Ant Colony Optimization Scheduling

Jaouaher Belgacem
Department of Electronic Engineering
Hochschule of Hamm-Lippstadt
Lippstadt, Germany
jaouaher.belgacem@stud.hshl.de

***Abstract*—Ant Colony Optimization (ACO) is a meta-heuristic-based algorithm which can solve optimization problems in computer science, mathematics, and other fields. Optimizing high-performance computing problems is about finding a practical, relevant, and low-cost solution. The optimization of an ant colony is driven by an animal's natural behaviour in their habitat, particularly ant behaviour, which travels the shortest distance between food sources and their colonies [1]. This document is about a discussion of implementing the Ant Colony Optimization algorithm using hardware/software co-design techniques.**

***Index Terms*—ACO, Hardware/Software Codesign, Ant colony Optimization**

I. Introduction

In 1991, Marco Dorigo and his co-workers presented the Ant Colony Optimization (ACO) after studying and observing the behaviour of real ants in nature. In solving optimization problems, the ACO algorithm performs better than other algorithms. Individuals and feedback of information are essential to the ACO algorithm, where the ants are collaborating in an organized way through the pheromone in a polyphony. Hence, optimization-related problems can be solved by imitating the ants' behaviour [2] [2].

In the recent couple of years, hardware elements are getting complicated especially as they have a predominant component that includes a hardware platform that executes software programs. Thus, Hardware/Software Codesign means meeting those two parts together. Hence, researchers have been trying to use different optimization methods to maximize the performance of the circuits. Thus, ACO has shown good optimization results in solving optimization-related problems such as the traveling salesman problem, job-shop scheduling problem, graph coloring problem, and so on [2] this high performance is achieved due to the meta-heuristic feature that ACO has, by combining prior information about a potential solution with a posteriors information about previously successful solutions [4].

In order to escape local optima, meta-heuristic algorithms rely on some basic heuristics to help them escape them. In a constructive heuristic, elements are added to build a good complete solution from a null solution, in a local search heuristic, some elements of a complete solution are modified within several iterations to make it better. In spite of iterating, the meta-heuristic part enables the low-level heuristic to find better solutions than alone [**?**].

A lot of challenging problems could face the designer like partitioning or scheduling during high-level synthesis [1] [5] [4]. In the first section, we will discuss different terminologies related to the topic, in section two we will provide a deep explanation of how the ant colony optimization is working and in the last section we will show how this algorithm could be integrated to solve optimization challenges related to High-level synthesis.

II. Background

A. High Level Synthesis

The automated process of converting an algorithm into a dedicated digital circuit is called high-level synthesis or in other words architectural synthesis [6]. HLS concept is used to produce an efficient Electronic System-level design on a hardware and software level [7]. Thus, it is responsible for decreasing the production time and verification time as well as facilitating the flow of the power analysis and as a consequence reducing the production costs [6] [7]. This High-level synthesis design is performed in different steps as shown in Figure 1 [12]: A High-Level synthesis tool executes the following tasks

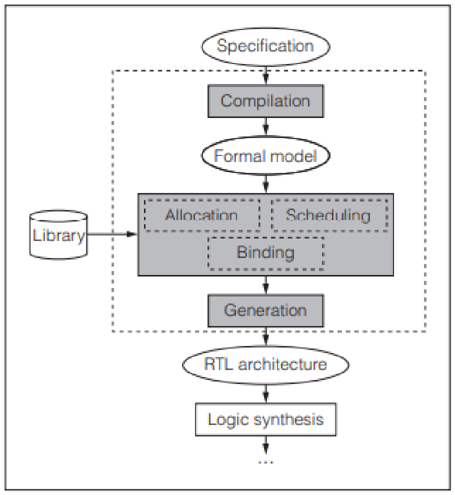

Fig. 1. The different steps of HLS design [7]

[7]:

- The compilation of the different specifications
- The allocation of hardware resources such as memory, buses and so on

- The scheduling of the different operations to clock cycles
- The binding of these operations to functional units
- The binding of variables to the existing storage units
- The binding of buses' transfer
- The generation of the RTL architecture

B. Hardware Software Co-design

The task of designing an embedded system's architecture is a complicated task as it is composed of different elements such as hardware and software. Thus, to make this task more flexible and relatively more straightforward, researchers proposed an appealing design technique which is called Hardware/Software Co-design [8]. The traditional way to define Hardware/software co-design is about putting the effort of designing hardware elements in a collaborating way in a single design effort [9]. This approach aims to merge the different design processes which are namely hardware and software designs. Where, the software design utilizes temporal decomposition and is well suited for flexibility, while hardware design utilizes spatial decomposition. Hence, It is possible to get solutions that are both flexible and effective when hardware and software are combined successfully [8].

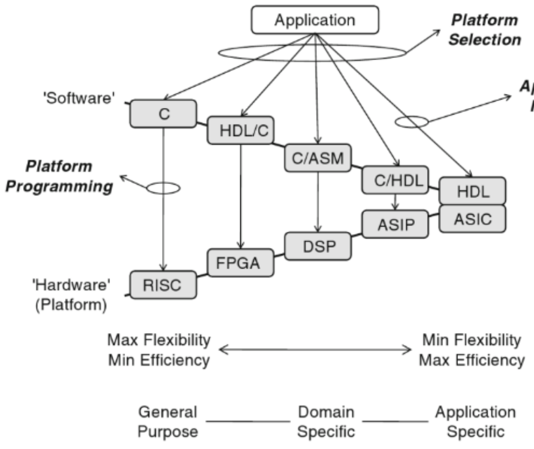

Fig. 2. Hardware/Software Codesign space [9]

In the figure above, we can illustrate some examples of programmable devices such as FPGA, ASIP, DSP, RISC and ASIC and the software is mapped to the hardware platforms using programming them. Besides, Application mapping means writing software programs using suitable programming languages for the aimed platform [9].

III. Ant Colony Optimization Algorithm

The ACO concept is that the ants have to find a way to reach their target which is mainly reaching the food that is placed on the other side of their nest. The ants are guided while exploring the path with the help of the pheromone which is a special chemical trail [2]. As shown in the figure, ants have to decide on the path they will follow once they reach the decision node A. There are two paths, one is longer than the

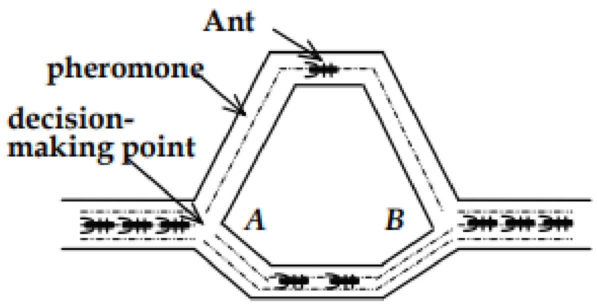

Fig. 3. Ants choose their path according to the pheromone [2]

other. Since the ants have the same crawling speed, the group of ants who choose the shortest path will reach destination node B first. It is important to note that, the choice is made randomly [2].

Mainly, the ACO algorithm is based on a number of iterations. For each iteration, several ants use heuristic information as well as previous populations' experience to construct complete solutions [2]. Below is a simple pseudo-code that illustrates a meta-heuristic approach to ant colony optimization, which can be applied to a range of optimization problems [10]:

```
ACO meta-heuristic Algorithm
Set parameters, initialize pheromone trails
while termination condition not met do ConstructAntSolutions
ApplyLocalSearch optional
UpdatePheromones
endwhile
```

The three main components of the algorithm are [10]:
ApplyLocalSearch: After the construction of solutions, it is a common practice to enhance the solutions generated by the ants through a process known as local search, before updating the pheromone. This particular phase, which is tailored to the specific problem at hand, is not obligatory but is typically incorporated into advanced ACO (Ant Colony Optimization) algorithms [10].

ConstructAntSolutions: A group of m simulated ants creates solutions using elements from a finite set of available solution components $C = c_{ij}$, where i ranges from 1 to n and j ranges from 1 to $|D_i|$. The construction of a solution begins with an empty partial solution sp. During each construction step, the partial solution sp is expanded by adding a feasible solution component from the set N(sp), which is defined as the collection of components that can be incorporated into the current partial solution sp without violating any of the constraints present.
UpdatePheromones: The objective of updating the pheromone is to amplify the pheromone values linked to favourable or promising solutions while reducing those associated with unfavourable ones. This is typically accomplished by (i) diminishing all pheromone values through pheromone evaporation,

and (ii) elevating the pheromone levels connected to a selected group of desirable solutions.

A. Explaining ACO through TSP example

At first, the ACO was applied to solve the Travelling Salesman Problem (TSP) which is composed of a set of cities whew the distances between the different cities are known. The ultimate goal was to find the shortest Hamiltonian tour on a fully connected map, where each city is allowed to be visited only once. Each edge of the graph represents the cities' connection and every vertex indicates the city itself. Besides, Ants can read and modify a variable called a pheromone associated with each edge [10].

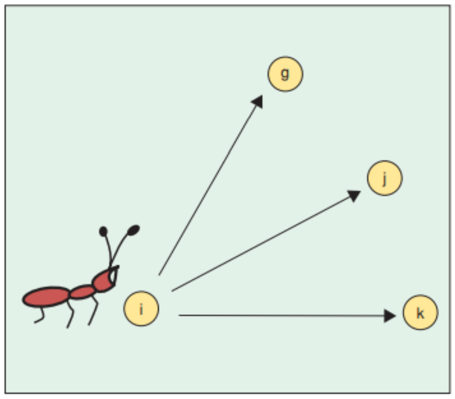

Fig. 4. The ant in the city i decides which city to visit next by using a stochastic process. If j has never been visited before, it will be selected with a probability proportional to the pheromone associated with an edge (i,j). [10]

The TSP problem is a famous optimization challenge that we use as an application to illustrate how the ACO is used to overcome this challenge. Let N be the number of towns, let i be the city edge and city j. Let the length d_{ij} be the distance between cities i and j to each edge(i,j). One of the best ACO algorithms is the Max-Min Ant system (MMAS) [2] The algorithm is described using different rules such as:

- **The rule of updating Pheromone:** While the ants are moving toward the destination node, they should leave their pheromone on the edges. Every completed trip by the ants is considered an iteration. Thus, the sum pheromone of one boundary is determined by the following equation [2]:
$$\tau_{ij}(t+1) = \Delta\tau_{ij} + (1-\rho)\tau_{ij}(t) \text{ (1)}$$
Where 1-ρ represents the persistence rate of the prior pheromone and $\rho \in (0,1)$. It represents the evaporation rate of the pheromone. However, in MMAS, only the best updates the trails of the pheromone which will give the value of [2]:
$$\tau_{ij}(t+1) = [\Delta\tau_{ij}^{best} + (1-\rho)\tau_{ij}(t)]_{\tau_{min}}^{\tau_{max}} \text{ (2)}$$
where tau_max and tau_min represent respectively the upper and lower bounds of the pheromone and the $\Delta\tau_{ij}^{b}est$ is [2]:
$$\Delta\tau_{ij}^{best} = \begin{cases} [1/L_{best} if(i,j) \text{ belongs to the best path} \\ else \\ 0 \end{cases} \quad (3)$$
Let L_best be the cost of the best solution achieved or the best solution until a specific iteration and it could be a mix of both [2].
- **The Ants' movement rule:** The transition of the ants from one city to another is happening randomly. Hence, The cities that have been entered by the ants have to be registered in a table. However, the set of cities that have never been accessed of the Kth ants must be defined as $allowed_k$ and the next step is to determine the visible degree: η_{ij}, $\eta_{ij} = 1/d_{ij}$. Thus, the probability of Kth ants choosing a specific city is determined by [2]:
$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{K\in allowed_k}[\eta_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}} \\ else \\ 0 \end{cases} \quad (4)$$
Where the most important variables are α and β which determine the relative influence of the trail pheromone and the heuristic information [2]. However, we can achieve a better move with the MMAS version of ACO which will be as follows [2]:
$$j = \begin{cases} \arg_{k\in allowed_k} \tau_{ij}(t)[\eta_{ik}]^{\beta} \text{ if } p \le p_0 \\ else \\ J \end{cases} \quad (5)$$
where p is an arbitrary number in [0,1]. Hence, with the probability of $0 \le p_0 \le 1$ and based on the pheromone trail and heuristic information, make the best move. However, according to the random variable J with the distribution given by the rule (4) (biased exploration), the move is made with the probability $1 - p_0$ [2].
- **The Initialization of the Pheromone Trail:** The $\tau_{max} = 1/((1-\rho)C_{nn})$, $\tau_{min} = \tau_{max}/(2N)$ where the initial pheromone values $\tau_{ij}(0) = \tau_{max}$ are set at the beginning of the run. C_{nn} is the length of a trip generated by the nearest neighbour heuristic and N represents the number of cities [2].
- **The Stopping rule**: To stop the ants from travelling there are two conditions, which are we reach the limited iteration number, CPU time limitation, or the best solution is found [2].
An overview of the Ant Colony Algorithm flow is the following:

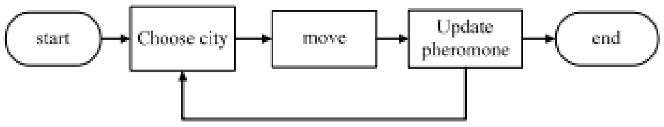

Fig. 5. Fundamental Flow of ACO Algorithm [11]

Once the different ACO parameters are defined, then the program will start the probabilities of the next transitions to

choose the optimal path and start the journey. Once, the path of the journey is calculated and the pheromones of the route are updated, then the ants will begin searching for food. As a final step, all the journeys will be compared to each other to find the shortest path and give the final result.
Choosing the path is the most fundamental step of this algorithm. Hence we will explain How it is made through the ACO algorithm. Supposing that S is the starting point of the next move and a set of other possible paths: [A, B, C, D, E, F, G] [11].

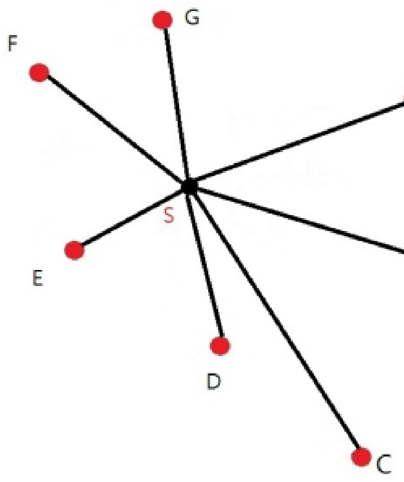

Fig. 6. The Schematic of Path possibilities [11]

Thus the possible transitions are: $P_{SA}(t)$, $P_{SB}(t)$, $P_{SC}(t)$, $P_{SD}(t)$, $P_{SE}(t)$, $P_{SF}(t)$, $P_{SG}(t)$. Given, parameters which are the R that will be generated randomly and SUM. The process will start with respect to the order of the alphabet. Therefore, the first possibility $P_{SA}(t)$ will be added to SUM and then compare to the value R. In case $SUM < R$, then the algorithm will proceed with the next step which is adding the next possibility in the order, $P_{SB}(t)$ to the output SUM of $P_{SA}(t)$ and have a total SUM of both of possibilities and compare it to R. The procedure will be repeated as long as SUM is smaller than the value R. However, if $SUM > R$, then node A will be selected as the next [11]. The path choice selection process is shown in the following figure:

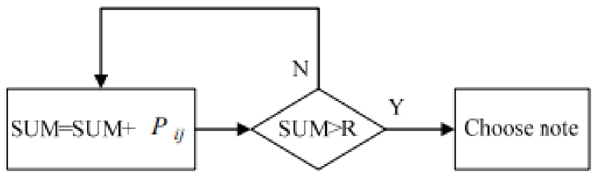

Fig. 7. Fundamental Flow of the Ant Colony Optimization Algorithm. [11]

IV. Integration of Ant Colony Optimization and the High-Level Synthesis

A. System on Programmable Chip

The choice of design modalities is a critical step for designing embedded products. Hence, several FPGAs are based on the SOPC modality which gives them flexibility for customising both software and hardware infrastructures contrarily to the traditional modality ASIC which lacks the hardware flexibility feature. Therefore, SOPC offers re-configurable, flexible designs with a short development process [12].
Furthermore, Combining microprocessors and FPGA is fundamental for High-Level applications where a programmable system is integrated into FPGAs through the SOPC-based platform [11]. Thus, the associated software tools to microprocessors enable the SOPC development by including register file size, interrupts, I/O and hardware that supports logic operations such as multiplication, division of integers and floating numbers. Accordingly, the output of this synthesis tool is a synthesized Hardware Description Model (HDL) model of the microprocessor in Verilog or VHDL which will be translated to corresponding logic gates that form the FPGA technology-based circuits where CAD tools such as Xilinx or Altera's Quartus are used for FPGA programming. However, the software part of the embedded application (Processor) is developed using C/C++ programming languages executed, debugged and compiled using customized tools [12].

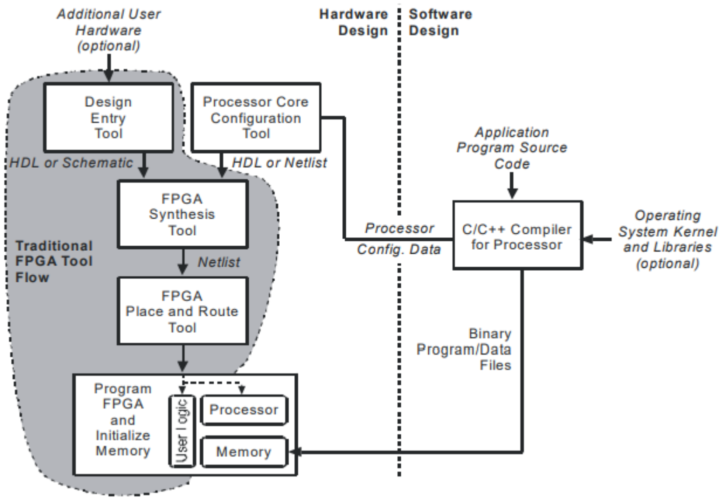
Additional User Hardware (optional)
Hardware Design
Software Design
Design Entry Tool
Processor Core Configuration Tool
HDL or Schematic
HDL or Netlist
Application Program Source Code
FPGA Synthesis Tool
Traditional FPGA Tool Flow
Processor Config. Data
C/C++ Compiler for Processor
Operating System Kernel and Libraries (optional)
Netlist
FPGA Place and Route Tool
Binary Program/Data Files
Program FPGA and Initialize Memory
User logic
Processor
Memory

Fig. 8. SOPC system hardware-software co-design structure [12]

Figure 8 shows a practical example of SOPC design flow of an FPGA-based system that compromises hardware and software designing [12].

B. Mapping ACO to FPGA

In this section, we will explain how ACO is integrated into hardware. Hence, the figure below shows how the ACO algorithm circuit is mapped to Hardware Software Co-design [11]: As shown in the figure, the FPGA of the ACO is divided

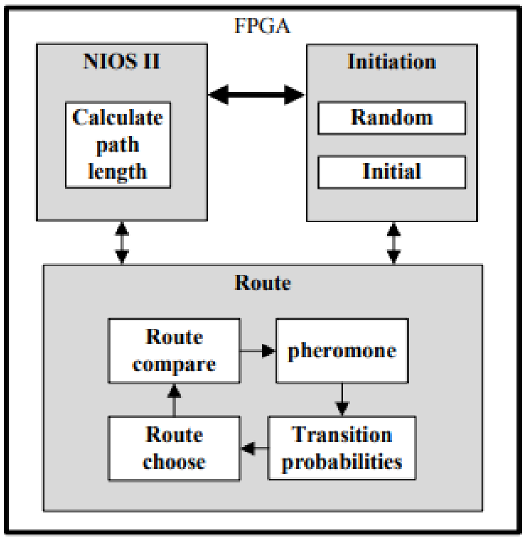

Fig. 9. The Circuit chart of Ant Colony algorithm [11]

into three main parts. The first module is the initiation block which is responsible for initialising signals and data, that will be transferred to the route selection module and the NIOS II Block. The route selection block will start choosing a path that will be sent to the NIOS II module in order to calculate the length of the path. Then, it will be transferred back to the path selection block to update the pheromones and signals. Thus, to detail the different modules and their functionalities in the ACO FPGA circuit, we will explain how the hardware part of ACO which is mainly focusing on modules' initialization and the software side is responsible for selecting the path and computing its length as follows [11]:

1) Initiation Block: The initiation of the parameterised parameters is performed in this module. The ACO parameters are the number of iterations, the number of ants and their definition, absolute distance β, pheromones parameter α, coefficient of disappeared pheromones ρ, and $\eta_{ik}(t)$. Thus, the sub-module Random is used to randomise a random value that the route selection block needs to use.

2) Route Block: This block is used to construct the pheromone matrix circuit which includes the value of the pheromone concentration between all nodes $\tau_{ij}(t)$. If the receives an update signal then, it automatically triggers the update function of the pheromone. The sub-module of the transition possibilities is implemented to compute the pheromone matrix and achieves equation (4) to determine the different probabilities of the conversion rule for every node which will be sent to the route choose sub-block that will decide for the next transition. Finally, the route compare section of this module will determine and update the best route and create a matrix to store the best route.

3) NIOS II Processor Block: The third module of the ACO FPGA circuit is dedicated to calculating the best path and the length of each path. Therefore, all the outputs will be transmitted back to the route block for updating data and more specifically the pheromones.

It is important to note that the straightforward implementation of the ACO on FPGA restricts the potential of the meta-heuristic approach due to the provided hardware resources of some commercial FPGA architectures. According to Bernd Scheuermann et al, the challenges are [13]:

- The floating point representations that are required for the pheromone values and the randomized numbers. Thus, the implementation of a fine-grained programmable logic from such a representation is difficult.
- The multiplication circuits are supported by some FPGA circuits that have a restricted size comparing the majority of the available FPGAs where the available computational resources are not efficient for the multiplication operations that, the integration of heuristic data and the evaporation require.
- The selection rule of the paths of the standard ACO algorithm leads to some complexities in terms of time and space for the programmable gate arrays.

Therefore, scientists in [13] have proposed another version of the ACO algorithm which is based on the original ACO which is named, population-based ACO in order to overcome these challenges. The P-ACO approach has proven a tremendous reduction concerning the runtime compared to the original ACO [13].

V. Conclusion

The Ant Colony Optimization technique is inspired by the primitive behaviour of animals and specifically by the ants during the process of food searching. Hence, this technique has helped scientists to overcome optimization challenges in different kinds of domains. In this paper, we focused on the integration of HW/SW codesign based on SOPC techniques to achieve the standard ACO algorithm where an FPGA-based circuit was designed to speed up the route selection process of the ACO and a c program was designed to enable the NIOS II processor to analyse the path within an optimal execution time with the help of used methods [11]. To conclude, the original ACO algorithm has been a reference algorithm that scientists have used to invent different ACO algorithm derivatives to solve some hardware restrictions and enhanced optimization performance that standard ACO failed to meet such as P-ACO [13], Max-Min ant system [2], [10] and Multi-Colony Ant systems [2].

References

[1] Kopuri, Shekhar, and Nazanin Mansouri. "Enhancing scheduling solutions through ant colony optimization." 2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04CH37512). Vol. 5. IEEE, 2004.
[2] Book ANT COLONY OPTIMIZATION METHODS AND APPLICATIONS Edited by Avi Ostf eld
[3] Deng, Wu, Junjie Xu, and Huimin Zhao. "An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem." IEEE access 7 (2019): 20281-20292.
[4] Maniezzo, Vittorio, Luca Maria Gambardella, and Fabio De Luigi. "Ant colony optimization." New optimization techniques in engineering 1.5 (2004).

[5] Koudil, Mouloud, et al. ”Solving partitioning problem in codesign with ant colonies.” Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach: First International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2005, Las Palmas, Canary Islands, Spain, June 15-18, 2005, Proceedings, Part II 1. Springer Berlin Heidelberg, 2005.

[6] Elie Torbey and John Knight. High-level synthesis of digital circuits using genetic algorithms. In 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), pages 224–229. IEEE, 1998.

[7] Coussy, Philippe, et al. ”An introduction to high-level synthesis.” IEEE Design & Test of Computers 26.4 (2009): 8-17.

[8] Schaumont, Patrick. ”Hardware/software co-design is a starting point in embedded systems architecture education.” Proceedings of the Workshop on Embedded Systems Education. 2008.

[9] Practical Introduction to Hardware/Software Codesign By Patrick R. Schaumont book p11-p18

[10] Dorigo, Marco, Mauro Birattari, and Thomas Stutzle. ”Ant colony optimization.” IEEE computational intelligence magazine 1.4 (2006): 28-39.

[11] Shih-An Li, Min-Hao Yang, Chung-Wei Weng, Yi-Hong Chen, Chia-Hung Lo, and Ching-Chang Wong. Ant colony optimization algorithm design and its fpga implementation. In IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS 2012), 2012.

[12] Hall, Tyson S., and James O. Hamblen. ”Using system-on-a-programmable-chip technology to design embedded systems.” (2006).

[13] Scheuermann, Bernd, et al. ”FPGA implementation of population-based ant colony optimization.” Applied Soft Computing 4.3 (2004): 303-322.

VI. Declaration of Originality

I, Jaouaher Belgacem, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

2023 Lippstadt - Jaouaher Belgacem