# Implementation of Integrated Design Space Exploration of Scheduling, Allocation and Binding in High Level Synthesis using Multi Structure Genetic Algorithm

By

**Michael Gebremariam**

Bachelor of Science in

Electrical and Electronics Engineering

Addis Ababa University

Addis Ababa, Ethiopia, 2000

A project report

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2010

# Author's Declaration

I hereby declare that I am the sole author of this project report.

I authorize Ryerson University to lend this project report to other institutions or individuals for the purpose of scholarly research.

* Signature

Michael Gebremariam

I further authorize Ryerson University to reproduce this project report by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

* Signature

Michael Gebremariam

# ABSTRACT


**Implementation of Integrated Design Space Exploration of Scheduling, Allocation and Binding in High Level Synthesis using Multi Structure Genetic Algorithm**

Michael Gebremariam, Master of Engineering, 2010

Optimization Problems Research and Application Laboratory (OPR-AL)
Electrical and Computer Engineering Department, Ryerson University

Project Directed By: Dr. Reza Sedaghat,
Department of Electrical and Computer Engineering, Ryerson University

The objective of this project is to develop a software tool which assists in comparison of a work known as "M-GenESys: Multi Structure Genetic Algorithm based Design Space Exploration System for Integrated Scheduling, Allocation and Binding in High Level Synthesis" with another well established GA approach known as "A Genetic Algorithm for the Design Space Exploration of Data paths During High-Level Synthesis".

Two sets of Software are developed based on both approaches using Microsoft visual 2005 C# language. The C# language is an object-oriented language that is aimed at enabling programmers to quickly develop a wide range of applications on the Microsoft .NET platform. The goal of C# and the .NET platform is to shorten development time by freeing the developer from worrying about several low level plumbing issues such as memory management, type safety issues, building low level libraries, array bounds checking, etc. thus allowing developers to actually spend their time and energy working on the application and business logic .

# Acknowledgement

I would like to thank Dr. Reza Sedaghat and Anirban Sengupta for their guidance and support.

I am also grateful to my wife, my daughter and my parents for the sacrifice and constant source of love and motivation they gave me throughout my life and studies, particularly in times of hardships and difficulty.

I am also very thankful to my brothers and sisters, who provided me with encouraging words to accomplish my goals.

# Table of Contents

# List of Figures

# Nomenclature

| | |
|---|---|
| **A** | Total Area of the resources |
| **$R_i$** | The resources available for system designing |
| ASAP | As soon as possible |
| ALAP | As late as possible |
| **n** | Functional resources |
| **L** | Latency of execution |
| **$T_c$** | Cycle time of execution |
| **N** | Number of data elements to be processed |
| **$T_p$** | Time period of the clock |
| **$p_c$** | Power consumed per area unit resource at a particular frequency |
| **FU** | Functional units |
| **$P_{constraint}$** | Power constraint |
| **$T_{constraint}$** | Execution Time constraint |
| **P** | Total power consumption |
| **$T_{exe}$** | Total execution time |
| **DFG** | Data Flow Graph |
| **GA** | Genetic Algorithm |

# Chapter 1

# Introduction

Advances in VLSI technology have made high-level synthesis process more complicated. Recent advances in high-level synthesis have to find the most effective approach to deal with the complexity of today's increasing System-on-Chip (SoC) design demand. High-level synthesis starts from an abstract behavioral description and automatically generates a structural description of a digital circuit that realizes the desired behavior. High-level synthesis process involves three interdependent and NP-complete optimization problems:

(i) Operation scheduling

(ii) Resource allocation

(iii)Synthesis

Evolutionary algorithms have been effectively employed to high level synthesis in existence of conflicting design objectives for finding good tradeoffs in the design space exploration.

Because of the diversity of the parameters, and also due to the variety in architecture for implementation; the design and development of systems with diverse performance optimization objective requires broad analysis and assessment of the design space. The system designer has to follow a divide-and-conquer approach to tackle the large and complex design space problems. The problems have to be scaled down into a set of manageable and realistic design sets in order to meet the system performance objectives and functionality. Design space architecture can have numerous design and implementation alternatives based on the parameters of optimization. For this reason, selection of the optimal architecture from the design space which satisfies all the performance objectives is vital, especially in recent generation of System-on-Chip (SoC) designs

[7]. As it is always possible to implement different functions of a system on different hardware components, the architecture design space has become more complex to analyze [6]. In the case of high level synthesis, performing design space exploration to choose the best candidate architecture by concurrently satisfying many operating constraints and optimization parameters is considered the most important stage in the whole design flow. Since the design space is huge and complex, there needs to be an efficient way to explore the best candidate architecture for the system design based on the application to be executed. The method for exploration of the best candidate micro architecture should not only be less in terms of complexity factor and time but also explore the variant in an efficient way meeting all the specifications provided. The process of high level synthesis design is very complicated and descriptive and is usually performed by system architects. Depending on the application, the process of defining the problem, performing design space exploration and the other steps required for its successful accomplishment are very time consuming. Modern high level synthesis design flow should be multi-parametric optimized in terms of area occupied, execution time and power consumption. Furthermore, recent advancements in areas of communications and multimedia have led to the growth of a wide array of applications requiring huge data processing at minimal power expense. Such data hungry applications demand satisfactory performance with power efficient hardware solutions. Hardware solutions should satisfy multiple contradictory performance parameters such as power consumption and time of execution. Since the selection process for the best design architecture is complex, an efficient approach to explore the design space for selecting the best design option is needed.

# Chapter 2

# Theory

VLSI digital circuit design has becomes more and more complex. In the early 2000's, VLSI technology was in the range of ten millions logics per chip. But now VLSI technology has reached densities of billion transistor logics per chip and this trend will be increasing at Moor's law rate for the next decades. For such complexity, it has become a difficult task to write RTL description for the system and would be even more challenging for system verification.

The design space exploration embraces a multitude of different optimization scenarios on varying abstraction levels. Typical parameters of this exploration are timing, power, and area. Additionally, how to limit this parameter and how to find or approximate Pareto Points in the Design Space quickly falls in the realm of design space exploration.

High-Level-Synthesis tools are developed to solve all these optimization problems. High-Level-Synthesis is a translation process from behavioral description into RTL description in an automatic way. HLS is a complex problem. It is either partitioned into several sub-tasks and executes the tasks one by one, or partitioned into a sequence of transformation steps each of which make a small change to the intermediate result of the earlier step. The former one is the general approach to implement HLS. The later one is called transformational approach. The general approach is easier to implement than the transformation approach. Most HLS tools use the general approach to synthesize.

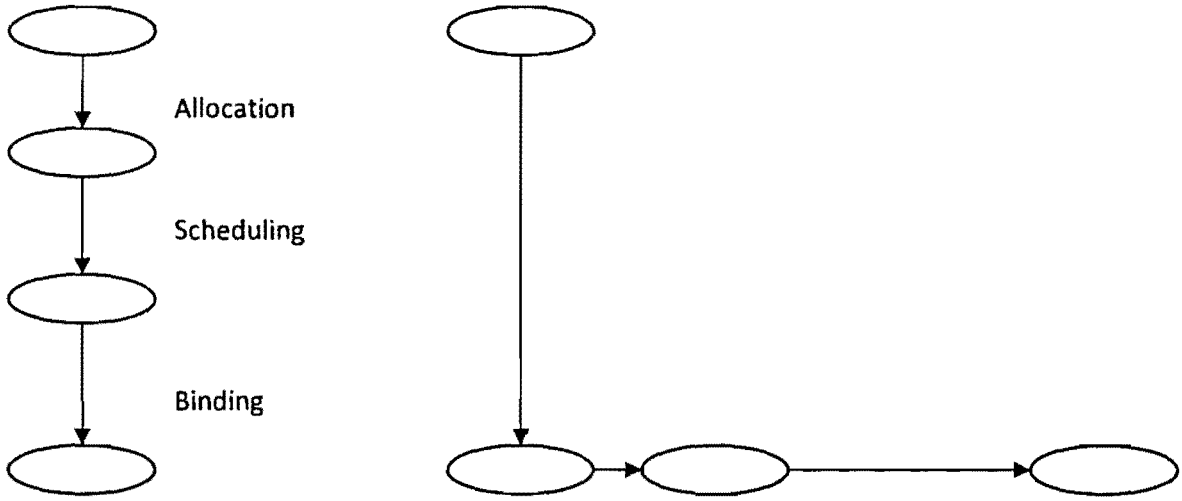



Figure 1- General approach vs. transformational approach

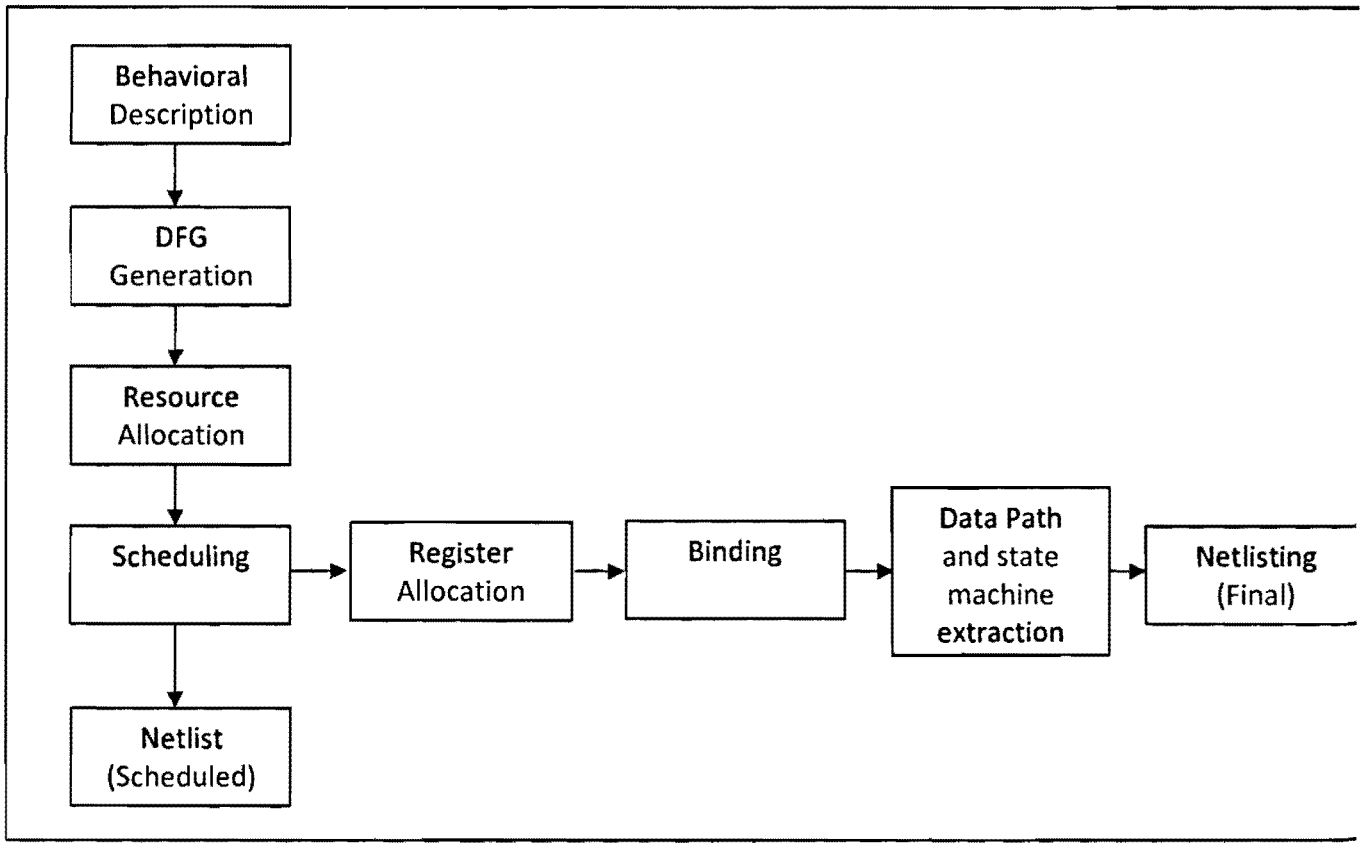



Figure 2- General steps in the Behavioral Synthesis process

### *2.1 Data flow graph*

The first step in High-Level-Synthesis is to translate the behavioral description into an intermediate representation as shown in Figure 3.1 and Figure 3.2. A Data Flow Graph (DFG) is commonly used as intermediate representation to capture the behavior such as data dependence, control structure etc. Figure 3.2 shows an example of DFG. At this step, data-based transformations and control-based transformations are used to optimize the intermediate representation. Data-based transformations include tree-height reduction, constant and variable propagation, common sub-expression elimination, dead-code elimination, operator-strength reduction, code motion etc. And control-based transformations include model expansion, conditional expansion, loop expansion, block-level transformations etc [24].

(A x (B x C) - D x (E x F)) + (D x (E x F) - G x (H + I) )

Figure 3.1 - Behavioral Specification

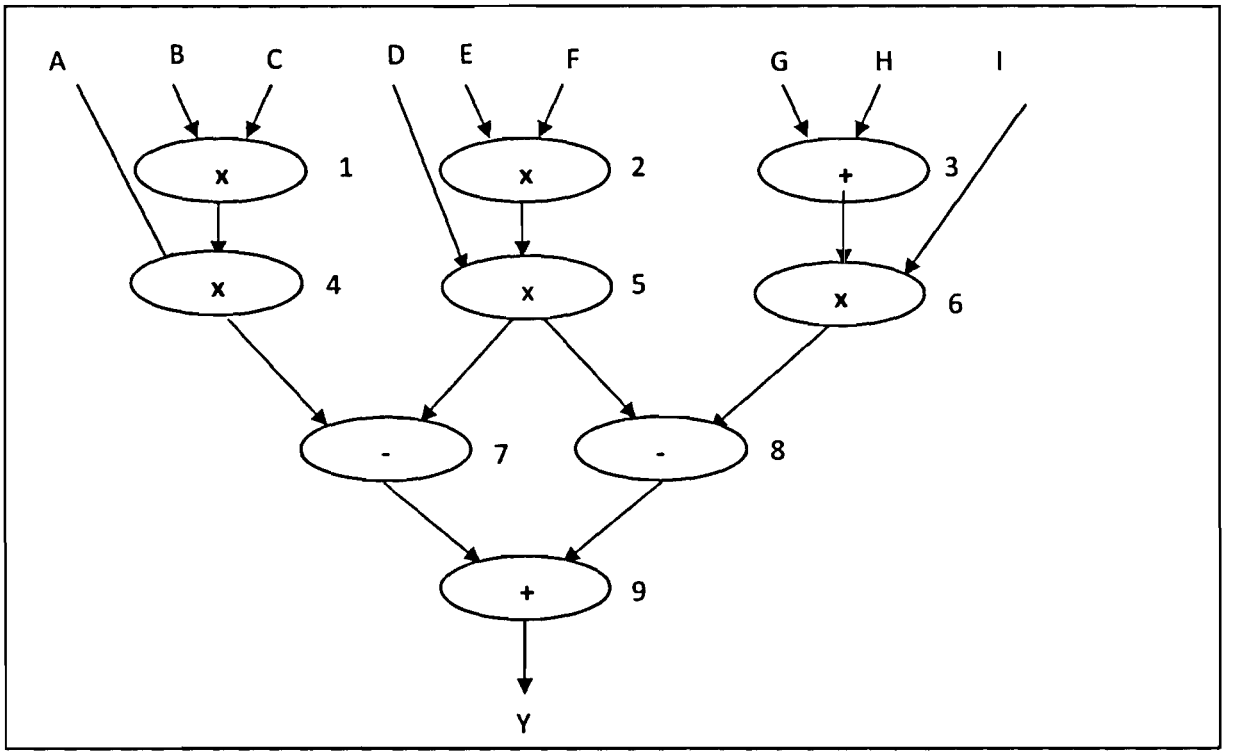



Figure 3.2 - dataflow graph

**2.2 Resource allocation**

It is to decide how many and which kind of resources can be used in the design. These are the resource constraints. For example, in Figure 3.2, 3 multiplier, 1 adder and 2 subractor could be allocated. If the resource allocation is not sufficient, the operation can't be scheduled. Also if there are less resources and the cycle time constraint is small the scheduling will fail. This implies that allocation and scheduling are correlated. There may be some reiteration among them as well.

***2.3 Scheduling***

It is the most important step of high level synthesis. The quality of scheduling influences the final result. It largely determines the trade-off between area and latency. Scheduling is used to

determine start time of each operation according to data dependence, resource constraints and timing constraints. There are three main categories of scheduling problem: scheduling without constraints, scheduling under resource constraints and scheduling under time constraint. Various kinds of algorithms can be used to tackle these problems. The algorithms can also be divided into two categories: the exact ones and the heuristic ones. Integer linear program (ILP) [25] formulation is an exact solution for both resource and timing constraints scheduling problems - which only be suitable for small design problems. Its complexity increases exponentially as the scale of the design increases. There are many heuristic algorithms to solve these problems in the acceptable accuracy. ASAP (as soon as possible) and ALAP (as late as possible) [25] algorithms are examples of the simplest constructive algorithms that can be used to solve scheduling problems. List scheduling [25] and Forced-directed [25] scheduling resolve the scheduling problems under resource constraints. Genetic algorithm is also one of the heuristic algorithms used the design space optimization problem. [2] and [1] use Genetic algorithm with multi chromosome.

### *2.4 Register allocation*

After scheduling, if a data transfer crosses the cycle boundary it means that it has to be stored in memory or a register. There is a possibility of sharing the register, if the lifetime of the data has not overlapped. The lifetime of the data refers to the interval from its first appearance to the last use. There are algorithms based on the graph theory to serve the register sharing such as left-edge algorithm [2], coloring conflict graph etc. Register sharing can significantly save area cost. In left-edge algorithm, all data transfers between control steps (which represent intermediate variables in the input algorithmic description) are stored in registers. At the end, all output data

are also saved in registers. The number of registers required in a data path implementation is determined by the maximum number of concurrent data transfers between any two control steps, which in turn relies on the operation schedule [2]. Hence, the number of registers needed in a data path can be determined only after operation scheduling is completed. The birth time of a data transfer is the control step in a schedule when it is created by a producer. Likewise, the end time of a data transfer is the control step when it is used by the last consumer.

### *2.5 Binding*

Binding is the process of grouping each scheduled operation with a concrete component. Allocation insures that there are enough resources for scheduling. But at this stage, it hasn't been decided which resource to be used for which operation. Binding resolves this issue and influences the number of multiplexers and quantity of interconnect that needs to take place. For example, in Figure 3.2 after scheduling if we have 1 multipler,1 adder and 1 subtractor, during binding node (1, 2,4,5), (7,8) and (3,9) will bind together.

### *2.6 State machine extracting and net- list generation*

The final step is extraction of data path and state machine and also generation of the control unit. Finite State Machine is an effective method to generate a control unit. In HLS, generation of FSM as a control unit is done automatically. But if we use RTL design methodology, designers have to complete the process manually. This is tedious and error-prone job at the same time.

The last step is net-list generation. The net-list generated by this step is used to transfer the design data to another tool, such as the RTL synthesizer.

# Chapter 3

# Genetic algorithms

Genetic algorithms are stochastic combinatorial optimization techniques based on evolutionary improvements. It operates on a population of knowledge structures, chromosomes that represent candidate solutions. On every generation, a number of chromosomes with the worst fitness values are removed from the population and replaced by new chromosomes obtained through applying genetic operators. The execution of the algorithm is iterated until either the best chromosome, representing an optimal solution, is found or a predetermined terminating condition such as maximum number of generations or maximum number of fitness evaluations has been satisfied. In the later case, the chromosome with the best fitness in all generations is considered as a final solution.

An implementation of a genetic algorithm begins with a population of (typically random) chromosomes. One then evaluates these structures and allocates reproductive opportunities in such a way that those chromosomes, which represent a better solution to the target problem, are given more chances to "reproduce" than those chromosomes with poorer solutions. The "goodness" of a solution is typically defined with respect to the current population.

### *3.1 Initial population*

In Genetics algorithm initial population are usually generated randomly. However in some circumstances the initial population can be acquired from:-

- A previously saved population

- A set of solutions provided by a human expert
- A set of solutions provided by another heuristic algorithm

The initial population in both [2] and [1] are generated randomly.

### *3.2 Encoding of the Chromosome*

Usually there are only two main components of most genetic algorithms that are problem dependent: problem encoding and evaluation function.

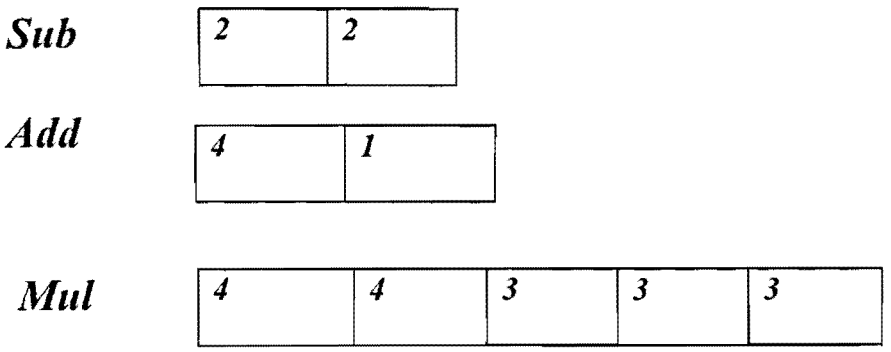
Consider a parameter optimization problem where we must optimize a set of variables either to maximize or to minimize cost or some measure of error. The goal is to set the various parameters so as to optimize an output. In more traditional terms, we wish to minimize (or maximize) some function F1(X1,X2...Xm)

Most users of genetic algorithm typically are concerned with problems that are nonlinear. This also often implies that it is not possible to treat each parameter as an independent variable which can be solved in isolation from the other variables. There are interactions such that the combined effects of the parameters must be considered in order to maximize or minimize the output of the black box.

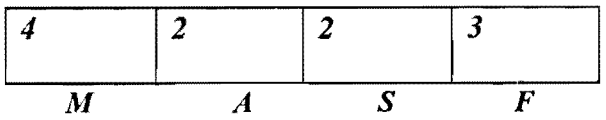
Possible individual's encoding can be represented as Bit strings, Real numbers, Permutations of element, Lists of rules (R1 R2 R3 ... R22 R23) etc.

When choosing an encoding method the data structure has to be as close as possible to the natural representation. If possible, make sure that all genotypes correspond to feasible solutions and also genetic operators preserve feasibility.

[1] Uses 'nodal string' - contains the load-factor values of each node which will determine the priority of the nodes during scheduling. The 'resource allocation string' contains list of integers which indicates the maximum number of resources allowed during scheduling. Figure 4 shows encoding of the DFG based on [1] assuming execution time for all operation is 1cc.

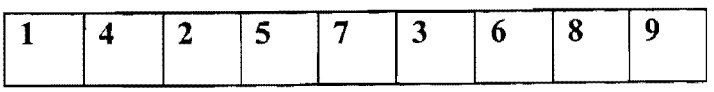



A. Nodal string representation example

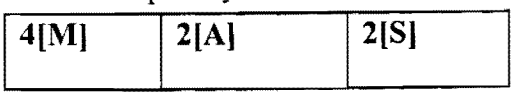



*B. Resource allocation string example*

Figure 4 - M-GenESys Chromosome encoding scheme




a. Node priority field




b. Resource allocation field

Figure 5 - GA Chromosome encoding scheme

### *3.3 Crossover Scheme*

Genetic algorithms have the recombination operation which probably comes closest to the natural model. When performing single-point crossover, both parental chromosomes are split at a randomly determined crossover point. Subsequently, a new child genotype is created by appending the second part of the second parent to the first part of the first parent. In two-point crossover, both parental genotypes are split at two points and a new offspring is created by using part number one and three from the first, and the middle part from the second parent chromosome. The n-point crossover operation is also called multi-point crossover. For fixed-length strings, the crossover points for both parents are always identical. Many crossover techniques exist for organisms which use different data structures to store themselves.

***One-point crossover***: - A single random crossover point on both parents' organism strings is selected. All data beyond that point in either organism string are swapped between the two parent organisms. The resulting organisms are the children:

Parent1: XXX|XXXXXXX

Parent2: YYY|YYYYYYY

Offspring1: XXX|YYYYYYY

Offspring2: YYY|XXXXXXX

*Two-point crossover.* - Two-point crossover calls for two random points to be selected on the parent organism strings. Everything between the two points is swapped between the parent organisms, rendering two child organisms:

Parent1: XXX|XXXXXXX|XXXXX

Parent2: YYY|YYYYYYY|YYYYY

Offspring1: XXX|YYYYYYY|XXXXX

Offspring2: YYY|XXXXXXX|YYYYY

*Cut and splice.* - Another crossover variant, the "cut and splice" approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point.

Parent1: XXX|XXXXXXXXXXX

Parent2: YYYYYYYYY|YYYYY

Offspring1: XXX|YYYYYYYYYYY

Offspring2: YYYYYYYYY|XXXXXXXXXXX

***Uniform Crossover and Half Uniform Crossover.*** In both these schemes, the two parents are combined to produce two new offspring. In the uniform crossover scheme, individual bits in the string are compared between two parents. The bits are swapped with a fixed probability, typically 0.5.

In the half uniform crossover scheme, exactly half of the non-matching bits are swapped. Thus first the Hamming distance (the number of differing bits) is calculated. This number is divided by two and from the result, analyzes how many of the bits do not match between the two parents. If any, they will be swapped.

***Crossover for Ordered Chromosomes:*** Depending on how the chromosome represents the solution, a direct swap may not be possible. One such case is when the chromosome is in an ordered list, such as an ordered list of the cities to be travelled by the traveling salesman problem. A crossover point is selected on the parents. Since the chromosome is an ordered list, a direct swap would introduce duplicates and remove necessary candidates from the list. Instead, the chromosome up to the crossover point is retained for each parent. The information after the crossover point is ordered as it is ordered in the other parent. For example, if our two parents are ABCDEFGHI and IGAHFDBEC and our crossover point is after the fourth character, then the resulting children would be ABCDIGHFE and IGAHBCDEF.

[2] and [1] use One-point crossover in all the strings.

### *3.4 Mutation Operation*

Mutation is an important method for preserving the diversity of the solution candidates by introducing small and random changes into them. In fixed-length string chromosomes, this can be achieved by randomly modifying the value of a gene.

A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified. This mutation procedure, based on the biological point mutation, is called single point mutation. Other types are inversion and floating point mutation. When the gene encoding is restrictive as in permutation problems, mutations types will be swaps, inversions and scrambles.

The purpose of mutation in GAs is preserving and introducing diversity. Mutation should allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.

### *3.5 Global Cost Function and Fitness Evaluation Methodology*

A fitness function is a particular type of objective function that prescribes the optimality of a solution (that is, a chromosome) in a genetic algorithm so that that particular chromosome may be ranked against all the other chromosomes. Optimal chromosomes, or at least chromosomes which are more optimal, are allowed to breed and mix their datasets by any of several techniques, producing a new generation that will (hopefully) be even better.

An ideal fitness function correlates closely with the algorithm's goal, and yet may be computed quickly. Speed of execution is very important, as a typical genetic algorithm must be iterated many, many times in order to produce a usable result for a non-trivial problem. This is one of the main drawbacks of GAs in real world applications and limits their applicability in some industries. It is apparent that amalgamation of approximate models may be one of the most promising approaches, especially in the following cases:

- Fitness computation time of a single solution is extremely high
- Precise model for fitness computation is missing
- The fitness function is uncertain or noisy

Two main classes of fitness functions exist: one where the fitness function does not change, as in optimizing a fixed function or testing with a fixed set of test cases; and one where the fitness function is mutable, as in niche differentiation or co-evolving the set of test cases.

Another way of looking at fitness functions is in terms of a fitness landscape, which shows the fitness for each possible chromosome.

Definition of the fitness function is not straightforward in many cases and often is performed iteratively if the fittest solutions produced by GA are not what are desired. In some cases, it is very hard or impossible to come up even with a guess of what fitness function definition might be. Interactive genetic algorithms address this difficulty by outsourcing evaluation to external agents (normally humans).

### *3.6 Terminating*

Termination is the criterion by which the genetic algorithm decides whether to continue searching or stop the search. Each of the enabled termination criterion is checked after each generation to see if it is time to stop. Genetic Server and Genetic Library include the following types of terminations:

***Generation Number*** - A termination method that stops the evolution when the user-specified maximum number of evolutions have been run. This termination method is always active.

***Evolution Time*** - A termination method that stops the evolution when the elapsed evolution time exceeds the user-specified max evolution time. By default, the evolution is not stopped until the evolution of the current generation has been completed, but this behavior can be changed so that the evolution can be stopped within a generation.

***Fitness Threshold*** - A termination method that stops the evolution when the best fitness in the current population becomes less than the user-specified fitness threshold and the objective is set to minimize the fitness. This termination method also stops the evolution when the best fitness in the current population becomes greater than the user-specified fitness threshold when the objective is to maximize the fitness.

***Fitness Convergence*** - A termination method that stops the evolution when the fitness is deemed as converged. Two filters of different lengths are used to smooth the best fitness across the generations. When the smoothed best fitness from the long filter is less than a user-specified percentage away from the smoothed best fitness from the short filter, the fitness is deemed as converged and the evolution terminates.

***Population Convergence*** - A termination method that stops the evolution when the population is deemed as converged. The population is deemed as converged when the average fitness across the current population is less than a user-specified percentage away from the best fitness of the current population.

***Gene Convergence*** - A termination method that stops the evolution when a user-specified percentage of the genes that make up a chromosome are deemed as converged. A gene is deemed as converged when the average value of that gene across all of the chromosomes in the current population is less than a user-specified percentage away from the maximum gene value across the chromosomes.

The terminating criterion for both [1] and [2] is user defined Number of Generation.

# Chapter 4

# Software Implementation

The software implementation for the M-GENESYS and GA consists of five major components as shown in Figure 6. These are:-

- Module library
- Input file
- User configurable parameters
- Main application software
- Output result

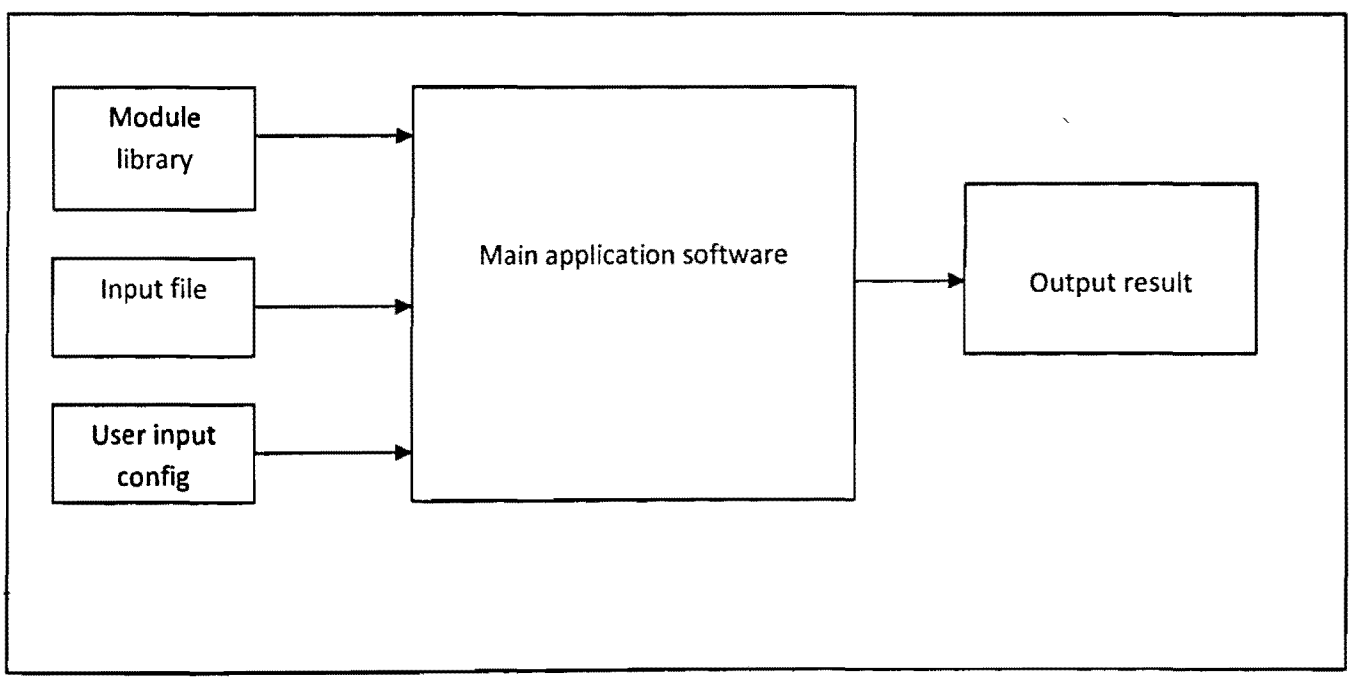



Figure 6 - Major components of the implementation software

### 4.1 Module library

The module library is user configurable values which contains the following resource information:-

1. Versions of Functional Units
2. Maximum resources available for each version of the Functional Units(FU)
3. Clock cycle of each available FUs
4. Power consumption per unit area at a specific frequency
5. Total area of the FUs

```
adder
1,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,50au,1cc,1
2,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,30au,2cc,1
3,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,15au,3cc,2
subtractor
1,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,50au,1cc,0
2,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,30au,2cc,0
3,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,15au,3cc,0
multiplier
1,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,120au,2cc,0
2,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,80au,3cc,1
3,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,50au,4cc,1
comparator
1,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,50au,1cc,0
2,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,30au,2cc,0
3,50MHZ,3mW,100MHZ,6mW,200MHZ,12mW,15au,3cc,0
```

Figure 7 - Module library format

A sample of the module library format is shown in Figure 7. On this particular module library, there is only one adder of version 1 and this adder takes 50au area unit and the execution time for this adder is 1cc. This unit consumes 3mw per unit area at 50MHZ.

### 4.2 Input File

The input file contains the Data Flow Graph (DFG) which represent the behavioral description of the data path in ASAP (as soon as possible) format.

```
+,I,I,1,+,I,I,2
+,2,2,3
+,3,3,4
+,4,4,5
*,5,5,6,*,5,5,7
+,1,6,8,+,7,7,9
+,1,8,10,+,9,9,11,+,3,9,12
*,10,10,13,+,8,11,14,*,12,12,15
+,13,13,16,+,2,15,17
+,16,16,18,+,8,16,19,+,9,17,20,+,2,17,21
*,18,18,22,+,19,19,23,+,20,20,24,*,21,21,25
+,16,22,26,*,23,23,27,*,24,24,28,+,25,25,29
+,27,27,30,+,28,28,31,+,17,29,32
+,23,30,33,+,31,31,34
```

Figure 8 - Data Flow Graph representation of The Elliptic Wave Filter

Figure 8 shows the representation of the Elliptic Wave Filter benchmark DFG which is shown in Figure 9. Each node is represented with the following format:-

*Functional unit, input1, input2, output*

The Functional Unit can be adder, multiplier, subtractor, and comparator. If either input1 or input 2 are nodes which don't have parents, they should be represented by the letter I as shown in Figure 8. This informs the software that these are inputs to the DFG from outside. This input format was chosen for its simplicity in implementation compared to a graphical one.

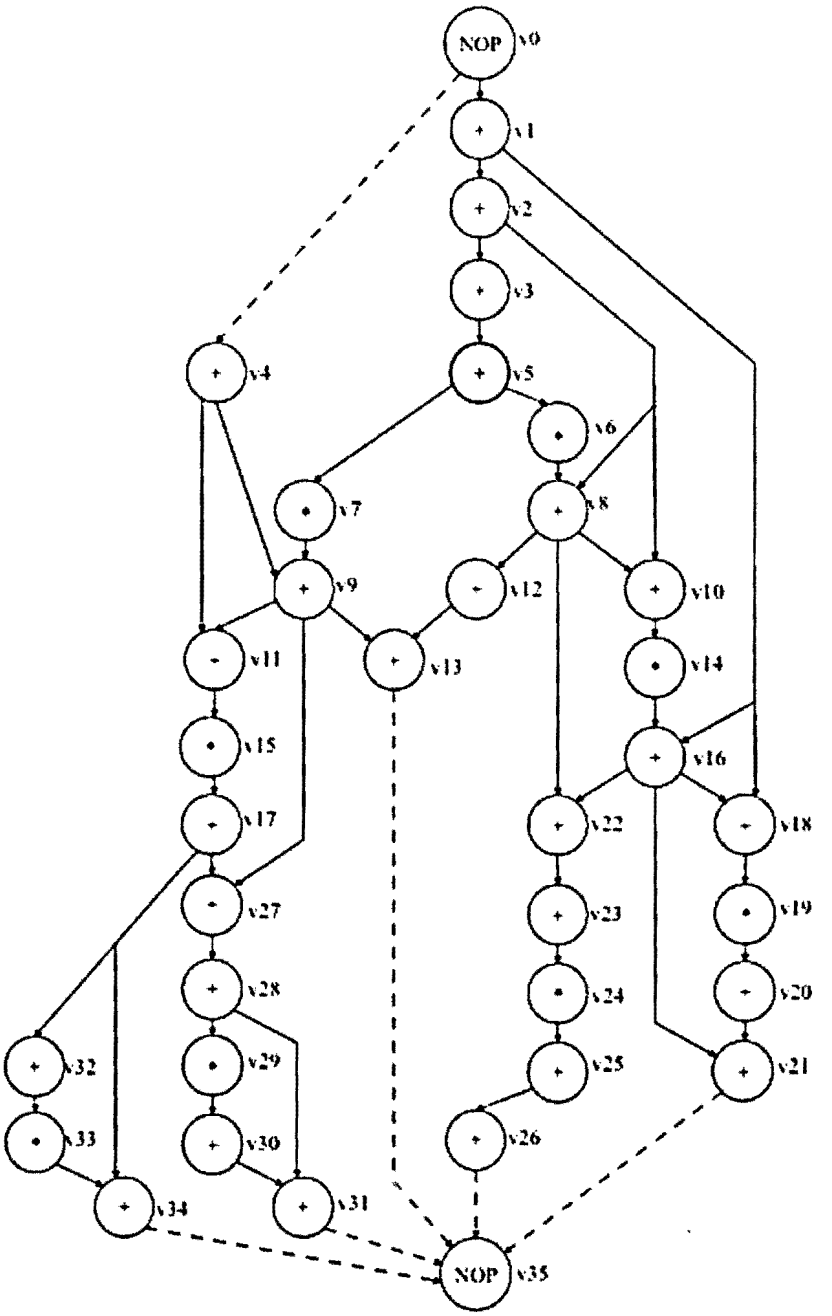



Figure 9 - Elliptic wave filter (EWF)

### 4.3 User configurable parameters

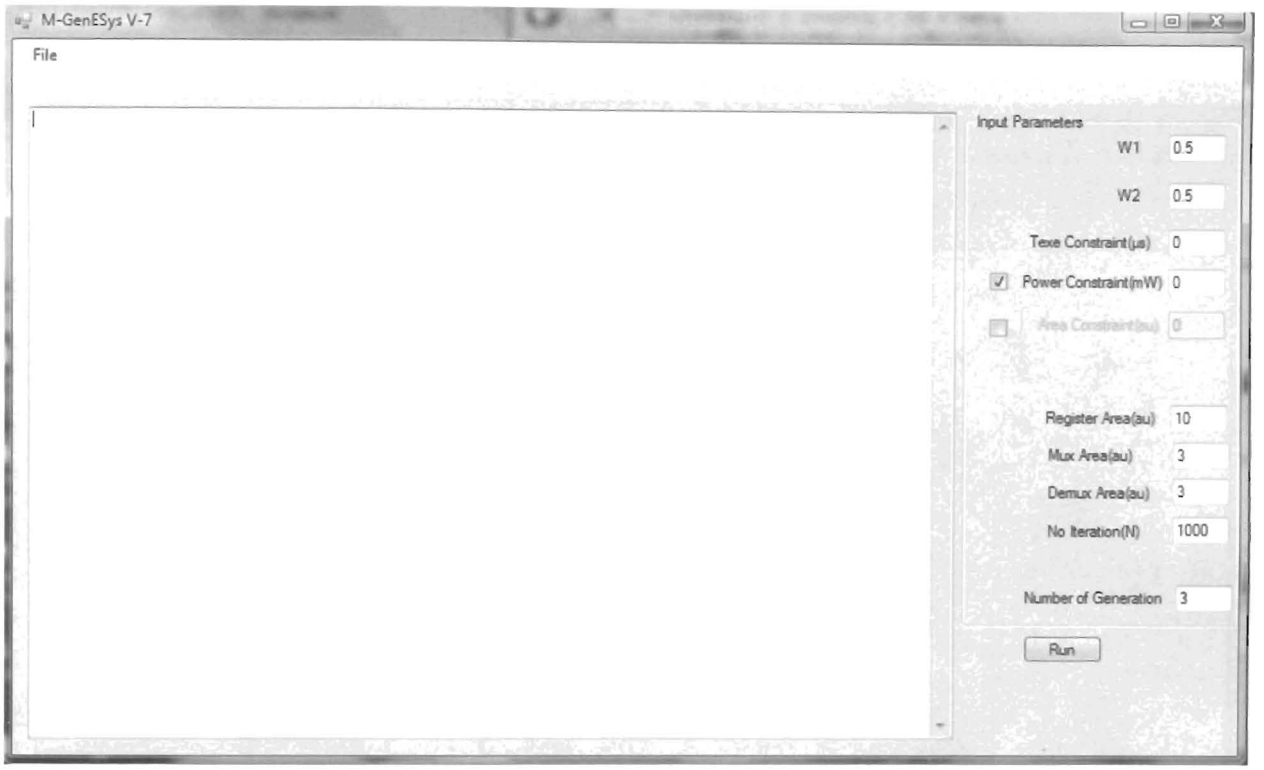



Figure 10 - User interface for M-GENESYS

The following parameters can be configured by the user:-

1. ***W1 and W2*** - The weight values used in the cost calculation which either favor $T_{exe}$ or power/area. W1is for $T_{exe}$ and W2 is for power/area

2. ***$T_{exe}$ constraint*** – User define $T_{exe}$ user constraint

3. ***Power constrain*** – User define power constraint

4. ***Register, Mux, Demux unit areas***– Unit area for Registers, Mux and Demux

5. ***Number of iteration*** - Is used in the $T_{exe}$ calculation ($T_{exe}$ calculation is described in[1])

6. ***Number of Generation*** – Is the number of generation explored before the genetic algorithm terminates

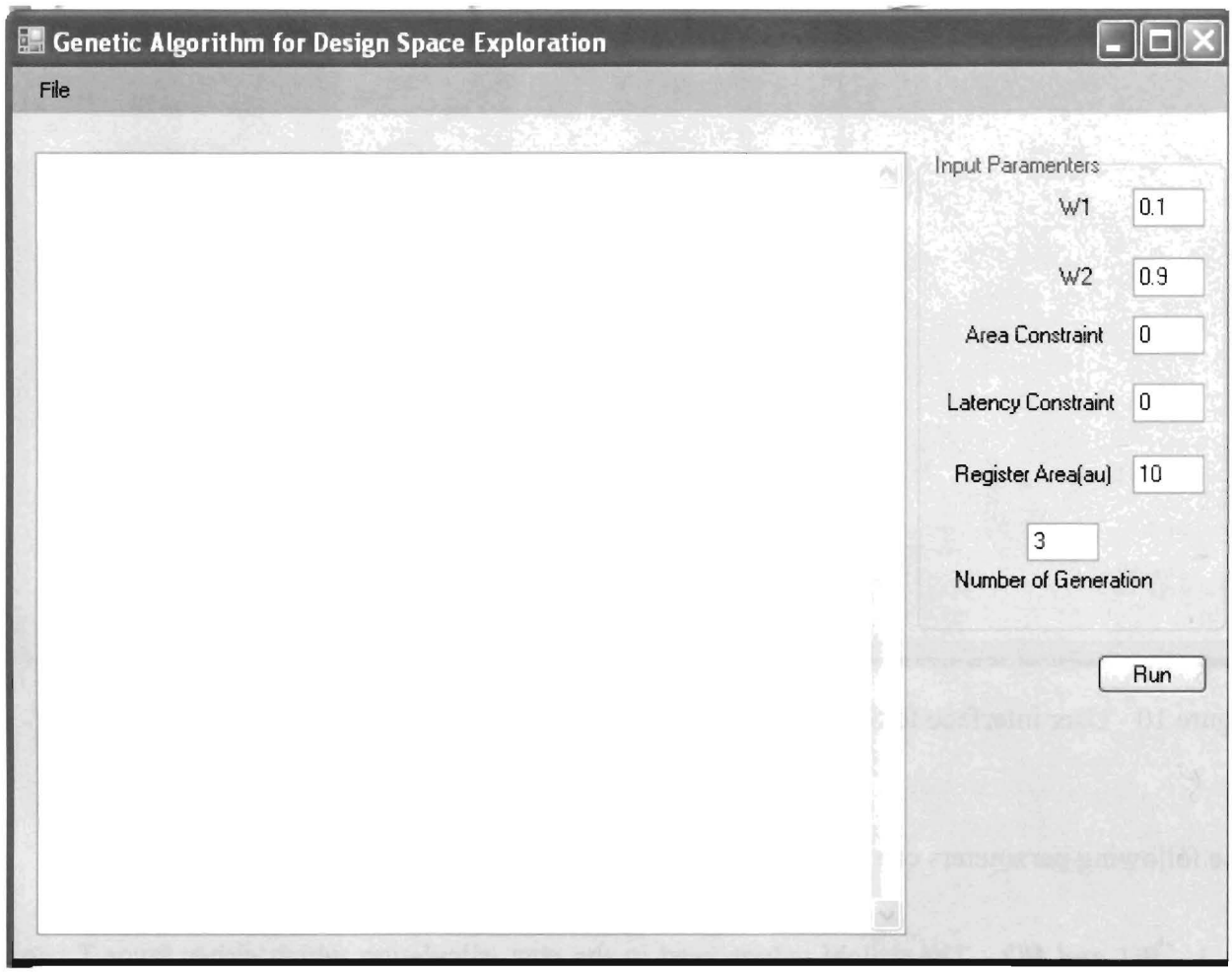



Figure 11 - User interface for GA

### 4.4 Main application software

The main application software flow is shown in Figure 12. In this section a detailed implementation of the each module will be explained.

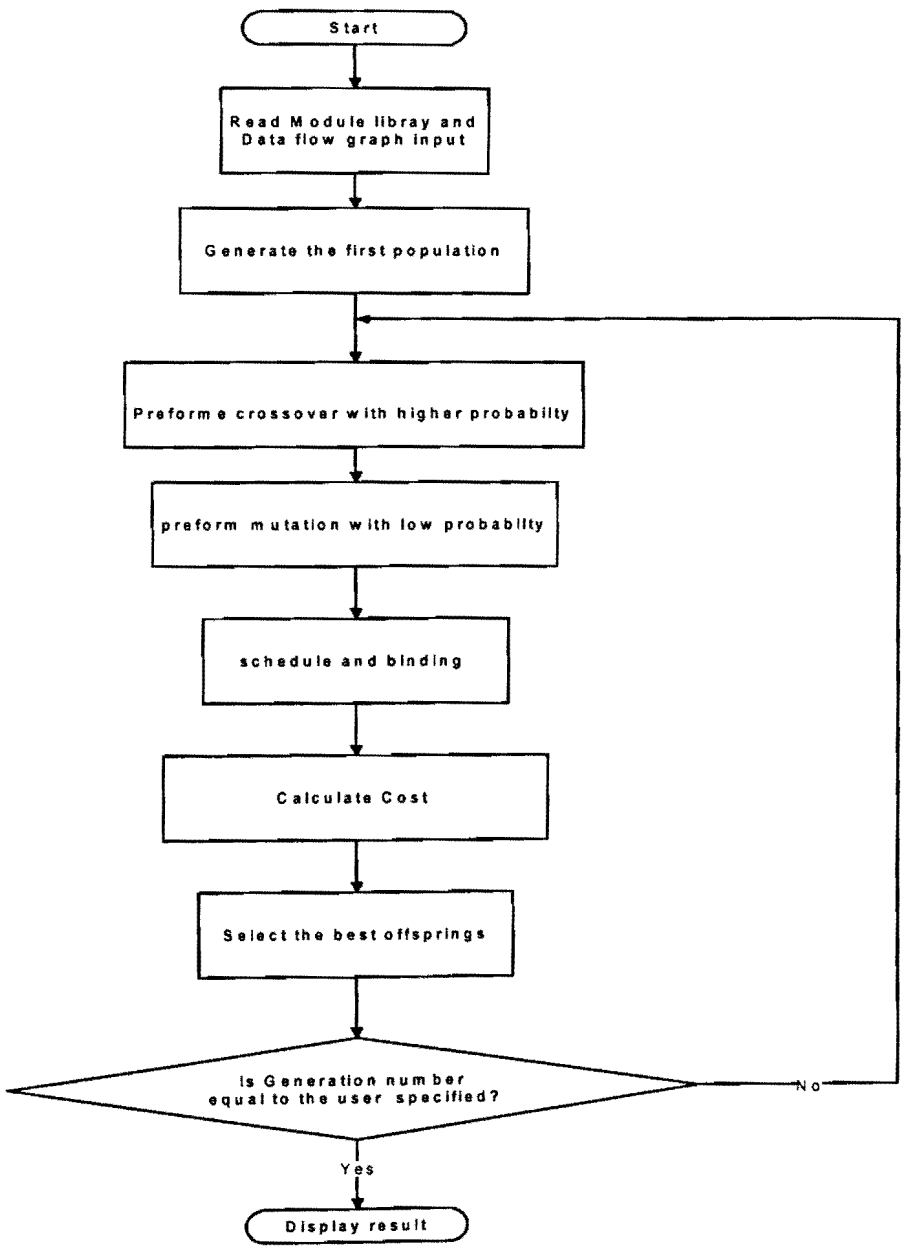



Figure 12 - General flow of the main software

### 4.5 Reading module library and data flow graph

During start up, the module library file is read and parsed. The information is stored in the memory for later use. The user also needs to select a specific Data Flow Graph (one of the benchmarks) on which the genetic algorithm will be preformed.

### 4.6 Generation of the first population

In the case of the M-GENESYS, initially a total of 40 parents are created based on the following algorithm. P1 is created as follows:-

- *Encode the first parent (P1) of the nodal string using the load factor (α) metric based on the ASAP schedule with maximum resources.*

```
Load factor computation for each node

For each node in the DFG v[i]=0 to v[i]= max
  While( end of tree)
    go down the DFG tree to the next node
    Load factor (LF)= latency of the FU
    LF= LF + latency of the FU
   v[i]_LF = LF
  End while
End for
```

- *Encode the first parent (P1) of the resource allocation string with maximum resources acquied from the ASAP schedule*

Figure 13 - First parent generation

The second parent is created as follows:-

- *Create the second parent nodal string using load factor (β) calculated as: $L^{ASAP\,(max)}$-α (oi) based on minimum resources.*
- *Resource allocatin string is created with the minimum resource*

Figure 14 - Second parent generation

The remaining parents are created as follows:-

- *Create the remaing of the parent (P3...Pn) of the nodal string based on the perturbation function = (α + β)/2 ± μ; where 'μ' is a random value between 'α' and 'β'.*

```
for i=3 to i=Pn
   for each nodal stringof the workload
      generate random number μ  between 'α' and 'β'.
      nodal string =(α + β)/2 ± μ
End for
```

- *Resource allocation chromosome for the remianing parents. Randomly select two nodes vi and vj from the chromosome that represent the resource allocaltion then generate a random value between vi and vj*

```
for i=3 to i=Pn
    for each resource allocation
       generate random number k  between vi and vj
       resource allocation =k
End for
```

Figure 15 - (*P3...Pn)* parent generation

In the case of the GA algorithm the first $P_n$ parents are created as follows:-

```
For i =0 to i = max parent
    While (node priority field is complete)
        Generate random number
        if  Random number topologically correct
            add to the node priority field
    end while
End for
```

Figure 16 - First $P_n$ Parent generation for GA implementation

### 4.7 Cross over and mutations

Crossover for M-GENESYS and the GA implementations are as follows:-

```
Algorithm M-GENESYS Crossover
    Radomaly generate an integer r, where 1<r<N, and N= number of tasks
    For i=1 to r do
        Offspring1[i]=parent1[i]
        Offspring2[i]=parent2[i]
    End for
    For i=R to n do
        Offspring1[i]=parent2[i]
        Offspring2[i]=parent1[i]
    End for
```

Figure 17 - M-GENESYS crossover for nodal string and resource allocation

```
Algorithm One-Point Topological Crossover
    Radomaly generate an integer r, where 1<r<N, and N= number of tasks
   For i=1 to r do
       Offspring1[i]=parent1[i]
       Offspring2[i]=parent2[i]
   End for
   Pos1= r+1
   Pos2=r+1
  For j=1 to N do
        If offspring1 does not contain parent2[j] then
          Offspring1[pos1]= paent2[j]
          Pos1 = pos2 +1
        End if
 End for
 For j=1 to N do
       If offspring2  does not contain parent [j] then
            Offspring2[pos2]= parent1[j]
           Pos2=pos2+1
       End if
End for
```

Figure 18 - One-point topological crossover for GA implementation

The Mutation scheme proposed in [1] used for the cost calculations

```
Algorithm Precedence Preserving Shift Mutation
   Randomly pick a task T to shift
   Compute p[Pmax]and p[SMIN] for task T
   Randomly pick a postion r, where p[PMAX] < r < p[SMIN]
  Shift task T to positon r
```

Figure 19 - Precedence preserving shift mutation for GA implementation

## 4.8 Schedule and Binding

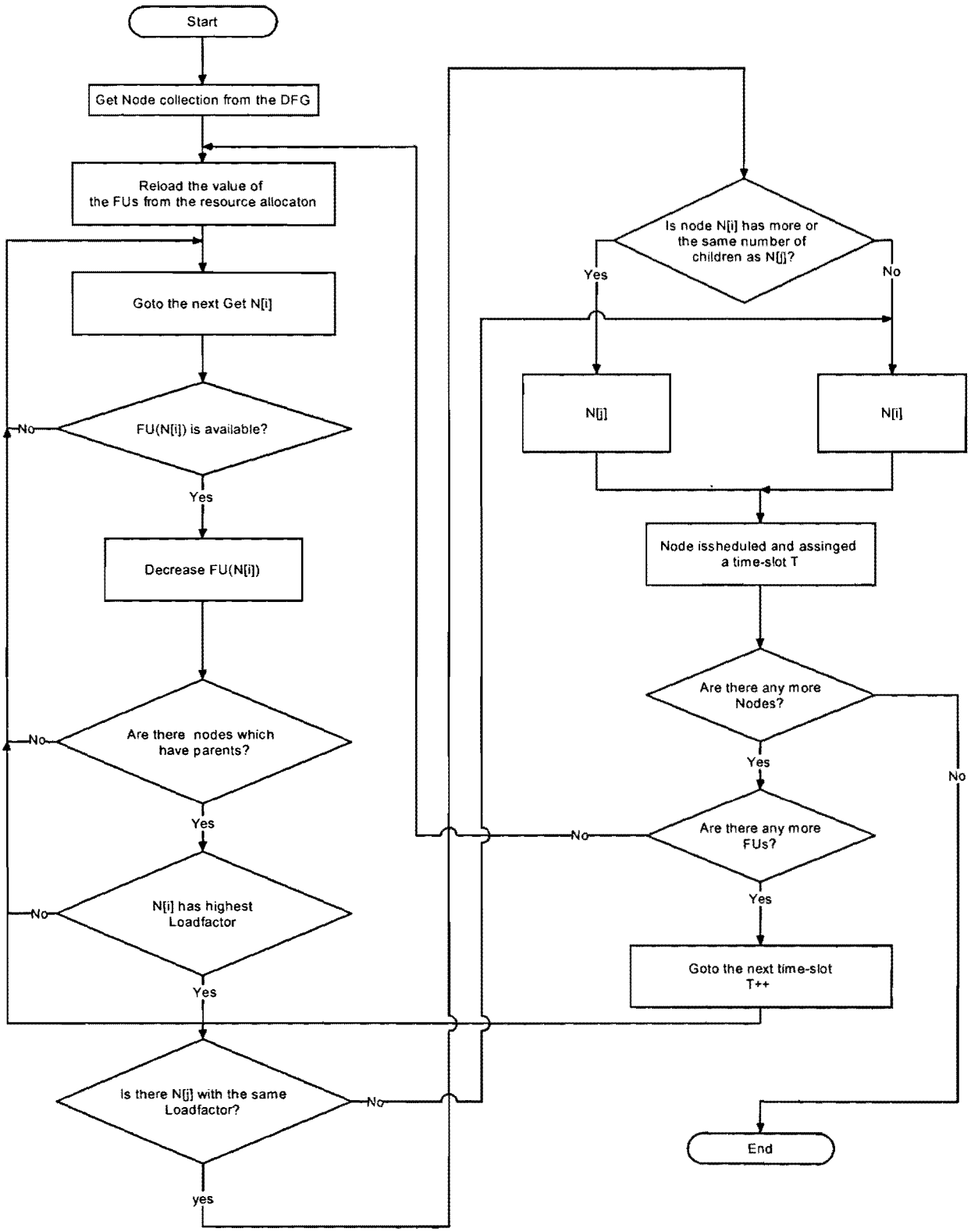



Figure 20 - Scheduling scheme for the M-GENESYS

Scheduling algorithm for GA algorithm is shown in Figure 21

```
Algorithm :- Modified list scheduling algorithm
    Input: Dataflow Graph, Chromosome
   Control step = 1
   While not all tasks scheduled
     For i= 1 to N do
       If p[i] is unscheduled and a corresponding functional unit available then
          Assign task p[i] to the functional unit in the current control step
       End if
     End for
     Control step = control step +1
   End while
   Schedule length = control step
   Return schedule length
```

Figure 21 - scheduling algorithm for GA implementation

### 4.9 Cost calculation

The cost function used during implementation has been adopted from authors work in [1] and [2] as shown below. Please refer to [1] and [2] where the authors have explained in detail about the cost function.

$$C_G = f(T_{EXE}, A_{FU}, A_{REG}, A_{MUX}, A_{DEMUX}) \quad [1]$$

$$C_G = f(T_{LATENCY}, A_{FU}, A_{REG}) \quad [2]$$

Implementation of Latency and $T_C$ is shown in Figure 22. $T_{EXE}$ is only required for the M-GENESYS implementation. The GA implementation only uses latency for time calculation.

```
Algorithm :- T LATENCY and Tc computation
    Input: Scheduled Dataflow Graph, Resource allocation string

  For i=0 to Max_Node
    {
        N[i].t1 = cycle_time from module library
        N[i].t2 = cycle_time from module library
    }

Do {
    For i=0 to Max_Node
      {
        If N[i].t1>0 AND no node higher in the DFG has t1>0 AND free FU available AND
         parent1[N[i]].t1 = 0 AND parent2[N[i]] = 0
          {
            N[i].t1--
          }
      }
     If N[i] ≠ 0 for atleast one node in DFG
          TLATENCY++

    For i=0 to Max_Node
      {
        If N[i].t2>0 AND no node higher in the DFG has t2>0 AND free FU available AND
         parent1[N[i]].t2 = 0 AND parent2[N[i]] = 0
          {
            N[i].t2--
          }
      }
    T++
} while all the nodes in the DFG are analysed

     Tc = T - TLATENCY
```

Figure 22 - Latency and $T_C$ computation

Both M-GENESYS and GA implementation use Left Edge Algorithm to determine the number of registers. Figure 23 shows a pseudo code for the Left Edge Algorithm

```
Algorithm :- Left Edge Algorithm
    Input: Scheduled Dataflow Graph
   Sort data transfers in increasing order of their birth time
   K = 1, where k is the register count
  Do{
       Assign the first unassinged data transfer to regiser k;
      Scan the sorted list for the next unassigned data transfer whose birth time
     Is ≥the end time of the pervious value;
    Assign this data transfer to the current register;
   Scan the list until no more non- overlapping data transfers can share the same register;
  k=k+1
  } unitl all data transfers are assigned to registers;
```

Figure 23 - Left Edge Algorithm

Number of Mux and Demux calculation is depicted in Figure 24. This calculation is used by the M-GENESYS implementation only.

```
Algorithm :- Number of mux and demux determination
Input : Schedule DataFlow Graph
 Mux=0 and Demux=0,i =0
 Do {
      Get the number N of nodes assigned to a FU(i)
       If N> 1
          {
           Mux = Mux + 2
          Demux = Mux + 1
       }
      i=i+1
   } Untill all the FU in the scheduled DFG are done
```

Figure 24 - number of Mux and Demux computation algorithm

```
For i=0 to i= max_offspring
{
   2  P[i] = P_offspring
    3 Texe[i] =Texe_offspring
}
sort(P[i]) in increasing order
sort(Texe[i]) in increasing order
select the highest value of Pmax
select  the highest value for Texec_max
calculate Global cost using the equations [1] and [2]
```

Figure 25 - Cost calculation for M-GENESYS

Cost calculation for the GA implementation follows the same method in determining the $L_{max}$ and $A_{max}$ which are calculated during scheduling.

### 4.10 Termination criteria

The termination stage for both M-GENESYS and GA implementation is user configurable. As shown in Figure 10 and Figure 11 depending on the number entered in the "Number of Generation" text box, the program terminates the iteration of the generation and displays the least cost solution on the user interface display textbox.

# Chapter 5

# Results

In this section the results for the following benchmarks from [1] are shown:-

- Elliptic Wave Filter (EWF) shown in Figure 9
- Discrete Wavelet Transformation (DWT) shown in Figure 28
- Fast Fourier Transformation (FFT) shown in Figure 31
- Finite Impulse Response Filter (FIR) shown in Figure 34

During generation of the results, a default value of user interface was taken except for the generation number. 10 generation is used for M-GenESys and 100 generation is used for GA.

```
T1 =1  2
T2 =3
T3 =4
T4 =5
T5 =7
T6 =6  9
T7 =8  12
T8 =10  11  15
T9 =13  14  17
T10 =20  21
T11 =16  24  25
T12 =19  28  29
T13 =23  31
T14 =18  27  32
T15 =22  30  34
T16 =26  33

Latency=  460
Area=  160
cost=  69
Number of Population=  12000


   Resource allocation
+  =2
*  =1

   Node Priority
 1  2  3  4  5  7  6  8  9  12  10  11  15  13  17  20  21  25  14  24  29  28  16  19  31  23  18  32  27  22  34
30  26  33
```

Figure 26 - output result for Elliptic Wave Filter (EWF) benchmark using M-GENESYS implementation

```
Processing....
   Scheduling solution
(1, +, V-3,1)  (2, +, V-3,2)
(3, +, V-3,1)
(4, +, V-3,2)
(5, +, V-3,1)
(7, *, V-3,1)
(6, *, V-3,1)  (9, +, V-3,2)
(8, +, V-3,2)  (12, +, V-3,1)
(10, +, V-3,1)  (11, +, V-3,2)  (15, *, V-3,1)
(13, *, V-3,1)  (14, +, V-3,2)  (17, +, V-3,1)
(20, +, V-3,1)  (21, +, V-3,2)
(16, +, V-3,2)  (24, +, V-3,1)  (25, *, V-3,1)
(18, +, V-3,2)  (19, +, V-3,1)  (28, *, V-3,1)
(22, *, V-3,1)  (23, +, V-3,1)  (29, +, V-3,2)
(26, +, V-3,2)  (27, *, V-3,1)  (31, +, V-3,1)
(30, +, V-3,1)  (32, +, V-3,2)
(33, +, V-3,1)  (34, +, V-3,2)


Texe= 906 µs
Latency = 56 cycle
Power= 969 mW
cost= 37
Totoal Area= 323au
Number of Mux= 6
Number of Demux= 3
Number of Register= 9
Number of Population= 360


   Resource allocation
* =1
+ =2
f =1
```

Figure 27 - output result for Elliptic Wave Filter (EWF) benchmark using GA implementation

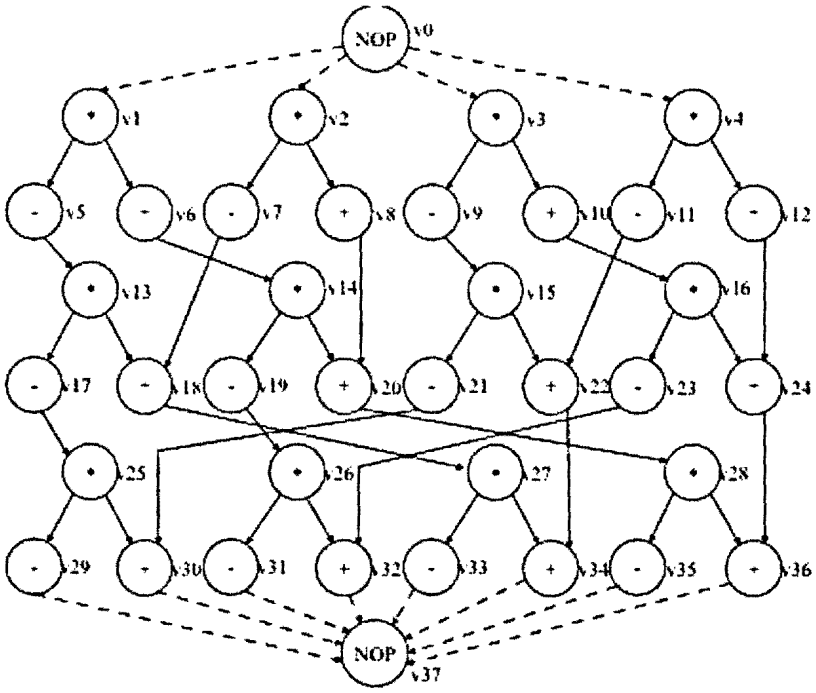



Figure 28 - Fast Fourier Transformation (FFT)

```
Processing....
   Scheduling solution
(1, *, V-3,1)
(2, *, V-3,1)  (5, -, V-3,1)  (6, +, V-3,1)
(3, *, V-3,1)  (7, -, V-3,1)  (8, +, V-3,1)
(4, *, V-3,1)  (9, -, V-3,1)  (10, +, V-3,1)
(11, -, V-3,1)  (12, +, V-3,1)  (13, *, V-3,1)
(14, *, V-3,1)  (17, -, V-3,1)  (18, +, V-3,1)
(19, -, V-3,1)  (20, +, V-3,1)  (25, *, V-3,1)
(28, *, V-3,1)  (29, -, V-3,1)
(16, *, V-3,1)  (35, -, V-3,1)
(15, *, V-3,1)  (23, -, V-3,1)  (24, +, V-3,1)
(21, -, V-3,1)  (22, +, V-3,1)  (26, *, V-3,1)
(27, *, V-3,1)  (30, +, V-3,1)  (31, -, V-3,1)
(33, -, V-3,1)  (34, +, V-3,1)
(32, +, V-3,1)
(36, +, V-3,1)


Texe= 2400 µs
Latency = 57 cycle
Power= 1992 mW
cost= 100
Totoal Area= 332au
Number of Mux= 6
Number of Demux= 3
Number of Register= 9
Number of Population= 1200


   Resource allocation
* =1
+ =1
- =1
f =2
```

Figure 29 - Output result for Fast Fourier Transformation (FFT) benchmark using M-GENESYS implementation

```
T1 =1  2
T2 =3  4  5  6  7  8
T3 =9  10  11  12  13  14
T4 =15  16  17  18  19  20
T5 =21  22  23  24  25  27
T6 =26  28  29  30  33  34
T7 =31  32  35  36

Latency= 124
Area= 1300
cost= 75
Number of Population= 12000


   Resource allocation
* =2
- =2
+ =2

   Node Priority
 1 5 2 4 3 11 9 6 10 16 8 7 15 12 14 24 20 23 21 13 19 17 22 18 25 30 29 27 33 28 34
36 26 31 32 35
```

Figure 30 - output result for Fast Fourier Transformation (FFT) benchmark using GA implementation

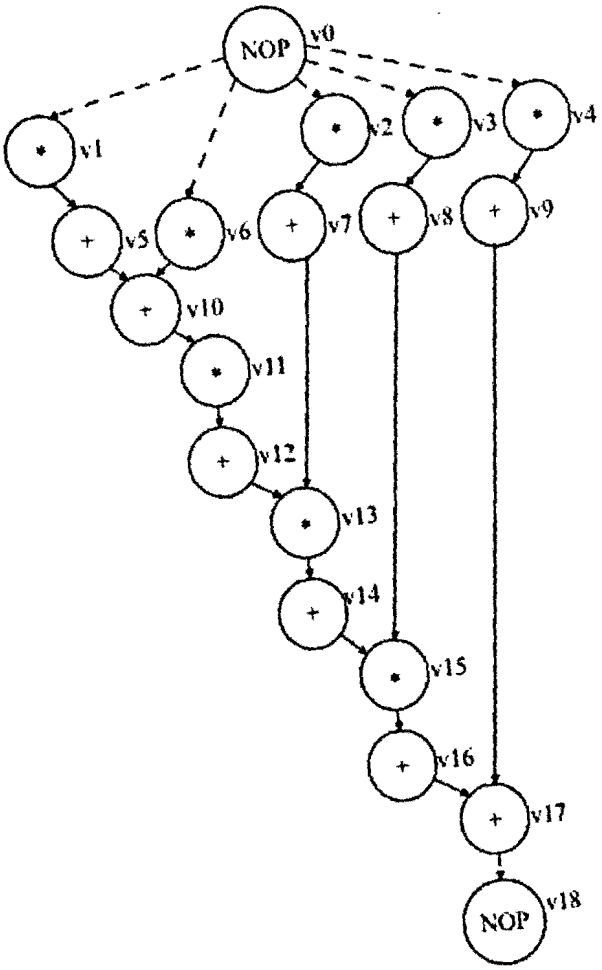



Figure 31 - Discrete Wavelet Transformation (DWT)

```
Processing....
   Scheduling solution
(1, *, V-3,1)
(2, *, V-3,1)  (6, +, V-3,1)
(3, *, V-3,1)  (10, +, V-3,2)
(5, *, V-3,1)  (7, +, V-3,1)
(4, *, V-3,1)  (9, +, V-3,2)
(8, +, V-3,1)  (11, *, V-3,1)
(12, +, V-3,2)
(13, *, V-3,1)
(14, +, V-3,1)
(15, *, V-3,1)
(16, +, V-3,2)
(17, +, V-3,1)


Texe= 203 µs
Latency = 44 cycle
Power= 1566 mW
cost= 100
Totoal Area= 261au
Number of Mux= 6
Number of Demux= 3
Number of Register= 10
Number of Population= 1200


   Resource allocation
* =1
+ =2
f =2
```

Figure 32 - Output result for Discrete Wavelet Transformation (DWT) benchmark using M-GENESYS implementation

```
T1 =1
T2 =3  6
T3 =4  7
T4 =2  8
T5 =5  10
T6 =9  11
T7 =12
T8 =13
T9 =14
T10 =15
T11 =16
T12 =17

Latency= 44
Area= 500
cost= 77
Number of Population= 12000


   Resource allocation
* =1
+ =1

   Node Priority
 1 3 4 7 6 2 5 8 10 9 11 12 13 14 15 16 17
```

Figure 33 - Output result for Discrete Wavelet Transformation (DWT) benchmark using GA implementation

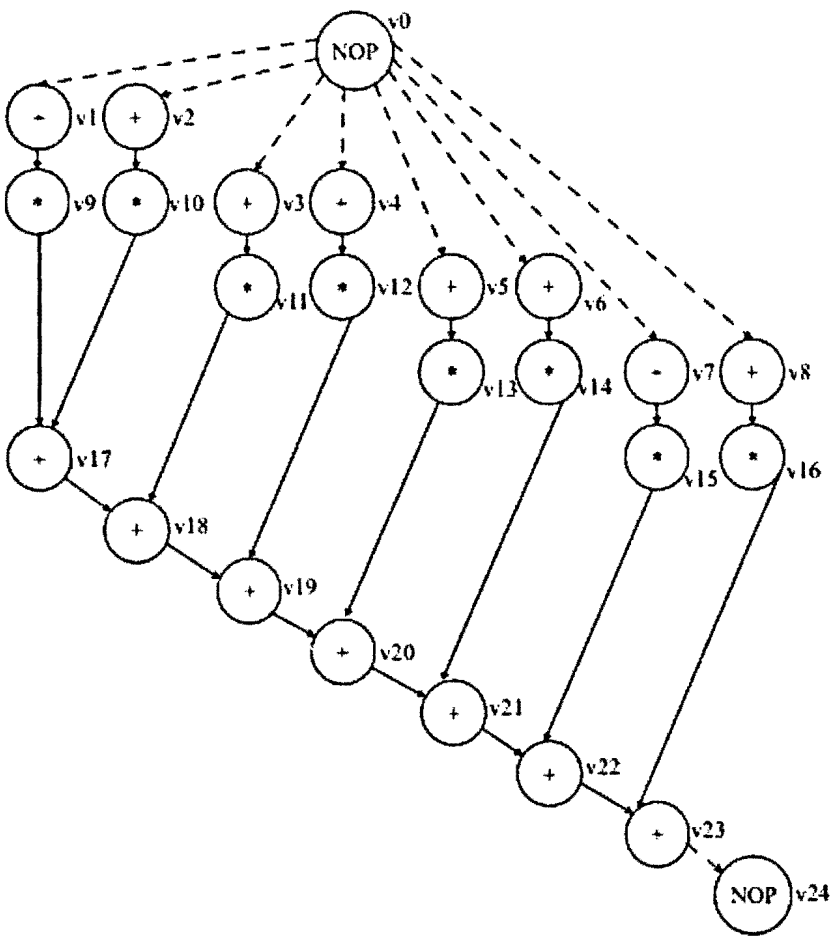



Figure 34 - Finite Impulse Response filter (FIR)

```
Processing....
   Scheduling solution
(1, *, V-3,1)
(2, *, V-3,1)  (5, -, V-3,1)  (6, +, V-3,1)
(3, *, V-3,1)  (7, -, V-3,1)  (8, +, V-3,1)
(4, *, V-3,1)  (9, -, V-3,1)  (10, +, V-3,1)
(11, -, V-3,1)  (12, +, V-3,1)  (13, *, V-3,1)
(14, *, V-3,1)  (17, -, V-3,1)  (18, +, V-3,1)
(15, *, V-3,1)  (19, -, V-3,1)  (20, +, V-3,1)
(21, -, V-3,1)  (22, +, V-3,1)  (28, *, V-3,1)
(16, *, V-3,1)  (35, -, V-3,1)
(23, -, V-3,1)  (24, +, V-3,1)  (25, *, V-3,1)
(26, *, V-3,1)  (29, -, V-3,1)  (30, +, V-3,1)
(27, *, V-3,1)  (31, -, V-3,1)  (36, +, V-3,1)
(32, +, V-3,1)  (33, -, V-3,1)
(34, +, V-3,1)


Texe= 2400 µs
Latency = 54 cycle
Power= 1992 mW
cost= 100
Totoal Area= 332au
Number of Mux= 6
Number of Demux= 3
Number of Register= 9
Number of Population= 1200


   Resource allocation
* =1
+ =1
- =1
f =2
```

Figure 35- Output result for Finite Impulse Response filter (FIR) benchmark using M-GENESYS implementation

```
T1 =4
T2 =8  12
T3 =1  16
T4 =3  9
T5 =2  11
T6 =6  10
T7 =5  14
T8 =7  13
T9 =15  17
T10 =18
T11 =19
T12 =20
T13 =21
T14 =22
T15 =23

Latency= 53
Area= 560
cost= 83
Number of Population= 12000


   Resource allocation
+ =1
* =1

   Node Priority
 4 8 1 3 16 11 2 6 5 10 12 7 9 13 15 14 17 18 19 20 21 22 23
```

Figure 36 - Output result for Finite Impulse Response filter (FIR) benchmark using GA implementation

# Chapter 6

# Conclusions

Both M-GenESys and GA implementations use genetic algorithms to solve the interdependent problem of scheduling and allocation of high level synthesis during the design space exploration. The M-GenESys cost function implementation takes into consideration $T_{exe}$ rather than only taking Latency as the GA algorithm which makes the implementation more robust and practical. But this comes at the cost of higher CPU execution time.

M-GenESys also takes into consideration the number of registers, Mux and Demux (other than FU) in the total area calculation as opposed to the GA implementation which only takes register area. This also have an advantage of minimizing the number of Mux and Demux at slight penalty of CPU execution time.

The chromosome representation of the GA encodes the precedence relationships among the tasks, in the input behavioral specification with a topological order-based representation, has made the implementation simple and consumes less CPU time compared to the M-GenESys which uses a complicated scheduling technique based on the load-factor heuristics. Moreover the M-GenESys scheduling selects the best Functional Unit (FU) type, based on the user's defined module library.

Generally, M-GenESys takes lots of practical parameters into consideration which has not been consider by the GA implementations. This makes the M-GenESys result more realistic compared to the GA implementation. Additionally, the results of the M-GenESys for most benchmarks are competitive and even better than GA implementations (Please refer to the M-GenESys paper).

# References

[1] Anirban Sengupta and Reza Sedaghat "M-GenESys: Multi Structure Genetic Algorithm based Design Space Exploration System for Integrated Scheduling, Allocation and Binding in High Level Synthesis", IEEE Transactions on Evolutionary Computation, Submitted, 2010.

[2] Vyas Krishnan and Srinivas Katkoori, "A Genetic Algorithm for the Design Space Exploration of Datapaths During High-Level Synthesis, IEEE Transactions on Evolutionary Computation, vol. 10, no. 3, June 2006.

[3] Anirban Sengupta, Reza Sedaghat, Zhipeng Zeng, "A High Level Synthesis design flow with a novel approach for Efficient Design Space Exploration in case of multi parametric optimization objective", International Journal of Microelectronics Reliability, Science Direct, Elsevier, Volume 50, Issue 3, 2010, Pages 424-437.

[4] Anirban Sengupta, Reza Sedaghat, "Multi Objective Design Space Exploration and Architectural Synthesis of an Application Specific Processor with Multi Parametric Objective", Journal of Computer and Electrical Engineering, Elsevier, Submitted, 2010.

[5] Pilato, D. Loiacono, F. Ferrandi, P.L. Lanzi and D. Sciuto (June 2008), "High-level Synthesis with Multi-objective Genetic Algorithm: a Comparative Encoding Analysis", Proc. IEEE CEC 2008 Congress on Evolutionary Computation, Hong Kong (China).

[6] S. Ghaemi, M. T. Vakili, and A. Aghagolzadeh, "Using a Genetics Algorithm Optimizer Tool to Solve University Timetable Scheduling Problem," 9th International Symposium on Signal Processing and its Applications (ISSPA2007), Sharjah, United Arab Emirates, 12-15 February, 2007.

[7] A. Banaiyan, H. Esmaeilzadeh, and S. Safari, "Co-Evolution Scheduling and Mapping for High-Level Synthesis, " IEEE ICEIS 2006, Islamabad, Pakistan, pp. 269-273, Apr. 2006.

[8] C. Bolchini , W. Fornaciari , F. Salice , D. Sciuto, "Concurrent error detection at architectural level", Proceedings of the 11th international symposium on System synthesis, p.72-75, December 02-04, 1998, Hsinchu, Taiwan, China

[9] Author(s): Grewal, G; O'Cleirigh, M; Wineberg, M," An evolutionary approach to behavioral-level synthesis" , 2003 CONGRESS ON EVOLUTIONARY COMPUTATION, VOLS 1-4, PROCEEDINGS Pages: 264-272, 2003.

[10] Mandal, C, Chakrabarti, P.P., "Genetic Algorithms for High-Level Synthesis in VLSI Design", Materials and Manufacturing Processes, 18(3), pp 355 – 383, 2003

[11] Elgamel, MA; Bayoumi, MA, "On low power high level synthesis using genetic algorithms", ICES 2002: 9TH IEEE INTERNATIONAL CONFERENCE ON ELECTRONICS, CIRCUITS AND SYSTEMS, VOLS I-111, CONFERENCE PROCEEDINGS Pages: 725-728, 2002

[12] Elie Torbey , John Knight, Performing Scheduling and Storage Optimization Simultaneously Using Genetic Algorithms, Proceedings of the 1998 Midwest Symposium on Systems and Circuits, p.284, August 09-12, 1998

[13] Whitely, D.: A Genetic Algorithm Tutorial, Computer Science department, Colorado State University, 2003.

[14] Dianati, M., Song, I., and Treiber, M. An introduction to genetic algorithms and evolution strategies. Technical report, University of Waterloo, Ontario, N2L 3G1, Canada, July 2002.

[15] M. Holzer, B. Knerr, M. Rupp "Design Space Exploration with Evolutionary Multi-Objective Optimisation", IEEE Second International Symposium on Industrial Embedded Systems, Lisbon, Portugal , S. 126 – 133, 2007

[16] E. Torbey and J. Knight, "High-level synthesis of digital circuits using genetic algorithms," in Proc. Int. Conf. Evol. Comput., May 1998, pp.224–229.

[17] E. Torbey and J. Knight, "Performing scheduling and storage optimization simultaneously using genetic algorithms," in Proc. IEEE Midwest Symp. Circuits Systems, 1998, pp. 284–287.

[18] Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo, Maurizio Palesi, Davide Patti, "Effcient design space exploration for application specific systems-on-a-chip" Journal of Systems Architecture 53 (2007) pages: 733–750.

[19] Williams, A. C., Brown, A. D. and Zwolinski, M, "Simultaneous Optimisation of Dynamic Power, Area and Delay in Behavioural Synthesis", IEE Proceedings Computers and Digital Techniques, 2000, Volume: 147, Issue: 6, On page(s): 383-390.

[20]De Micheli,G.(1994) Synthesis and Optimization of Digital Systems, McGraw-Hill Inc., 580 p.

[21] McFarland, Parker, A.C, Camposano, R. "Tutorial on high-level synthesis" Proceedings of the 25th ACM/IEEE Design Automation Conference, 1988, Atlantic City, New Jersey, United States, Pages: 330 – 336.

[22]Giuseppe Ascia, Vincenzo Catania, Alessandro G. Di Nuovo Maurizio Palesi, Davide Patti, "Effcient design space exploration for application specific systems-on-a-chip" Journal of Systems Architecture 53, Science Direct, Elsevier, 2007, Pages: 733-750

[23] Pierre G. Paulin and John P. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis, 26th conference on Design Automation, 1988, Pages: 1-6

[24] Maurizio Palesi, Tony Givargis, "Multi-Objective Design Space Exploration Using Genetic Algorithms", Proceedings of the tenth international symposium on Hardware/software codesign", Estes Park, Colorado, 2002, Pages: 67 – 72.

[25] "Synthesis and optimization of digital circuits" Giovanni De Micheli, 1993