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 that 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 behavior in their habitat, particularly ant behavior, 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 behavior of real ants in nature. In solving optimization problems, the ACO algorithm performs better than other algorithms. Furthermore, individuals and information feedback are essential to the ACO algorithm, where the ants collaborate in an organized way through the pheromone in a polyphony. Hence, optimization-related problems can be solved by imitating the ants' behavior [2] [3].

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. Therefore, 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 posteriors information about previously successful solutions [4].

In order to escape the local optima, meta-heuristic algorithms rely on some basic heuristics that could be constructive or local search heuristics. Thus, for the first type of heuristics, elements are added to build a good complete solution from a null solution. However, in a local search heuristics, some elements of a complete solution are modified within several iterations to enhance it. In spite of iterating, the meta-heuristic part enables the low-level heuristic to find better solutions [4].

A lot of challenging problems could face designers while developing the outline of the system, like partitioning or scheduling during high-level synthesis. Hence these problems can be solved using ACO algorithms [1] [5] [4].
The structure of this paper is composed of five parts. After introducing the topic, we will discuss different terminologies related to ACO and HLS. In the next section, we will provide a deep explanation of how the ant colony optimization is working and then, show how it is integrated with HLS in section four while the last section concludes the paper.

II. Background

A. High Level Synthesis

The automated process of converting an algorithm into a dedicated digital circuit is called high-level synthesis (HLS) or in other words architectural synthesis [6]. This 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].
The High-level synthesis design is performed in different steps as shown in Figure 1 [12]:

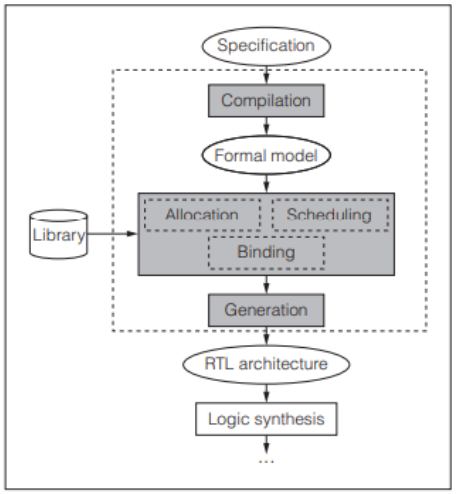

Fig. 1. The different steps of HLS design [7].

A High-Level synthesis tool executes the following tasks [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 sums up in putting the effort of designing both hardware and software elements in a collaborating way into one single design effort [9]. Therefore, this approach aims to merge the different design processes which are namely hardware and software designs. Where, the software design utilizes temporal decomposition, while the hardware design utilizes spatial decomposition. Consequently, It is possible to get solutions that are both flexible and effective when hardware and software are combined successfully [8].

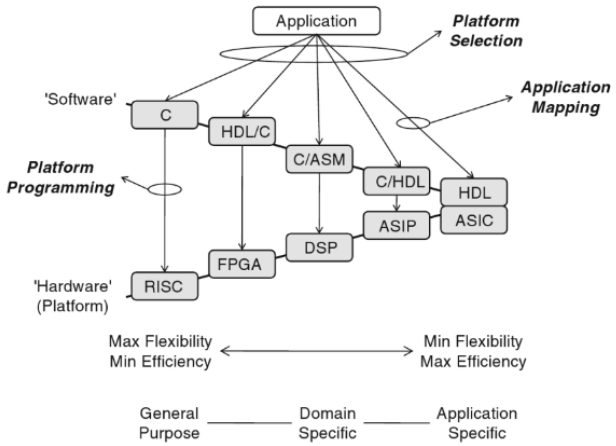

Fig. 2. Hardware/Software Codesign space [9].

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

III. Ant Colony Optimization Algorithm

In the ACO concept, the ants are required to find a way to reach their target or in other words reach the food source which is placed on the other side of their nest or colonies. Additionally, the ant's exploration of the path is guided by a chemical trail known as a pheromone. [2].

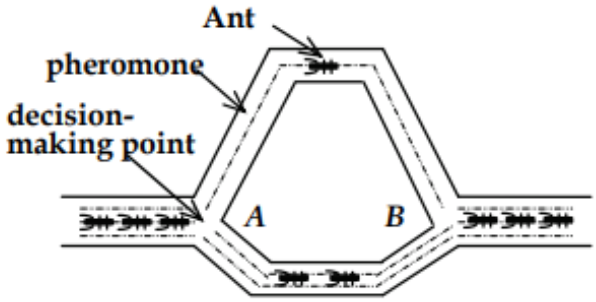

Fig. 3. Ants choose their path according to the pheromone [2].

As shown in the figure, ants have to decide on one path they will follow once they reach decision node *A* where the length of both paths is different. Since the ants have the same crawling speed, the group of ants who chose the shortest path will reach the destination node *B* first. It is important to note that, the route's choice is made randomly [2].

In order to build the solution using the ACO algorithm, a number of iterations will be performed. For each iteration, several ants use heuristic information as well as previous populations' experience to construct complete solutions which will be used afterwards to decide on the optimal one [2]. In the following pseudo-code, the ACO meta-heuristic approach is illustrated [10]:

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

Where the three main components of this 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 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)*. This set can be incorporated within the partial solution *sp* without violating any of its constraints.

UpdatePheromones: The objective of updating the pheromone is to amplify the pheromone values linked to favorable or promising solutions while reducing those that are associated with unfavorable ones. Thus, it is typically accomplished by

first diminishing all pheromone values through pheromone evaporation, and second elevating the pheromone levels that are 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 where the distances between them are given. 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 whereas every vertex indicates the city itself. Moreover, each edge is associated with a pheromone, which can be read and modified by Ants [10].

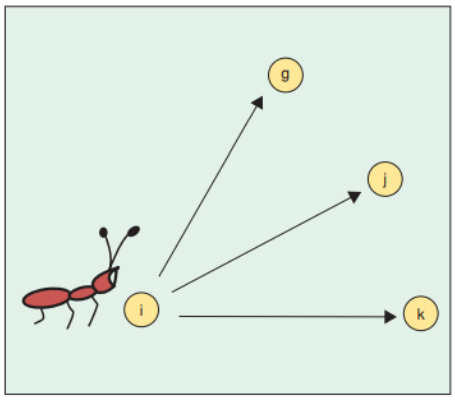

Fig. 4. The ant in the city *i* decides which city to visit next by using a stochastic process. If *j* has never been visited before, it will be selected with a probability proportional to the pheromone associated with an edge *(i,j)* [10].

To illustrate how the ACO algorithm is implemented in practice, we utilize the TSP problem as a famous optimization challenge. Let *N* be the number of towns, let *i* be the city edge, and the city *j*. Let the length d_{ij} be the distance between them. Since the Min-Max ACO (MMAS) is one of the best ACO variations, we will present some of its rules besides the standard ACO one. [2] Our algorithm can be described using different rules, which are namely:

- **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 one single iteration. Thus, the pheromone's sum of one boundary is determined by the following equation [2]:

$$\tau_{ij}(t+1) = \Delta\tau_{ij} + (1-\rho)\tau_{ij}(t) \tag{1}$$

Where 1-ρ represents the persistence rate of the prior pheromone noting that $\rho \in (0,1)$ as it represents the evaporation rate of the pheromone. However, in MMAS, only the best pheromone updates are considered. Thus, the MMAS trail updating rule is determined by [2]:

$$\tau_{ij}(t+1) = [\Delta\tau_{ij}^{best} + (1-\rho)\tau_{ij}(t)]_{\tau_{min}}^{\tau_{max}} \tag{2}$$

where τ_{max} and τ_{min} represent respectively the upper and lower bounds of the pheromone. Consequently, the $\Delta\tau_{ij}^{best}$ is given by [2]:

$$\Delta\tau_{ij}^{best} = \begin{cases} [1/L_{best}, & \text{if } (i,j) \text{ belongs to the best path} \\ \text{else} \\ 0 \end{cases} \tag{3}$$

Let L_{best} This can be the cost of reaching the best solution or the cost of staying at the best solution until a particular iteration or a combination of the two [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 K_{th} 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 K_{th} ants choosing a specific city is determined by [2]:

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{K\in allowed_k}[\eta_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}} \\ \text{else} \\ 0 \end{cases} \tag{4}$$

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]:

$$j = \begin{cases} \arg_{k\in allowed_k} \tau_{ij}(t)[\eta_{ik}]^{\beta} \text{ if } p \le p_0 \\ \text{else} \\ J \end{cases} \tag{5}$$

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 neighbor heuristic and N represents the number of cities [2].
- **The Stopping rule**: To stop the ants from traveling there are two conditions, which are whether the algorithm reaches 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:

After the different ACO parameters are defined, the program will calculate the probabilities of the next transition to choose

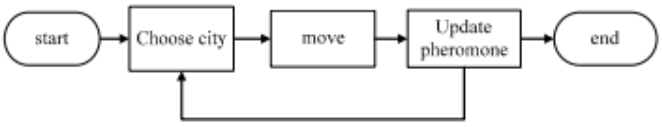

Fig. 5. Fundamental Flow of ACO Algorithm [11].

the optimal path and begin the journey. Once, the path of the journey is determined and the pheromones of the route are updated, then the ants will start searching for food. As a final step, all the journeys will be compared to each other to find the shortest path and produce 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].

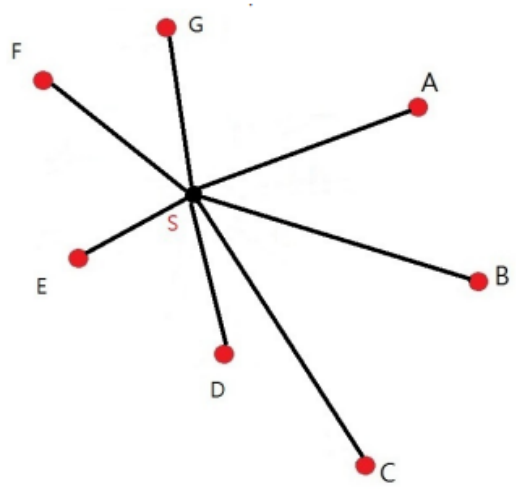

Fig. 6. The Schematic of Path possibilities [11].

Thus the possible transitions are: $P_{SA}(t)$, $P_{SB}(t)$, $P_{SC}(t)$, $P_{SD}(t)$, $P_{SE}(t)$, $P_{SF}(t)$ and, $P_{SG}(t)$. In the following step, the algorithm will generate a random value R, and sum the values of SUM. The process of different possibilities will be made according to the alphabet order. Therefore, the first possibility $P_{SA}(t)$ will be added to SUM and then compared to the value R. In case $SUM < R$, then the algorithm will proceed to the next step by adding the next possibility in the order, $P_{SB}(t)$ to the output SUM of $P_{SA}(t)$ and compare that to R. The procedure will be repeated as long as SUM is smaller than R. However, if $SUM > R$, then node A will be selected as the next route [11]. The selection process of paths is explained in the following figure:

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. The System On Programmable Chip

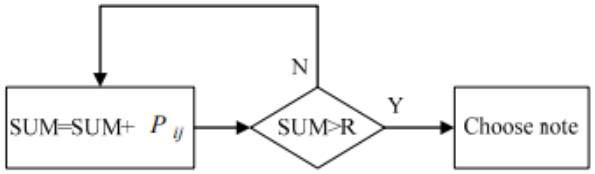

Fig. 7. Fundamental Flow of the Ant Colony Optimization Algorithm. [11].

(SOPC) modality is therefore used in a number of FPGAs, which allows both software and hardware infrastructures to be customized, whereas hardware customization is not permitted with traditional ASICs. In sum, SOPC offers re-configurable, flexible designs with a short development process [12].

Additionally, integrating a programmable system with an FPGA via an SOPC-based platform is fundamental for High-Level applications [11]. Thus, the associated software tools to microprocessors enable SOPC development by including register file size, interrupts, I/O, and hardware elements that support 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). FPGA technology-based circuits are formed from logic gates constructed from the HLS model developed in Verilog or VHDL. In order to program the FPGAs we need to use CAD tools such as Xilinx or Altera's Quartus. However, the software of embedded applications is typically developed using C/C++ scripting languages, which are then executed, debugged, and compiled using customized tools [12].

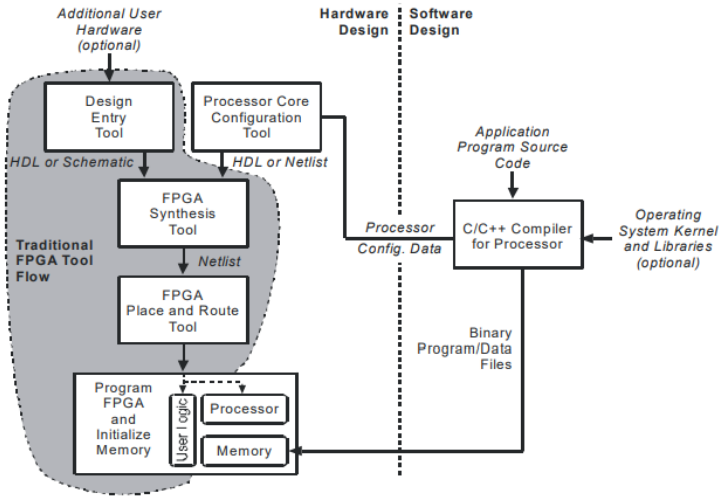

Fig. 8. SOPC system hardware-software co-design structure [12].

Figure 8 shows a practical example of the 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 in Hardware Software Co-design [11]:

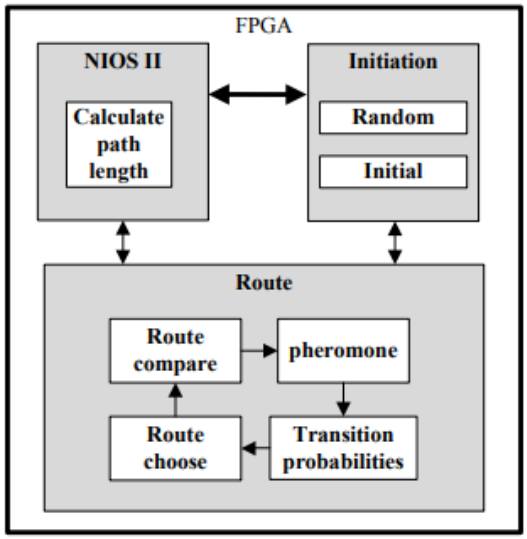

Fig. 9. The Circuit chart of Ant Colony algorithm [11]

Figure 9 shows the overall mechanism of the ACO FPGA. One whole, circuit is divided into three blocks. The first module is the initiation block which is responsible for initializing signals and data, that will be transferred to the route selection and the NIOS II modules. Then, the route selection block will select a path and send it to the NIOS II module. Which will calculate the length of the chosen path. Then, it will be transferred back to the path selection block in order to update the pheromones and signals. Briefly, in the ACO FPGA circuit, the hardware part is mainly responsible for initializing modules, while the software side determines the path and calculates its length. In order to get a better understanding of how ACO is integrated into FPGA circuits, different modules' functionalities are explained below [11]:

1) Initiation Block: The ACO parameters are generated 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 randomize 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 $\tau_{ij}(t)$. In case, this block 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 achieve equation (4). Each node's probability of a conversion rule is determined by this function, which is sent to the *[*route choose] sub-block that decides 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: Calculating the best path and determining the length of each path is performed in the third block of the ACO FPGA circuit. Therefore, all the outputs will be transmitted back to the route block for updating data or more specifically the pheromones.

Due to the hardware resources of some commercial FPGA architectures, the straightforward implementation of the ACO on FPGA limits the potential of the meta-heuristic approach. According to Bernd Scheuermann et al, the imposed challenges with this approach are [13]:

- Pheromone values and randomized numbers require floating point representations. Thus, the implementation of a fine-grained programmable logic from such a representation is difficult.
- Compared to the majority of available FPGAs, some of the multiplication circuits have a limited size and are not efficient to perform the multiplication operations, the integration of heuristic data, and evaporation required by the multiplication circuits.
- When it comes to programmable gate arrays, the selection rules of the standard ACO algorithm are complicated in terms of time and space.

To overcome these challenges, scientists in [13] have proposed a version of the ACO algorithm based on the original ACO, which is known as population-based ACO. 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 behavior of ants during the process of food searching. Consequently, this method has assisted scientists in overcoming optimization difficulties in several types of domains. In this paper, we concentrated on the HW/SW codesign of the ACO algorithm integration. As a result, the ACO FPGA circuit was created to accelerate the route selection process. Additionally, a C program was created to allow the NIOS II processor to analyze the path with the aid of the employed approaches in an optimal execution time [11]. To sum up, scientists have used the original ACO algorithm as a reference algorithm to create various ACO algorithm derivatives, such as P-ACO [13], Max-Min ant systems [10] [2], and Multi-Colony Ant systems [2], to overcome hardware limitations and improve optimization performance where standard ACO fell short.

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] Enxiu Chen and Xiyu Liu, "Multi-Colony Ant Algorithm," in Ant Colony Optimization Methods and Applications, A. Ostfeld, Ed. Rijeka, Croatia: InTech, 2011, pp. 3-6.

[3] W. Deng, J. Xu and H. Zhao, "An Improved Ant Colony Optimization Algorithm Based on Hybrid Strategies for Scheduling Problem," in IEEE Access, vol. 7, pp. 20281-20292, 2019, doi: 10.1109/ACCESS.2019.2897580.
[4] V. Maniezzo, L. M. Gambardella, and F. De Luigi, "Ant colony optimization," in New Optimization Techniques in Engineering, vol. 1, no. 5, 2004, pp 101–121.
[5] M. Koudil, et al., "Solving partitioning problem in codesign with ant colonies," in 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, vol. 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] P. Coussy, D. D. Gajski, M. Meredith and A. Takach, "An Introduction to High-Level Synthesis," in IEEE Design & Test of Computers, vol. 26, no. 4, pp. 8-17, July-Aug. 2009, doi: 10.1109/MDT.2009.69.
[8] P. Schaumont, "Hardware/software co-design is a starting point in embedded systems architecture education," in Proceedings of the Workshop on Embedded Systems Education, 2008.
[9] P. R. Schaumont, "The nature of hardware and software" in Practical Introduction to Hardware/Software Codesign. Boston, MA: Springer, pp. 11-18, 2012
[10] M. Dorigo, M. Birattari and T. Stutzle, "Ant colony optimization," in IEEE Computational Intelligence Magazine, vol. 1, no. 4, pp. 28-39, Nov. 2006, doi: 10.1109/MCI.2006.329691.
[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," 2012 International Symposium on Intelligent Signal Processing and Communications Systems, Tamsui, Taiwan, 2012, pp. 262-265, doi: 10.1109/ISPACS.2012.6473492.
[12] T. S. Hall and J. O. Hamblen, "Using system-on-a-programmable-chip technology to design embedded systems," in Proceedings of the IEEE, vol. 13, no. 3, pp. 142-152, Sep. 2006.
[13] B. Scheuermann, et al., "FPGA implementation of population-based ant colony optimization," Applied Soft Computing, vol. 4, no. 3, pp. 303-322, 2004, doi: 10.1016/j.asoc.2004.03.008.

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