary, a smaller swarm size needs more iteration to explore a better result for larger problem size. Therefore, three different swarm sizes have been analyzed and their impacts on

Table 7.20 Comparison of cost and exploration time with respect to swarm size (S)

Benchmark	Swarm Size	Cost	Exploration Run time (millisecond)
IIR Butterworth	3	-0.14	65
	5	-0.14	85
	7	-0.14	93
DCT	3	-0.175	1538
	5	-0.175	1675
	7	-0.175	1882
MESA	3	-0.132	216
	5	-0.132	245
	7	-0.132	252
ARF	3	-0.199	1728
	5	-0.199	1704
	7	-0.199	2086
EWF	3	-0.132	344
	5	-0.132	314
	7	-0.132	324
DWT	3	-0.169	311
	5	-0.169	231
	7	-0.169	290
MPEG	3	-0.88	1593
	5	-0.88	1711
	7	-0.88	2051
IDCT	3	-0.167	1407
	5	-0.167	1330
	7	-0.167	1718
BPF	3	-0.93	426
	5	-0.93	506
	7	-0.93	495
JPEG Downsample	3	-0.17	107
	5	-0.17	103
	7	-0.17	148
WDF	3	-0.155	548
	5	-0.155	576
	7	-0.155	642
FIR	3	-0.223	1866
	5	-0.223	2399
	7	-0.223	1924
FFT	3	-0.224	39999
	5	-0.23	91888
	7	-0.23	125548
Differential eqn	3	-0.197	14704
	5	-0.222	44245
	7	-0.222	56784
Testcase	3	-0.107	5120
	5	-0.107	5157
	7	-0.107	8145

cost and exploration time are reported (as shown in Table 7.20). At the time of selection, first priority is given to lower cost solution (higher quality result), followed by the second priority given to the exploration time. *(Note:-based on this analysis, the selected swarm sizes for benchmarks used as our base line parameter are underlined.)*

As evident from Table 7.20, the best tradeoff between fast exploration and searching optimal solution can be obtained by setting S=3 in case of small benchmarks (with smaller design space) and S=5 or 7 in case of larger benchmark (with larger design space). For example, in case of MESA, S=3 give an optimal solution (cost = -0.132) in minimum exploration run time of 216ms. However, in case of FFT, S=5 give an optimal solution with least cost/best quality (cost = -0.23) in minimum exploration time of 91888 ms.

7.3.2. Results obtained through proposed algorithm

As evident from Table 7.21 and 7.22 the solution explored by the proposed approach comprehensively meets the user defined constraints for area and execution time as well as minimizes the hybrid cost. For example, in case of DCT, the explored solution 4(*),1(+) has execution time 33356us and area of 28140au based on the user constraints of 60000us and 35000au respectively. Moreover, in case of FIR, the explored solution 4(*), 1(+), 1(<) and UF=4 has execution time of 24.24us and area of 16184au which satisfies (as well as minimizes) the given user constraints. (*Note 1*: that, since PSO is a parallel evolutionary algorithm, where multiple particles participate in exploration process therefore, it assures escaping the local optima. Moreover, inclusion of proposed mutation strongly reduces any chance of local optimal convergence. Further, the solutions obtained for the tested benchmarks are real optimal solutions in most of the cases (except FFT) which can be verified by comparing with the golden solutions found by exhaustive analysis. *(Note 2:-The proposed approach also has the capability to explore multiple optimal solutions as reported in Table 7.21 and 7.22).*

The Figure 7.7 shows the variation in area, execution time and number of control steps (number of control steps needed to execute loop body C_{body}) with respect to the unrolling factor. *(Note:- the corresponding resource combination for respective UFs are also indicated below the X axis).*We analyzed the following three aspects of our solution found by the proposed algorithm:

a) When a solution with higher UF is found at unchanged resource combination during exploration process: In this case, a steady rise in area and # of control steps are noted with a

Table 7.21 Results of Estimated Area and Execution Time Using Proposed Approach for DFGs

Note: for proposed approached baseline parameters : $\varphi_1 = \varphi_2 = 0.5$, the value of ω will be linearly decreased between [0.9-0.1],value of b = 2, swarm size = 3,5 M = 100, λ=1000, £=10					
Bench-mark		**Area (au)**		**Execution Time(us)**	
	Resources found	**Const raint**	**Proposed solution**	**Const raint**	**Proposed solution**
ARF	4(*),1(+)	30000	22092	75000	43741
IDCT	3(*),1(+)	30000	26810	70000	33377.8
BPF	4(*),2(+)	25000	24500	30000	11507
IIR Butterworth	2(*),1(+)	12000	9604	30000	22150
DCT	4(*),1(+)	35000	28140	60000	33356
EWF	2(*),1(+)	25000	20944	50000	39969
DWT	4(*),2(+)	22000	19208	30000	10979
	4(*),1(+)	22000	19824	30000	10979
JPEG Downsample	2(*),1(+)	25000	19054	15000	10953
MPEG MMV	4(*),1(+)	30000	22092	36000	33355
	6(*),1(+)	30000	26264	36000	22420
MESA	3(*),1(+)	16000	15092	30000	10979
WDF	4(*),1(+)	27000	25872	35000	11511

simultaneous decline in the execution time value. This is due to the fact that a solution with higher UF (but with same resource combination) will have more code density (resources) as well as greater number of multiplexers for switching operation resulting in higher value of area and control steps. However, higher UF will provide a better performance than its counterpart with lower UF (under same resource constraint). This trend can be observed in case of differential equation benchmark as shown in Figure 7.7 (a), (b) and (c) where as the value of UF increases from 2 to 4 under same resource combination of (1(+), 1(-), 4(*), 1(<)), the rise in area and # of CS as well as drop in execution time value are observed.

b) When a solution with higher resource combination is found at unchanged value of UF: in such a case, area increases because of augmentation in number of resources. On other hand,# of control steps required and final execution time decreases (i.e. performance improves). This trend can be observed in case of differential equation, as shown in Figure 7.7 (a), (b), (c), a new solution with resource combination of (1(+),1(-), 4(*), 1(<)) is explored by the algorithm compared to previous solution of (1(+), 1(-), 2(*), 1(<)); whereby in both cases the UF = 4 did not change while exploration. The area value increases and at the same time execution time and control steps decreases.

Table 7.22 Results of Estimated Area and Execution Time Using Proposed Approach for CDFGs

Note: For proposed approached baseline parameters : $\varphi_1 = \varphi_2 = 0.5$, the value of ϖ will be linearly decreased between [0.9 - 0.1],value of b = 2, swarm size = 3,5 M = 100, £=10					
Benchmark		**Area (au)**		**Execution Time (us)**	
	Resources found	**Constraint**	**Proposed solution**	**Constraint**	**Proposed solution**
FIR	4(*), 1(+),1(<), UF=4	18000	16184	60us	24.24
FFT	4(*), 1(+), 1(-), 1(<), UF=4	69000	49588	800us	358.56
	5(*), 1(+), 2(-), 1(<), UF=2	69000	34426	800us	498.4
Differential equation	6(*), 1(+), 1(-), 1(<), UF=2	40000	25032	600us	277.4
	4(*), 1(+), 1(-), 1(<), UF=2	40000	20860	600us	360.96
Test case	2(*), 1(+),1(<), UF=1	20000	8988	500us	406.08
	3(*), 1(+),1(<), UF=2	20000	13342	500us	401.04

c) In the third case, when solution explored have redundant resources for a problem. In such a case, increment in the resources has no impact on execution time and control steps but area steadily increases. For example in case of FIR reported, as shown in Figure 7.7 (d), (e), (f) at UF = 4, the resource configuration changes from 2(+),4(*),1(-) to 1(+),4(*),1(-) during exploration. As shown in CDFG (Figure 5.3) of the FIR, there are no parallel addition operations being performed in case of UF=2 (hence will be the same for UF=4), therefore, one extra adder in the solution becomes redundant (*Note: A simple observation will reveal that the input of the adder of next iteration always requires the output from previous loop iteration, thereby rendering the additions to be sequential in nature*). Therefore, no impact of higher resource number in the improvement of performance and control step can be noted.

Thus from the empirical evidences obtained it is clearly shown that proposed algorithm responds as per the theoretical calculation (as well as expectation) and is able to perform a simultaneous exploration of optimal datapath and UF under multi-objective user constraints at minimum exploration time.

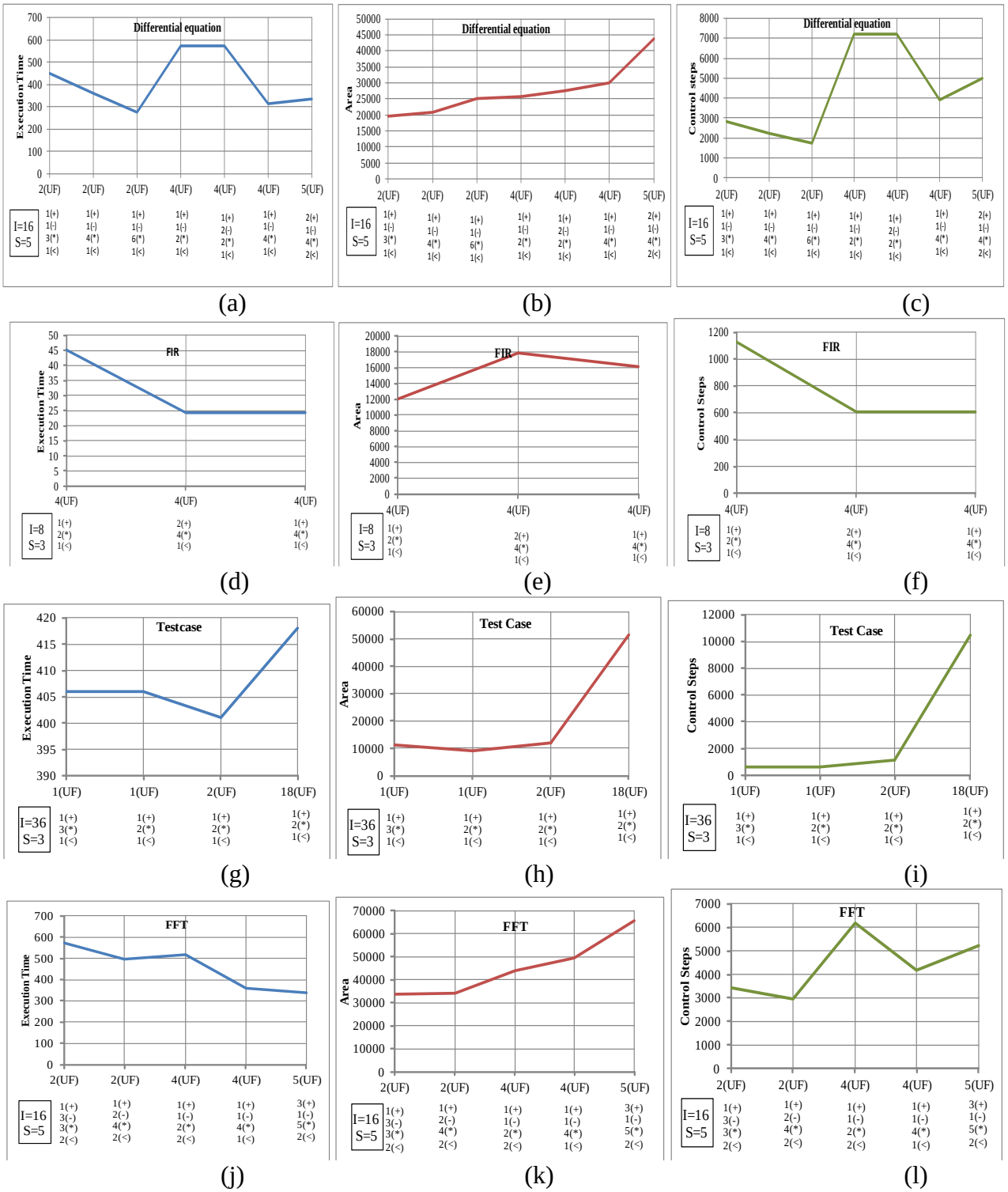


Figure 7.7 Analysis of area, execution time, control steps of global best particle w.r.t unrolling factor

7.3.3. Comparison of proposed approach with [27] and [59]

Table 7.23 and Table 7.24 shows the comparative qualitative analysis with [27] and [59] for DFGs and CDFG benchmarks. *(Note: - For the sake of reporting comparative results, completely unrolled CDFG (flattened version of the application), is taken as an input for [27] and [59] as it is not directly handled by them).* As evident from Table 7.23 and Table 7.24, the QoR as well as the exploration speed of proposed approach is significantly better than [27] and [59] respectively. The QoR is determined as:

$$QoR = \frac{1}{2}\left(\frac{A_T}{A_{\max}} + \frac{T_E}{T_{\max}} \right) \tag{7.2}$$

Where, the variables A_{max} and T_{Max} are defined in chapter 5.

Therefore, simple calculation revels that proposed approach is simultaneously able to achieve average improvement in QoR of greater than 23% and 35% as well as average reduction in exploration time of greater than 99% and 92% when compared with [59] and [27] respectively as shown in Table 7.23 and Table 7.24.

Table 7.23 Comparison of proposed approach with [27] in terms of exploration run time and QoR

Benchmark	Resource combination		Exploration Runtime		QoR (cost)	
	Proposed approach	**[27]**	**Proposed approach (sec)**	**[27] (sec)**	**Proposed approach**	**[27]**
IIR Butterworth	2(*),1(+)	3(*),1(+)	0.065	2.38	0.47	0.52
DCT	4(*),1(+)	1(*),1(+)	1.538	25.5	0.44	0.80
MESA Horner	3(*),1(+)	1(*),1(+)	0.216	4.4	0.45	0.75
EWF	2(*),1(+)	1(*),1(+)	0.314	20.6	0.60	0.84
DWT	4(*),2(+)	1(*),1(+)	0.231	5.9	0.47	0.77
ARF	4(*),1(+)	2(*),1(+)	1.704	11.34	0.43	0.50
MPEG MMV	4(*),1(+)	5(*),1(+)	1.593	12.0	0.33	0.35
IDCT	3(*),1(+)	2(*),2(+)	1.330	21.8	0.43	0.55
BPF	4(*),2(+)	2(*),1(+)	0.426	13.5	0.51	0.76
JPEG Downsample	2(*),1(+)	1(*),1(+)	0.103	20.6	0.43	0.71
WDF	4(*),1(+)	2(*),1(+)	0.548	22.4	0.49	0.75
FIR (I=8)	4(*), 1(+),1(<), UF=4	4(*), 1(+),1(<), UF=8	1.86	4.23	0.36	0.38
FFT (I=16)	4(*), 1(+), 1(-), 1(<), UF=4	2(*), 1(+), 1(-), 1(<), UF=16	91.88	1259	0.29	0.96
Differential equation (I=16)	6(*), 1(+), 1(-), 1(<), UF=2	3(*), 1(+), 1(-), 1(<), UF=16	44.245	322	0.25	0.51
Test case (I=36)	2(*), 1(+),1(<), UF=1	2(*), 1(+),1(<), UF=36	5.12	321	0.29	0.82
Average decrement in exploration run time W.R.T [27]=92.68 %			**Average reduction in cost W.R.T [27]=35.33 %**			

Table 7.24 Comparison of proposed approach with [59] in terms of exploration run time and QoR

Benchmark	Resource combination		Exploration Runtime		QoR (cost)	
	Proposed approach	**[59]**	**Proposed approach (sec)**	**[59] (Min)**	**Proposed approach**	**[59]**
IIR Butterworth	2(*),1(+)	4(*),1(+)	0.065	1.01	0.47	0.48
DCT	4(*),1(+)	4(*),1(+)	1.538	16.5	0.44	0.44
MESA Horner	3(*),1(+)	1(*),1(+)	0.216	2.4	0.45	0.75
EWF	2(*),1(+)	2(*),2(+)	0.314	7.4	0.60	0.64
DWT	4(*),2(+)	4(*),1(+)	0.231	2.24	0.47	0.476
ARF	4(*),1(+)	3(*),1(+)	1.704	9.11	0.43	0.47
MPEG MMV	4(*),1(+)	6(*),1(+)	1.593	4.3	0.33	0.33
IDCT	3(*),1(+)	1(*),1(+)	1.330	18.5	0.43	0.74
BPF	4(*),2(+)	4(*),1(+)	0.426	4.2	0.51	0.54
JPEG Downsample	2(*),1(+)	1(*),1(+)	0.103	2.2	0.43	0.58
WDF	4(*),1(+)	4(*),1(+)	0.548	10.0	0.49	0.48
FIR (I=8)	4(*), 1(+),1(<), UF=4	4(*), 1(+),1(<), UF=8	1.86	3.78	0.36	0.38
FFT (I=16)	4(*), 1(+), 1(-), 1(<), UF=4	2(*),1(+), 1(-), 1(<), UF=16	91.88	>1hr	0.29	0.63
Differential equation (I=16)	6(*), 1(+), 1(-), 1(<), UF=2	4(*), 1(+), 1(-), 1(<), UF=16	44.245	>1hr	0.25	0.48
Test case (I=36)	2(*), 1(+),1(<), UF=1	2(*), 1(+),1(<), UF=36	5.12	>1hr	0.29	0.70
Average decrement in exploration run time W.R.T. [59]= 99.03%			**Average reduction in cost W.R.T. [59]= 23.01%**			

7.4 Experimental results: the proposed approach 'Swarm Inspired Exploration of Architecture and Unrolling Factors for Nested Loop Based Application in Architectural Synthesis'

This section describes the experimental results of the proposed approach explained in Chapter 6 and the improvements obtained compared to recent approaches [27,31, 39]. The proposed approach has been implemented in Java and run on Intel core i5-2450M processor, 2.5 GHz with 3MB L3 cache memory and 4GB DDR3 RAM. Multiple benchmarks were chosen for testing that include CDFGs with nested loops and single loop [55, 56, 71]. *Note- the CDFGs are generated from the VHDL format of the benchmarks shown in [55, 71] however generation of CDFG is also possible from high level language (such as C).* The results are divided into two phases. a) The results obtained through proposed approach are shown in Table 7.25. As evident from Table 7.25 the solution explored by the proposed approach comprehensively meets the user defined constraints for power and execution time as well as minimizes the hybrid cost. For example, in case of Autocorrelation, the explored solution 1(+), 4(*), 1(<) and UF_1=1,UF_2=8 has execution time of 96.96us and power of 0.65mW which satisfies the given user constraints. The proposed approach also has the capability to explore multiple optimal solutions as reported in Table 7.25.

b) Moreover, Table 7.26 and Fig 7.8 and 7.9 show the comparative analysis with [27], [31] and [39] for CDFGs. As evident from Table 7.26, the QoR of proposed approach is much better than [27], [31] and [39], simultaneously exploration speed of proposed approach is multiple times higher than [27], [31] and [39] as shown in Figure 7.9. QoR is determined as eqn. (7.1):

A simple calculation with w_1=0.5 and w_2=0.5 revels that proposed approach is simultaneously able to achieve average improvement in QoR of more than 33% as well as average decrement in exploration time is more than 34% as shown in Figure 7.8 and Figure 7.9. *(Note: - For the sake of comparison, completely unrolled CDFG, is taken as an input because CDFGs were not directly handled by [27] and [31]).* This is because in [27] and [31] optimal solution were not explored as seen from Table 7.26 (QoR values). Further [27] and [31] are not capable exploring optimal unrolling factor. Besides above these approaches uses GA which have exponential time complexity. Moreover, even the approach [39] able to handle unrolling factor during exploration but required manual intervention to decide unrolling factor. Thus approach [39] not able to achieve optimal solution (composition of

datapath and loop unrolling factor) and it is also clear from the Table 7.26 that our approach achieve more than 33 % QoR in lesser exploration time (as shown in Figure 7.9). In other hand our algorithm is able to achieve optimal solution for most of the benchmark and this has been proved when we compared our results with golden solution found with exhaustive search method. Besides above these approaches uses EA which has exponential time complexity and do not have time model to predict delay of solution without physically unrolling CDFG.

Table 7.25 Results of Estimated Power and Execution Time Using Proposed Approach for CDFGs

Note: For proposed approached baseline parameters : $\varphi_1 = \varphi_2 = 0.5$, the value of ω will be linearly decreased between [0.9-0.1],value of b = 2, swarm size = 3,5 M = 100, £=10					
Bench-mark [55, 5671]		**Power**		**Execution Time**	
	Resources found	**Constraint**	**Proposed solution**	**Constraint**	**Proposed solution**
Autocor-relation (I_1=8,I_2= 8)	1(+),4(*), 1(<), UF_1=1,UF_2=8	1.25mW	0.65mW	250us	96.96us
	1(+),4(*), 1(<), UF_1=1,UF_2=4	1.25mW	0.47mW	250us	184.96us
DHMC (I_1=4,I_2= 4)	1(+),5(*), 1(<), UF_1=1,UF_2=2	3.0mW	0.82mW	600us	363.2us
	1(+),4(*), 1(<), UF_1=1,UF_2=2	3.0mW	0.81mW	600us	448.96us
FIR (I= 8)	1(+),4(*), 1(<), UF=4	0.6mW	0.47mW	40us	24.24us
	1(+),3(*), 1(<), UF=2	0.6mW	0.34mW	40us	46.24us
FFT (I= 16)	3(+),4(*), 2(-), 1(<), UF=4	1.5mW	1.44mW	500us	348.48us
	1(+),5(*), 1(-), 1(<), UF=4	1.5mW	1.51mW	500us	358.5us
Differential equation (I= 16)	1(+),6(*), 1(-), 1(<), UF=2	1.5mW	0.73mW	400us	277.4us
	1(+),6(*), 1(-), 1(<), UF=4	1.5mW	0.95mW	400us	225.6us

Table 7.26 Comparison of proposed approach with [27] , [31] and [39] in terms solution found and respective QoR

Bench-mark [55, 56, 71]	Resource combination				QoR (cost)			
	PSDSE	[27] GA	[31]WSPSO	[39]EA	PSDSE	[27]	[31]	[39]
Autocor-relation	1(+),4(*), 1(<), UF_1=1, UF_2=8	1(+),5(*), 1(<), UF_1=8, UF_2=8	1(+),4(*), 1(<), UF_1=8, UF_2=8	4(+),4(*), 4(<), UF_1=2, UF_2=2	0.15	0.38	0.37	0.28
DHMC	1(+),5(*), 1(<), UF_1=1, UF_2=2	2(+),3(*), 1(<), UF_1=4, UF_2=4	1(+),4(*), 1(<), UF_1=4, UF_2=4	4(+),12(*), 4(<), UF_1=2, UF_2=2	0.16	0.46	0.49	0.29
FIR	1(+), 4(*),1(<), UF=4	1(+), 4(*),1(<), UF=8	1(+), 3(*),1(<), UF=8	2(+), 2(*),2(<), UF=2	0.35	0.38	0.40	0.49
FFT	3(+), 4(*),2(-), 1(<), UF=4	1(+), 2(*),1(-), 1(<), UF=16	1(+), 4(*),1(-), 1(<), UF=16	4(+), 8(*),4(-), 4(<), UF=4	0.28	0.70	0.56	0.34
Diffe-rential equation	1(+), 6(*),1(-), 1(<), UF=2	1(+),3(*),1(-), 1(<), UF=16	1(+), 5(*),1(-), 1(<), UF=16	4(+), 12(*),4(-), 4(<), UF=4	0.24	0.51	0.52	0.38
Average reduction in cost w.r.t [27]= 51.4% w.r.t. [31]= 49.57% w.r.t. [39]= 33.74%								

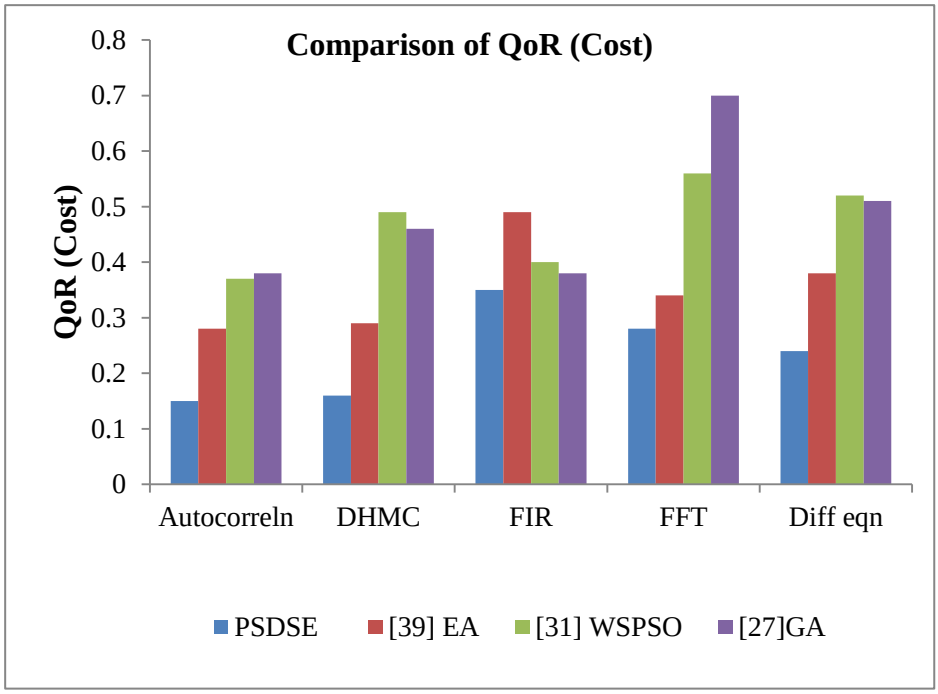


Figure 7.8 Comparison of PSDSE with [39], [31], and [27] in term of QoR

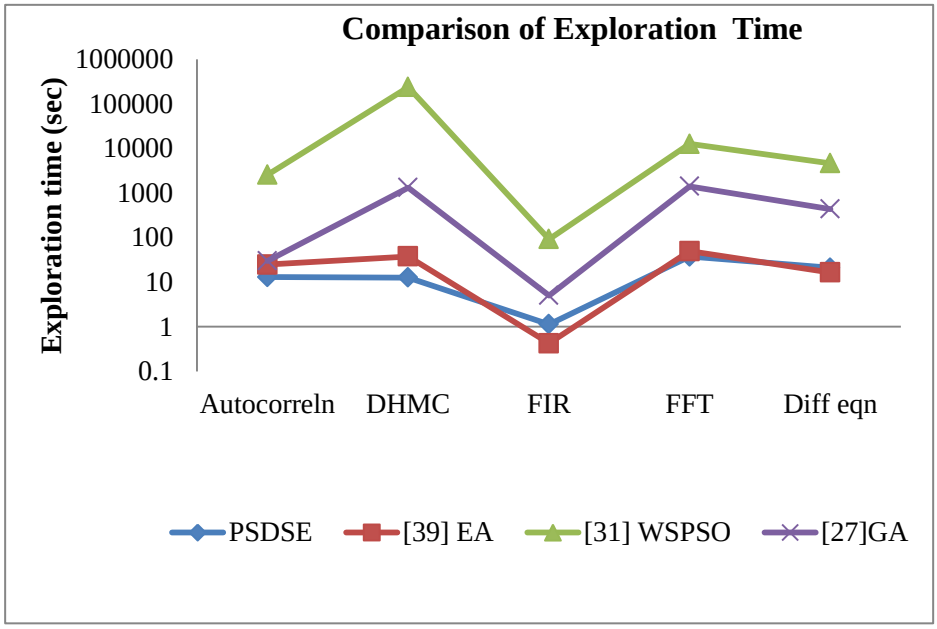


Figure 7.9 Comparison of PSDSE with [39], [31], and [27] in term of Exploration time

Chapter 8
Conclusion and Future work

8.1. Conclusion

This thesis presented different methodologies for automated multi-objective design space exploration problem in high level synthesis for application specific computing. Each methodology is unique based on class of application handled by approach and design metrics optimized by the approach. The main aim was faster exploration of good quality solutions. In order to achieve this goal, many milestones were crossed, listed as follows:

1. Proposed a novel particle swarm optimization based methodology for design space exploration of datapath during power-performance trade-off for data intensive application specific processor in high level synthesis. The proposed methodology is 49% faster and produced 10% better quality solutions as compared to previous GA approaches.
2. Proposed an automated framework for simultaneous exploration of datapath and loop unrolling factor during power performance duality in high level synthesis. The framework utilized the exploration capability of swarm intelligence to solve this twofold problem. The proposed framework produced average 28% better solutions with 94% lesser exploration time as compared to previous GA approaches.
3. Proposed a novel methodology for automated design space exploration of datapath and unrolling factor during area-delay trade-off using hyper dimensional particle swarm encoding in high level synthesis for application specific computing. As compared to previous GA approaches the proposed methodology produced average 23% better solutions with 92% faster exploration speed.
4. Proposed a novel framework for automated exploration of datapath and unrolling factors for nested loop based applications during power performance trade-off in high level synthesis. The proposed framework explored average 33% better solutions with 34% higher speed compared to previous approaches.
5. Proposed a novel power and cost model for assessment of design points.

6. Proposed an execution time estimation model for single loop based applications based on resource constraints without tediously unrolling loop.
7. Proposed a delay estimation model for nested loop based applications based on resource constraints without necessity of complete loop unrolling.
8. Proposed an adaptive end terminal perturbation algorithm to handle boundary outreach problem.
9. Presented a novel sensitivity analysis of PSO parameters for solving design space exploration problem in high level synthesis. This sensitivity analysis helps to the designer for pre-tuning the control parameters of PSO for getting high quality solution in lesser exploration time.
10. Presented an analysis of power/area, performance, control step based on unrolling factors in case of control and data intensive application.

Therefore, this thesis presented multiple design space exploration methodologies in high level synthesis, which have capability to handle data intensive applications as well as data and control intensive applications. The proposed methodologies can efficiently apply for exploration problem in HLS for any user criterion. Moreover, the execution time models (for control and data intensive application) presented in the thesis can widely applicable to determine execution time of an application in design space exploration process.

8.2. Future work

a) The area of design space exploration and high level synthesis is still required more research efforts for making high level synthesis as efficient as RTL synthesis, logic synthesis. There are various aspects which required more attention by the researcher such as handling reliability and temperature during design space exploration in high level synthesis. During handling temperature, the investigation required to handle two aspects a) reducing the peak temperature of the design which directly impact on the reliability of the design b) reduce the average temperature of the circuit which impacts on the leakage power and also impacts on the cooling and packaging of the circuit. These algorithms can be integrated with existing high level synthesis techniques for generation of optimized RTL circuits. This will allow system architects to design systems based on performance-temperature trade-offs.

b) Another aspect of the design space exploration problem is, reducing the exploration time for finding the final design architecture, and thereby accelerates the exploration process.

Further research can introduce another approach which required lesser evaluation of architectural variants to be during the exploration process for searching a high quality solution. Reducing the analysis of the architectural variants directly reduces the exploration time which in turn impacts the design time and hence will help in faster designing without compromising quality of the solution.

c) In order to improving quality of solution, further research on high level synthesis to incorporate lower level information such as gate level or physical level information during DSE for improving accuracy of evaluation models which directly effect on the quality of solution.

References

[1] Coussy, P., & Morawiec, A. (2008). High-Level Synthesis: From Algorithm to Digital Circuits. Springer, Berlin, Germany.

[2] Micheli, G. D. (1994). Synthesis and optimization of digital circuits. McGraw-Hill Higher Education. New York.

[3] Dutt, N. D., Wu, A. C. H., & Lin, S. Y. L. (1992). High-level synthesis: introduction to chip and system design, Kluwer Academic Publishers, Norwell, MA, USA.

[4] Weste, N., & Harris, D. (2010). CMOS VLSI Design: A Circuits And Systems Perspective Author: Neil Weste, David Harris, Publisher: Addison We, pp. 1-5.

[5] Mohanty, S. P., Ranganathan, N., Kougianos, E., & Patra, P. (2008). Low-power high-level synthesis for nanoscale CMOS circuits. Springer Science & Business Media.

[6] ITRS. International Technology Roadmap for Semiconductors, 2013. http://www.itrs.net/Links/2013ITRS/Home2013.htm.

[7] Parker, A., Thomas, D., Siewiorek, D., Barbacci, M., Hafer, L., Leive, G., & Kim, J. (1979, June). The CMU design automation system: An example of automated data path design. In Proceedings of the 16th Design Automation Conference, pp. 73-80.

[8] Director, S. W., Parker, A. C., Siewiorek, D. P., & Thomas Jr, D. (1981). A design methodology and computer aids for digital VLSI systems. IEEE Transactions on Circuits and Systems, 28(7), pp. 634-645.

[9] Paulin, P. G., Knight, J. P., & Girczyc, E. F. (1986, July). HAL: a multi-paradigm approach to automatic data path synthesis. In Proceedings of the 23rd ACM/IEEE Design Automation Conference, pp. 263-270.

[10] De Micheli, G., & Ku, D. C. (1988, June). HERCULES-a system for high-level synthesis. In Proceedings of the 25th ACM/IEEE Design Automation Conference, pp. 483-488.

[11] Yassa, F. F., Jasica, J. R., Hartley, R. I., & Noujaim, S. E. (1987). A silicon compiler for digital signal processing: Methodology, implementation, and applications. Proceedings of the IEEE, 75(9), pp. 1272-1282.

[12] Coussy, P., Gajski, D. D., Meredith, M., & Takach, A. (2009). An introduction to high-level synthesis. IEEE Design & Test of Computers, (4), pp. 8-17.

[13] Parker, A. C., Pizarro, J. T., & Mlinar, M. (1986, July). MAHA: a program for datapath synthesis. In Proceedings of the 23rd ACM/IEEE Design Automation Conference pp. 461-466.

[14] Paulin, P. G., & Knight, J. P. (1989). Force-directed scheduling for the behavioral synthesis of ASICs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 8(6), pp. 661-679.

[15] Knapp, D. W. (1996). Behavioral synthesis: digital system design using the synopsys behavioral compiler. Prentice-Hall, Inc.

[16] Elliott, J. P. (1999). Understanding behavioral synthesis: A practical guide to high-level design. Springer Science & Business Media.

[17] Kress, R., Pyttel, A., & Sedlmeier, A. (2000). FPGA-based prototyping for product definition. In Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, Springer Berlin Heidelberg, pp. 78-86.

[18] Gupta, S., Dutt, N., Gupta, R., & Nicolau, A. (2004, February). Loop shifting and compaction for the high-level synthesis of designs with complex control flow. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2004. pp. 114-119.

[19] Villarreal, J., Park, A., Najjar, W., & Halstead, R. (2010, May). Designing modular hardware accelerators in C with ROCCC 2.0. In Proceedings of the 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 127-134.

[20] Coussy, P., Chavet, C., Bomel, P., Heller, D., Senn, E., & Martin, E. (2008). GAUT: A high-level synthesis tool for DSP applications. In High-Level Synthesis From Algorithm to Digital Circuits, Springer Netherlands, pp. 147-169.

[21] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., & Czajkowski, T. (2011, February). LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays, pp. 33-36.

[22] Pilato, C., Loiacono, D., Ferrandi, F., Lanzi, P. L., & Sciuto, D. (2008, June). High-level synthesis with multi-objective genetic algorithm: A comparative encoding analysis. In Proceedings of the IEEE World Congress on Computational Intelligence. pp. 3334-3341.

[23] Ferrandi, F., Lanzi, P. L., Loiacono, D., Pilato, C., & Sciuto, D. (2008, April). A multi-objective genetic algorithm for design space exploration in high-level synthesis. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2008. ISVLSI'08. pp. 417-422.

[24] Yang, H., Wang, C., & Du, N. (2012). High Level Synthesis using Learning Automata Genetic Algorithm. Journal of Computers, 7(10), pp. 2534-2541.

[25] Torbey, E., & Knight, J. (1998, May). High-level synthesis of digital circuits using genetic algorithms. In Proceedings of the IEEE International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp. 224-229.

[26] Torbey, E., & Knight, J. (1998, August). Performing scheduling and storage optimization simultaneously using genetic algorithms. In Proceedings of the Midwest Symposium on Circuits and Systems, pp. 284-287.

[27] Krishnan, V., & Katkoori, S. (2006). A genetic algorithm for the design space exploration of datapaths during high-level synthesis. IEEE Transactions on Evolutionary Computation,10(3), pp. 213-229.

[28] Mandal, C., Chakrabarti, P. P., & Ghose, S. (2000). GABIND: a GA approach to allocation and binding for the high-level synthesis of data paths. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 8(6), pp. 747-750.

[29] Sengupta, A., Sedaghat, R., & Sarkar, P. (2012). A multi structure genetic algorithm for integrated design space exploration of scheduling and allocation in high level synthesis for DSP kernels, Swarm and Evolutionary Computation, 7, pp. 35-46.

[30] Palermo, G., Silvano, C., & Zaccaria, V. (2008, September). Discrete particle swarm optimization for multi-objective design space exploration. In Proceedings of the 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools, pp. 641-644.

[31] Ram, D. S., Bhuvaneswari, M. C., & Prabhu, S. S. (2012). A novel framework for applying multi objective GA and PSO based approaches for simultaneous area, delay, and power optimization in high level synthesis of datapaths. VLSI design, 2012, pages-12.

[32] Sengupta, A., Sedaghat, R., & Zeng, Z. (2010). A high level synthesis design flow with a novel approach for efficient design space exploration in case of multi-parametric optimization objective. Microelectronics Reliability, 50(3), pp. 424-437.

[33] Liu, H. Y., & Carloni, L. P. (2013, May). On learning-based methods for design-space exploration with high-level synthesis. In Proceedings of the 50th Annual Design Automation Conference, pp. 1-7.

[34] McFarland, M. C. (1986, July). Using bottom-up design techniques in the synthesis of digital hardware from abstract behavioral descriptions. In Proceedings of the 23rd ACM/IEEE Design Automation Conference, pp. 474-480.

[35] Sengupta, A., Sedaghat, R., & Zeng, Z. (2011). Rapid design space exploration by hybrid fuzzy search approach for optimal architecture determination of multi objective computing systems. Microelectronics Reliability, 51(2), pp. 502-512.

[36] Haubelt, C., & Teich, J. (2003, January). Accelerating design space exploration using pareto-front arithmetics. In Proceedings of the Asia and South Pacific Design Automation Conference, pp. 525-531.

[37] Wang, G., Gong, W., DeRenzi, B., & Kastner, R. (2006, July). Design space exploration using time and resource duality with the ant colony optimization. In Proceedings of the 43rd annual Design Automation Conference, pp. 451-454.

[38] Schafer, B. C., & Wakabayashi, K. (2010). Design space exploration acceleration through operation clustering. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 29(1), pp. 153-157.

[39] Holzer, M., Knerr, B., & Rupp, M. (2007, July). Design space exploration with evolutionary multi-objective optimisation. In Proceedings of the International Symposium on Industrial Embedded Systems, pp. 126-133.

[40] Zhang, Z., Fan, Y., Jiang, W., Han, G., Yang, C., & Cong, J. (2008). AutoPilot: A platform-based ESL synthesis system. In High-Level Synthesis: From Algorithm to Digital Circuits, Springer Netherlands, pp. 99-112.

[41] Haubelt, C., Schlichter, T., Keinert, J., & Meredith, M. (2008, June). SystemCoDesigner: automatic design space exploration and rapid prototyping from behavioral models. In Proceedings of the 45th annual Design Automation Conference, pp. 580-585.

[42] Sengupta, A., & Sedaghat, R. (2011, March). Integrated scheduling, allocation and binding in High Level Synthesis using multi structure genetic algorithm based design space exploration. In Proceedings of the 12th International Symposium on Quality Electronic Design (ISQED), pp. 1-9.

[43] Bollaert, T. (2008). Catapult synthesis: a practical introduction to interactive C synthesis. In High-Level Synthesis: From Algorithm to Digital Circuits, Springer Netherlands, pp. 29-52.

[44] Meredith, M. (2008). High-level SystemC synthesis with forte's cynthesizer. InHigh-Level Synthesis: From Algorithm to Digital Circuits, Springer Netherlands, pp. 75-97.

[45] Wakabayashi, K., & Schafer, B. C. (2008). "All-in-C" Behavioral Synthesis and Verification with CyberWorkBench. In High-Level Synthesis: From Algorithm to Digital Circuits, Springer Netherlands, pp. 113-127.

[46] Feist, T. (2012). Vivado design suite. White Paper, Xilinx

[47] Cadence C-to-Silicon White Paper, 2008 http://www.cadence.com/rl/resources/technical_papers/c_to_silicon_tp.pdf

[48] Kennedy, J. (1995). Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, pp. 1942-1948.

[49] Engelbrecht A.P., (2005) "fundamental of computational swarm intelligence", John Wiley and sons limited, England.

[50] Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information processing letters, 85(6), pp. 317-325.

[51] Ratnaweera, A., Halgamuge, S., & Watson, H. C. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Transactions on Evolutionary Computation, 8(3), pp. 240-255.

[52] Reynders, N., & Dehaene, W. (2011, November). A 190mV supply, 10MHz, 90nm CMOS, pipelined sub-threshold adder using variation-resilient circuit techniques. In Proceedings of the IEEE Asian Solid State Circuits Conference (A-SSCC), pp. 113-116.

[53] University of California, Santa Barbara, Express Benchmarks: http://express.ece.ucsb.edu/benchmark/.

[54] Kumar, A. K., Somasundareswari, D., Duraisamy, V., & Pradeepkumar, M. (2010). Low power multiplier design using complementary pass-transistor asynchronous adiabatic logic. International Journal on Computer Science and Engineering, 2(07), pp. 2291-2297.

[55] Texas Instruments: 'Benchmarks - C674x Low Power DSP - TI.Com', Aug 2014, http://www.ti.com/lsds/ti/dsp/c6000dsp/c674x/benchmarks.page.

[56] Namballa, R. K. (2003). CHESS: A tool for CDFG extraction and high-level synthesis of VLSI systems (Doctoral dissertation, University of South Florida).

[57] Elgamel, M., & Bayoumi, M. A. (2002). On low power high level synthesis using genetic algorithms. In Proceedings of the 9th International Conference on Electronics, Circuits and Systems, pp. 725-728.

[58] Nikara, J., Takola, J., Akopian, D., & Saarinen, J. (2001, May). Pipeline architecture for DCT/IDCT. In Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 902-905.

[59] Sengupta, A., Sedaghat, R., Sarkar, P., & Sehgal, S. (2011, May). Integrated scheduling, allocation and binding in High Level Synthesis for performance-area tradeoff of digital media applications. In Proceedings of the 24th Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 533-537.

[60] Pilato, C., Loiacono, D., Ferrandi, F., Lanzi, P. L., & Sciuto, D. (2008, June). High-level synthesis with multi-objective genetic algorithm: A comparative encoding analysis. In Proceedings of the IEEE Congress on Evolutionary Computation, IEEE World Congress on Computational Intelligence. pp. 3334-3341.

[61] Chang, Jui-Ming, and Massoud P. (1997) Energy minimization using multiple supply voltages, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 5(4), pp. 436-443.

[62] Lee, Y.T.; Park, I.C.; Kyung, C.M., (April 1993) Design of compact static CMOS carry look-ahead adder using recursive output property, Electronics Letters, 29(9), pp.794-796.

[63] Salman, A., Ahmad I., and Sabah Al-Madani. (2002) Particle swarm optimization for task assignment problem. Microprocessors and Microsystems, 26(8), pp. 363-371.

[64] Tasgetiren, M. Fatih, Yun-Chia Liang, Mehmet Sevkli, and Gunes Gencyilmaz. (2007) A particle swarm optimization algorithm for make span and total flow time minimization in the permutation flow shop sequencing problem." European Journal of Operational Research 177(3) pp. 1930-1947.

[65] Goldberg, David E., and Robert Lingle. (1985) Alleles, loci, and the traveling salesman problem, In Proceedings of the first international conference on genetic algorithms and their applications, pp. 154-159.

[66] Kirkpatrick, S. (1984). Optimization by simulated annealing: Quantitative studies. Journal of statistical physics, 34(5-6), pp. 975-986.

[67] Boeringer, D. W., & Werner, D. H. (2004). Particle swarm optimization versus genetic algorithms for phased array synthesis. IEEE Transactions on Antennas and Propagation, 52(3), pp. 771-779.

[68] Jarboui, B., Damak, N., Siarry, P., & Rebai, A. (2008). A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation, 195(1), pp. 299-308.

[69] Zheng, Y. L., Ma, L. H., Zhang, L. Y., & Qian, J. X. (2003, November). On the convergence analysis and parameter selection in particle swarm optimization. In Proceeding of the IEEE International Conference on Machine Learning and Cybernetics, pp. 1802-1807.

[70] Wang, J., and Su, B. (1998, May). Software pipelining of nested loops for real-time DSP applications. In Proceedings of the Acoustics, Speech and Signal Processing, pp. 3065-3068.

[71] High-Level Synthesis Benchmark Circuits : https://filebox.ece.vt.edu/~mhsiao/hlsyn.html

[72] Sengupta, A., and Bhadauria, S. (2014). Exploration of multi-objective tradeoff during high level synthesis using bacterial chemotaxis and dispersal. Procedia Computer Science, 35, pp. 63-72.

[73] K. M. Passino,(2002) Biomimicry of Bacterial Foraging for Distributed Optimization and Control, IEEE Control Systems Magazine, pp. 52-67.

[74] Lee, Y. T., Park, I. C., & Kyung, C. M. (1993). Design of compact static CMOS carry look-ahead adder using recursive output property. Electronics Letters, 29(9), pp. 794-796.

[75] Paulin, P.G., Knight, J.P. (1989) Force directed scheduling for the behavioral synthesis of ASICs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 8(6), pp.661–679.

[76] Paulin, P.G., Knight, J.P. (1999) Algorithms for high-level synthesis. IEEE Design and Test of Computers 6(6), pp.18–31.

[77] Papachristou, C.A., Konuk, H. (1990) A linear program driven scheduling and allocation method.In: Proceedings of the 27th ACM/IEEE Design Automation Conference, pp. 77–83.

[78] Kollig, P., Al-Hashimi, B.M. (1997) Simultaneous scheduling, allocation and binding in high level synthesis. IEE Electronics Letters 33(18), pp.1516–1518.

[79] Kopuri, S., Mansouri, N. (2004) Enhancing scheduling solutions through ant colony optimization. In: Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp. 257–260.

[80] Walker, R.A., Chaudhuri, S. (1995) Introduction to the scheduling problems. IEEE Design and Test of Computers 12(2), pp.60–69.

[81] Heijligers, M.J.M., Cluitmans, L.J.M., Jess, J.A.G. (1991) High-level synthesis scheduling and allocation using genetic algorithms. In: Proceedings of the 28th Design Automation Conference, pp. 61–66.

[82] Mohanty, S.P. (2003) Energy and Transient Power Minimization During Behavioral Synthesis. Ph.D. thesis, University of South Florida.

[83] Mohanty, S.P., Rangnathan, N., Chappidi, S.K. (2003) An ILP-based scheduling scheme for energy efficient high performance datapath synthesis. In: Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp. 313–316.

[84] Abdel-Kader, R.F. (2005) Resource-constrained loop scheduling in high-level synthesis. In: Proceedings of the 43rd ACM Annual Southeast Regional Conference, pp. 195–200.

[85] Narasimhan, M., Ramanujam, J. (2000) On lower bounds for scheduling problems in high-level synthesis. In: Proceedings of the Design Automation Conference (DAC), pp. 546–551.

[86] Gerez, S.H. (2004) Algorithms for VLSI Design Automation. Wiley

[87] Springer, D.L., Thomas, D.E. (1994) Exploiting the special structure of conflict and compatibility graphs in high-level synthesis. IEEE Transactions on CAD of Integrated Circuits and Systems 13(7), pp.843–856.

[88] Paulin, P.G., Knight, J.P. (1989) Scheduling and binding algorithms for high-level synthesis. In: Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 1–6.

[89] Raje, S., Bergamaschi, R.A. (1997) Generalized resource sharing. In: Proceedings of the Design Automation Conference (DAC), pp. 326–332.

[90] Mohanty, S.P., Velagapudi, R., Kougianos, E. (2006) Physical-aware simulated annealing optimization of gate leakage in nanoscale datapath circuits. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE), pp. 1191–1196.

[91] Al-Mouhamed, M., & Al-Massarani, A. (2000). Scheduling optimization through iterative refinement. Journal of systems architecture, 46(10), pp. 851-871.

[92] Crop, J., Fairbanks, S., Pawlowski, R., & Chiang, P. (2010, April). 150mV sub-threshold Asynchronous multiplier for low-power sensor applications. In Proceeding of the IEEE International Symposium on VLSI Design Automation and Test (VLSI-DAT), pp. 254-257.

[93] Eberhart, R. C., & Shi, Y. (2001). Particle swarm optimization: developments, applications and resources. In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on Vol. 1, pp. 81-86.

[94] J.H. Holland (1975), Adaptation in Natural and Artificial Systems, Univ. of Michigan Press, Ann Arbor, Mich.

[95] Bäck, T., & Schwefel, H. P. (1993). An overview of evolutionary algorithms for parameter optimization. Evolutionary computation, 1(1), pp.1-23.

[96] Hassan, R., Cohanim, B., De Weck, O., & Venter, G. (2005). A comparison of particle swarm optimization and the genetic algorithm. InProceedings of the 1st AIAA multidisciplinary design optimization specialist conference pp. 1-13.

[97] Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, 8(2), pp. 173-195.

[98] Eglese, R. W. (1990): "Simulated annealing: a tool for operational research." European journal of operational research 46.3 pp. 271-281.

[99] E. Aarts, J. Korst (1989), Simulated annealing and boltzmann machines: A stochastic approach to combinatorial optimization and neural computing, Wiley, Chichester

[100] S. Kirkpatrick Jr., C.D. Gelatt, M.P. Vecchi (1983), Optimization by simulated annealing, Science, 220 pp. 671‑680

[101] P.J.M. Laarhoven, E.H.L. Aarts (1987) Simulated annealing: Theory and applications Reidel, Dordrecht

[102] Dorigo, M., & Birattari, M. (2010). Ant colony optimization. In Encyclopedia of machine learning, Springer US. pp. 36-39.

[103] Dorigo, M., Birattari, M., & Stützle, T. (2006). Ant colony optimization: Computational Intelligence Magazine, IEEE, 1(4), pp. 28-39.

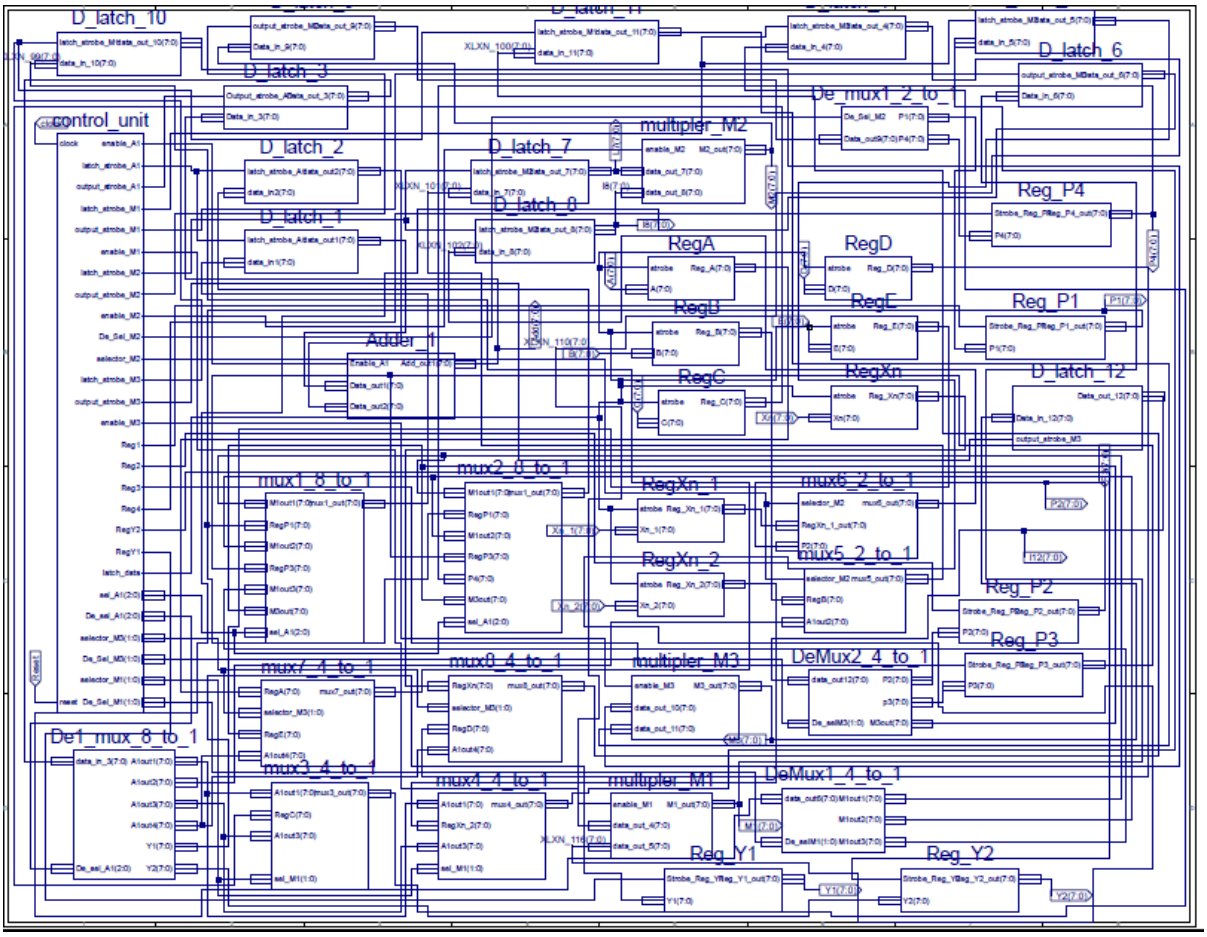
[104] Dorigo, M., & Blum, C. (2005). Ant colony optimization theory: A survey: Theoretical computer science, 344(2), pp. 243-278.

[105] Kennedy, J., Kennedy, J. F., Eberhart, R. C., & Shi, Y. (2001). Swarm intelligence. Morgan Kaufmann.

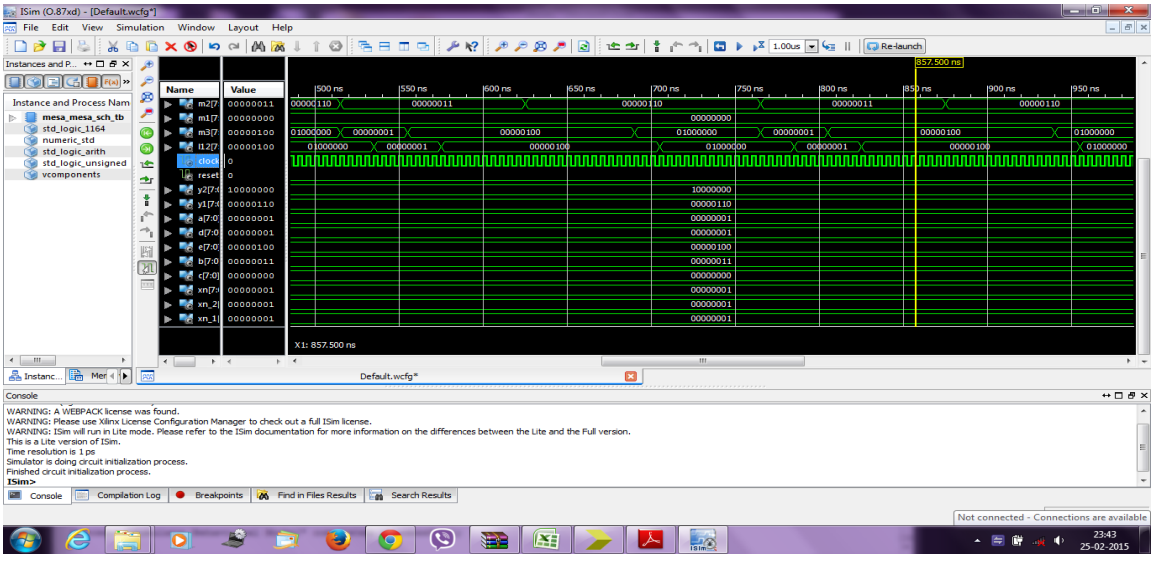
[106] Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In Evolutionary Computation, 2000. Proceedings of the 2000 Congress on (Vol. 1, pp. 84-88). IEEE.

[107] Eberhart, R. C., & Shi, Y. (1998). Comparison between genetic algorithms and particle swarm optimization. In Evolutionary Programming VII (pp. 611-616). Springer Berlin Heidelberg.

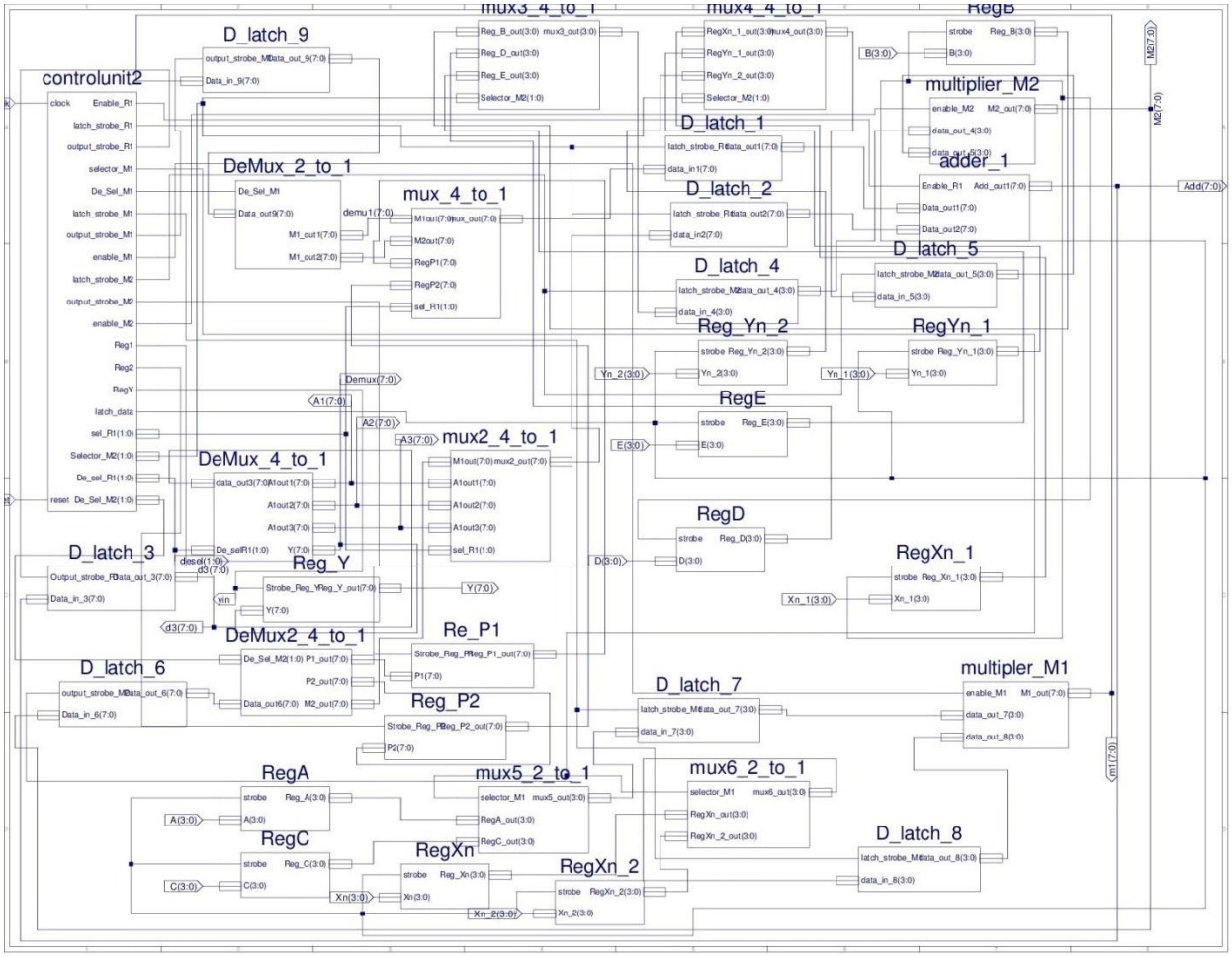
Appendix A



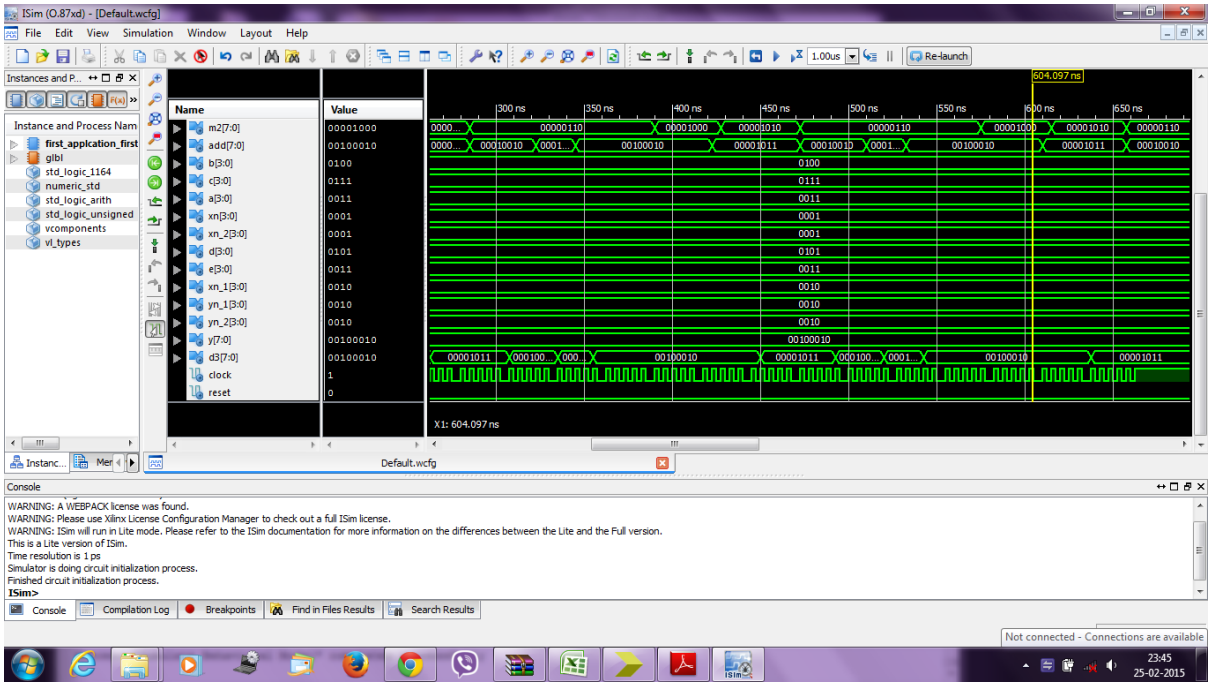
Schematic view of the designed MESA Horner in Xilinx ISE tool



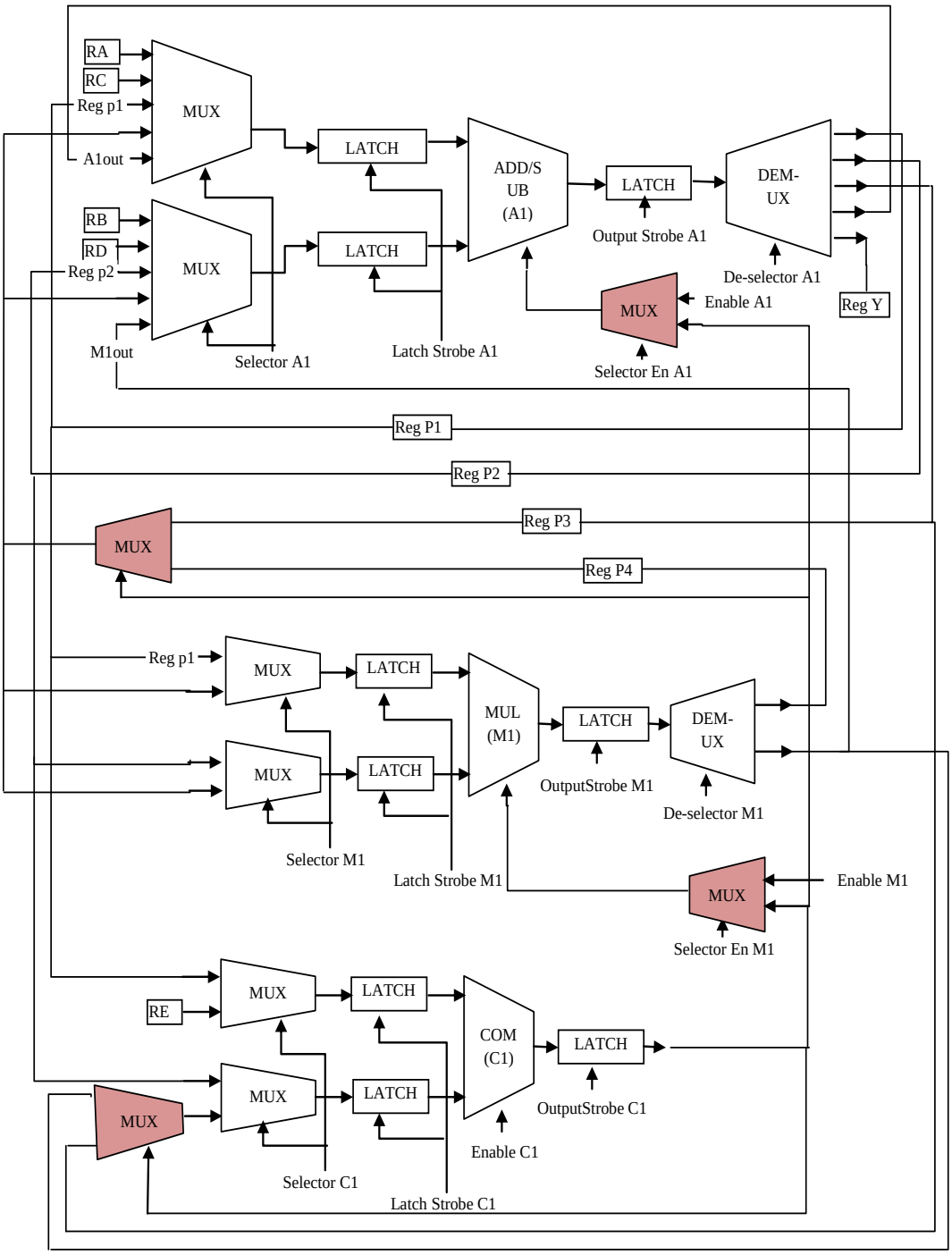
Simulation result of MESA Horner



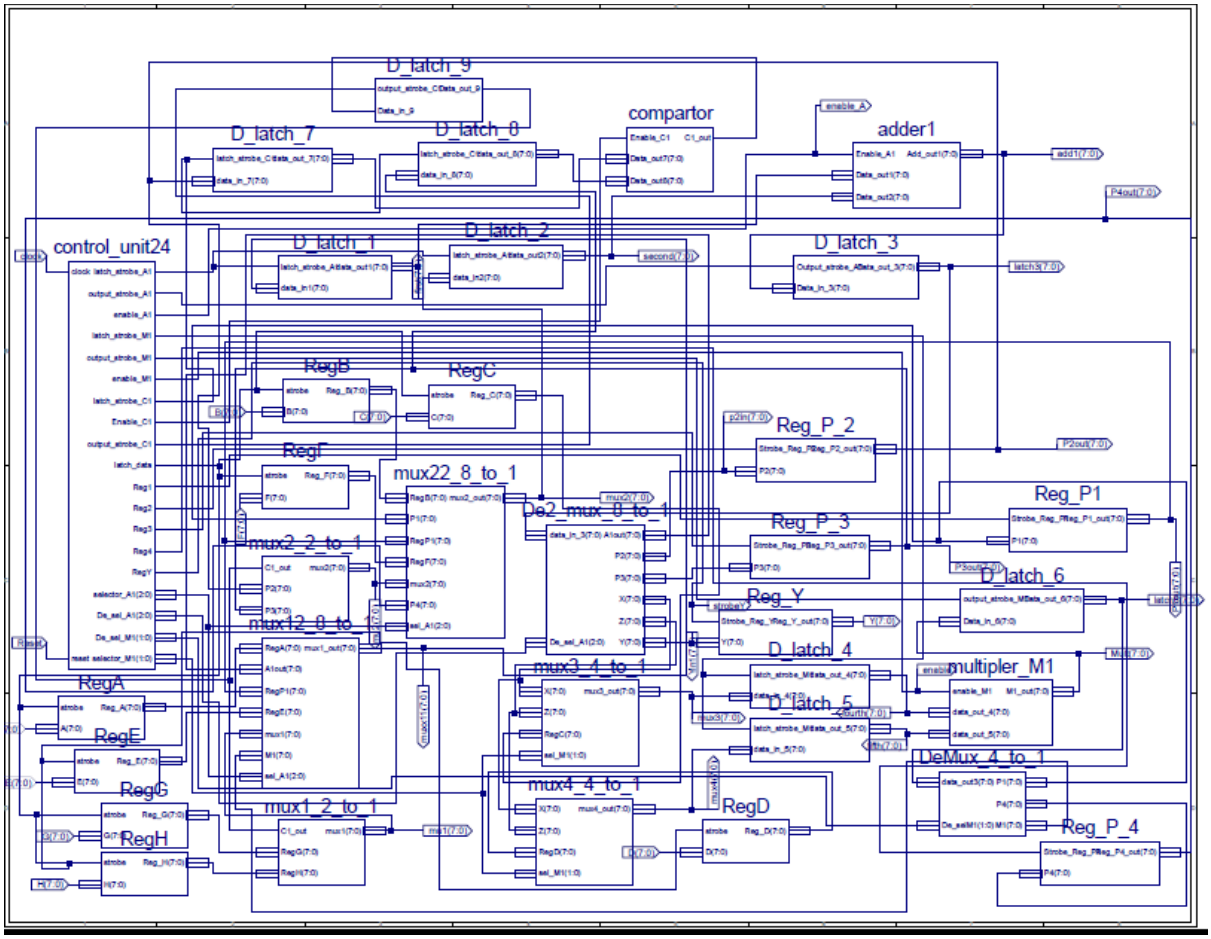
RTL design of IIR Butterworth in Xilinx ISE tool



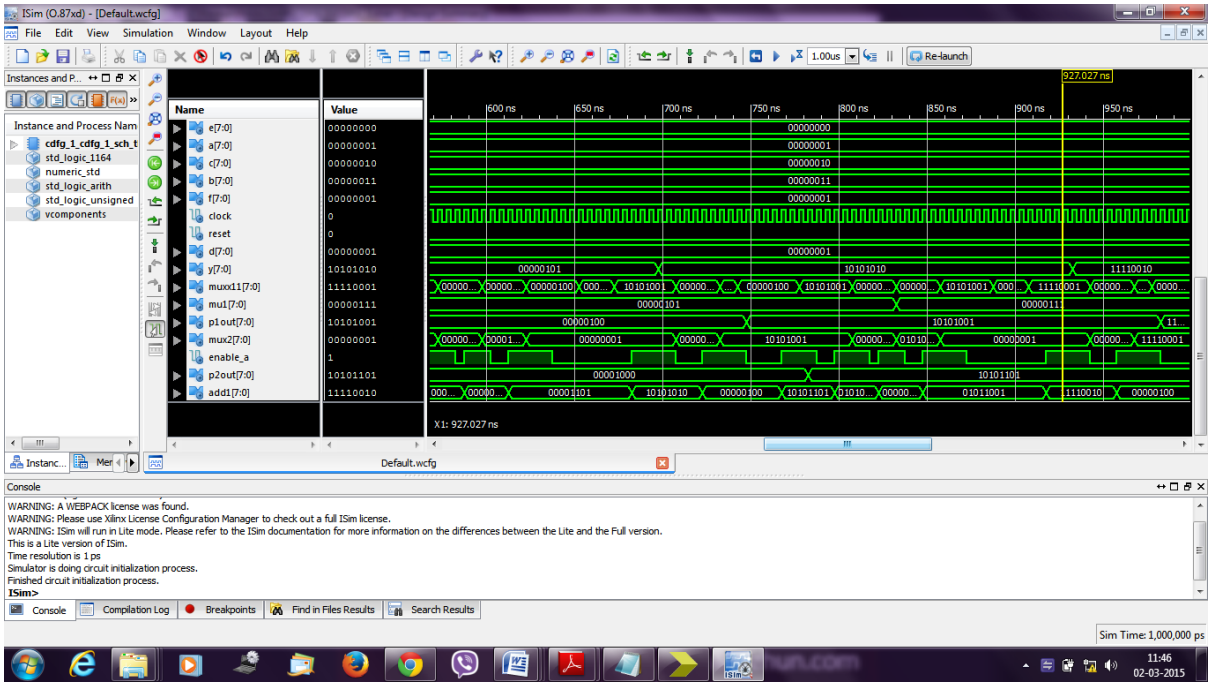
Simulation result of IIR Butterworth



Data path circuit for scheduled CDFG1 with 1(*), 1(+), 1(<) resource configuration presented in chapter 3 Figure 3.9



RTL Diagram of the CDFG1 presented in chapter 3 (datapath given in previous figure)

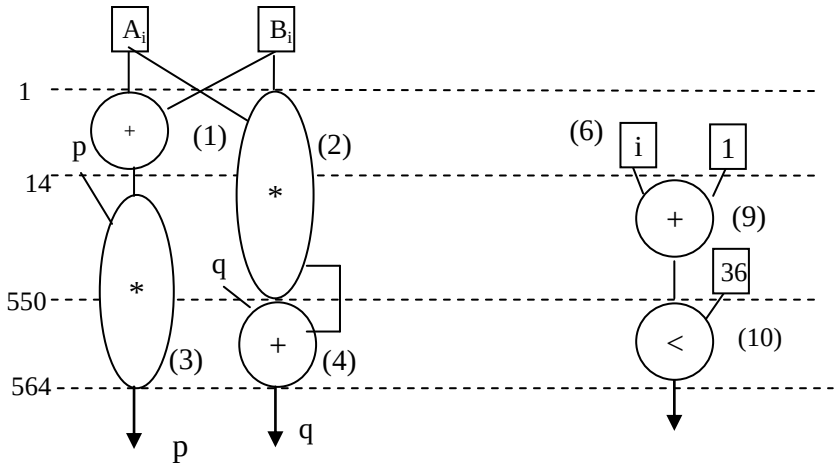


Simulation result of CDFG1

CDFG of Testcase used in Chapter 4 and Chapter 5

```
For(i=0; i<36; i++) {
        P = P * (Ai + Bi)
        q = q + (Ai * Bi)
}
```

(a) 'C' code of original loop



(b) Control and data flow graph of Testcase

Detail of Testcase used in the thesis