2025 33rd Mediterranean Conference on Control and Automation (MED)
June 10 - 13, 2025. Tangier,Morocco

An FPGA-Based SoC Accelerator for Ant Colony Optimisation in Bearings-Only Target Motion Analysis

Kyriakos M. Deliparaschos[1], Roberto Setola[2], and Gabriele Oliva[2*]

2025 33rd Mediterranean Conference on Control and Automation (MED) | 979-8-3315-7719-3/25/$31.00 ©2025 IEEE | DOI: 10.1109/MED64031.2025.11073252

***Abstract*— Bearings-only Target Motion Analysis (BOTMA) is a challenging problem in signal processing due to its nonlinear nature and reliance on noisy angular measurements. Traditional iterative methods, such as Maximum Likelihood Estimation (MLE), are computationally demanding and sensitive to initialization. This paper presents a novel approach integrating Ant Colony optimisation (ACO) with a System-on-Chip Field Programmable Gate Array (SoC FPGA) to efficiently solve the MLE problem in BOTMA. The implementation is deployed on the PYNQ-Z1 platform, featuring an AMD Zynq-7000 SoC FPGA, which combines an ARM processing system with programmable logic. The FPGA design uses the hardware's concurrent processing capabilities to accelerate the ACO algorithm, achieving low-latency and high-throughput performance. The proposed solution is validated through simulations, demonstrating improved execution performance and low power consumption compared to the software model executed on a general-purpose processor. This work highlights the effectiveness of integrating advanced optimisation techniques with an FPGA-based SoC platform, enabling real-time BOTMA applications in resource-constrained environments.**

***Index Terms*— SoC FPGA, High-Level Synthesis (HLS), Target Motion Analysis, Ant Colony optimisation**

I. Introduction

Bearings-only target motion analysis (BOTMA) is a critical problem in signal processing, with applications ranging from defence systems to autonomous navigation and surveillance. The challenge lies in estimating target motion using only angular measurements, often requiring sophisticated algorithms to deal with the inherent non-linearity and noise in the data. Early works addressed this problem by proposing closed-form solutions and iterative estimation methods, such as Maximum Likelihood Estimation (MLE), which, despite their effectiveness, are computationally demanding and often require careful initialization to ensure convergence [1]–[3]. In this context, evolutionary algorithms such as Ant Colony optimisation (ACO) [4], [5], exhibit a large potential for tackling these computational challenges.

At the same time, Field Programmable Gate Array (FPGA) technology has become an attractive platform for implementing computationally intensive algorithms due to its inherent parallelism, low latency, and energy efficiency. FPGA-based optimisation approaches have demonstrated significant speedups in solving non-convex and real-time optimisation problems. For example, particle swarm optimisation (PSO) and Big Bang-Big Crunch (BB-BC) algorithms have been implemented on FPGA platforms for real-time nonlinear model predictive control (NMPC) and image processing tasks, achieving notable improvements in execution speed and resource utilisation [6], [7]. Neural network-based approaches, such as Lagrange Programming Neural Networks (LPNN) and Proximal Projection Neural Networks (PPNN), have also been effectively implemented on FPGAs to solve smooth and nonsmooth optimisation problems, emphasising fixed-point arithmetic and efficient circuit design [7].

Despite these advancements, a gap remains in leveraging FPGAs for integrating BOTMA algorithms with state-of-the-art optimisation techniques like ACO. While FPGAs offer unparalleled real-time processing capabilities, few frameworks exist to systematically bridge the theoretical development of such algorithms with their practical hardware implementation. Moreover, existing FPGA implementations often focus on specific applications or optimisation techniques, leaving room for generalizable solutions that address diverse operational scenarios.

In this paper, we propose a SoC FPGA-based ACO implementation for solving the Maximum Likelihood Estimation problem in BOTMA. Our approach combines the flexibility of ACO with the real-time processing capabilities of the SoC FPGA, providing a practical solution to the computational challenges inherent in BOTMA. The implementation is deployed on the PYNQ-Z1 platform, which integrates an AMD Zynq-7000 SoC, combining an ARM processing system with programmable logic. Simulation results demonstrate the efficacy of our approach, highlighting its advantages in terms of improved execution performance and low power consumption compared to the software model executed on a general-purpose processor.

II. Problem Setting

We examine a scenario where a target moves linearly on a plane under discrete-time sampling. Specifically, the target's motion is given by

$$\begin{cases} x_t(k) = x_{t0} + \dot{x}_{t0}kT + \frac{1}{2}\ddot{x}_t k^2 T^2 \\ y_t(k) = y_{t0} + \dot{y}_{t0}kT + \frac{1}{2}\ddot{y}_t k^2 T^2 \\ \dot{x}_t(k) = \dot{x}_{t0} + \ddot{x}_t kT \\ \dot{y}_t(k) = \dot{y}_{t0} + \ddot{y}_t kT, \end{cases} \tag{1}$$

[1] Department of Electrical Engineering and Computer Engineering and Informatics, Cyprus University of Technology, Limassol, Cyprus, Email: k.deliparaschos@cut.ac.cy

[2] Department of Engineering, Università Campus Bio-Medico di Roma, via Álvaro del Portillo 21, 00128, Rome, Italy. Email:{r.setola,g.oliva}@unicampus.it

[*] corresponding author.

where the sampling time is denoted by T. An own-ship platform seeks to estimate the parameter vector:

$$\boldsymbol{\psi} = \begin{bmatrix} x_{t0} & y_{t0} & \dot{x}_{t0} & \dot{y}_{t0} & \ddot{x}_t & \ddot{y}_t \end{bmatrix}^T, \tag{2}$$

using uniformly spaced measurements sampled during own-ship motion over $[0, k_{\max}T]$. In this problem, we assume the own-ship receives noisy measurements. In particular, the *nominal measurement* (e.g. see [8]) is given by

$$h(\boldsymbol{\psi}, k) = \operatorname{atan}\left(y_t(k) - y_o(k), x_t(k) - x_o(k)\right) \tag{3}$$

where the coordinates of the own-ship at time $t = kT$ along the x and y axes, are denoted by $x_o(k)$ and $y_o(k)$, respectively. The noisy measurements obtained by the own-ship are as follows:

$$z(k) = h(\boldsymbol{\psi}, k) + w(k),$$

where the terms $w(k) \sim \mathcal{N}\left(0, \sigma^2\right)$ are zero mean, independent and identically distributed Gaussian noises having a variance equal to σ^2.

We now turn to the problem of estimating the parameter vector $\boldsymbol{\psi}$ using the *Maximum Likelihood Estimation* (MLE) method (refer to [9], p. 182, for details).

The MLE for the parameter vector $\boldsymbol{\psi}$ is defined as the value $\boldsymbol{\theta}^*$ that maximizes the likelihood function $p(z_1, z_2, \ldots, z_m \mid \boldsymbol{\theta})$, where the maximization is performed over the feasible set of $\boldsymbol{\theta}$.

For convenience, when there is no risk of confusion, we use the simplified notation $p(\boldsymbol{\theta})$ to represent the likelihood function.

When the likelihood function $p(\boldsymbol{\theta})$ is differentiable, the MLE $\boldsymbol{\theta}^*$ is obtained as the solution to the equation:

$$\frac{\partial \ln(p(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^*} = \mathbf{0}_n. \tag{4}$$

It is important to note that, in this case, the solution to the above equation is both unique and theoretically asymptotically unbiased. Specifically, for our scenario, the likelihood function is given by the following expression [3]:

$$p(\boldsymbol{\theta}) = \frac{1}{2\pi\sigma} \prod_{k=1}^{k_{\max}} \exp\left(-\frac{(z(k) - h(\boldsymbol{\theta}, k))^2}{2\sigma^2}\right). \tag{5}$$

A key point to highlight is that the maximization problem can be expressed as:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \ln(p(\boldsymbol{\theta})). \tag{6}$$

Let us define $\lambda(\boldsymbol{\theta}) = -\ln(p(\boldsymbol{\theta}))$. The aforementioned problem can be alternatively represented as:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \lambda(\boldsymbol{\theta}) \tag{7}$$

which, through straightforward computations [3], reduces to solving:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{k=1}^{k_{\max}} \frac{(z(k) - h(\boldsymbol{\theta}, k))^2}{2\sigma^2}. \tag{8}$$

Notably, Eq. (8) corresponds to the classical least squares (LS) solution.

As highlighted in [10], it is important to note that the MLE for bearings-only target motion analysis does not have a closed-form solution. Instead, it must be solved iteratively, with an initialization close to the true solution to prevent divergence.

Furthermore, we emphasise that the MLE minimisation problem is generally nonconvex. Consequently, solving it often requires approximate methods designed to find a suitable local optimum.

III. FPGA-based SoC Implementation of Ant Colony optimisation (ACO)

This section outlines the implementation of the FPGA-based SoC accelerator for the Ant Colony optimisation (ACO) algorithm on the PYNQ-Z1 development board. The approach combines hardware acceleration with the PYNQ platform's software interface, offering efficient execution of computationally intensive tasks.

A. Overview of the Implementation

The implementation begins with the creation and simulation of virtual representations of sensor measurements on a host computer. The host communicates with the PYNQ-Z1 board via Ethernet, enabling seamless data exchange. The ACO algorithm, known for its high computational complexity, is offloaded to the programmable logic (PL) part of the Zynq System on Chip (SoC). Acting as an FPGA accelerator, the PL communicates with the processing system (PS) through AXI-based channels, as shown in Fig. 1. The PYNQ framework simplifies this interaction, providing a Python-based interface to manage data transfers and control the FPGA accelerator.

This integration enhances computational performance by utilising hardware acceleration, ensuring the ACO algorithm executes efficiently. The implementation details of the ACO algorithm are shown in Algorithm 1.

B. Developing the ACO IP Core using High-Level Synthesis

The first step in developing the FPGA accelerator involves designing the ACO IP core using AMD Vitis High-Level Synthesis (HLS) [11] tool. The algorithm, implemented in C++, focuses on optimising computationally intensive operations such as pheromone updates and path evaluations. These operations are synthesised into hardware logic using Vitis HLS, resulting in a reusable and highly efficient RTL module.

The synthesised IP core is then exported to Vivado for integration into a complete hardware design.

C. Integrating the IP Core into Vivado Design Software

In AMD Vivado design software, the ACO IP core is integrated into a custom block design alongside the Zynq Processing System (PS). The IP core features a single high-speed input interface implemented using a 128-bit AXI-Stream for data transfer (4 inputs of 32 bits each) and one 64-bit AXI-Stream output interface (2 outputs of 32 bits each).

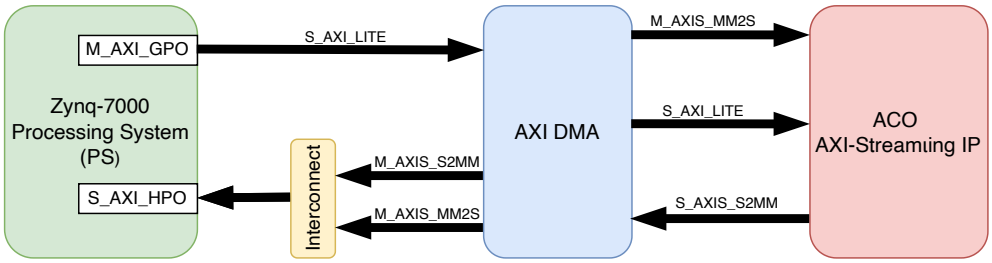

Fig. 1: SoC FPGA-based ACO implementation flow.

Additionally, one AXI-Lite interface is used for control and configuration. Data movement between the Zynq PS and the ACO IP core is handled by a single AXI DMA block, which manages the 128-bit input stream and the 64-bit output stream, ensuring efficient data transfer and minimising processor overhead. The AXI interconnect remains configured to enable efficient communication between the Zynq PS and the Programmable Logic (PL), with the AXI-Lite interface directly managed by the Zynq PS for configuration and status monitoring.

The block diagram topology for the presented accelerator, which integrates the Zynq PS with the ACO IP core featuring a single AXI-Stream input (128-bit)[1], a single AXI-Stream output (64-bit), and one AXI-Lite interface, requires only one AXI DMA block for efficient data handling. This AXI DMA block provides one AXI-Stream Slave Interface (S_AXIS_S2MM) for stream-to-memory transfers and one AXI-Stream Master Interface (M_AXIS_MM2S) for memory-to-stream transfers. The 128-bit AXI-Stream input is connected to the S_AXIS interface of the AXI DMA block, while the 64-bit AXI-Stream output is connected to the M_AXIS interface of the same DMA block. The AXI-Lite interface remains unchanged and is managed separately through the AXI interconnect, directly controlled by the Zynq PS for configuration and status management. This simplified topology reduces hardware complexity while maintaining efficient high-speed data throughput.

Once the design is completed, an HDL wrapper is generated, followed by synthesis, implementation, and bitstream generation. The resulting bitstream serves as a PYNQ overlay, which can be loaded onto the FPGA for execution.

The overall system architecture and data flow are depicted in Figure 2.

D. Deploying and Accelerating ACO on PYNQ

The generated PYNQ overlay is deployed on the PYNQ-Z1 and controlled using the PYNQ's Python-based interface. This interface allows the ACO algorithm to utilise the FPGA accelerator transparently, facilitating efficient execution of optimisation tasks.

The Bearings-Only Target Motion Analysis problem, which requires solving computationally demanding optimisation tasks, benefits significantly from this hardware-accelerated implementation. Parallel execution of pheromone updates and path evaluations in the PL enhances overall performance compared to a software-only solution.

[1]The input stream carries 3 inputs of 32 bits each (96 bits). However, since the AXI DMA requires widths in multiples of 32, the input stream is declared as 128 bits, with only the lower 96 bits utilised.

E. ACO Algorithm Implementation

The implementation of the ACO algorithm follows the pseudocode outlined in Algorithm 1. The algorithm iteratively builds and evaluates solutions, updating pheromone levels to guide future iterations towards the optimal result.

Algorithm 1 Ant Colony Optimisation for minimisation.

Require: Objective function $f(x)$, dimensions D, number of ants N, maximum iterations T, evaporation rate ρ, pheromone influence α, heuristic influence β.
Ensure: Best solution x_{best}.
Initialise pheromone levels $\tau_d \leftarrow 1.0, \forall d \in \{1, \ldots, D\}$.
Set $x_{\text{best}} \leftarrow \emptyset$ and $f(x_{\text{best}}) \leftarrow \infty$.
for $t \leftarrow 1$ to T **do**
  Initialise solutions $X \leftarrow \emptyset$.
  for $i \leftarrow 1$ to N **do**
    % Construct solutions for each ant
    for $d \leftarrow 1$ to D **do**
      Sample random value $r \in [0, 1)$.
      Select $x_{i,d} \sim \text{probability}(\tau_d^{\alpha})$.
    end for
    Add solution $x_i \leftarrow [x_{i,1}, \ldots, x_{i,D}]$ to X.
    Compute fitness $f(x_i)$.
  end for
  Find local best $x_{\text{local}} \leftarrow \arg\min_{x_i \in X} f(x_i)$.
  if $f(x_{\text{local}}) < f(x_{\text{best}})$ **then**
    Update $x_{\text{best}} \leftarrow x_{\text{local}}$.
  end if
  Update pheromone levels:
  for $d \leftarrow 1$ to D **do**
    $\tau_d \leftarrow (1 - \rho) \cdot \tau_d + \sum_{i=1}^{N} \frac{1}{1+f(x_i)}$.
  end for
end for
return x_{best}

IV. Case Study

In this section, we analyse the performance of the proposed ACO implementation with respect to the scenario reported in Figure 3. Specifically, we consider a scenario set

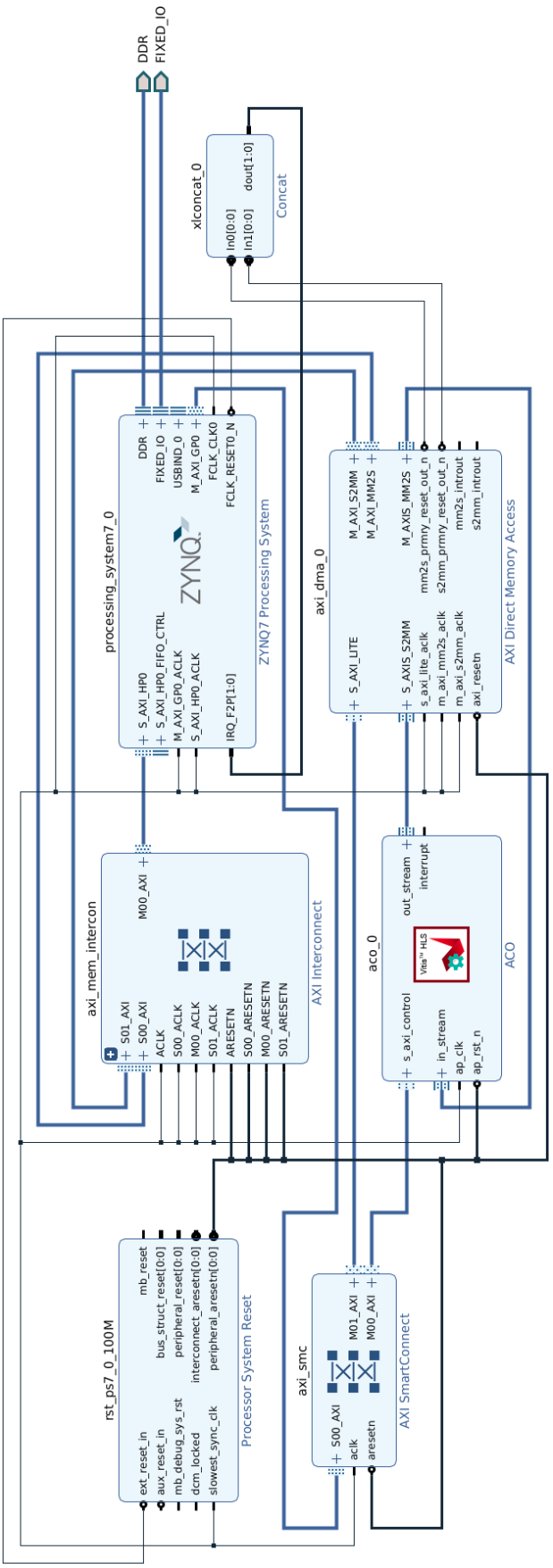

Fig. 2: Overall system architecture and data flow.

in the Aden Gulf, where the target ship (red curve) moves with constant velocity (i.e. we assume the acceleration is zero). The ownship (blue curve) moves with a sinusoidal trajectory. Both trajectories involve 1800 samples, obtained with a sampling time of $t = 2s$ (i.e. we simulate one hour). Assuming the frame of reference is centred at the ownship's initial location and has axes that are parallel to those of Figure 3, the parameters for the target ship motion are: $x_{t0} = 30\,Km$, $y_{t0} = 30\,Km$, $\dot{x}_{t0} = 8\,m/s$, $\dot{y}_{t0} = 7\,m/s$. In our simulation, we set the standard deviation to $\sigma = 0.01$.

Table I presents the resource utilisation summary of the implemented FPGA-based SoC accelerator designed for Ant Colony optimisation (ACO) in Bearings-Only Target Motion Analysis (BOTMA). The architecture integrates the Processing System (PS) and Programmable Logic (PL) components, with the ACO module representing the computational core of the system. The implementation was carried out on the PYNQ-Z1 platform, which features an AMD Zynq-7000 SoC (part number xc7z020-clg400-1) with an ARM Cortex-A9 processor and FPGA fabric.

This resource utilisation corresponds to an ACO implementation configured with three input data sets consisting of 1801 entries: *ownship_x* (i.e. the position of the ownship along the x axis), *ownship_y* (i.e. the position of the ownship along the y axis), and *measure* (i.e. the measured relative positions). Moreover, we consider two outputs: *best_solution* and *best_fitness*, which correspond to the array containing the six parameter results and the associated objective function cost. The implemented algorithm uses 100 ants, 2000 iterations, and a pheromone evaporation rate of 0.1.

The ACO instance utilizes approximately 18.13% of the available slice LUTs, 11.07% of Flip-Flops (FFs), 5.36% of block RAM (BRAM) tiles, and 12.73% of DSP resources, highlighting its computational demands. The remaining peripherals, including DMA controllers, AXI interconnects, and system management cores, contribute to the overall resource utilisation with relatively lower resource occupancy. It is important to note that the "SoC FPGA design" row in the table represents the total resource utilisation for the entire design, including the ACO instance and all supporting components. The table depicts the resource distribution between the ACO module and the supporting components within the SoC architecture.

TABLE I: Resource utilisation.

Name	LUT	FF	BRAM	DSP
SoC FPGA design	13322.00	16564.00	11.50	28.00
aco_0	9646.00	11775.00	7.50	28.00
axi_dma_0	1565.00	2329.00	4.00	0.00
axi_mem_intercon	1609.00	1805.00	0.00	0.00
axi_smc	487.00	622.00	0.00	0.00
proc_system7_0	0.00	0.00	0.00	0.00
rst_ps7_0_100M	17.00	33.00	0.00	0.00
xlconcat_0	0.00	0.00	0.00	0.00
aco_0 Util. (%)	18.13	11.07	5.36	12.73
Total Util (%)	25.04	15.57	8.21	12.73

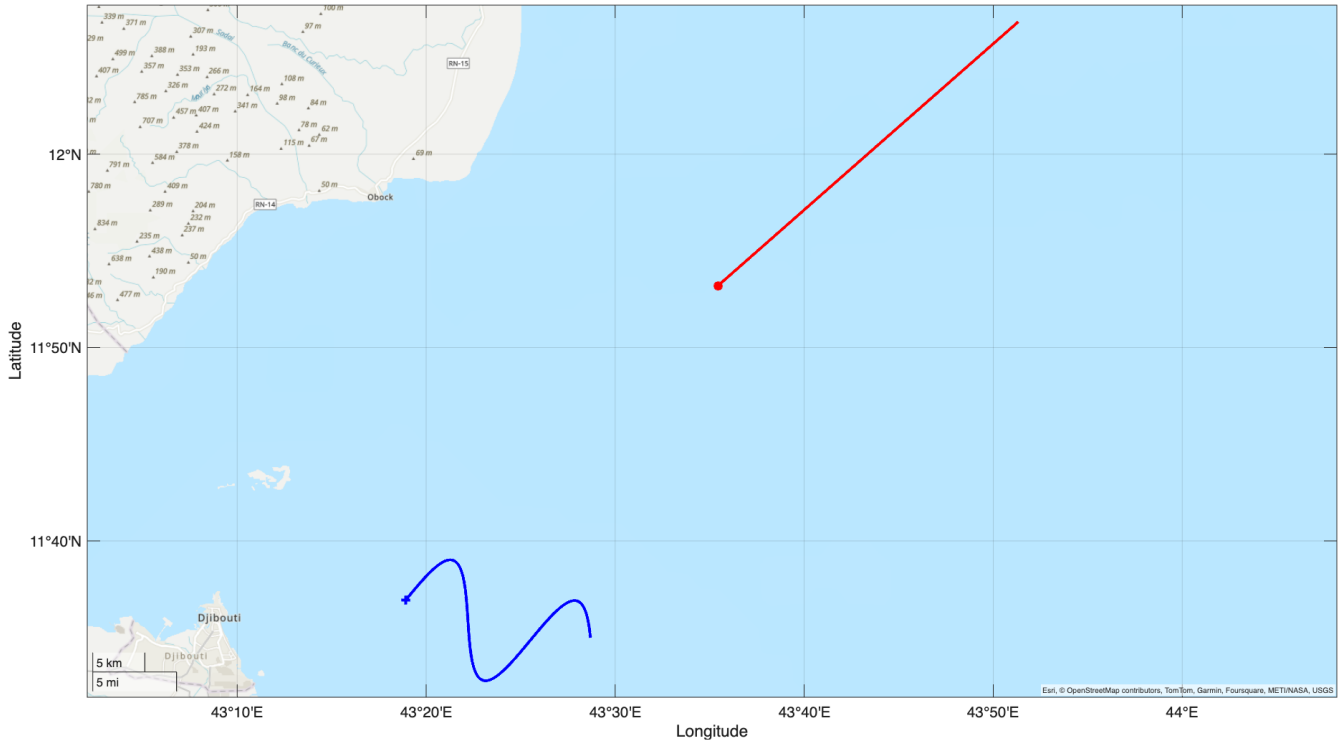

Fig. 3: Example considered in Section IV. The ownship is shown with a blue line and its start point is denoted by an asterisk. The motion of the target ship is shown in red and the initial point is shown with a circle.

Table II presents the power analysis of the FPGA-based SoC implementation for the ACO algorithm applied to the BOTMA problem. The total on-chip power consumption is 1.683 W, of which 1.540 W (92%) is dynamic power and 0.142 W (8%) is device static power. The dynamic power is further distributed among different components, with the highest contribution coming from the Zynq-7 Processing System (PS), consuming 1.260 W, which corresponds to 80% of the dynamic power.

It is important to note that the FPGA design, which includes clocks, signals, logic, BRAM, and DSP blocks, consumes only 12% of the total power. The power analysis assumes an ambient temperature of 25°C, providing a realistic estimate of power consumption under typical operating conditions. This analysis shows that most of the power usage comes from the Zynq-7 PS, while the FPGA-based ACO accelerator is power-efficient, using a relatively small portion of the total power.

Figure 4 depicts the floorplan of the implemented SoC FPGA, i.e. the physical locations and relationships of the blocks as well as the routing resources available for connecting them.

Table III presents the execution times and speedup comparisons for different implementations of the ACO algorithm applied to the BOTMA problem. The FPGA-based SoC accelerator, implemented on the PYNQ-Z1 platform, operates at a clock frequency of 100 MHz.

The FPGA implementation completes the ACO algorithm in 22 seconds, significantly outperforming the Python and C++ implementations, which take 1088.76 seconds and 459.48 seconds, respectively. The Python and C++ software models were executed on a computer with an Intel®Core™i7-9750H processor (6 cores, maximum frequency 4.50 GHz) and 32 GB of RAM. Compared to the Python implementation, the FPGA-based SoC accelerator achieves a 49.49× speedup, while compared to the C++ implementation, it achieves a 20.88× speedup.

TABLE II: Power analysis of the SoC FPGA implementation.

Power Component	Power (W)	Percentage (%)
Total On-Chip Power: 1.683 W		
Dynamic Power	1.540	92
Clocks	0.063	4
Signals	0.103	7
Logic	0.104	7
BRAM	0.001	<1
DSP	0.009	1
Zynq-7 PS	1.260	80
Device Static Power	0.142	8

This performance gain highlights the advantages of hardware-based execution, specifically FPGAs, which accelerate computations by executing multiple tasks concurrently through dedicated hardware resources. The results demonstrate the effectiveness of the FPGA-based SoC solution in accelerating the ACO algorithm for the BOTMA problem, making it well-suited for real-time or near-real-time applications.

To conclude the section, Table IV shows the results in

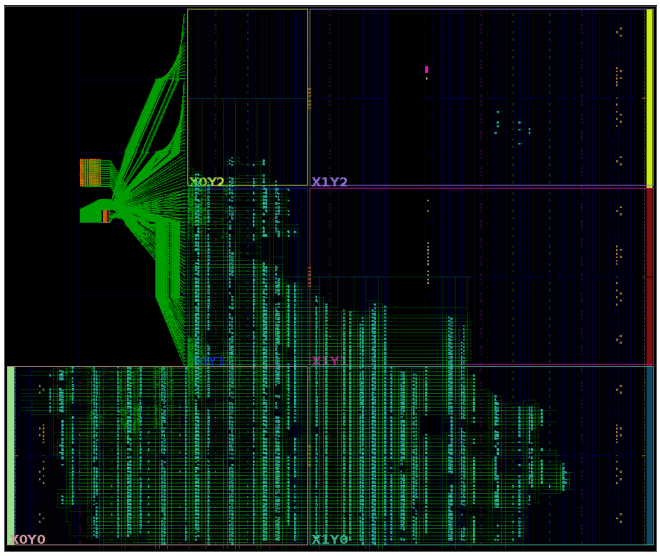

Fig. 4: Floorplan of the implemented design on the AMD XC7Z020 SoC.

TABLE III: ACO execution times and speedup comparison.

Impl.	Time (s)	Speedup (vs Py)	Speedup (vs C++)
Python	1088.76	1.00	0.42
C++	459.48	2.37	1.00
SoC Design	22.00	49.49	20.88

terms of the parameters being estimated and of the objective function of the MLE problem for the Python, C++ and FPGA implementation. A first interesting consideration is that, given the non-convex nature of the problem at hand and the presence of noise, the nominal solution for the unknown parameters does not correspond to a zero objective function value, but is close to the cost of the ACO solutions. As for the different implementations, we observe that the results are comparable, even though the C++ and FPGA implementation exhibit a slight increase in the objective function (i.e. about $+6.94\%$ with respect to the Python implementation and $+12.57\%$ with respect to the nominal solution); this is due to the fact that, within the C++ and FPGA implementation, for the sake of simplicity, we have used a random number generation technique, namely *linear congruential generator*[2] (LCG, see [12] and references therein for more details), that is simpler than Python's rand library.

Overall, the results suggest that the proposed FPGA implementation is remarkably faster while maintaining comparable performance with respect to Python and C++.

V. Conclusion

In this work, we presented a novel FPGA-based SoC accelerator for ACO applied to the BOTMA problem. Our implementation, deployed on the PYNQ-Z1 platform, demonstrates the effectiveness of integrating evolutionary optimisation algorithms with hardware accelerators for real-time signal processing tasks. Through simulations, we validated the proposed solution's ability to efficiently solve the Maximum Likelihood Estimation (MLE) problem in BOTMA, achieving significant performance improvements over traditional software-based implementations.

[2]The LCG is computationally inexpensive in terms of hardware, as it only requires one multiplication, one addition, and one modulo operation.

TABLE IV: Comparison of the different implementations in terms of the quality of the solution.

Impl.	$x_{t0}[km]$	$y_{t0}[km]$	$\dot{x}_{t0}[m/s]$	$\dot{y}_{t0}[m/s]$	fitness
Nominal	30	30	8	7	903.00
Python	29.76	29.52	9.18	8.73	950.58
C++	32.00	31.55	8.6	8.42	1016,54
SoC Design	32.00	31.55	8.6	8.42	1016,54

The experimental results highlight the benefits of our FPGA-accelerated ACO approach. Specifically, the FPGA implementation achieved up to a $49.49\times$ speedup over a Python-based solution and a $20.88\times$ speedup compared to a C++ software model, all while maintaining low power consumption and efficient resource utilisation. The power analysis revealed that the FPGA design accounted for only 12% of total dynamic power, underscoring the efficiency of hardware-based parallelism in executing computationally intensive optimisation tasks.

Future work will explore additional enhancements, such as integrating multi-objective optimisation techniques, refining pheromone update strategies for faster convergence, and extending the approach to multi-target tracking scenarios. Additionally, we plan to investigate adaptive hardware configurations using partial reconfiguration techniques to further optimize resource utilisation and power efficiency.

References

[1] S. Nardone and M. Graham, "A closed-form solution to bearings-only target motion analysis," *IEEE Journal of Oceanic Engineering*, vol. 22, pp. 168–178, 1997.

[2] T. Song and T. Um, "Practical guidance for homing missiles with bearings-only measurements," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, pp. 434–443, 1996.

[3] A. Farina, "Target tracking with bearings–only measurements," *Signal processing*, vol. 78, pp. 61–78, 1999.

[4] M. Dorigo and T. Stützle, *Ant colony optimization*. Cambridge, Mass: MIT Press, 2004.

[5] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE computational intelligence magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[6] M. Psarakis, A. Dounis, A. Almabrok, S. Stavrinidis, and G. Gkekas, "An FPGA-based accelerated optimization algorithm for real-time applications," *Journal of Signal Processing Systems*, vol. 92, pp. 1155–1176, 2020.

[7] R. Xiao, X. He, T. Huang, and J. Yu, "FPGA implementation of classical dynamic neural networks for smooth and nonsmooth optimization problems," *IEEE Transactions on Sustainable Computing*, vol. 9, no. 2, pp. 197–210, 2024.

[8] M. Mallick, "A note on bearing measurement model," *Mach. Eng*, vol. 10, 2018.

[9] S. M. Kay, "Statistical signal processing: estimation theory," *Prentice Hall*, vol. 1, pp. Chapter–3, 1993.

[10] K. Doğançay, "On the efficiency of a bearings-only instrumental variable estimator for target motion analysis," *Signal processing*, vol. 85, no. 3, pp. 481–490, 2005.

[11] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473–491, 2011.

[12] S. Tezuka, "Linear congruential generators," in *Uniform Random Numbers: Theory and Practice*. Springer, 1995, pp. 57–82.