DESIGN SPACE EXPLORATION: COMPARAITIVE STUDY OF SIMULATED ANNEALING AND PARTICLE SWARM OPTIMIZATION

ELG6158
Digital Systems Architecture

Professor M. Bolic

Vishal Thareja
2964965

Monday December 3, 2007

Abstract

As the complexity of designs implemented in embedded system increases, there needs to be some sort of exploration of hardware configurations to find the best design that will meet the design constraints. Design Space Exploration is an exploration technique should by many to find the best hardware configuration to fit in some design constraints. In this project, a popular algorithm, Simulated Annealing and a relatively new algorithm, Particle Swarm Optimization are analyzed and implemented to perform space exploration on a Fuzzy Logic Coprocessor. Results from both algorithms are observed and compared to suggest a more efficient algorithm. The Fuzzy Logic Coprocessor is a small design with very low complexity. This makes the analysis of Simulated Annealing and Particle Swarm Optimization much easier to understand and then with a strong foundation, the problem complexity can increase and more results can be gathered from the algorithms.

Table of Contents

Table of Figures

Table of Tables

Introduction

Design Space Exploration (DSE) is a technique used by designers to determine the most effective hardware configuration to meet the design constraints. The most effective combination of components may be the utilization of 4 pipeline stages instead of 5, using a parallel adder instead of a serial one, etc. The design constraints may include system performance measures such as power consumption, data throughput rate, or chip area. Design constraints may also be a combination of system performance measures.

Nowadays DSE has evolved from designers manually performing the technique of determining the best hardware configuration to automated design tools which present the designers with reports and analysis of the DSE on a design. These tools allow the designer to select the design most suitable to meet the design constraints. Some of these automated design tools are: CoWare's Platform Designer, CoWare Processor Designer, Celoxica's DK Design Suite, and Tensilica's Xtensa. These tools allow designers to design their systems and once compiled, these tools return reports and analysis of DSE and presents a variety of options for the designer to select to meet their specific design constraints.

Although many tools are available, DSE still remains a challenge when designing for embedded systems. In embedded system design it is crucial to have a good hardware configuration and meet the design constraints in the early stages of the design flow; otherwise more cost and time will be wasted on redesigns for a system that does not meet the constraints. The main challenge here is to identify all possible hardware configurations and select one configuration for implementation. [Anderson06]

Motivation

The motivation of this project is to understand the techniques of DSE and learn how to apply it to hardware designs. A literature review was used as an aid in the analysis of what algorithms are used today and what problems are solved. The literature review pointed out a popular algorithm used to perform DSE, Simulated Annealing (SA). Academic examples helped further understand why and how SA was used. A further analysis of DSE lead to another algorithm called Particle Swarm Optimization (PSO). This is a relatively new optimization algorithm used to solve many different problems. However, in terms of DSE, there have only been a few successfully implementations of PSO.

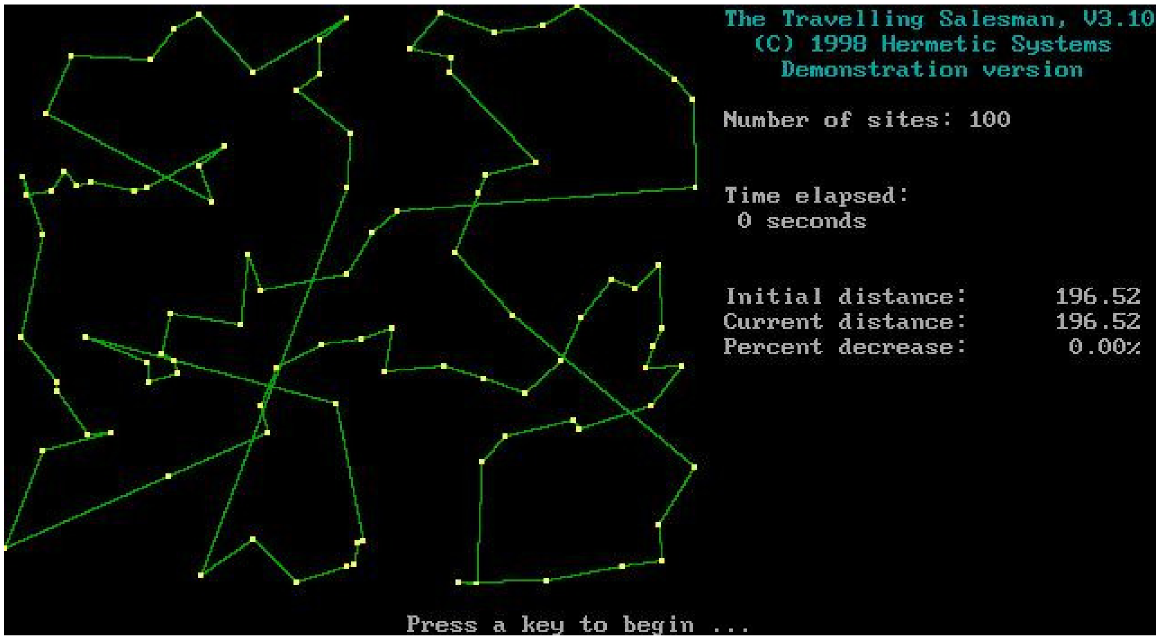
Comparing these algorithm will aid in the understanding of what algorithm should be used. The comparison can involve algorithm performance, minimum time required to find the solution, or how good the solution is as the problem complexity increases. As a stepping stone, the algorithms will be applied to a Fuzzy Logic (FL) Coprocessor. Furthermore, within the coprocessor, there are a number of modules that may be implemented in one of several ways. For example, a multiplier may be implemented as a hardware multiplier or you can use the onboard multiplier within the Field Programmable Gate Array (FPGA). The DSE technique determines which way the multiplier should be implemented. Parameters such as execution time and area are associated with both the hardware multiplier and the onboard multiplier. With the parameters, the algorithms can mathematically select the best implementation depending on constraints. The task here is DSE and for much larger problems, the difficulty increases. Therefore, in order to understanding the strengths and weaknesses of both algorithms we will apply them to the coprocessor problem. The two design constraints that will be used in this project are finding the minimum execution time and minimum area cost hardware configurations.

Prior Art

Simulated Annealing

Simulated Annealing (SA) was first presented in 1983 by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi and again in 1985 by V. Cerny. This algorithm was a generalization of the Monte Carlo method invented by N. Metropolis et al in 1953 and was used for examining n-body systems with equations of states. [Kirkpatrick83]

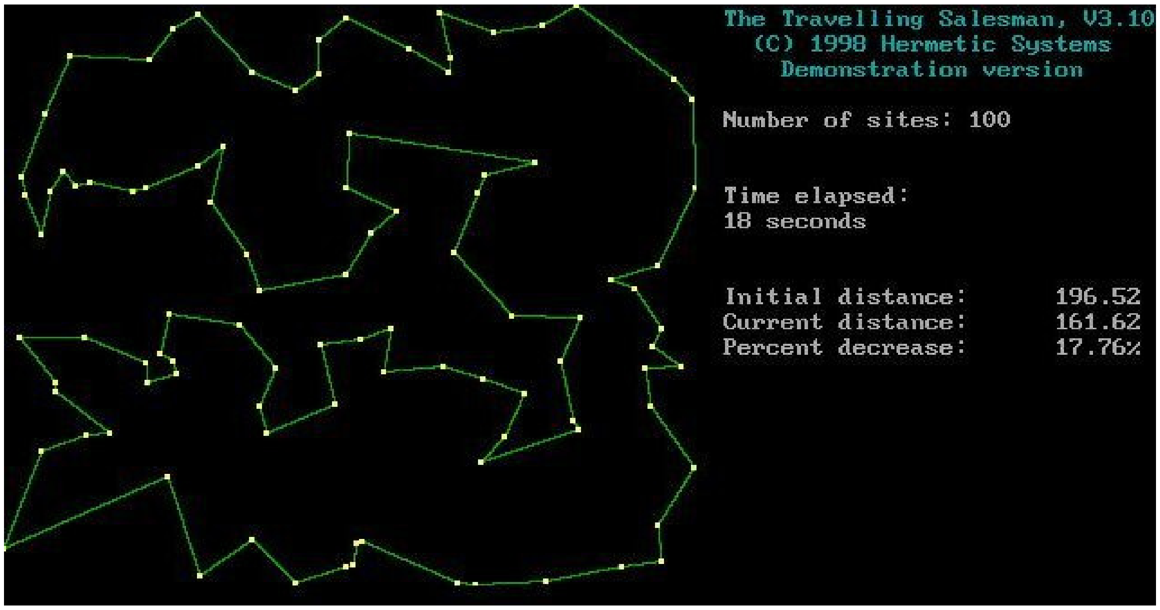
SA is a probabilistic algorithm and works by determining a good approximation of the global optimum given a function and design space. To illustrate how the algorithm works, it will be applied to the traveling salesman problem. In this problem, the salesman has to visit a number of cities while minimizing the total mileage traveled. To start, let's assume a random itinerary (random ordered cities) as shown in Figure 1, and then perform pairwise trades the order of the cities to visit, hoping to reduce the total mileage traveled.



```
The Travelling Salesman, V3.10
  (C) 1998 Hermetic Systems
    Demonstration version

Number of sites: 100

Time elapsed:
 0 seconds

Initial distance:      196.52
Current distance:      196.52
Percent decrease:       0.00%

Press a key to begin ...
```

Figure 1: Traveling Salesman Problem. [Mayer98]

If changing the order results to a shorter path, accept the change. If the change yields a longer path, apply a probability of accepting the change. This probability will likely be less than the proposed increase in path length. Over time, this probability reduces and rules out shorter path increases. Thus, in the end the minimum path is likely to be determined. An example of a minimum path from Figure 1 is shown in Figure 2. [Carr05]



```
The Travelling Salesman, V3.10
  (C) 1998 Hermetic Systems
    Demonstration version

Number of sites: 100

Time elapsed:
18 seconds

Initial distance:       196.52
Current distance:       161.62
Percent decrease:       17.76%
```

Figure 2: Traveling Salesman Problem Solved. [Mayer98]

A practical example of SA used in DSE is presented in [Saraswat07]. The simulated annealing algorithm was utilized in order to determine the best VLIW (Very Long Instruction Word) architecture for a Global System for Mobile communications (GSM) Decoder application. Both SA and Exhaustive Search (ES) were utilized and compared against each other. With a defined cost function (design constraints) and design parameters (design space), both algorithms were tested. Along with the cost function, the associated costs for certain components were also identified and shown in Figure 3 and 4 respectively.

Parameters	*Cost*
Issue width	4
ALU/cluster	2+1 = 3
Mpy/Cluster	4+2 = 6
MemLoad	4+3 = 7
MemStore	(same as memload) 4+3 = 7
Memory Ports/ Cluster	5
I-cache	8
D-cahce	8

Figure 3: GSM Decoder parameters and costs. [Saraswat07]

```
The cost =
  (2^ICacheSize x costIcache) + (2^DCacheSize x costDcache)
+ (costALU x numALUs x numClusters)  +  (costMpy x numMpy x numClusters)
+ (issueWidth x Registers x costRegisters x numClusters)  +  (numMemStores x costMemStores)
+ (numMemLoads x costMemStores ) +  (memPorts x  costMemPorts x numClusters)
```

Figure 4: GSM Decoder cost function. [Saraswat07]

The design constraint was to achieve minimum execution time and minimum cost in hardware. SA determined the absolute minimum in approximately 2.4 hours, exploring 192x4 total permutations, whereas ES determined the absolute minimum in 22 hours executing in parallel on 2 machines exploring the same number of total permutations. [Saraswat07]

Particle Swarm Optimization

The particle swarm optimization (PSO) algorithm was first introduced by James Kennedy and Russell Eberhart in 1995. Initially, this algorithm was to be used a simulation of social systems such as flocks of birds or schools of fish. The goal was to understand these social systems behaviour and how they function with such precision and accuracy. The first simulations of these systems incorporated nearest-neighbor velocity and multidimensional search and distance. By understanding these simulations carefully, it was deduced that this model of social systems was in fact an optimizer. [Eberhart01]

To understand how the PSO algorithm works, an N-variable optimization problem will be discussed. Given a population-based problem space, a swarm of particles are assigned random positions such that each particle's position represents a possible solution in the solution space. Depending on the position of each particle in the solution space, a solution will be presented and it will show how well the algorithm solved the problem. The particles will start to swarm the solution space, while remembering their own personal best position. This personal best is called *pbest* of the solution space.

Furthermore, each particle knows the best position found by any particle in the swarm. This is called *gbest* of the solution space. During the iterations, the particles slowly swarm around the global best. Each particle will observe its own personal best and the global best, and then adjust itself to try to meet the global best. Once the particles start settling near the global best, the optimization problem is solved. [Boeringer04]

A detailed breakdown of the PSO algorithm is as follows [Eberhart01]:

1. Initialize population of particles with random positions and velocities of the problem space.
2. Compare each particle's fitness with particle's *pbest*. If current value is better, then set *pbest* value to the current value.
3. Compare each particle's fitness with all particles *gbest*. If any particle's current value is better, then set to current particle's array index and value.
4. Change each particle's velocity and position.
5. Loop to step 2 until a criterion is met.

Many researchers have used the PSO algorithm for optimizing many different problems. One example is presented in [Farmahini-Farahani07] deals with hardware/software partitioning using the PSO algorithm. The goal to achieve here was to perform hardware/software partitioning of a given task graph with area and performance constraints. The PSO algorithm was enhanced and compared against Genetic Algorithm

(GA). Results from both algorithms were compared against each other on the criteria of optimal solution, executing time, and overall performance.

In [Farmahini-Farahani07], the problem was defined to be given a set of hardware and software parameters, PSO must partition the given software to achieve optimal solution. To solve the partitioning problem, 4 approaches were defined:

1. Given hardware and software constraints, partition software such that the hardware cost should be less than the hardware constraint and the execution time should be less than the software constraint.
2. Given a hardware constraint, partition software such that hardware cost should be less than hardware constraint and have minimum execution time.
3. Given a software constraint, partition software such that execution time should be less than software constraint and have minimum hardware cost.
4. Partition software such that minimum execution time and minimum hardware cost is achieved.

Given the defined approaches to solve the hardware/software partitioning problem, the problem was represented as a task graph with given hardware costs, hardware execution time, and software execution time.

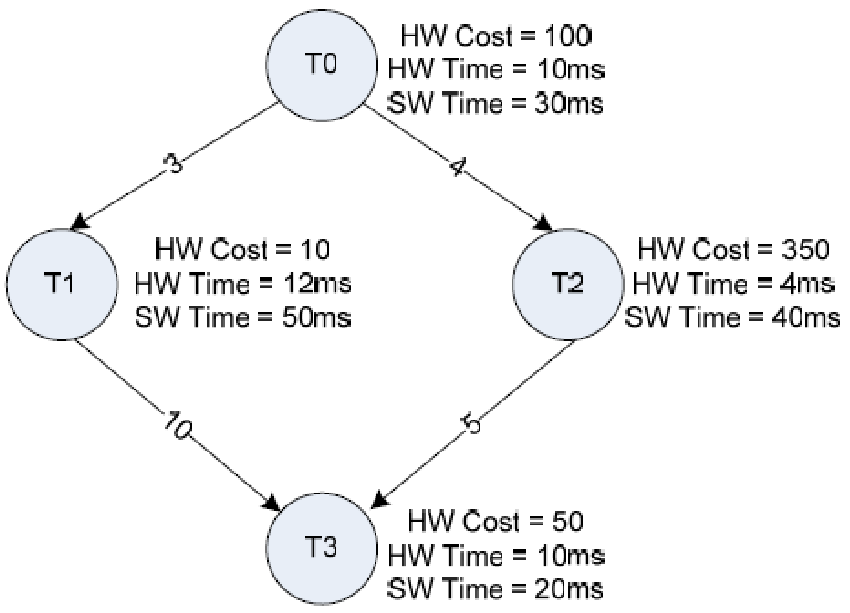


Figure 5: Typical Task Graph for Hardware/Software partitioning. [Farmahini-Farahani07]

Utilizing both PSO algorithm and GA algorithm, the results after 20 trails and over 1000 generations showed that PSO out-preformed GA in all test cases. The run-time of GA and PSO were equal. The PSO algorithm found better hardware/software partitions than GA in a fixed run time. For fixed number of generations, GA converged faster than PSO to determine the optimal solution. [Farmahini-Farahani07]

Problem: Fuzzy Logic Coprocessor

For the purposes for this project, DSE will be performed on a Fuzzy Logic (FL) Coprocessor. The FL coprocessor is designed to work with a Reduced Instruction Set Computer (RISC) with custom FL instructions. The FL coprocessor supports a 2 input / 1 output FL algorithm with a Mamdani-type inference engine with no limit to the number of inference rules. The FL coprocessor only supports FL algorithm with triangular and trapezoid input functions. The FL coprocessor considered implements a FL inverted pendulum problem with 2 inputs, 25 inference rules, and 1 output. [Thareja07]

The DSE problem that has to be solved is to find a configuration of the FL coprocessor that yields only minimum execution time, only minimum area costs, or minimum execution time and area. For example, if we want a configuration to have minimum execution time, we would assume to select all implementations with the minimum execution time. In the implementation section, all possible different implementations and associated execution time and area parameters will be discussed.

Overview of FL Coprocessor

The FL coprocessor is composed for 3 stages; fuzzification stage, inference rules stage; and defuzzification stage. The fuzzification stage converts a real input value to a crisp value. The inference rules stage evaluates the crisp value to a set of rules. The defuzzification stage converts the result from the inference rule stage back to a real output value. The overview of the FL coprocessor can be seen in Figure 6.

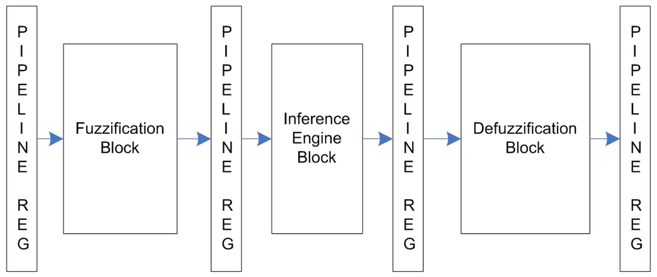


Figure 6: FL Coprocessor Overview. [Thareja07]

Fuzzification Stage

The fuzzification block, shown in Figure 7, is the first module inside the FL coprocessor where the crisp value is mapped to a fuzzy input value. For a two-input FL algorithm, the FL coprocessor will read a fuzzy instruction with 2 crisp values from the processor. This fuzzy instruction is a custom instruction which the FL coprocessor will read and execute. The FL coprocessor will “snoop” on the processor when instructions are read. Once a fuzzy instruction is read, the coprocessor will begin executing the instruction.

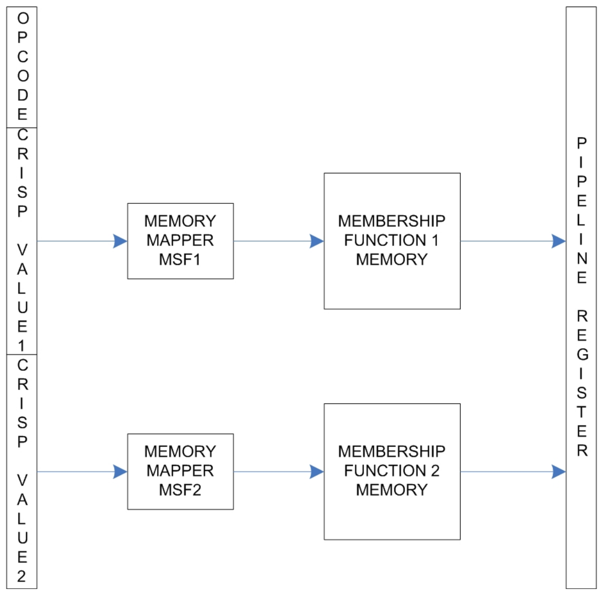


Figure 7: FL Coprocessor Fuzzification Stage. [Thareja07]

Since the crisp values can be negative or positive, a memory mapper component is needed to map the signed crisp value to an unsigned logical address that will be used as an index for the membership function memories. Once the address is determined, the address is used to retrieve the respective fuzzy input value from the specific membership function memory.

Inference Rules Stage

The inference engine block, shown in Figure 8, is where the inference rules are stored and valued against the fuzzy input values. Before the evaluation can take place, all possible fuzzy input values need to be determined. This is because FL works on vagueness, for example input 1 can be between 1 and 0. So, to clear the vagueness for the inference rules, the combinations need to be determined. For a general 2 input / 1 output FL controller, there can only be a maximum of 4 possible combinations. All the combinations are determined by the combination block inside the FL coprocessor. Once the combinations are determined, the next step is to evaluate each combination against the inference rules engine.

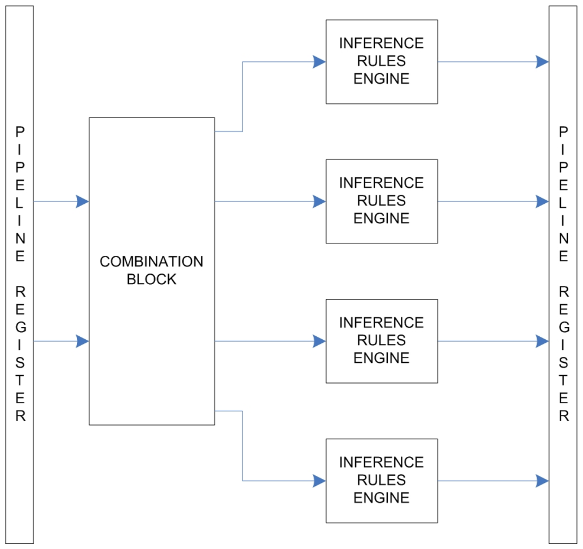


Figure 8: FL Coprocessor Inference Rules Stage. [Thareja07]

Defuzzification Stage

The third and final block of the FL co-processor is the defuzzification block. This block contains all of the ALU components and the complex arithmetic units to determine the crisp output value when using the Centre of Gravity (COG) method. Once the outputs from the inference engine block are read from the pipeline register between the inference rules stage and the defuzzification stage as shown in Figure 9, the weight strengths need to be determined through the weight mapper component. As discussed earlier, the weight strengths are determined by observing where each fuzzy subset is at its maximum. The weight mapper component reads in all 4 fuzzy outputs in parallel and output their respective weight strengths in parallel.

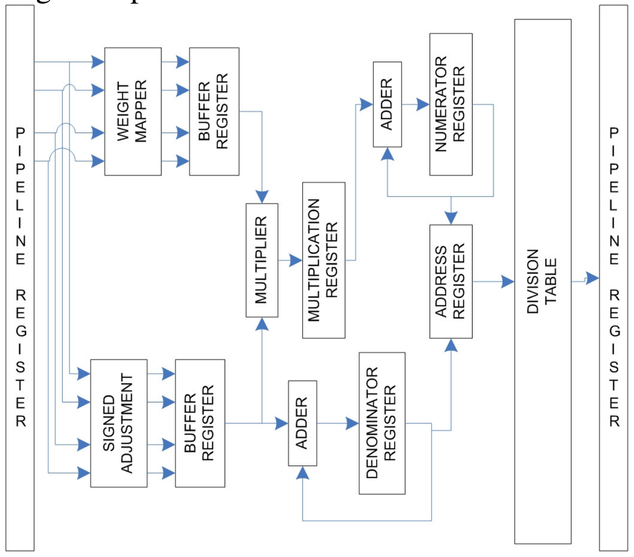


Figure 9: FL Coprocessor Defuzzification Stage. [Thareja07]

To determine the crisp value, the following COG formula is used:

$$FD = \frac{\sum \mu S}{\sum \mu}$$

Figure 10: FL Coprocessor Fuzzy Decision Equation. [Thareja07]

This equation represents the fuzzy decision (FD), where μ is the fuzzy output from the inference engine block and *S* is the weight strengths stored in the weight mapper component shown in Figure 9.

The first step is to determine the numerator of the equation. The numerator is the summation of the multiplication of each output value and their respective weight value. This is also known as Sum-of-Products (SOP). To perform this procedure in hardware, each output value is sequentially read from the buffer register and then multiplied. The result from this multiplication is stored in a multiplication register. Next, the value from the multiplication results is added together with the contents of the numerator register. Initially, the numerator register contains zero. This will repeat until the buffer registers are empty.

At the same time the numerator is determined, the denominator is determined in parallel. The denominator from the above equation is the summation of all the output values from the inference engine. To perform this procedure, an adder and accumulator registers are required. For example, a 2 input / 1 output FL algorithm will require 4 iterations to determine the output crisp value.

Implementations

Looking at the architecture of the FL coprocessor, you may suggest a different implementation for certain components. A different implementation may yield less execution time, lower area, lower power consumption, etc. To suggest different implementation, the FPGA platform for the FL coprocessor must be discussed to understand the physical constraints of the FL coprocessor once on chip. Originally, the FL coprocessor was designed using Celoxica's DK Design Suite and Celoxixa's RC10 Evaluation Board. The specifications of the FPGA on the RC10 are illustrated below in Table 1.

	System Gates	Logic Cells	18x18 Multipliers	Block RAM Bits	Distributed RAM Bits	DCMs	I/O Standard	Max Different I/O Pairs	Max Single Ended I/O
Xilinx Spartan 3 XC3S1500	1500K	29,952	32	576K	208K	4	24	221	487

Table 1: Xilinx Spartan 3 XC3S1500 Specifications. [Xilinx05]

To discuss all the components inside the FL coprocessor that may be implemented differently, we will look at each stage individually. Some implementations were determined through the Xilinx ISE and Xilinx Core Generator.

The first stage to be analyzed is the fuzzification stage. Referring back to the section on the fuzzification stage and Figure 7, the membership function memories can be implemented differently. The purpose of the membership function memories is to map an input real value to a fuzzy input value. This mapping can be implemented using 2 Single Port ROM membership function memories, 1 Dual Port ROM membership memory, or utilize the onboard Block RAM memory within the FPGA. Table 2 illustrates the execution time and area costs for each implementation.

Implementation	Execution time (cycles)	Area (gates)
2 Single Port Memories	1	131075
1 Dual Port Memory	1	65539
Onboard Block RAM	1	0

Table 2: Fuzzification Stage Implementations.

The first implementation, 2 Single Port Memories is what is shown in Figure 7. Since the FL coprocessor only supports a 2 input / 1 output FL algorithm, 2 Single Port Memories were selected to map each membership functions crisp input to a fuzzy input. The 2 Single Port Memories hold the mappings of membership function 1 and 2 respectively.

The second implementation, 1 Dual Port Memory can also be used to implement the mapping of crisp input to a fuzzy input. The block diagram of this implementation is shown in Figure 11.

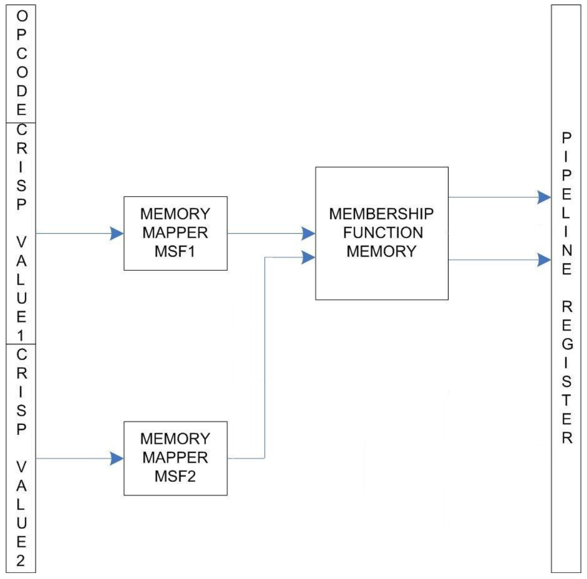


Figure 11: Fuzzification Stage Dual Port Memory Implementation.

Using the 1 Dual Port Memory, the memory can read in 2 address at the same clock edge and output the contents of that address the next cycle at the same time. When using either the Single Port Memory or Dual Port Memory, the memory cells are not considered in the area costs. This is because the synthesis tools use the FPGA Block RAM Bits for the logical cells. What is considered in the area costs is the hardware responsible to correctly select the logical cell. This is why the Dual Port Memory area cost is lower than the 2 Single Port Memories implementation because less hardware.

The last implementation is using the Onboard RAM block. The Xilinx Spartan 3 XC3S1500 has built in the hardware to implement a memory. This cuts down the area cost because it is not considered by the synthesis tool since no additional hardware is needed to use the Onboard RAM Block. Analyzing the execution time for all, the execution time is the same because memories usually have 1 cycle latency.

The second stage to be analyzed is the Inference Rules stage. Referring back to the section on the Inference Rules stage, there aren't any alternate ways to implement the comparison of fuzzy value to inference rules. Therefore, this stage is ignored complete in the DSE problem.

The third stage to be analyzed is the Defuzzification stage. Inside this stage, the two major components where different implementations can be used are the multiplier and the divider. Referring back to the overview of the Defuzzification stage, the multiplier and divider are used for determining the crisp output value. The multiplier can be implemented as a hardware multiplier or using the onboard multiplier. The divider can be implemented as a hardware divider, hardware pipelined divider, or as an onboard lookup table implemented as a memory. Table 3 illustrates the execution time and area costs for each implementation.

Implementation	Execution Time (cycles)	Area (gates)
Hardware Multiplier	18	4581
Onboard Multiplier	3	0
Hardware Divider	49	6209
Hardware Pipelined Divider	28	5833
Onboard Block RAM	1	1000000

Table 3: Defuzzification Stage Implementations.

The multiplier can be implemented as either a Hardware Multiplier or as an Onboard Multiplier. The Hardware Multiplier, designed and implemented using Xilinx ISE and Xilinx Core Generator, is a RTL based 12x12 multiplier, thus having an area of 4581 gates. The execution time of a RTL based multiplier changes as the input size changes, thus the 12x12 multiplier has a execution time of 18 cycles. The Xilinx Spartan 3 XC3S1500 supports 32 18x18 multipliers that also be used. Therefore, there is no area cost since it is already designed inside the FPGA and the execution time is 3 cycles.

The divider can be implemented as a Hardware Divider, Pipelined Hardware Divider, or as a look up table implemented as a Single Port Memory. The Hardware Divider and the Pipelined Hardware Divider were implemented using Xilinx ISE and Xilinx Core Generator. The major different between the two is one is pipelined, thus reducing the critical path and the overall latency. The divider component can also be implemented as a lookup table, where all possible division values are stored. This is not a very attractive implementation since the possibilities are end-less. This is why the area of the look up table implemented as a Single Port Memory is 1000000 gates. Actually, the size is infinity, but for the algorithms to solve the DSE problem, a numerical value is needed.

Design Considerations and Assumptions

The DSE problem itself is a very complex problem. This is why certain design considerations and assumptions were made to simplify the DSE problem. Obviously, these considerations and assumptions cannot keep very long since there are more and more complex problems. But, simplifying the problem will establish a starting point.

From the above implementations section, one can realize that there are 18 different implementations when mixing and matching the different implementations together. Again, the problem DSE will try to solve is to determine the best hardware configuration that yields minimum execution time, minimum area costs, or a tradeoff between execution time and area costs.

Assumption # 1

To simplify the DSE problem, the entire FL coprocessor datapath is not considered. To elaborate further, components that do not have alternate implementations will be removed from the problem because the associated execution time and area costs are static and will only create unnecessary overhead for DSE. To illustrate the only components that have alternate implementations, Figure 12 will show a simple data flow graph (DFG) of the FL coprocessor with alternate implementations.

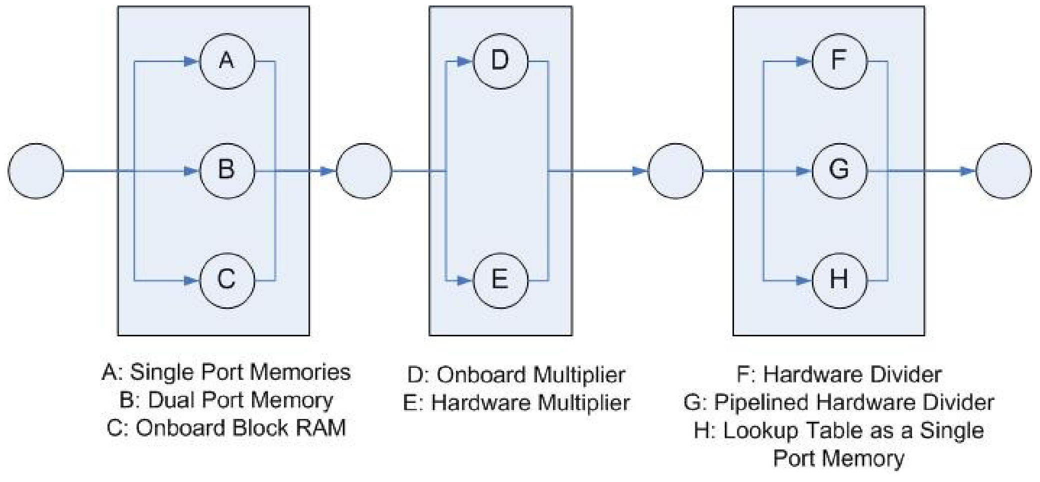


Figure 12: Data Flow Graph of FL Coprocessor Implementations.

Assumption # 2

Since the purpose of this project is to do a comparative analysis of both SA and PSO, the problem input must be the same for both algorithms. SA requires an input matrix that represents all possible solutions whereas PSO requires an input matrix that represents a solution space with all possible solutions. Using Figure 12, a matrix can be derived that represents all possible solutions.

$$\begin{bmatrix} A & D & F \\ A & D & G \\ A & D & H \\ A & E & F \\ A & E & G \\ A & E & H \\ B & D & F \\ B & D & G \\ B & D & H \\ B & E & F \\ B & E & G \\ B & E & H \\ C & D & F \\ C & D & G \\ C & D & H \\ C & E & F \\ C & E & G \\ C & E & H \end{bmatrix}$$

Figure 13: SA and PSO Input Matrix

Figure 13 shows all possible combinations of alternate implementations that make up a hardware configuration. Now, associated with the different implementations are respective execution time and area costs.

SA will utilize 2 matrices to solve DSE, whereas PSO will only use 1 matrix. SA will use 1 matrix that only stores execution times and 1 matrix that only stores area costs. On the other hand, PSO will only use 1 matrix that represents both execution time and area costs.

Assumption # 3

Based on assumption # 2, the following matrices for SA and PSO are shown in Figure 14.

$$\begin{bmatrix} 1 & 3 & 1 \\ 1 & 3 & 28 \\ 1 & 3 & 49 \\ 1 & 18 & 1 \\ 1 & 18 & 28 \\ 1 & 18 & 49 \\ 3 & 3 & 1 \\ 3 & 3 & 28 \\ 3 & 3 & 49 \\ 3 & 18 & 1 \\ 3 & 18 & 28 \\ 3 & 18 & 49 \\ 2 & 3 & 1 \\ 2 & 3 & 28 \\ 2 & 3 & 49 \\ 2 & 18 & 1 \\ 2 & 18 & 28 \\ 2 & 18 & 49 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1000000 \\ 0 & 0 & 5833 \\ 0 & 0 & 6209 \\ 0 & 4581 & 1000000 \\ 0 & 4581 & 5833 \\ 0 & 4581 & 6209 \\ 65539 & 0 & 1000000 \\ 65539 & 0 & 5833 \\ 65539 & 0 & 6209 \\ 65539 & 4581 & 1000000 \\ 65539 & 4581 & 5833 \\ 65539 & 4581 & 6209 \\ 131075 & 0 & 1000000 \\ 131075 & 0 & 5833 \\ 131075 & 0 & 6209 \\ 131075 & 4581 & 1000000 \\ 131075 & 4581 & 5833 \\ 131075 & 4581 & 6209 \end{bmatrix} \begin{bmatrix} 5 & 1000000 \\ 32 & 5833 \\ 53 & 6209 \\ 20 & 1004581 \\ 47 & 10414 \\ 68 & 10790 \\ 7 & 1065539 \\ 34 & 71372 \\ 55 & 71748 \\ 22 & 1070120 \\ 49 & 75953 \\ 70 & 76329 \\ 6 & 1131075 \\ 33 & 136908 \\ 54 & 137284 \\ 21 & 1135656 \\ 48 & 141489 \\ 69 & 141865 \end{bmatrix}$$

Figure 14: SA and PSO Input Matrices.

From Figure 14, the left-most matrix represents the execution time; the middle matrix represents the area costs; and the right-most matrix represents both execution time and area costs. The first 2 matrices are used for SA and the last matrix is used for PSO.

Since in the fuzzification stage, all alternate implementations yield an execution time of 1 cycle, to differentiate between the implementation, Single Port Memories are labeled as 1 cycle; Dual Port Memory is labeled as 3 cycles; and Onboard Block RAM is labeled as 2 cycles. Obviously, these are not the actual execution times, but for clarification and debugging this assumption helps.

Assumption # 4

The tradeoff between execution time and area costs must also be considered in the DSE problem. This is so because not all DSE problems can be solved by just determining a hardware configuration for minimum execution time or minimum area costs. There must exist a tradeoff because then full advantage of a platform's performance and capacity. In order to achieve this, there must be a conversion factor. Since both execution time and area costs have different units, a conversion factor is necessary to convert one matrix into the others units and combine them to form 1 matrix.

The conversion factor selected for this project is to equate clocks to gates. For example, equating 1 clock cycle to X number of gates will give the conversion factor and convert the execution time matrix to gates and then combined with the area cost matrix. Then DSE can compute on this matrix and find the best hardware configuration.

$$\begin{bmatrix} 0 & 0 & 1000000 \\ 0 & 0 & 5833 \\ 0 & 0 & 6209 \\ 0 & 4581 & 1000000 \\ 0 & 4581 & 5833 \\ 0 & 4581 & 6209 \\ 65539 & 0 & 1000000 \\ 65539 & 0 & 5833 \\ 65539 & 0 & 6209 \\ 65539 & 4581 & 1000000 \\ 65539 & 4581 & 5833 \\ 65539 & 4581 & 6209 \\ 131075 & 0 & 1000000 \\ 131075 & 0 & 5833 \\ 131075 & 0 & 6209 \\ 131075 & 4581 & 1000000 \\ 131075 & 4581 & 5833 \\ 131075 & 4581 & 6209 \end{bmatrix} + (3) \bullet \begin{bmatrix} 1 & 3 & 1 \\ 1 & 3 & 28 \\ 1 & 3 & 49 \\ 1 & 18 & 1 \\ 1 & 18 & 28 \\ 1 & 18 & 49 \\ 3 & 3 & 1 \\ 3 & 3 & 28 \\ 3 & 3 & 49 \\ 3 & 18 & 1 \\ 3 & 18 & 28 \\ 3 & 18 & 49 \\ 2 & 3 & 1 \\ 2 & 3 & 28 \\ 2 & 3 & 49 \\ 2 & 18 & 1 \\ 2 & 18 & 28 \\ 2 & 18 & 49 \end{bmatrix} = \begin{bmatrix} 3 & 9 & 1000003 \\ 3 & 9 & 5917 \\ 3 & 9 & 6356 \\ 3 & 4635 & 1000003 \\ 3 & 4635 & 5917 \\ 3 & 4635 & 6356 \\ 65548 & 9 & 1000003 \\ 65548 & 9 & 5917 \\ 65548 & 9 & 6356 \\ 65548 & 4635 & 1000003 \\ 65548 & 4635 & 5917 \\ 65548 & 4635 & 6356 \\ 131081 & 9 & 1000003 \\ 131081 & 9 & 5917 \\ 131081 & 9 & 6356 \\ 131081 & 4635 & 1000003 \\ 131081 & 4635 & 5917 \\ 131081 & 4635 & 6356 \end{bmatrix}$$

Figure 15: Execution Time and Area Costs Tradeoff

For example, from Figure 15, if the conversion factor is 3, (1 clock cycle = 3 gates), then the final matrix can be determined where DSE can be applied and the best hardware configuration can be determined.

Performing DSE on FL Coprocessor

With all the different implementations of components determined and discussed all the appropriate assumptions, how the algorithms will work can now be discussed. In this section, Simulated Annealing and Particle Swarm Optimization will be discussed with pseudo code and explanation of variables.

Simulated Annealing

Pseudo Code

The SA algorithm works as follows:

1. Compute randomly next position.
2. Determine the difference between the next position and current position, call this different delta.
3. If delta < 0, the assign the next position to the current position.
4. If delta > 0, then compute the probability of accepting the random next position.
5. If the probability is < the e^(-delta / temperature), then assign the next position to the current position.
6. Decrease temperature by a factor of alpha.
7. Loop to step 1 until temperature is not greater than epsilon.

The explanation of variables is illustrated as follows:

- Current position
    - This variable represents the current solution
- Next position
    - This variable represents the next possible solution
- Delta
    - This variable represents the difference between the next solution and current solution.
- Probability
    - This variable is a function where determines the probability of accepting the next solution or not.
- Temperature
    - This variable represents the number of iterations.
    - In terms of annealing, the temperature represents the temperature of the metal and after each iteration, the temperature is cooled, or decreased.
- e^(-delta / temperature)
    - This expression is compared against the probability to determine if the next solution should be accepted.
- Alpha
    - This variable is responsible for decrementing the temperature by a factor.
    - In terms of annealing, this factor demonstrates the cooling process after each iteration.

Particle Swarm Optimization

Pseudo Code

The PSO algorithm works as follows:

1. Initialize population of particles with random positions and velocities of the problem space.
2. Compare each particle's fitness with particle's *pbest*. If current value is better, then set *pbest* value to the current value.
3. Compare each particle's fitness with all particles *gbest*. If any particle's current value is better, then set to current particle's array index and value.
4. Change each particle's velocity and position.
5. Loop to step 2 until a criterion is met.

The explanation of variables is illustrated as follows:

- Initial population
    - This variable represents the initial solution at the start of the algorithm.
    - Random positions and velocities makeup the initial population.
- Position
    - This variable represents the position of a particle inside the solution space.
- Velocity
    - This variable represents the velocity of a particle inside the solution space.
- pBest
    - This variable represents a particle best position inside the solution space.
- gBest
    - This variable represents the best position inside the solution space.
- Criterion
    - This represents a condition that must be satisfied for the algorithm to stop.
    - This can be number of iterations or a required solution.
- Inertia weight factor (w)
    - Provides a balance between global and local exploration.
- Acceleration constants (c1, c2)
    - Represents how fast or slow should particles meet pBest and gBest.
- Uniform random numbers {0,1}(r1, r2)
    - Random numbers generated between {0,1}.

$$v_{id}^{k+1} = w.v_{id}^{k} + c_1 r_1 (pbest_{id} - x_{id}^{k}) + c_2 r_2 (gbest_{d} - x_{id}^{k})$$

$$x_{id}^{k+1} = x_{id}^{k} + v_{id}^{k+1}$$

Figure 16: Particle Swarm Optimization Distance and Velocity. [Farmahini-Farahani07]

Equation Vid represents the velocity of each particle and Xid represents the current position of the particle in the solution space.

Results

The results from the Simulated Annealing and Particle Swarm Optimization algorithms will be illustrated in this section. Results from minimum execution time, minimum area costs, and execution time and area costs tradeoff are shown.

Simulated Annealing

The following algorithm parameters were used to generate the results:

Alpha	0.999
Temperature	1.5
Epsilon	1
Delta	0
Probability	The probability used is The Maxwell-Boltzmann distribution which is the classical distribution function for distribution of an amount of energy between identical but distinguishable particles

Table 4: Simulated Annealing parameters.

Minimum Execution Time

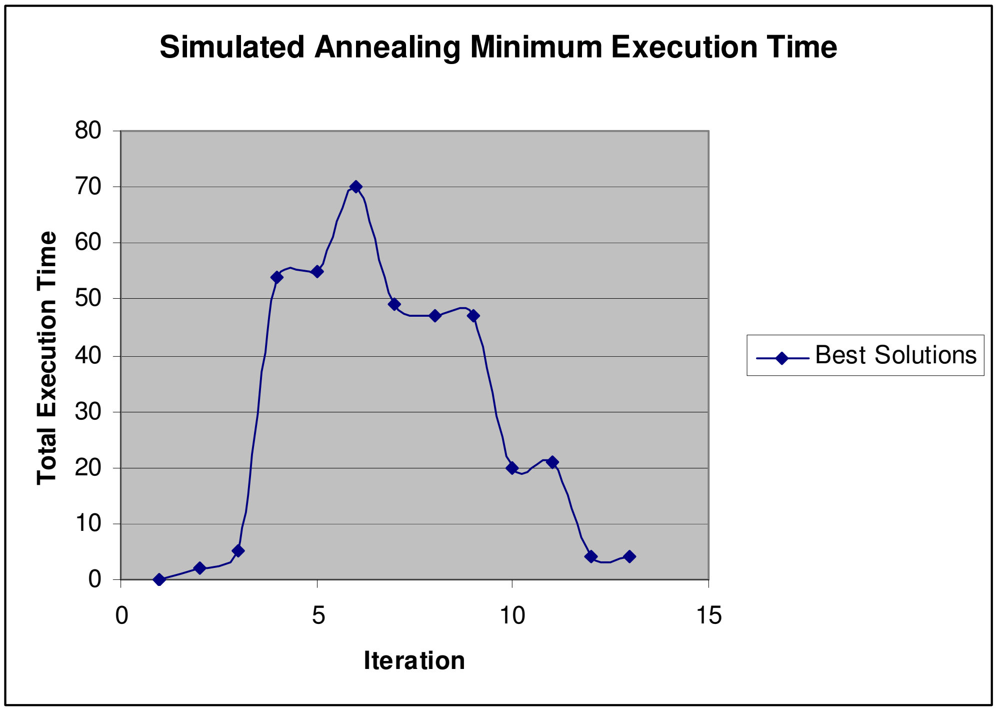


Figure 17: Simulated Annealing: Minimum Execution Time Results

The log file corresponding to Figure 17, from the SA algorithm is shown below:

```
best: 0,0,0,
best: 2,0,0,
best: 2,1,0,
best: 2,1,49,
best: 3,1,49,
best: 3,18,49,
best: 3,18,28,
best: 1,18,28,
best: 1,18,28,
best: 1,18,1,
best: 2,18,1,
best: 2,1,1,
best: 2,1,1,
```

Minimum Area Cost

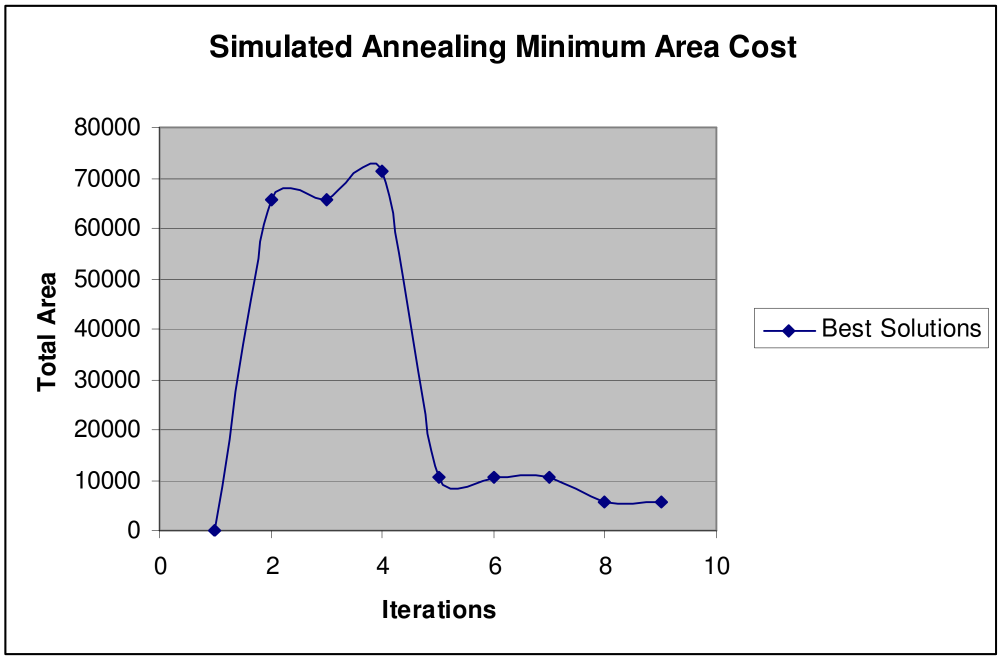


Figure 18: Simulated Annealing Minimum Area Cost

The log file corresponding to Figure 18, from the SA algorithm is shown below:

```
best: 0,0,0,
best: 65539,0,0,
best: 65539,0,0,
best: 65539,0,5833,
best: 0,4581,5833,
best: 0,4581,5833,
best: 0,4581,5833,
best: 0,0,5833,
best: 0,0,5833,
```

Execution Time & Area Cost Tradeoff

The tradeoff factor here is 1 clock cycle = 3 gates.

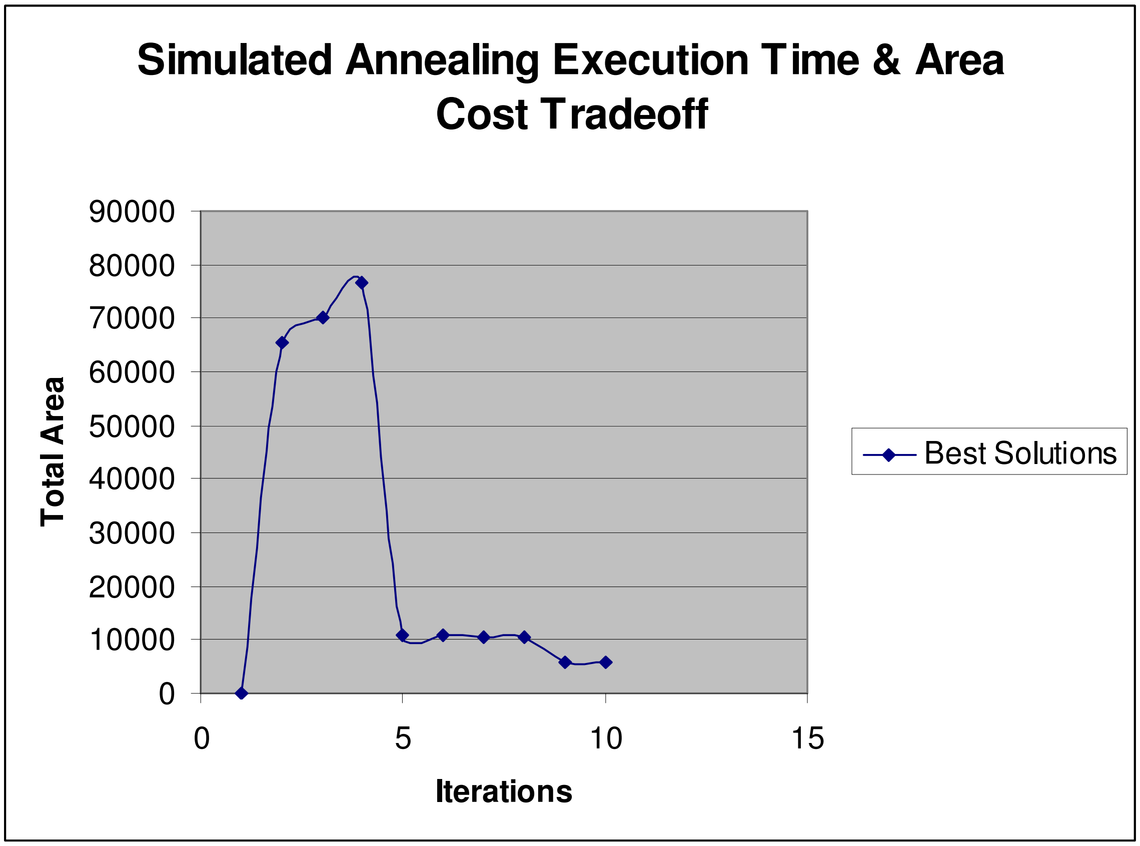


Figure 19: Execution Time & Area Cost Tradeoff

The log file corresponding to Figure 19, from the SA algorithm is shown below:

```
best: 0,0,0,
best: 65548,0,0,
best: 65548,4635,0,
best: 65548,4635,6356,
best: 3,4635,6356,
best: 3,4635,6356,
best: 3,4635,5917,
best: 3,4635,5917,
best: 3,9,5917,
best: 3,9,5917,
```

Particle Swarm Optimization

The following algorithm parameters were used to generate the results:

Inertia weight factor (w)	0
Min and Max weight factor	{0.4,0.9}
Acceleration constants(c1,c2)	{2.0, 2.0}
Min and Max Velocity Bounds	{-1,1}
Number of Particles	9
Criterion (iterations)	406

Table 5: Particle Swarm Optimization parameters.

Minimum Execution Time

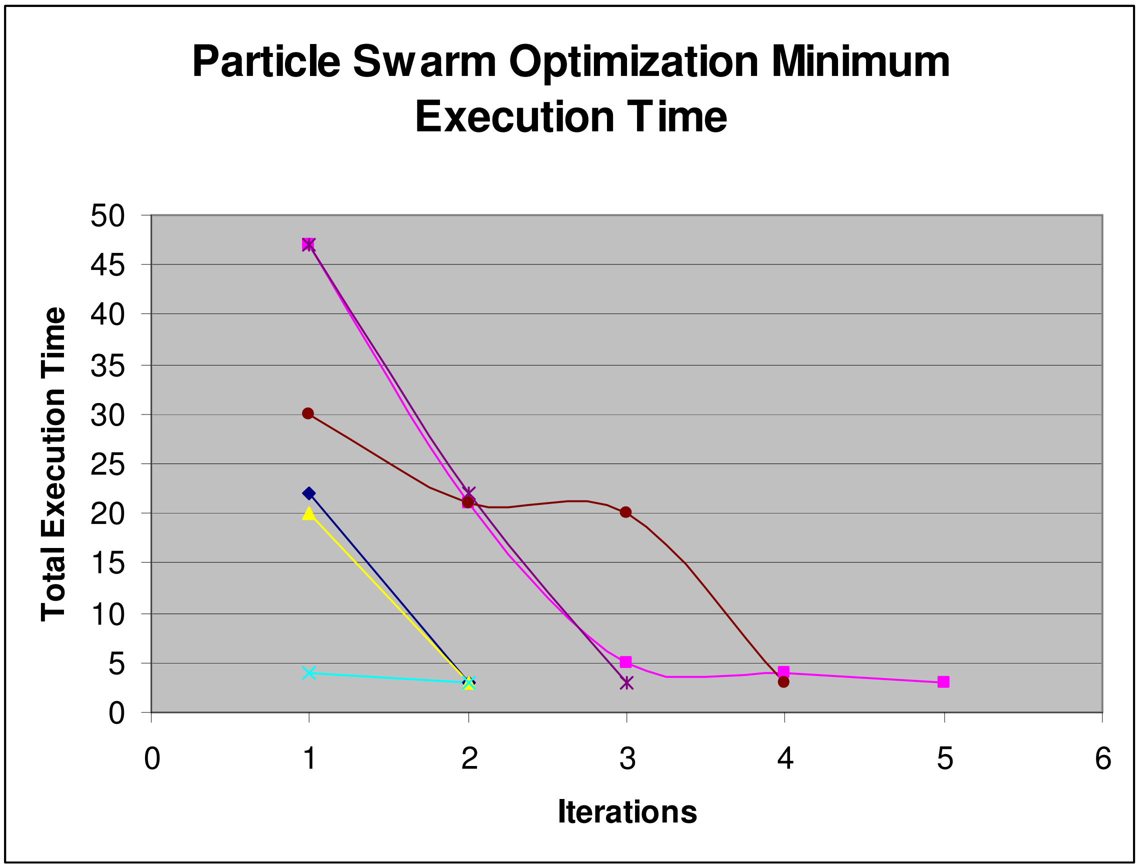


Figure 20: Particle Swarm Optimization Minimum Execution Time Multiple Cases

Minimum Area Cost

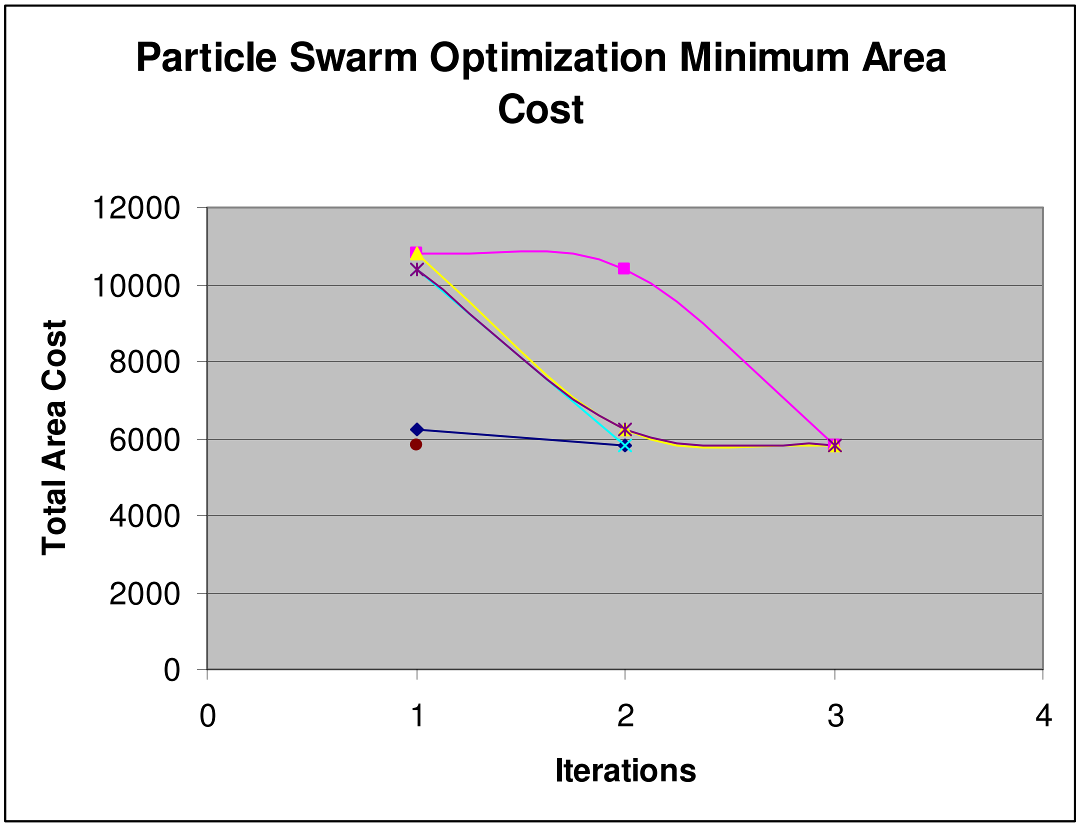


Figure 21: Particle Swarm Optimization Minimum Area Cost Multiple Cases

Discussion

In this section, the results from SA and PSO will be discussed. Furthermore, both algorithms will also be compared against each other to determine the most efficient algorithm.

Simulated Annealing

Minimum Execution Time

When determining the minimum execution time when using Simulated Annealing (SA), the results were found within 13 iterations in Figure 17. When SA is initialized, by default the best solution is 0. However, as random points are selected and compared to the best solution, the best solution changes and begins to converge to a minimum. Initially, the best solution execution time increases. This is probably because of the random selection of points. But soon after the execution time begins to decrease, the curve begins converging to the minimum. In this case, the minimum execution time is 3 cycles, which it finds in 13 iterations. On average, SA takes between 5-9 iterations before finding the minimum. Referring back to Figure 12, the minimum execution time hardware configuration is {A,D,H} or {Single Port Memories, Onboard Multiplier, Lookup Table as a Single Port Memory}.

Minimum Area Cost

Using SA to determine the minimum area cost, the algorithm found the minimum within 9 iterations in Figure 18. Again, initially, SA begins with a best solution of 0 and then begins to iterate through the possible solution set. The algorithm quickly converges to the minimum area cost of 5833 gates in 9 iterations. On average, SA takes between 5-9 iterations before finding the minimum. Referring back to Figure 12, the minimum area cost hardware configuration is {C,D,F} or {Onboard Block RAM, Onboard Multiplier, Hardware Divider}.

Execution Time & Area Cost Tradeoff

The conversion factor here was 1 clock cycle = 3 gates as shown in Figure 19. Using this factor, the execution time matrix was converted an area cost matrix. Then it was added to the area cost matrix, to make SA iterate through the possible solution set. Just like the other examples, the curve starts converging to the minimum within 10 iterations. Referring back to Figure 12, the tradeoff hardware configuration is {C,D,F} or {Onboard Block RAM, Onboard Multiplier, Hardware Divider}.

Particle Swarm Optimization

Minimum Execution Time and Minimum Area Cost

In order to fully understand the output from PSO, multiple cases were plotted to illustrate how the algorithm works. Initially, the algorithm places particles in random positions inside the solution space. Also, each particles personal best is also randomized as well as the global best. By randomizing both the personal best of all particles and the global best makes PSO a bit buggy. Because of the randomness of the personal best and global best, PSO sometimes does not find the minimum. Analyzing Figure 20, PSO determines the minimum execution time relatively quickly, within 1-5 iterations. The hardware configuration determined by PSO for minimum execution time was {A,D,H} or {Single Port Memories, Onboard Multiplier, Lookup Table as a Single Port Memory}.

Moreover, sometimes the particle is randomly placed at the minimum, so the PSO algorithm just outputs the minimum right away. Similar with determining the minimum area cost in Figure 21, the algorithm found the best hardware configuration within 3 iterations and the hardware configuration determined is {C,D,F} or {Onboard Block RAM, Onboard Multiplier, Hardware Divider}.

Simulated Annealing vs. Particle Swarm Optimization

Before plotting the results, the original hypothesis was that PSO would out perform SA. This is so because PSO uses multiple particles within the solution space, whereas SA randomly picks a possible solution inside the solution space and determines if it is better than the current best solution.

Keeping the hypothesis in mind, the results reaffirm it. Figure 17, 18, and 19 illustrate that multiple iterations are required in order to find the minimum. Analyzing Figure 20 and 21, PSO also takes iterations, however much less. On average, SA would like approximately 5-9 iterations, whereas PSO would take 3-5 iterations for this problem. At the worst case, PSO outperforms SA by a factor of approximately 2. Assuming as the problem becomes more complex, PSO would still outperform SA by a factor of 2.

This out performance is very important for complex problems where DSE can take hours or days using SA can easily be reduced by half when using PSO. Furthermore, the PSO results only utilized 9 particles. If the number of particles was increased, the assumption is that the number of iterations would decrease. As a test, 18 particles were used and the PSO algorithm found the minimum within 1 iteration. Again, this is so because with 18 particles within a solution space of 18, each particle is at each location. Therefore, running through the first iteration, the global best would be determined and would automatically be the minimum.

Conclusion

In conclusion, the project of implementing Simulated Annealing and Particle Swarm Optimization for perform Design Space Exploration of the Fuzzy Logic Coprocessor was a success. Performing the original literature review, a understanding was gathered of what is used today in the field of DSE and also new algorithms proposed for DSE. In this case, both SA and PSO were seen as attractive algorithms used by many for performing DSE.

With the understanding of SA and PSO, the parameters for each were also tested and determined to get the best output from the algorithm. The algorithms were implemented in C# with a user interface to select the input and algorithm to use to perform DSE. Further testing and gather results followed, where a trend can be seen that PSO outperforms SA by a factor of approximately 2. This makes PSO a much attractive algorithm to use for DSE when solving large, complex problems.

Future work for performing DSE is to remove some assumptions and modified the algorithms to accept more complex problems. Further testing also needs to be done on the parameter selection for both algorithms to get the best performance.

References

[Anderson06] Anderson, I.D.L.; Khalid, M.A.S.; "Design Space Exploration using Parameterized Cores: A Case Study," in Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE), May 2006, pp.1893-1896.

[Kirkpatrick83] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, 1983, pp. 671-680.

[Mayer98] Mayer. P; “Traveling Salesman Problem” [online], available: http://www.hermetic.ch/misc/ts3/ts3demo.htm

[Carr05] R. Carr, "Simulated Annealing" [online], MathWorld--A Wolfram Web Resource, Feb. 2005, available: http://mathworld.wolfram.com/SimulatedAnnealing.html

[Saraswat07] P.K. Saraswat, E. Zamsha and A. Jankovic, "A Fast and Efficient Simulated Annealing based Design Space Exploration for a Custom VLIW Architecture for GSM Decoder and Optimizations using VEX compiler" [online], Lugano, Switzerland: ALaRI - USI: Universita della Svizzera italiana, Jan. 2007, available: http://www.studentimaster.usilu.net/saraswap/prabhat/projects/SA_VLIW_Saraswat_Zamsha_jankovic_final.pdf

[Eberhart01] R.C. Eberhart and Y. Shi, "Particle Swarm Optimization:Developments, Applications and Resources," in Proceedings of the 2001 Congress on Evolutionary Computation, vol. 1, Seoul, South Korea, May 27-30, 2001, pp.81-86.

[Boeringer04] D.W. Boeringer and D.H. Werner, "Particle Swarm Optimization Versus Genetic Algorithms for Phased Array Synthesis," in Proceedings of IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, vol. 52, no. 3, March 2004, pp.771-779.

[Farmahini-Farahani07] A. Farmahini-Farahani, M. Kamal, S.M. Fakhraie, and S. Safari, "HW/SW Partitioning Using Discrete Particle Swarm," in Proceedings of the 17th great lakes symposium on Great lakes symposium on VLSI, vol. 1, Stresa-Lago Maggiore, Italy, 2007, pp.359-364.

[Thareja07] Thareja, Vishal; Bolic, Miodrag; Groza, Voicu, "Design of a Fuzzy Logic Coprocessor in Handel-C," in Proceedings of IEEE 2nd International Workshop on Soft Computing Applications (SOFA), Aug. 2007, pp. 83-88.

[Xilinx05] Xilinx Inc., "Spartan-3 FPGA Family: Complete Data Sheet," San Jose, CA: Xilinx Inc., May 2005, available: http://www.xilinx.com/support/documentation/data_sheets/ds099.pdf