République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique
Département d'Electronique

End of studies project dissertation in partial fulfilment of the requirements for the State Engineer Degree in Electronics

Hardware implementation of a bio-inspired MPPT technique for the control of a photovoltaic system

Mohamed BOUCHAKOUR

Under the supervision of Dr S.TITRI and Prof C.LARBES.

Publicly presented and defended on 25/06/2023

Composition of the jury :

President	Mr. Mourad	HADDADI	Prof.	ENP
Examiner	Mr. Mohamed Oussaid	TAGHI	Magister.	ENP
Supervisor	Mrs. Sabrina	TITRI	PhD.	CDTA
Supervisor	Mr. Cherif	LARBES	Prof.	ENP

ENP 2023

www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique
Département d'Electronique

End of studies project dissertation in partial fulfilment of the requirements for the State Engineer Degree in Electronics

Hardware implementation of a bio-inspired MPPT technique for the control of a photovoltaic system

Mohamed BOUCHAKOUR

Under the supervision of Dr S.TITRI and Prof C.LARBES.

Publicly presented and defended on 25/06/2023

Composition of the jury :

President	Mr. Mourad	HADDADI	Prof.	ENP
Examiner	Mr. Mohamed Oussaid	TAGHI	Magister.	ENP
Supervisor	Mrs. Sabrina	TITRI	PhD.	CDTA
Supervisor	Mr. Cherif	LARBES	Prof.	ENP

ENP 2023

www.enp.edu.dz

République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي
École nationale Polytechnique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique
Département d'Electronique

Mémoire de projet de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Électronique

Implémentation hardware d'une technique MPPT bio-inspirée pour le contrôle de systèmes photovoltaïques

Mohamed BOUCHAKOUR

Sous la direction de Dr S.TITRI et Prof C.LARBES.

Présenté et soutenu publiquement le 25/06/2023 auprès des membres du jury :

Président	M. Mourad	HADDADI	Prof.	ENP
Examinateur	M. Mohamed Oussaid	TAGHI	Magister.	ENP
Promotrice	Mme. Sabrina	TITRI	Dr.	CDTA
Promoteur	M. Cherif	LARBES	Prof.	ENP

ENP 2023

www.enp.edu.dz

Dedication

“

To my family, to my friends, to all my loved. ones

To you all, I dedicate this humble work

”

- Mohamed

Acknowledgements

I would like to thank Mrs Sabrina TITRI, a researcher at the CDTA research centre, for proposing this project and for her help and guidance.

I thank Mr Cherif LARBES, Professor at the Ecole Nationale Polytechnique, for his assistance and help with the project.

I also thank all the members of the jury for their time and effort in examining this work.

My thanks also go to all the teachers in the Electronics department, as well as all my classmates.

ملخص

في ضوء قضايا الاحتباس الحراري والتلوث ، أصبحت الطاقات المتجددة مثل الطاقة الشمسية تستخدم على نطاق واسع. يعد استخدام وحدات التحكم MPPT أمرًا ضروريًا للحصول على أقصى قدر من الكفاءة من الألواح الشمسية الكهروضوئية. في ظل ظروف التظليل الجزئي ، من الضروري استخدام تقنيات أكثر تقدمًا. أثبتت الأساليب المستوحاة من الكائنات الحية فعاليتها في ظل هذه الظروف. يهدف هذا العمل إلى تنفيذ تقنية SSA المستوحاة من كائنات حيوية. بدءًا من تنفيذ البرنامج ، نقوم بنمذجة ومحاكاة وحدة التحكم التي اخترناها. ثم ننتقل إلى تصميم الأجهزة باستخدام HLS. أخيرًا ، نقوم باختبار نظامنا في ظل ظروف العالم الحقيقي.

كلمات مفتاحية : لألواح الشمسية الكهروضوئية، MPPT, SSA, ستوحاة من الكائنات الحية .

Résumé

Face aux enjeux du réchauffement climatique et de la pollution, les énergies renouvelables telles que le solaire sont de plus en plus utilisées. L'utilisation de contrôleurs MPPT est essentielle pour obtenir une efficacité maximale des panneaux solaires photovoltaïques. Dans des conditions d'ombrage partiel, l'utilisation de techniques plus avancées est nécessaire. Les méthodes bio-inspirées ont prouvé leur efficacité dans ces conditions. Ce travail vise à mettre en œuvre la technique bio-inspirée SSA. A partir de l'implémentation du logiciel, nous modélisons et simulons notre contrôleur choisi. Nous passons ensuite à la conception matérielle à l'aide du flux de conception HLS. Enfin, nous testons notre système dans des conditions réelles.

Mots clés : MPPT, bio-inspirée, photovoltaïque, FPGA, Vivado, Vitis HLS.

Abstract

In light of global warming and pollution issues, renewable energies such as solar are becoming more widely used. The use of MPPT controllers is essential to obtain maximum efficiency from solar photovoltaic panels. Under partial shading conditions, the use of more advanced techniques is necessary. Bio-inspired methods proved their effectiveness under these conditions. This work aims to implement the SSA bio-inspired technique. Starting from the software implementation, we model and simulate our chosen controller. We then go through the hardware design using the HLS design flow. Finally, we test our system under real-world conditions.

Keywords : MPPT, Bio-Ispired, Photovoltaic, FPGA, HLS, Vivado, Vitis HLS.

Contents

List of Tables

List of Figures

List of abriviations

MPPT *Maximum Power Point Tracking*.

SSA *Salp Swarm Algorithm*.

MPP *Maximum Power Point*.

GMPP *Global Maximum Power Point*.

LMPP *Local Maximum Power Point*.

PV *Photovoltaic*.

BI *Bio-Inspired*.

PSO *Particle Swarm Optimisation*.

SSA *Salp Swarm Algorithm*.

FPGA *Field Programmable Gate Array*.

HLS *High Level Synthesis*.

RTL *Register Level Transfer*.

General introduction

The energy required to sustain life and modern civilization grows as the world population increases. Conventional fossil energy sources are not only limited in availability as it is none renewable, but they are also very polluting.

One of the most prominent challenges in our modern times is the issue of pollution and global warming, making it necessary to phase out these conventional energy sources. Many alternative and renewable sources exist : Wind, hydraulic, nuclear and solar.

In sunny regions, solar energy has attracted much attention in recent years. In Algeria, for example, with its substantial solar deposits, solar energy is rapidly gaining popularity. Many technologies are used to transform solar energy into electrical energy. One of the most popular methods is the use of Photovoltaic Panels. When light hits a semiconductor material, an electrical current is created.

Along with the countless advantages this method has, it presents many drawbacks, including the non-constant power output, which depends on weather conditions and the connected load. This is due to the non-constant P(V) characteristic.

To get maximum efficiency out of solar installations (Photovoltaic Panels), we try to always run our panels at their maximum power output (Maximum Power Point). The process of finding the MPP and keeping our system running at this point is called Maximum Power Point Tracking. Thus, tracking the maximum power point (MPP) of a photovoltaic panel (PV) is usually an essential part of a photovoltaic system (PVS).

However, variations in weather conditions or partial shading effect, complicate the task of tracking the MPPT due to the P(V) characteristic containing local Maximas.

In this context, the design and optimization of algorithm based Maximum Power Point Tracking handling the partial shading condition constitute a current research issue.

However, these MPPT controllers were implemented using the conventional methods (CM), which have the advantage of being simple, easy to implement with low cost. But, these CMs present a major drawback when partial shading occurs. In this case, the controller can easily fail to identify the global MPP.

This is where, recently, Bio-inspired methods emerged as new, more effective ones, which gained popularity. We can mention some of the popular bio-inspired algorithms in the field of MPPT : Ant Colony Algorithm (ACO) krishnan2020mppt,titri2017new, Particle Swarm Optimization (PSO) [1, 2, 3], Artificial Bee Colony (ABC) [4], Firefly Algorithm (FA)[5, 6] and many others.

Thus, in this work, we propose to apply the Salp Swarm bio-inspired algorithm for the maximum power point tracking of a Photovoltaic system subject to partial shading.

To highlight the performance of the proposed SSA controller, this later is compared with the Particle Swarm Optimisation (PSO) algorithm using MATLAB software.

Then, a hardware implementation on FPGA circuit is done using the new high-level synthesis design methodology with the VITIS HLS tool and the Basys3 prototyping board.

In addition to the results of the simulation investigation, the whole architecture of the photovoltaic system was verified in practice using an experimental circuit.

We organized our work into four chapters.

- Chapter 1 presents a general look at PV systems and their different components.
- Chapter 2 presents the software part, with the implementation of the proposed controller on a Matlab/Simulink environment with simulation to validate its performance. The tracking performance of the MPP using the SSA MPPT controller and the PSO MPPT controller are evaluated and compared in terms of tracking speed, accuracy and the ability to handle partial shading conditions.
- Chapter 3 presents the hardware part, where the HLS design methodology is discussed and used to design the proposed controller.
- Chapter 4 is dedicated to testing the proposed controller in real-time and real-world conditions.

Chapter 1

Photovoltaic Systems Overview

Introduction

In this chapter, the theoretical and practical bases necessary for the comprehension of our work will be presented.

We will introduce the basic concepts of a photovoltaic system as follows :

- The photovoltaic effect and the mathematical modelling of this conversion as well as the properties of this source of energy under different atmospheric conditions (light and temperature) and the influence of the shading on power production;
- The issue of maximum power transfer by illustrating an elementary conversion chain consisting of a static converter, which plays the role of a charging adapter during an indirect connection;
- The general operating principle of an MPPT control.

1.1 Principal of Photovoltaic Systems

The photovoltaic cell is the basic element in photovoltaic systems. A solar module groups together several cells which are electrically connected in series and/or in parallel, encapsulated and protected from external agents. Several modules form a solar panel, but in general, a photovoltaic system is composed of several panels, to which are added other protection devices, regulators, storage systems, controllers, measuring devices, inverters, etc...

1.1.1 The photovoltaic cell

1.1.1.1 Photovoltaic Principle

Discovered by E. Becquerel in 1839, the photovoltaic effect is the phenomenon by which certain materials generate a current when exposed to light. Based on this principle, the photovoltaic cell is the basic element of a solar system. Defined as a device that transforms energy from absorbed light radiation into electrical energy. When a photon hits a semiconductor atom, it liberates electrons that would be free to move. The movement of these free electrons is what generates a voltage at the cell terminals, which would ultimately result in a current flow if a load is connected. Figure 1.1 illustrates the basic principle of a silicon photovoltaic cell.

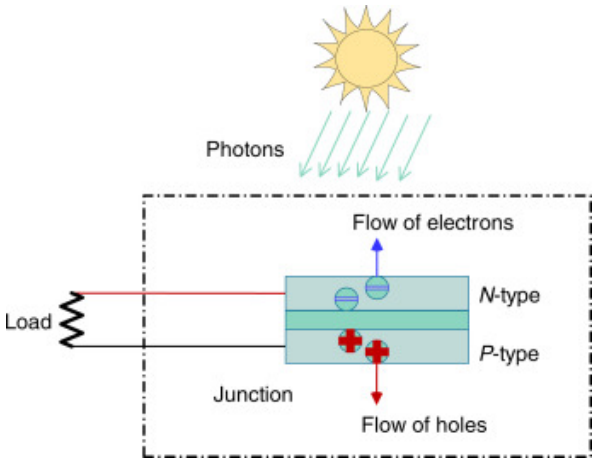


Figure 1.1 : Constitution of a PV cell [7]

1.1.1.2 Photovoltaic cell characteristics

The photovoltaic cells present a non-linear I(V) characteristic.
Figure 1.2 illustrates the I(V) and P(V) characteristic of a silicon-based PV cell for a given irradiance and temperature,

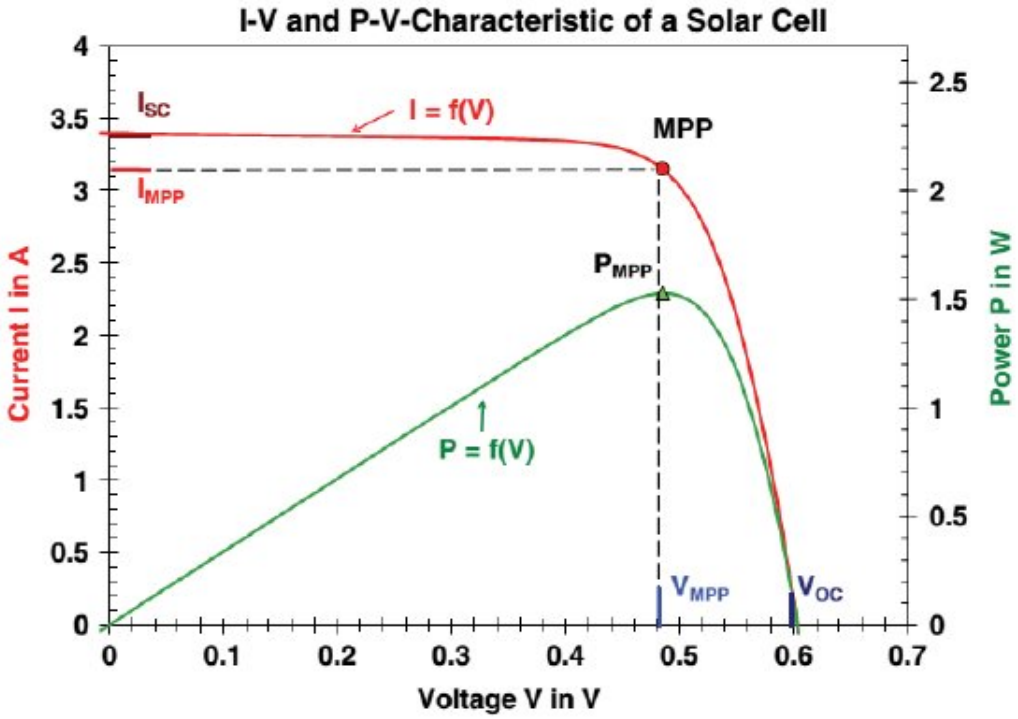


Figure 1.2 : I(V) and P(V) characteristic of a silicon PV cell [8]

We also noted MPP, which represents the Maximum Power Point of the cell. V_{MPP}, I_{MPP} and P_{MPP} represent the voltage, current and power values, respectively, at the maximum power point, where the cell outputs its maximum power.

1.1.2 Photovoltaic Generators

Under standard conditions of irradiance and temperature $(1000W/m^2;25°)$, the maximum power delivered by a 150 cm^2 cell is about 2.3W with a voltage of 0.5V. This power output is too small for any practical application, and thus, these cells need to be assembled to create a photovoltaic module. A photovoltaic generator is therefore made up of a series/parallel network of several photovoltaic modules grouped into panels, made up of identical modules.

These modules are assembled and connected together to create a panel, which would also be connected together to create an array.
The way the modules are connected (Series or parallel), and their number depend on the application and the needs. Figure 1.3 illustrates PV Cells, Modules, Panels and Arrays.

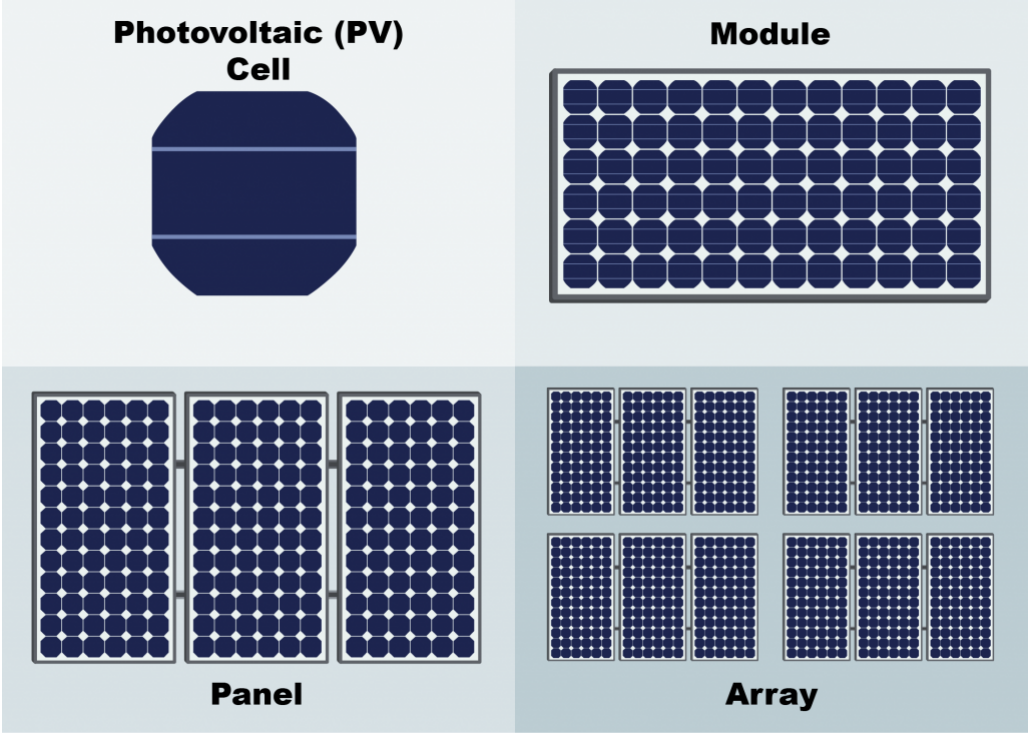
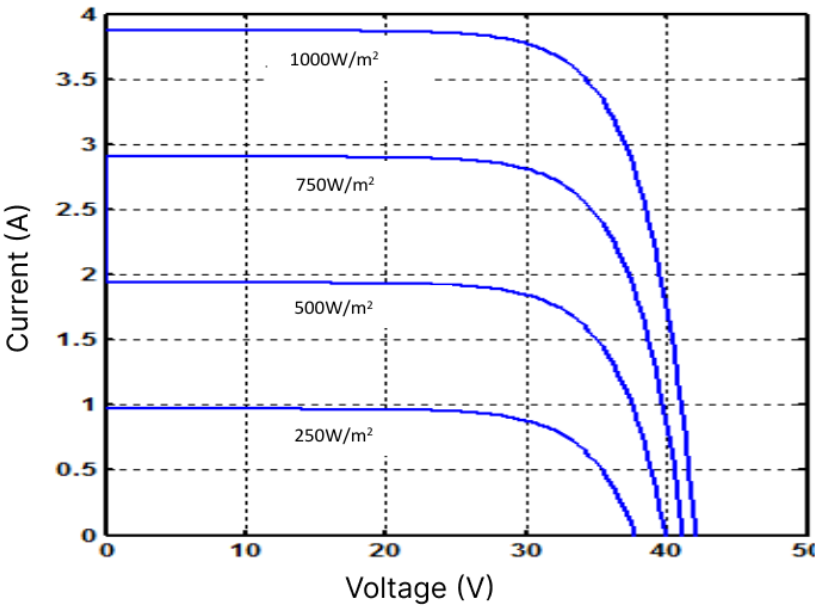


Figure 1.3 : PV cell, module, panel and array

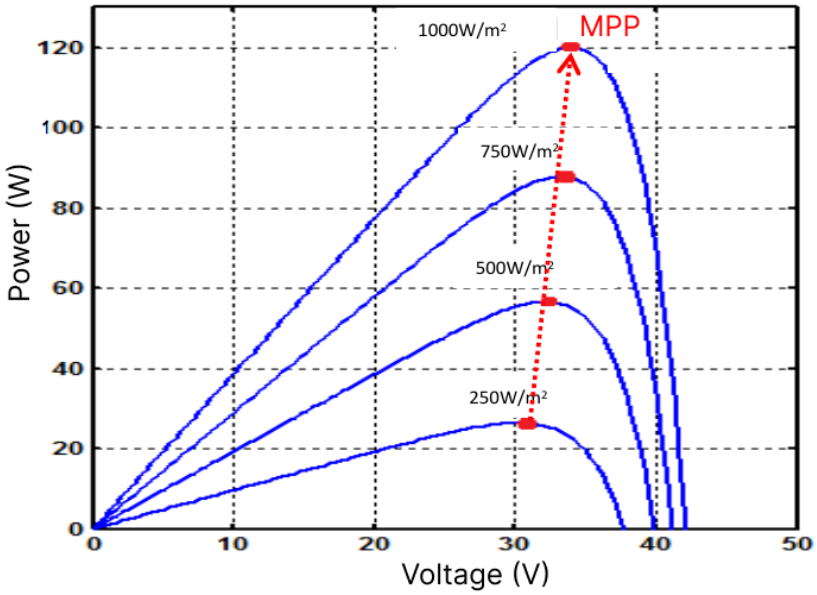
1.1.2.1 PV generator characteristics

a-Influence of irradiance

The I(V) characteristic of PV modules is affected by lighting conditions, which results in a variation in the power output. Figure 1.4 illustrates the I(V) and P(V) characteristics depending on the lighting condition, with a constant temperature $(25°)$.



(a) I-V characteristic



(b) P-V characteristic

Figure 1.4 : I(P) and P(V) characteristic under variation of irradiance

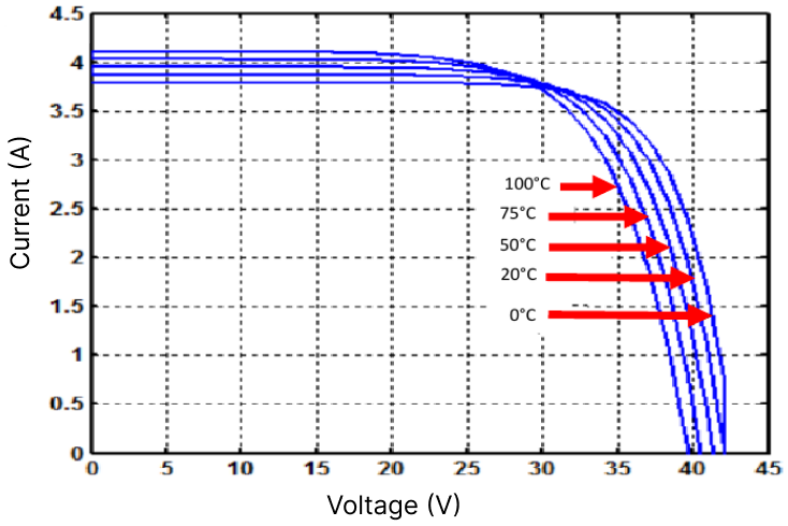
As shown in the figure, the short-circuit current is proportional to the lighting intensity, while the open-circuit voltage doesn't vary as much and stays close to the same value even under low lighting conditions.

This implies that the maximum power output of a panel is proportional to the lighting intensity, and the maximum power point is always located at approximately the same voltage regardless of lighting.

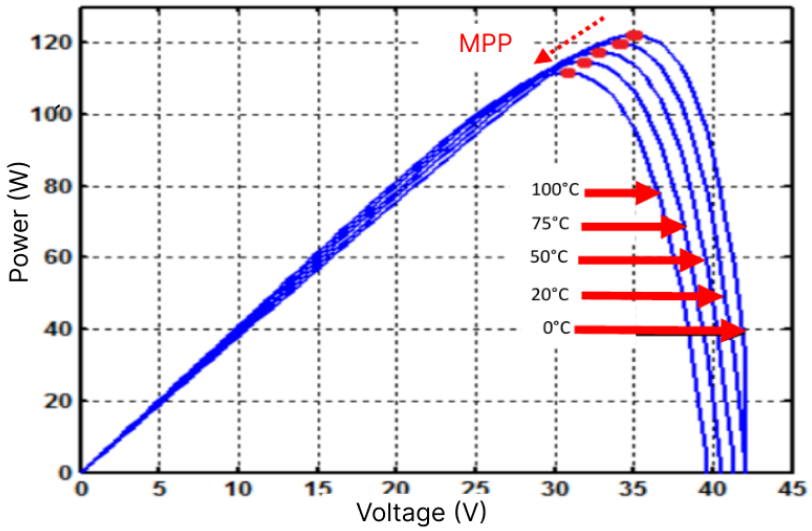
b-Influence of Temperature

Temperature also affects the characteristics of PV panels. Both short-circuit current and open-

circuit voltage are affected, though slightly.
As depicted in 1.5, we can see the effects of temperature change on P-V and I-V characteristics.



(a) I-V characteristic



(b) P-V characteristic

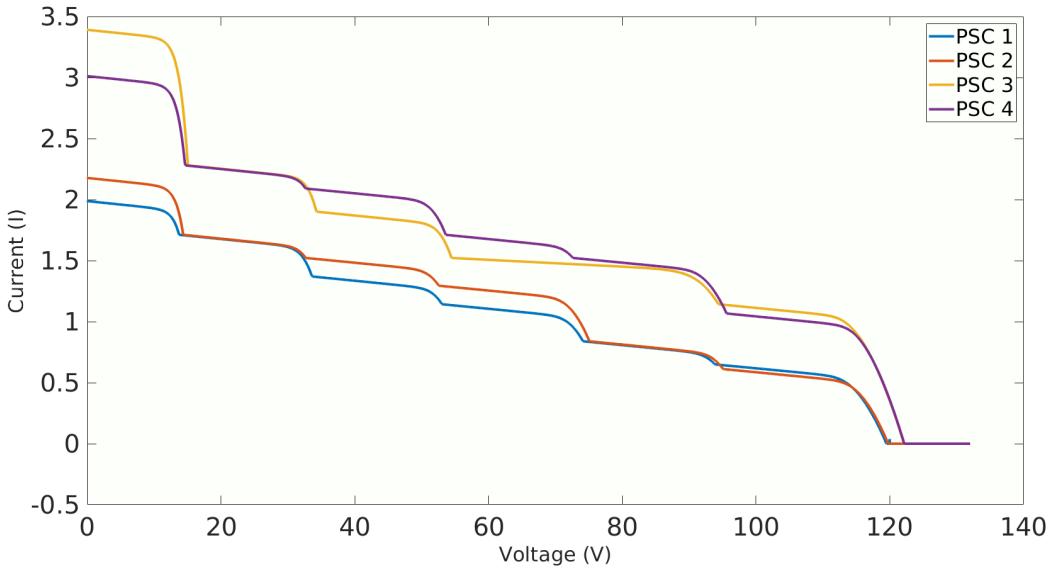
Figure 1.5 : I(V) and P(V) characteristic under variation of Temperature

c-Partial Shading Influence

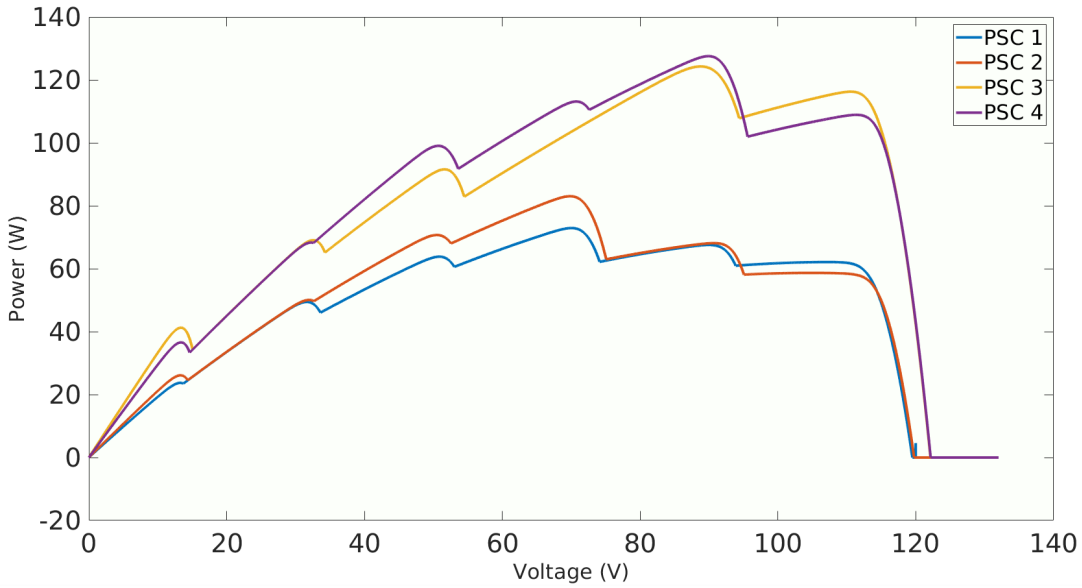
When cells of the same modules are under different irradiance conditions (similarly for different modules on a panel or different panels in an array), the PV modules react in a completely different manner.
The different irradiance conditions are the consequence of the partial shading of the panels. The shades can be caused by neighbouring buildings or trees, passing clouds and birds, or just by the ageing of the panels.

To illustrate this effect, figure 1.6 depicts an example of P-V and I-V characteristics for

different panels under different partial shading conditions.



(a) I-V characteristic



(b) P-V characteristic

Figure 1.6 : Characteristics of different panels under different partial shading conditions

As can be seen, and in contrast with normal lighting conditions, these characteristics present local maximas, which would result in more difficulty reaching the maximum power point.

1.1.3 Photovoltaic control systems

1.1.3.1 The need for a control system

The most basic way to harvest the power of a PV system is directly connecting a load at its output, as illustrated in figure 1.7.

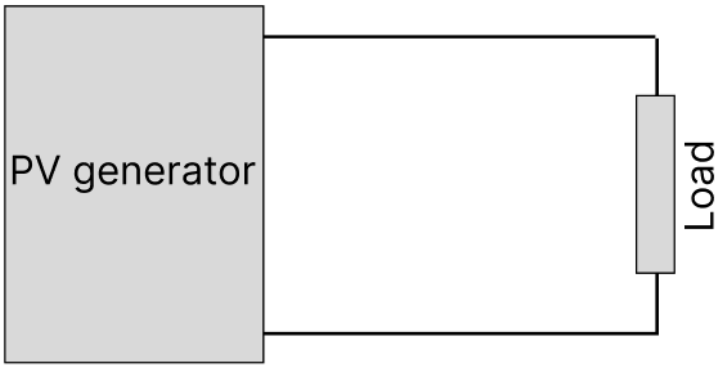


Figure 1.7 : PV generator-load direct connection

With this connection, and in order to be able to harvest the maximum amount of power and for the PV generator to operate at its maximum power point, the load needs to be properly adapted. This presents a few major issues.

- Only certain adapted loads can be used with a given panel.
- The conditions in which the panel is installed can change, which would result in a change of characteristics, and, thus, a change in the required load for maximum power output.

Knowing these problems, having a direct connection is not recommended for getting the maximum amount of power and reaching maximum efficiency, and thus, other ways to connect a load need to be used ; one such solution is the use of a Power Converter.

1.1.3.2 Power Converters

Power converters can be seen as impedance adapters. With a constant load connected at their output, and depending on the control signal provided, the equivalent impedance at their input varies.
Thus, by connecting the panels at the input, we can vary the control signal to reach the maximum power point of the generators, as can be seen in figure 1.8.

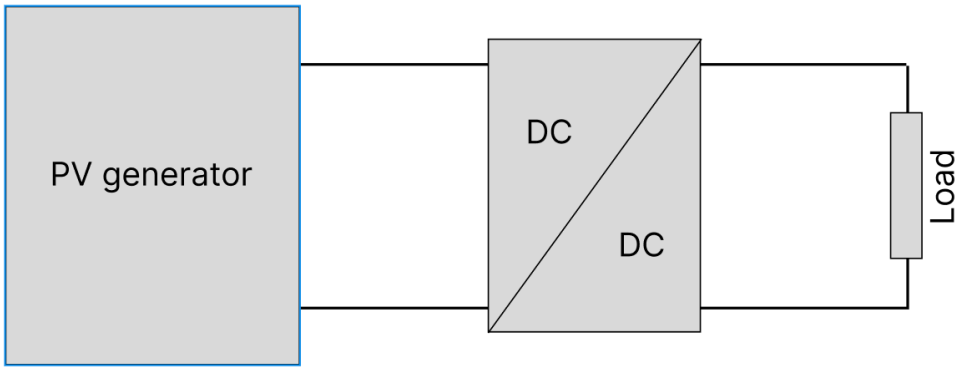


Figure 1.8 : PV generator-load Connection Through a DC/DC Converter

The range of impedance adaptation depends on the type of converter. We consider a resistive load with a value of R_{load} connected at the output of the converters.

- **Buck converters :** The input resistance ranges from R_{load} to $+\infty\ \Omega$.
- **Boost converters :** The input resistance ranges from $0\ \Omega$ to R_{load}.
- **Buck Boost converters :** The input resistance ranges from $0\ \Omega$ to $+\infty\ \Omega$.

1.2 MPPT : Maximum Power Point Tracking

1.2.1 MPPT basic concept

As mentioned in the previous section, with the use of a power converter, we can vary the impedance connected to the output of our PV generator. In order to extract the maximum amount of power, we need to generate the proper control signal that ensures that our system functions at the MPP.
That's where the concept of Maximum Power Point Tracking comes into play. The command based on MPPT aims to find the MPP of a given PV generator and generate the proper control signal that ensures maximum power extraction. Figure 1.8 can be modified to include an MPPT controller, as can be seen in 1.9

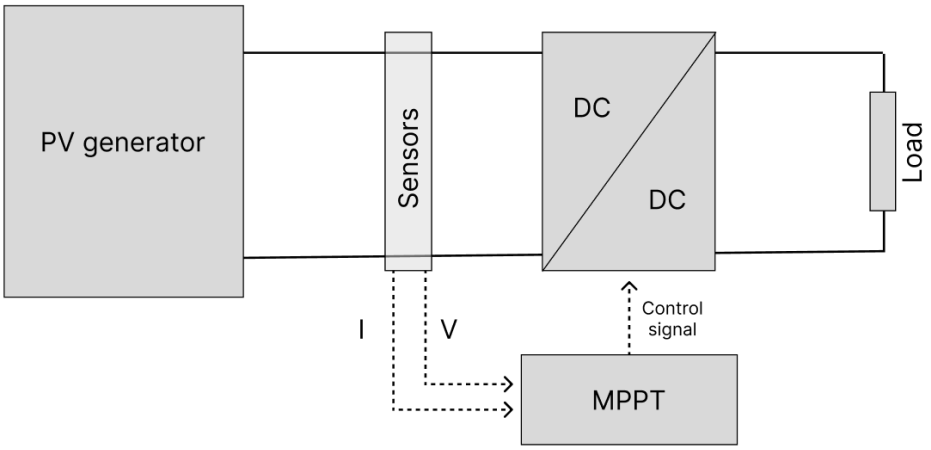


Figure 1.9 : PV Generator-Load Connection including MPPT

1.2.2 MPPT classification

Since the publication of the first command in 1968, various works on MPPT controllers have been carried out in order to constantly increase the effectiveness of photovoltaic systems.
There exist multiple ways to classify MPPT methods based on their performance, origin and computing type.
One such classification is the one proposed in [9], where MPPT controllers are divided into two main categories based on their computing complexity. Conventional methods contain simple methods, and soft computing methods that contain more complex algorithms. Figure 1.10 summarises this classification.

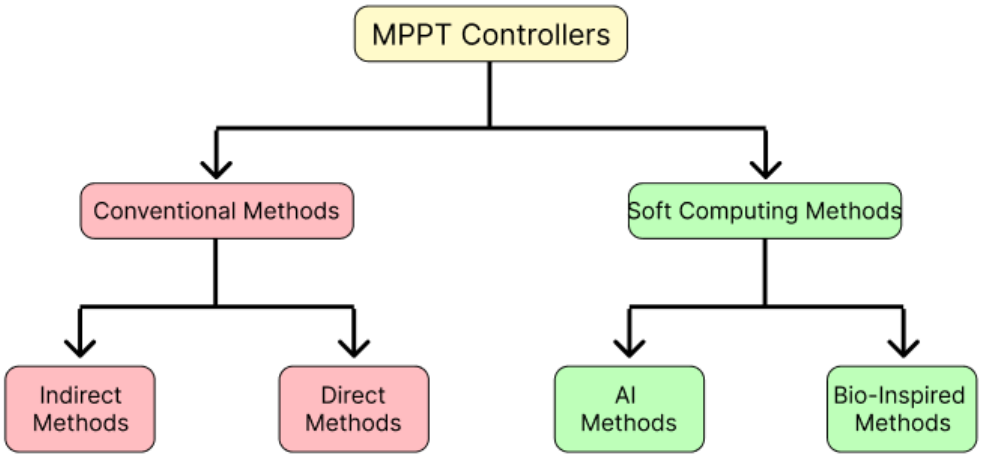


Figure 1.10 : MPPT Controllers Classification

We start by taking a global look at these methods; then, we'll detail the bio-inspired techniques, which are the object of this work.

1.2.2.1 Conventional Methods

These methods are the simplest and easiest to implement. Even though they are not as performant as other methods, they are widely used due to their simplicity, low cost and easy hardware implementation. Under uniform irradiance or slow irradiance changes, they provide a good tracking performance. However, these methods fail to find the Global Maximum Power Point (GMPP) under partial shading conditions (PSCs); even if they converge to the GMPP, they take a long time with low accuracy and with large steady-state oscillation.

The most applied conventional MPPT methods are :

- Perturn and Observe (P&O) [10, 11].
- Incremental Conductance (INC) [12, 13]
- Constant Voltage [14, 15]

1.2.2.2 Soft Computing Methods

These methods use more complicated models and algorithms to track the MPP. They're divided into two categories, Artificial intelligence methods and Bio-Inspired methods.

Artificial Intelligence based methods started appearing these last few years as alternatives to classical methods. They use different AI concepts, and they can be divided into three main categories : Artificial Neural Network (ANN) Based methods, Fuzzy Logic Based methods, and Hybrid methods combining different techniques. These methods present performance that is much superior to conventional methods and also manages to track the MPP under complex shading conditions; they present a much higher computational requirement, which makes their implementation harder and more costly.
A lot of research has been done with this kind of controller, and from which we can cite some that are based on ANNs [16, 17, 18], based on Fuzzy Logic [19, 20, 21], and Hybrid methods that combined different types of techniques [22, 23, 24, 25].

Another recent and emerging type of controller is called bio-inspired methods. As the name implies, they are inspired and based on natural phenomena and animal behaviour. In this work, we'll be interested in these methods, which we will further detail in the following section.

1.2.3 Bio-Inspired MPPT techniques

Recently, Bio-Ispired (BI) methods have attracted much attention due to their potential to find optimal solutions in complex optimization problems (such as multimodal objective functions). These methods are based on the iterative improvement of a population of solutions or a single solution (e.g., Tabu Search) and mainly use randomization and local search to solve a given optimization problem [3]. As illustrated in Figure 1.11, the two most predominant classes of bio-inspired algorithms are evolutionary algorithms and swarm intelligence-based algorithms [9]. These algorithms are derived from the study of natural evolution and the swarming behaviour of living beings [26].

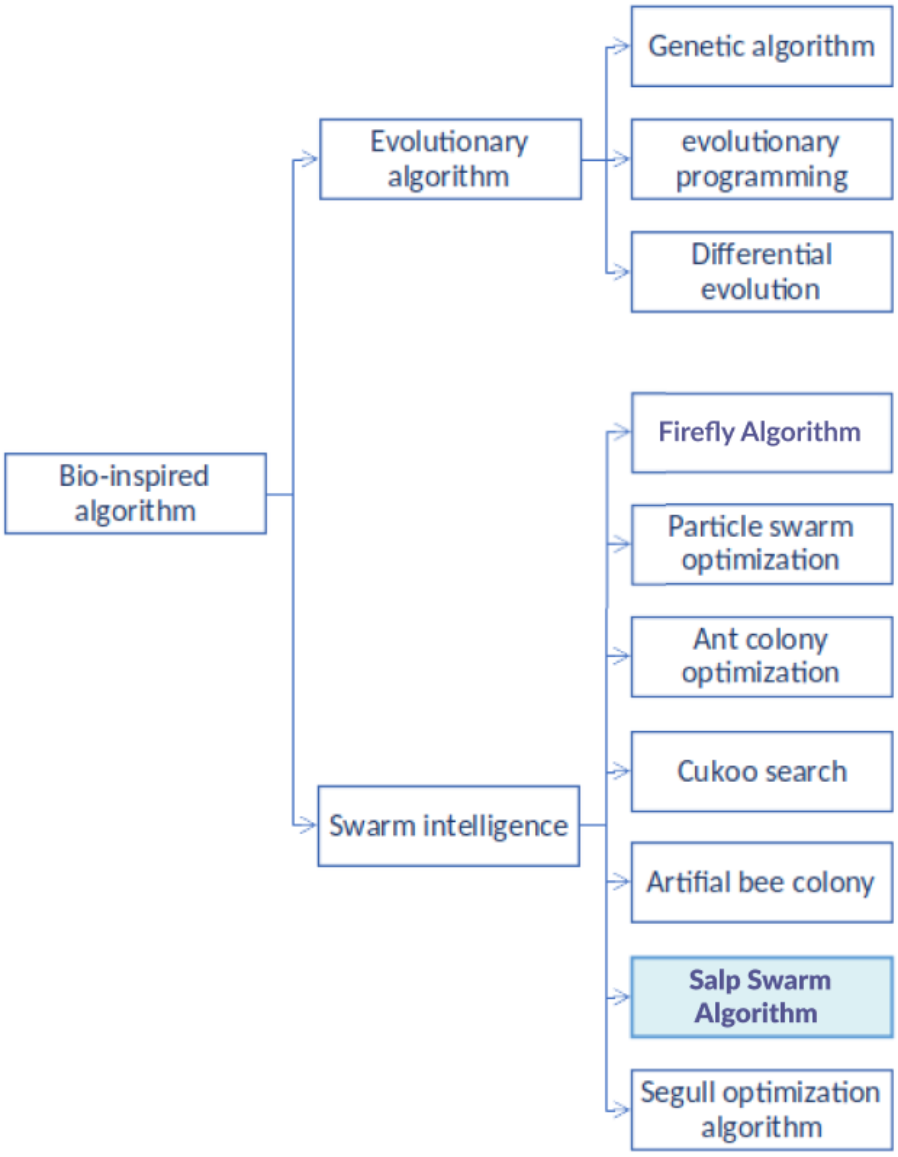


Figure 1.11 : Bio-inspired methods used with MPPT techniques in a PV system

1.2.3.1 Evolutionary Algorithms

These methods are based on the biological evolution of species. They rely on generating an initial population, selecting the best ones and duplicating them while making small adjustments, which would ultimately result in reaching the best solution. This can be summarised in figure 1.12.

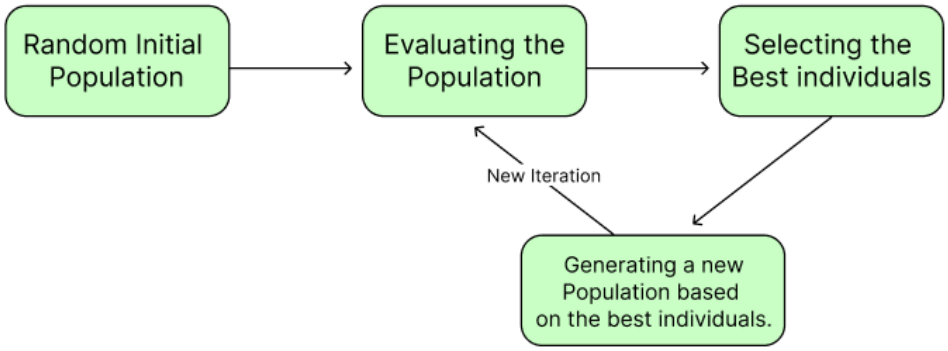


Figure 1.12 : Evolutionary Algorithms flow

Evolutionary algorithms can further be divided into two categories : Genetic algorithms and Differential Evolution Algorithms.
Some examples are discussed in [27, 28, 29] and [27, 28, 29] of Genetics Algorithms and Differential Evolution algorithm respectively, used in MPPT for PV systems.

1.2.3.2 Swarm Intelligence Algorithms

Swarm or collective intelligence represents the behaviour exhibited by certain species, where individuals perform simple and simple actions, all while communicating and coordinating with other members of the swarm. This would result in complex behaviour working towards a goal. In nature, this behaviour is generally exhibited while looking for food. We can see one such behaviour in figure 1.13, where a school of fish exhibit swarm behaviour.

Figure 1.13 : School of fish exhibiting swarming behaviour

Countless Swarm Intelligence methods have been used in PV systems, and it's still a very active research field where researchers keep trying new swarm intelligence techniques. Below, we cite some of the most recent and popular swarm intelligence algorithms.

- Particle Swarm Optimization (PSO) [1, 2, 3, 30]
- The Firefly Algorithm [5, 6]
- Ant Colony Algorithm [31, 32]
- Salp Swarm Algorithm (SSA) : [33, 34]
- Artificial Bee colony [4]
- The bad algorithm [35, 36]
- Seagull optimization algorithm [37]

The various works proposed in the literature on MPPT controllers that are designed around bio-inspired methods and, in view of their advantages, motivated us to choose one of these methods in the context of our work, namely the Salp Swarm Algorithm. It is a relatively new technique that was first introduced in 2017 by Seydali and al [38] and first applied in MPPT in 2019 [34], with ongoing research proving its effectiveness and performance, and proposing modified versions.

Given that this technique shows promise and potential, with proven results by multiple researchers, to date, no hardware implementations have been realized. It's for this reason, we chose the SSA for this work, which we'll further detail and explain in the following chapter.

Conclusion

In this chapter, we took a look into Photovoltaic systems as a whole and detailed their different components. We introduced the concept of Maximum Power Point Tracking by highlighting the need for it. And we took a look through the literature to explore the different techniques that exist.

For our work, we chose the Salp Swarm Algorithm (SSA). In the following chapter, we will explain this technique and realize a software implementation and check its effectiveness.

Chapter 2

Software implementation of Salp Swarm Algorithm

Introduction

This chapter aims to introduce the Salp Swarm Algorithm. We will then focus on applying this algorithm in Maximum Power Point Tracking (MPPT) for PV systems through the implementation of its mathematical model to find the MPP of a photovoltaic generator.

2.1 The Salp Swarm Algorithm

2.1.1 Basic Concept

The SSA algorithm is a swarm intelligence algorithm introduced in 2017 by Seyedali Mirjalili and al [38], inspired by the swarming behaviour of the salp fishes in the ocean.

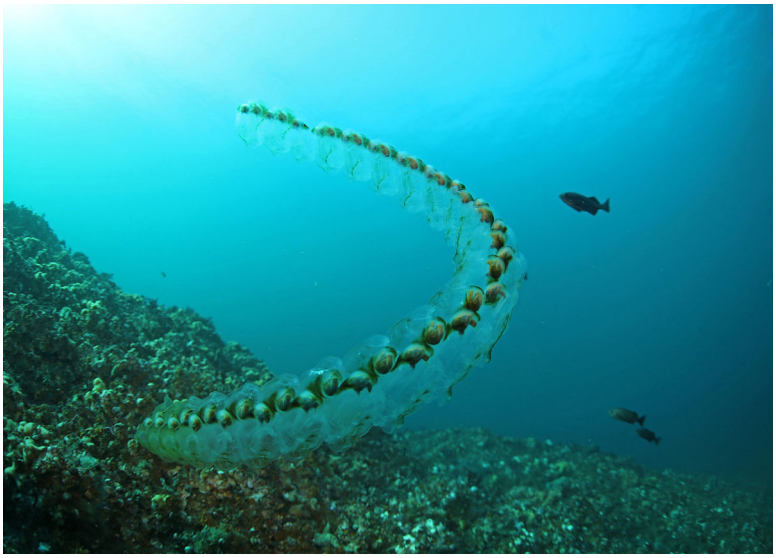


Figure 2.1 : A real salp chain [39]

The swarm in question is originally a kind of salp which are ocean creatures that tend to form chains or swarms (2.1) and exhibit a swarming behaviour : while looking for a food source, the position of each salp is affected by its neighbours' position. This natural algorithm is translated into a mathematical representation which will be detailed further down.
The three following rules govern the algorithm and create the behaviour of a modelled salp :

- The updated position of the salps will depend on the position of the previous salp in the chain.
- For each iteration, a leader is assigned, which has the highest fit value.
- All the salps will try to move towards the leader while searching the adjacent space for better solutions.

2.1.2 Mathematical model for representing and moving the salps

The mathematical model of this algorithm is not complex and easy to implement for engineering problems.

Initially, the salps are uniformly spread out across the search space, meaning they are equally spaced and cover the entire search space. These initial positions are then evaluated, and a fitness value is assigned to each salp to quantify the closeness of the salp to the food source.
Based on these values, they are then divided into two categories : a **leader** and **followers.** The leader is the salp closest to the food source. In other terms, the salp that's closest to fulfilling the constraint of the given optimization problem. The followers are the rest of the salps following the leader while trying to search the space for the food source.

The positions of the leader and followers are updated differently. First of all, the position of the leader is updated using the equation 2.1 [38].

$$x_j^1 = \begin{cases} F_j + c_1((ub_j - lb_j)c_2 + lb_j), c_3 \geq 0 \\ \\ F_j - c_1((ub_j - lb_j)c_2 + lb_j), c_3 \geq 0 \end{cases} \tag{2.1}$$

Where :

F_j : The food source : Meaning the position of the solution we're trying to reach.
c_2 and c_3 are randomly generated numbers between 0 and 1.
Ub_j : Upper bound of the search space in the j dimension.
Lb_j : Lower bound of the search space in the j dimension.
C1 is constant given with the formula 2.2 [38]

$$c_1 = 2e^{-(\frac{4l}{L})^2} \tag{2.2}$$

Where l is the current iteration and L is the maximum number of iterations.

As we can see, the position of the leader does not depend on the other salps and only depends on the position of the food source.

Secondly, the position of the following salps is updated using the equation 2.3 [38].

$$x_j^i = \frac{1}{2}(x_j^i + x_j^{i-1}) \tag{2.3}$$

Where x_j^i is the previous position of the current salp, and x_j^{i-1} is the position of the previous salp in the chain.

The previous equation poses the problem of the following salps only following the leader and not contributing to a better solution [33]. To eliminate this problem, an improved version was proposed, where a random variable was introduced. With this modification, the following salps will continue to follow the leader but will explore the adjacent space better. The proposed formula to be used after a given number of iterations m have passed is the formula 2.4 [33].

$$x_j^i = x_j^{i-1}(l) + r(x_j^{i-1}(l) - x_j^i) \tag{2.4}$$

2.1.3 Steps of the Salp Swarm Algorithm (SSA)

To understand how the SSA works in a single objective optimization problem, we present the following steps as follow :

1. Initialize the salps in predetermined positions or distributions. For example, they can be uniformly or randomly distributed through the search space ;
2. Explore the initial positions and assign fitness values to each one of the salps ;
3. Sort the salps according to their fitness values and optimization criteria, and determine the leader, the followers and the food source (The position of the leader in the case of the first iteration or if the fitness value is the best reached so far) ;
4. Update the position of the salps according to the equations given above (2.1 for the leader and 2.4 for the followers), and assign new fitness values to the new positions in the process. ;
5. Repeat steps 3 and 4 until the convergence criteria is reached or the max number of iterations is hit. Return the best position reached as the solution.

This algorithm can be summarised in the pseudo-code in figure 4.8

```
initialise salp positions ;
while ("Convergence condition unmet" AND "max number of iterations not reached")
        Calculate and assign fitness values for the salp population ;
        F=position of the best fitness value ;
        for i in (salp population)
                if (i==0)
                        Update the leading salp with the corresponding equation ;
                else
                        Update the following salp using the corresponding equation ;
                end if

        end for

end while
```

Figure 2.2 : Single objective SSA pseudo-code

2.2 Application of Salp Swarm Algorithm for MPPT controller

The performance of the Salp Swarm Algorithm described in the previous section is exploited for the design of a powerful MPPT controller to search for the optimal MPP in different working conditions.

For the implementation, the position of the Salps (the regulated variable) represents the duty cycle that controls the DC-DC converter, the fitness values would be the power output at

the given Duty Cycle and the objective function is chosen as the PV output power.

Thus, for the algorithm implementation, a finite state machine model is proposed. Where each state corresponds to a different step of the algorithm.

The flow chart below (figure 2.3) shows how the SSA is implemented for MPPT.

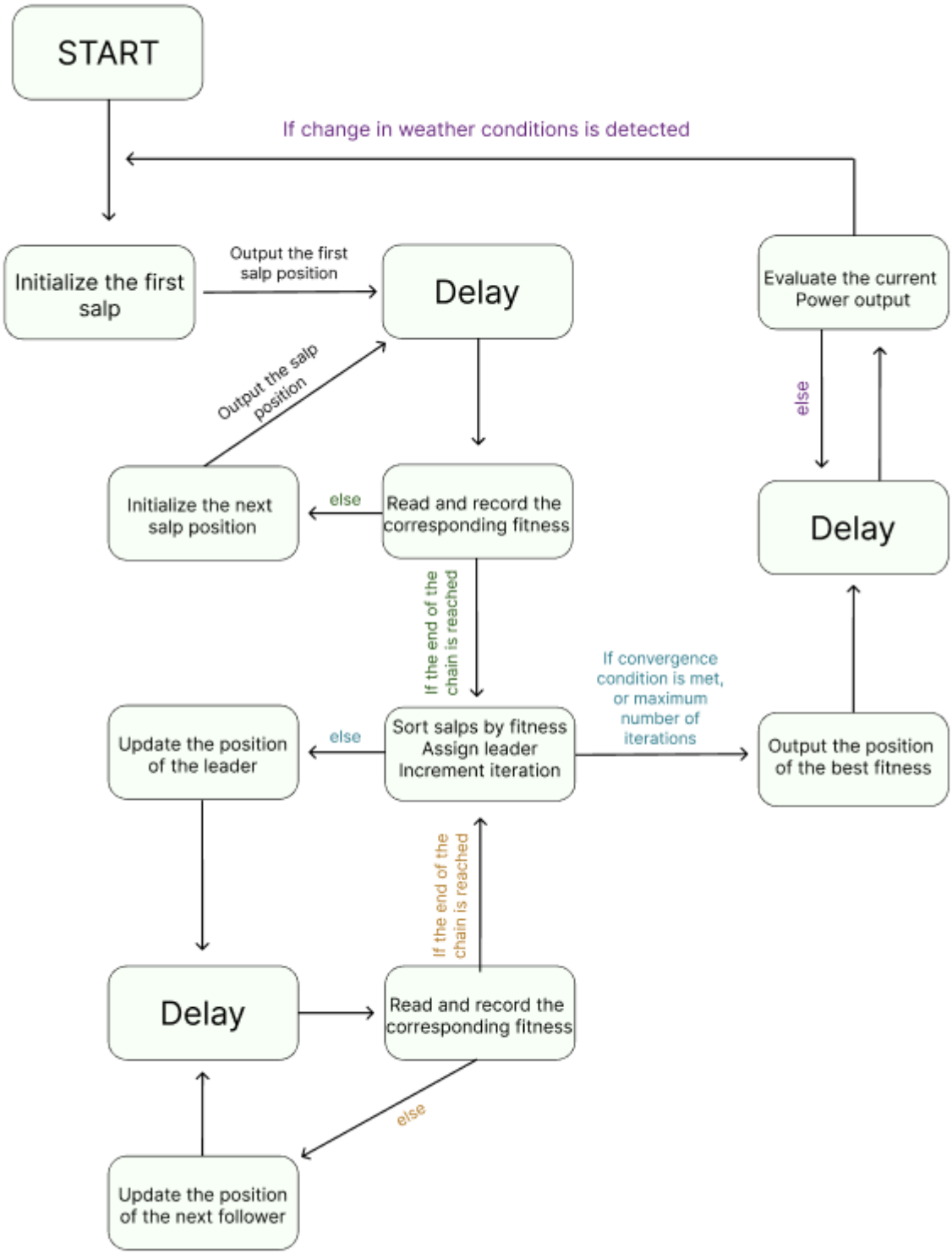


Figure 2.3 : SSA algorithm Flow-chart

2.3 Simulation of the proposed SSA MPPT controller

For the simulation of the proposed SSA_MPPT controller, we propose a design methodology, which consists, first, in modelling the PV system, then we proceeded to the selection of the

SSA_MPPT best parameters. Next, we tested several scenarios to study the response of the proposed controller under different atmospheric variations and partial shading conditions, taking into account the criteria related to the convergence time, steady-state oscillations, accuracy and robustness against atmospheric variations. Finally, we have carried out a comparative study in terms of performance with the bio-inspired PSO_MPPT controller.

2.3.1 Modeling of a PV system

The PV system is composed of the following components : a PV generator, a Buck-Boost converter, a load block and an MPPT controller.

2.3.1.1 Photovoltaic Array

We used a photovoltaic array containing 6 Simens SM50 modules, connected in two parallel branches with three modules in each. Each SM50 module contains 36 cells connected in series. The characteristic of the modules is summarized in table 2.1.

Maximum Power (W)	55
Max power voltage	17.4
Open Circuit Voltage (V)	21.7
Short Circuit Current (A)	3.4

Table 2.1 : SM50 module characteristics

We start by extracting the panel characteristics under various temperature and irradiance conditions.
Figure 2.4 shows the Simulink model that extracts panel data and saves them to the Matlab workplace environment.

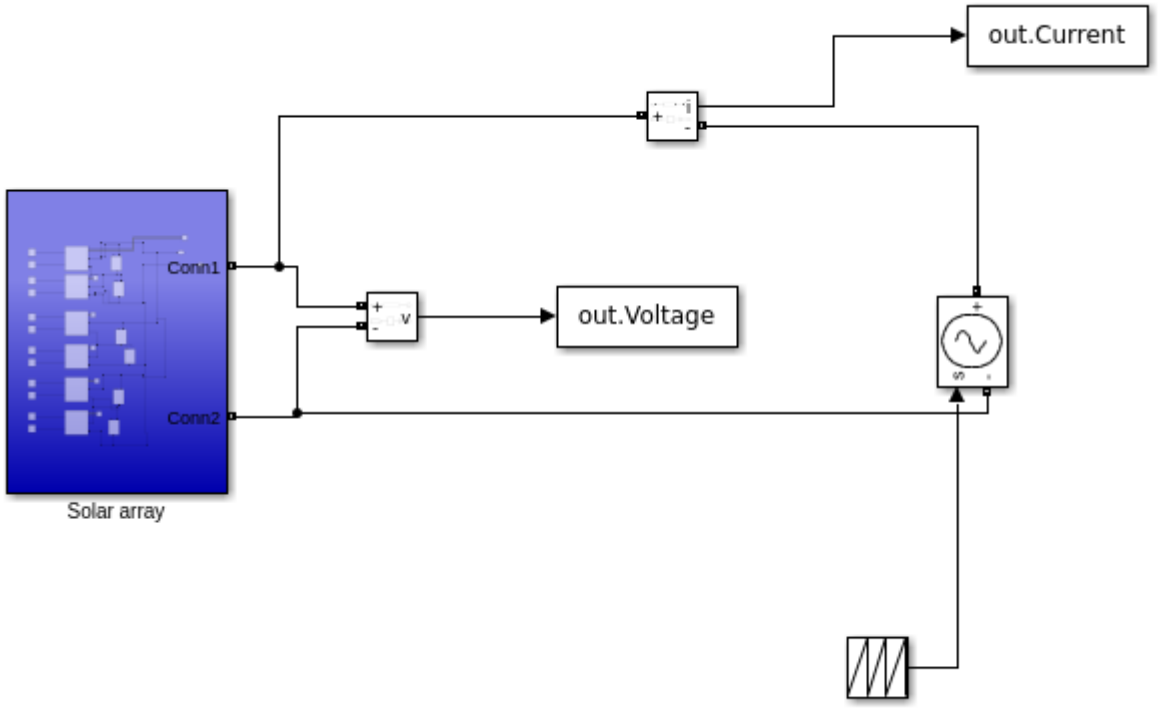


Figure 2.4 : Simulink model to extract panel data

These extracted panel characteristics are saved to be later used in simulating our solar system in a Matlab environment.

2.3.1.2 Buck Boost Converter

For a power converter, we chose to use a buck boost. It presents the widest conversion range, which makes it suitable for a variety of test environments and conditions.
The functionality of the converter and load are summarised in a single Matlab function we call "power_math". It takes as an input the duty cycle and the panel data we saved earlier in the previous section and returns a voltage and current value as an output.
To be able to assign a working point (Voltage and current) for a given duty cycle, we use a mathematical model for a buck-boost converter.
We assume a constant resistive load at the output of the converter, where, for each given duty cycle, the equivalent resistance at the converter's input is given by the equation 2.5 [40].

$$R_{in} = \frac{D^2}{(1 - D^2)} * R_{load} \tag{2.5}$$

Knowing the equivalent resistance value of our circuit, we can calculate the current for a given voltage. We look through our panel characteristics, calculating the current for each voltage value until we find the closest calculated current to the panel current, which will be our working point.

2.3.1.3 MPPT controller

For the proposed SSA_MPPT controller, we define a main function that takes the Voltage and Current as inputs and outputs the Duty-cycle. The function keeps track of iteration and previous states using persistent variables. Subfunctions are also implemented for repeating operations to update the salp position. This main function represents our MPPT controller

The declaration of the function is given in figure 2.5

```
function [Dcycle] = SSA(V,I)
```

Figure 2.5 : SSA function Matlab

The main variables used inside the function are :

- fit : array that keeps track of the fitness(power) values of the Salp chain.
- pos : array that keeps track of the position (duty-cycle) of the salps.
- hist : array that keeps track of results of previous iterations : to evaluate the convergence.
- idx, itr : they keep track of the current iteration and salp.

Along with these variables, the code uses some initial parameters that determine the configuration of the algorithm.

- ub and lb : Lowest and highest duty-cycle value.
- epsilon : threshold to consider the algorithm as converged.
- N : number of salps in the chain.
- delta_ P : change in power to consider a change in weather conditions.

Selection of parameters

The performances of an MPPT controller are strongly related to their tuning parameters (population size, coefficient constituting the algorithm and number of iterations) which directly affect the efficiency, accuracy and robustness of the MPP (i.e. slow convergence and being blocked in a local maximum). So, the parameterization constitutes a critical step for the design and implementation of the MPPT controller. To evaluate the influence of the SAA parameters, several tests were carried out by tuning the values of parameters (Max_iter, Nbr_particules) to select the most suitable ones, thus, reducing the processing time and improve the efficiency of the controller (i.e. reach the MPP). The various tests are performed under standard conditions (1000 W/m2, 25 °C) [41].

The values of SSA parameters are selected properly to balance the exploration and exploitation process. After the different simulations, we have selected the best parameters as follows : the population size N=3 with 20 maximum iterations, a lower bound of 0.02 and an upper bound of 0.98 for the search space.

2.3.1.4 Testbench script

The previously described component of a PV system will be assembled in a single testbench file that simulates its functionality.

The testbench script can be summarized in the pseudo-code shown in figure 2.6.

```
load(panel_characteristics) ;
V=0 ;
I=0 ;
for t=0 :step_size :simulation_time
        Duty_cycle=SSA(V,I) ;
        [V,I]=power_math(Duty_cycle,panel_characteristics) ;
        save V and I values to evaluate performance ;
end for ;
```

Figure 2.6 : Matlab testbench pseudo-code

2.3.2 Results and performance

In this section, we will test and evaluate the performance of the proposed controller.
To evaluate and compare the performance of the proposed SSA_MPPT_controller, different scenarios are developed. The behavioural study is based on these criteria :

- Convergence speed ;
- Accuracy ;
- Stability i.e. steady states oscillations ;
- Robustness.

To prove the effectiveness of our implementation, the proposed MPPT controller is tested under various conditions, including partial shading.
To validate the high performance of the proposed controller, a comparative study with PSO_MPPT controller is done.

2.3.2.1 Evaluation of Convergence

To study the convergence of our implementation, we simulate with a panel under standard temperature and irradiance conditions $\left(1000\text{W}/m^2,\ 25^\circ\right)$. We plot the evolution of power output and duty cycle over time. We also illustrate the characteristic of the panel as well as the functioning point reached by the controller. Results are shown in figure 2.7

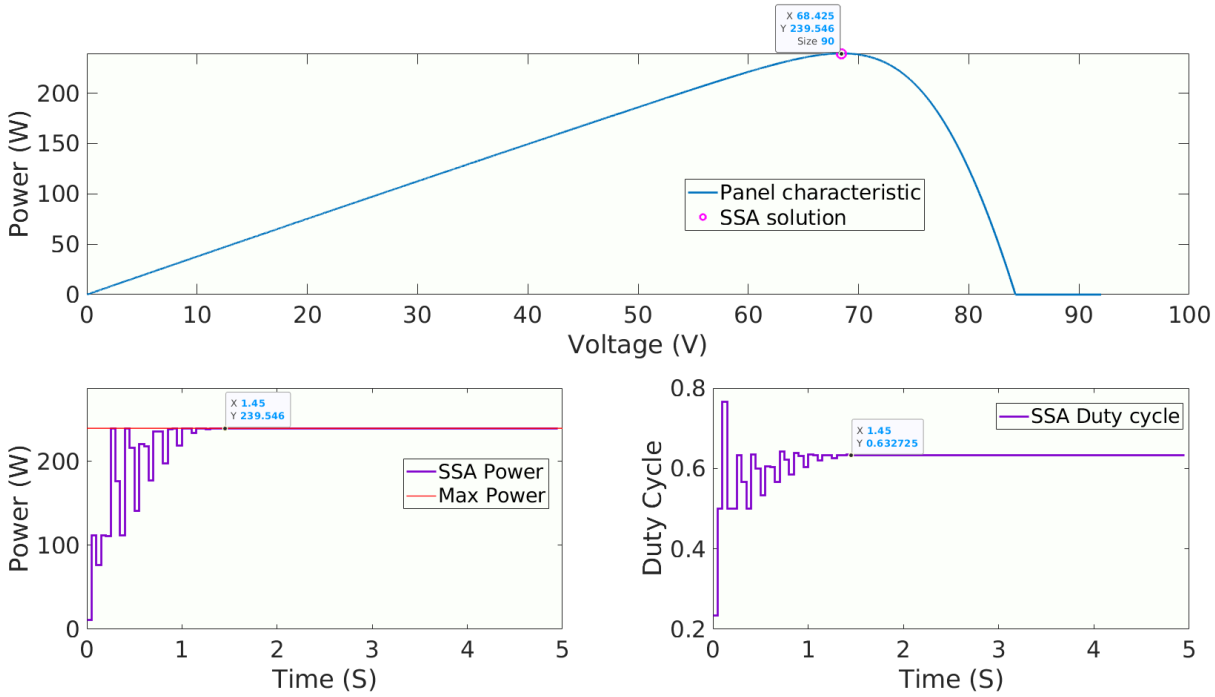


Figure 2.7 : SSA Performance, standard conditions

As can be seen in figure 2.7, our algorithm fully converges in less than 1.5 seconds and reaches the maximum power point of the panel.

To put this result into perspective and evaluate it, we try to compare it to a more conventional swarm method, the PSO algorithm. Figure 2.8 illustrates the results of the PSO controller under the same conditions.

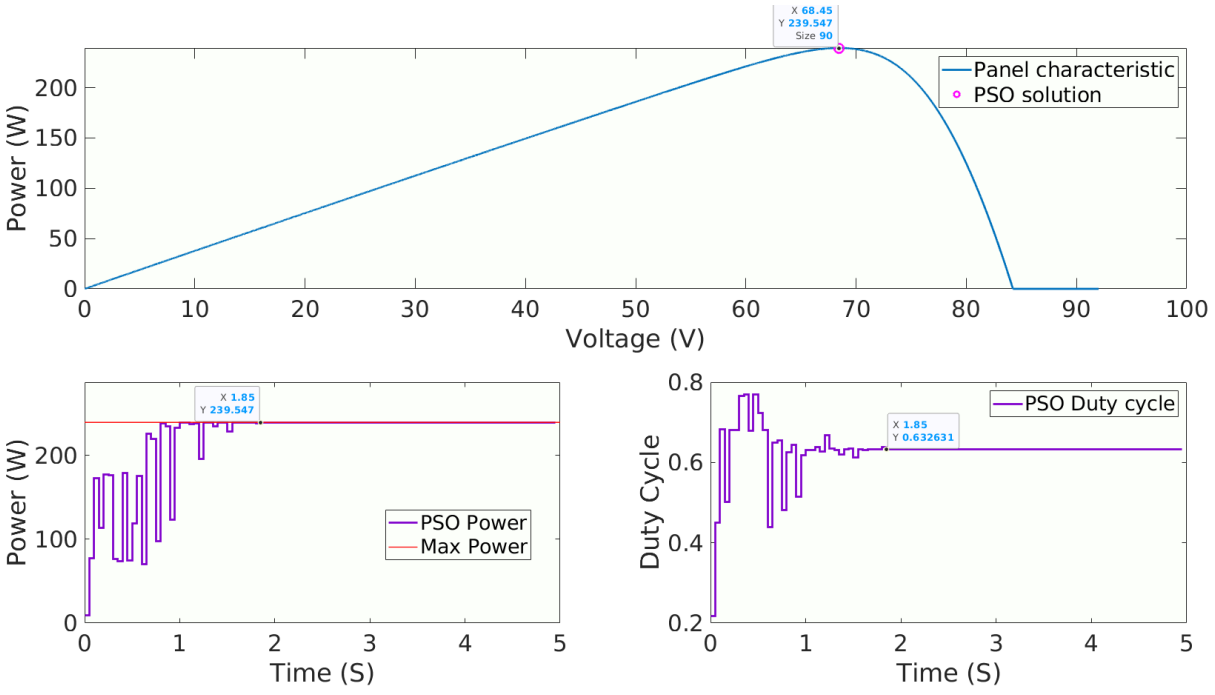


Figure 2.8 : PSO Performance, Standard conditions

The PSO algorithm converges to the same solution as the SSA, which represents the maximum power point. However, the SSA presents a much shorter convergence time, converging in just under 1.5 seconds, while the PSO algorithm needs more time, converging around 1.8 seconds.

2.3.2.2 Evaluation of robustness and stability

The robustness and stability of a tracker can be tested under unstable environments, meaning environments presenting a change in weather conditions : temperature and irradiance.

a-Change in temperature :
We first simulate our algorithm in an environment presenting a change in temperature.
We start at the standard temperature and irradiance conditions $(1000\ \mathrm{W}/m^w,\ 25^\circ)$, and after 2.5 seconds, we set the temperature at 35° while maintaining the radiance.
Figure 2.9 shows the results of the simulation.

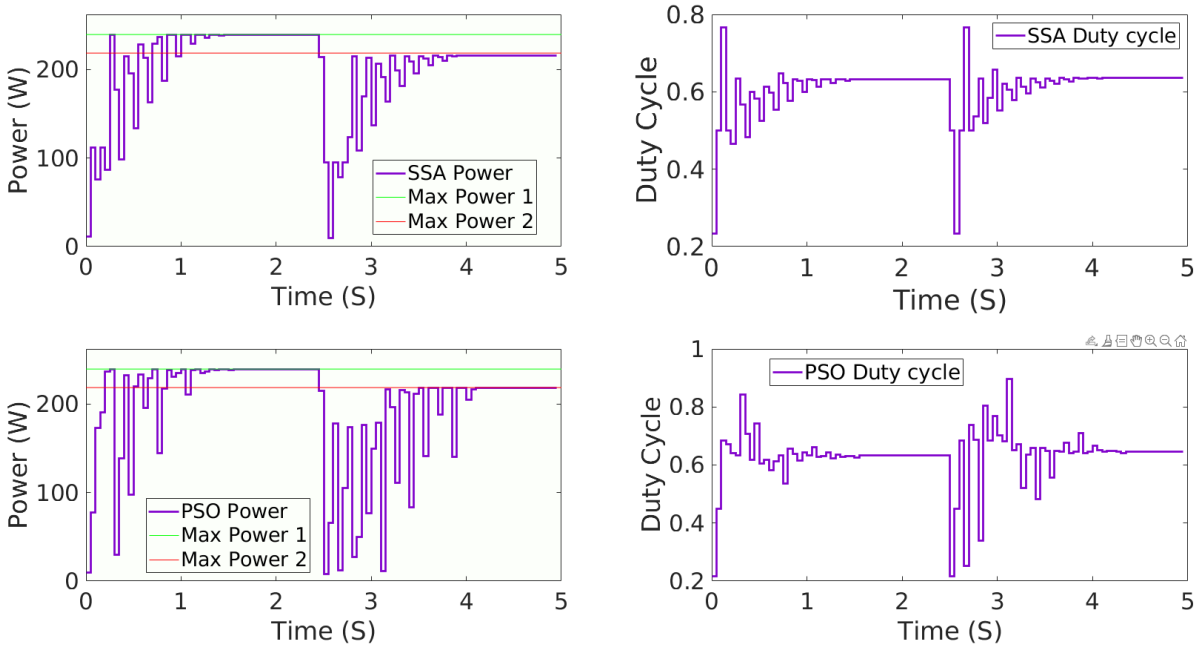


Figure 2.9 : SSA and PSO Performance : Change in temperature

After the temperature change, both the SSA and PSO algorithms restart to then converge to the new MPP. The SSA still shows shorter convergence times compared to the PSO.

a-Change in irrandiance :

In this simulation, we test our algorithm in a sudden change or irradiance, and we'll assert its performance. We start at the standard conditions $(1000\ \mathrm{W}/m^w,\ 25^\circ)$, then after 2.5 seconds,

we change the radiance to 800 W/m^2 while keeping the temperature at 25°.
The results are illustrated in figure 2.10.

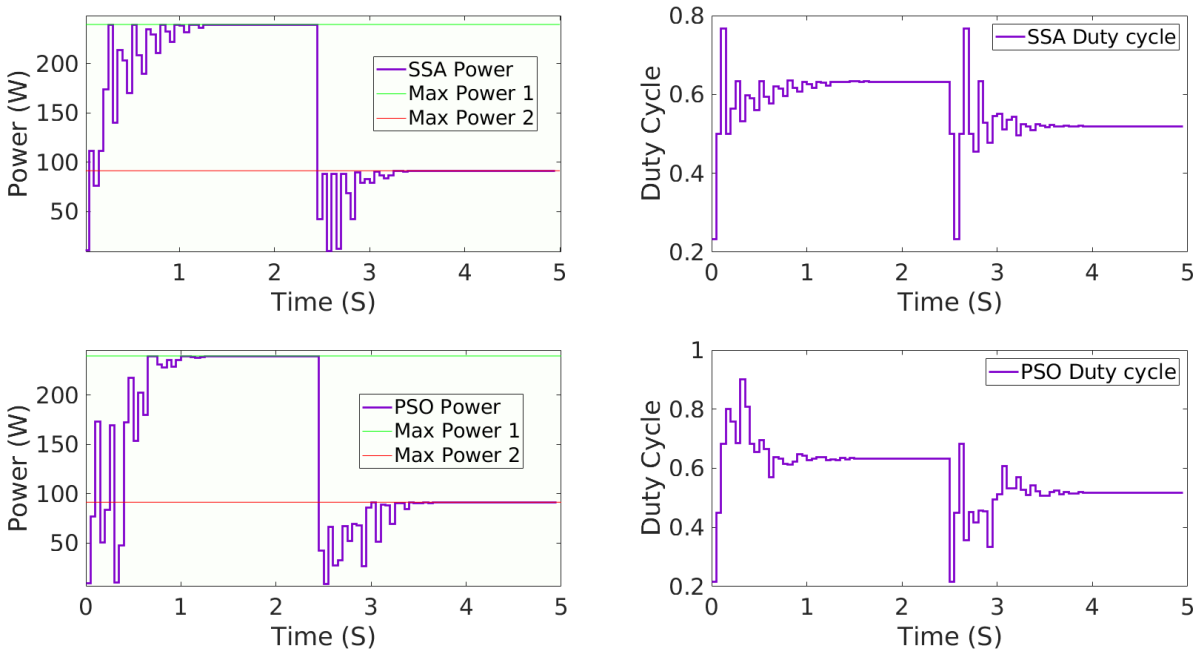


Figure 2.10 : SSA and PSO Performance : Change in irradiance

Both algorithms converge to the first MPP with standard conditions. After the irradiance change, both controllers restart and eventually converge to the new MPP.
Even tho both algorithms show perfect tracking performance, SSA performs slightly better in terms of convergence time.

2.3.2.3 performance under partial shading conditions

In this test, the goal is to determine the performance of our controller under non-uniform lighting conditions.
To do that, multiple partial shading scenarios were set up, where each module in our array is exposed to a different irradiance under standard temperature conditions.
Table 4.3 shows the four partial shading scenarios.

Room \ Date	1	2	3	4	5	6
PSC 1	1000	800	800	1000	800	1000
PSC 2	600	800	1000	800	600	400
PSC 3	1000	500	1000	1000	500	1000
PSC 4	300	500	800	1000	1000	1000
PSC 5	400	1000	100	800	800	600

Table 2.2 : Various partial shading conditions

After extracting the panel characteristics under these conditions, they're used to simulate our

controller, which will be compared to the PSO. Figures figs. 2.11 to 2.15 show the simulation results under the different shading conditions for both SSA and PSO while showing the panel characteristics under the same conditions.

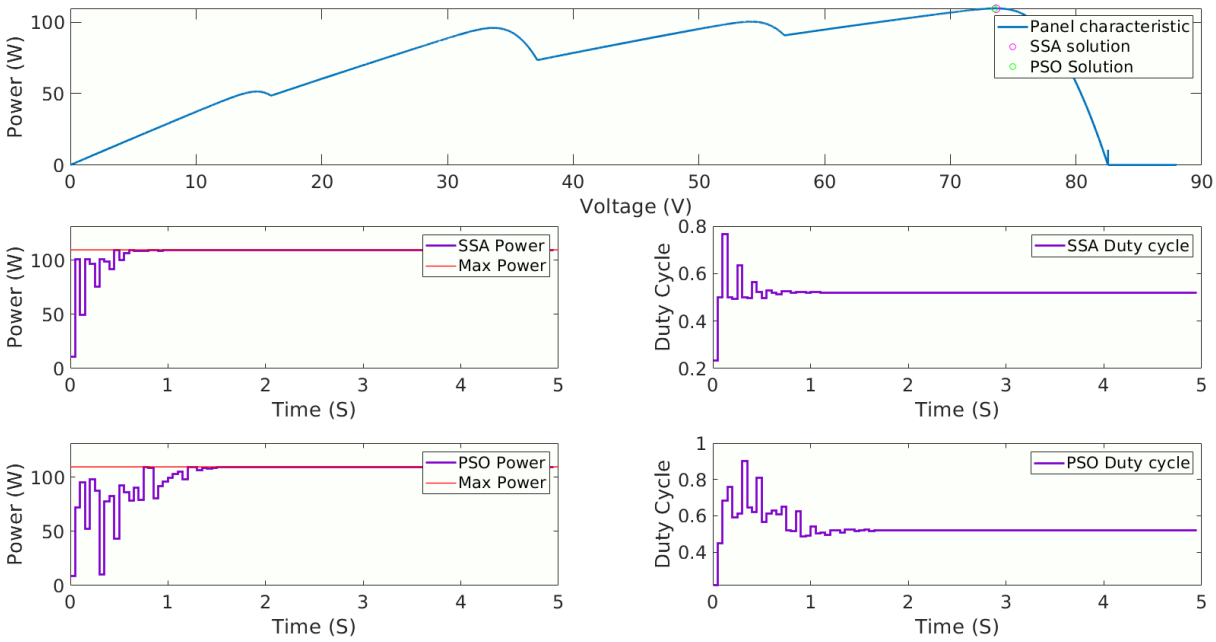


Figure 2.11 : PSO and SSA Performance : partial shading 1

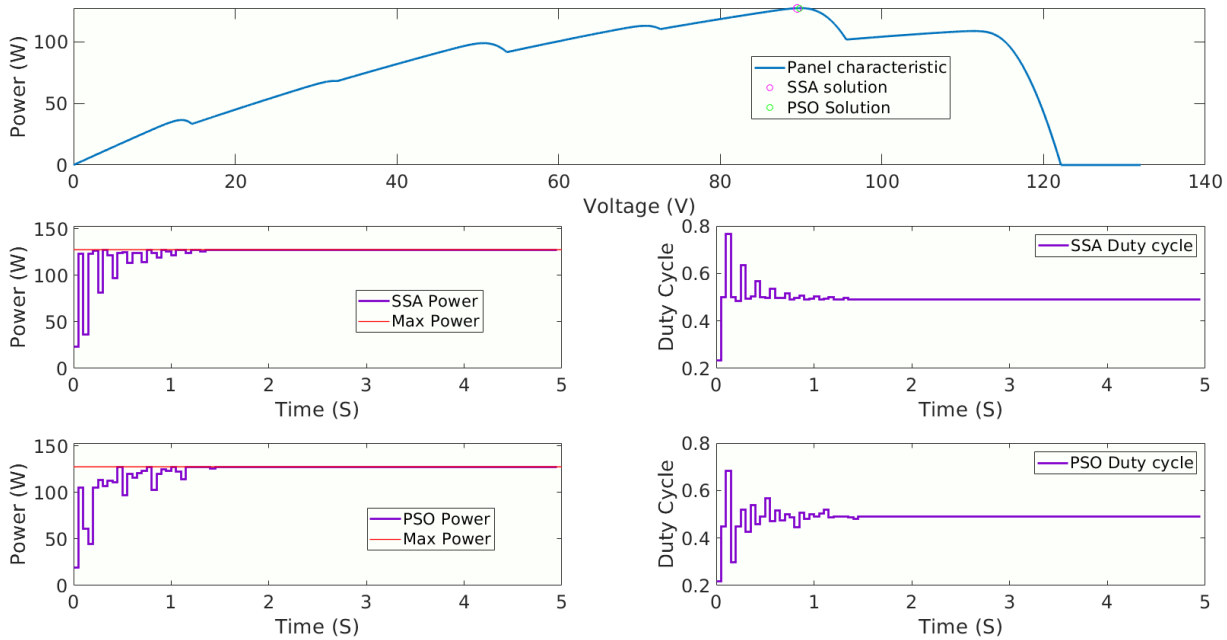


Figure 2.12 : PSO and SSA Performance : partial shading 2

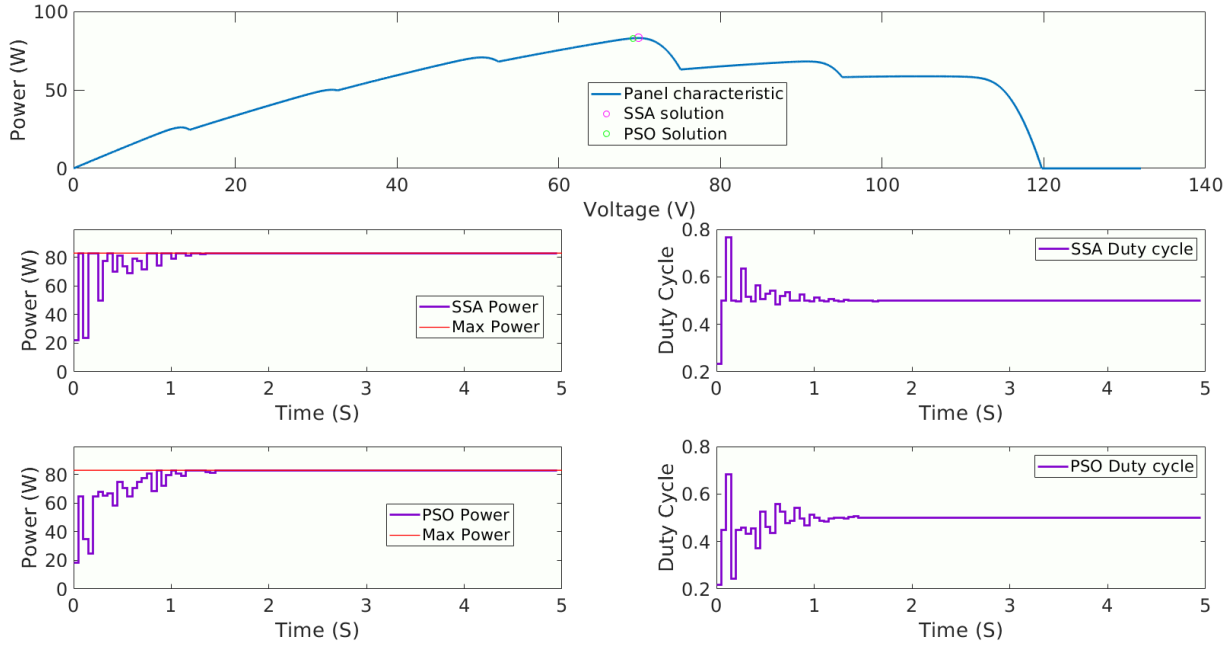


Figure 2.13 : PSO and SSA Performance : partial shading 3

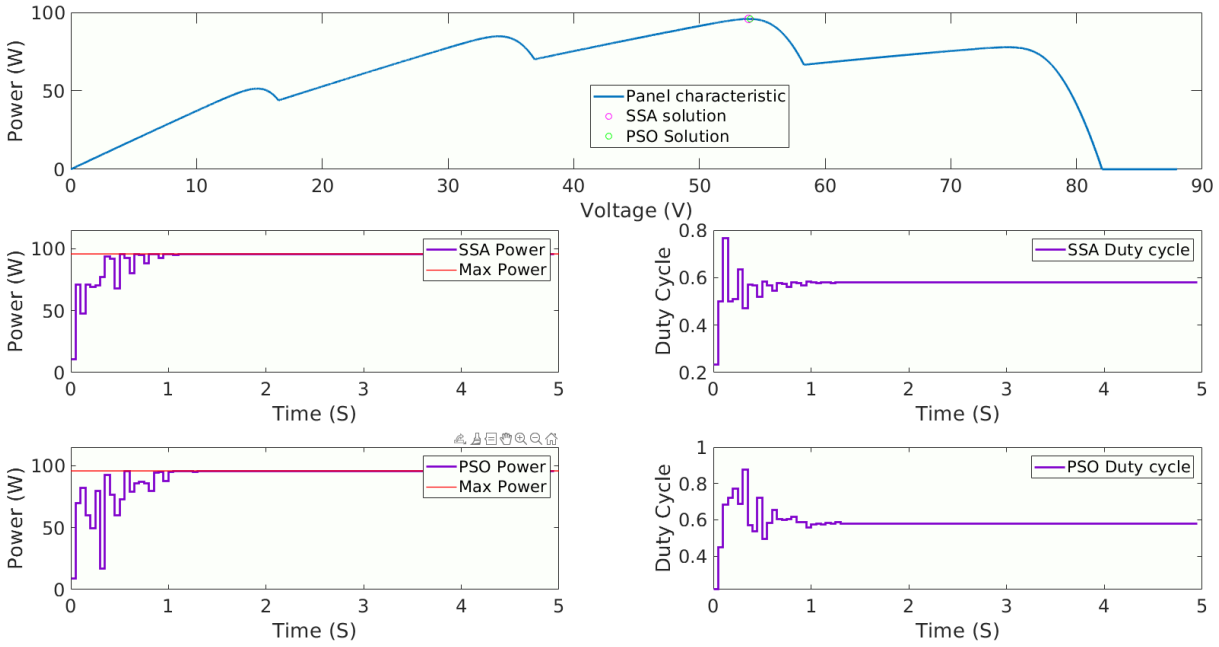


Figure 2.14 : PSO and SSA Performance : partial shading 4

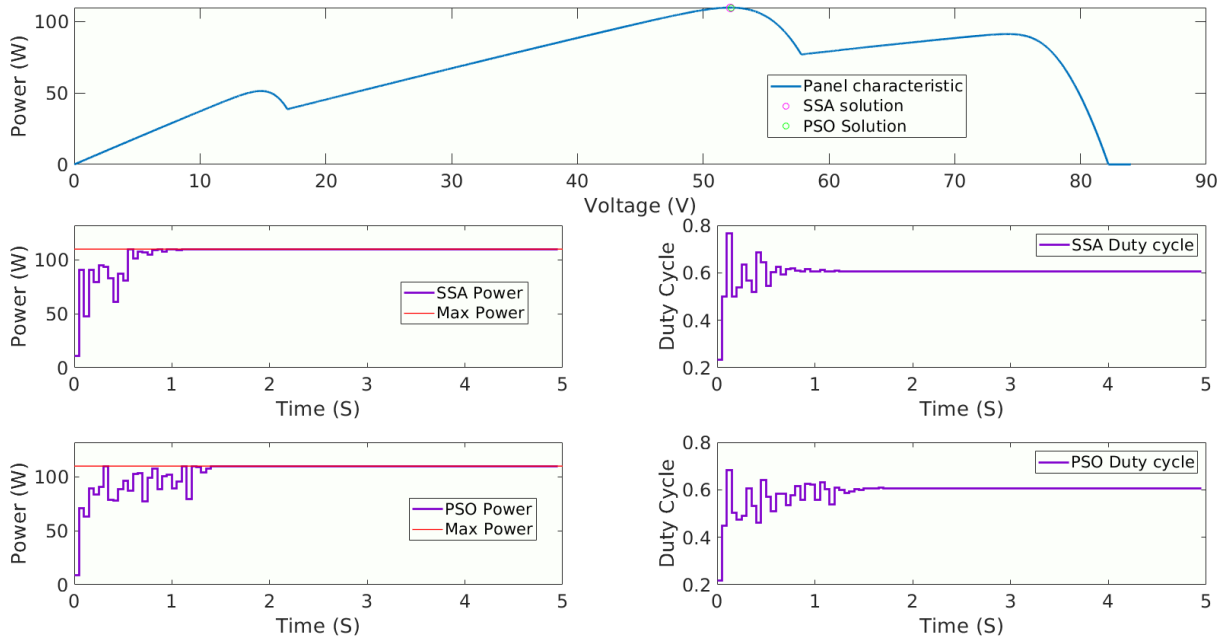


Figure 2.15 : PSO and SSA Performance : partial shading 5

As can be seen in the above simulations, both algorithms manage to converge to the maximum power point in all shading conditions, and the SSA consistently manages to converge in less time compared to the PSO.

To further test and check the robustness and consistency of the algorithms, we run both algorithms on all the above panels 50 times, and we record the outputs for tests, from which we can extract information that will indicate the performance. Mainly the average convergence time and the number of times the algorithm manages to reach the maximum power (within 2%). The results are summarised in the table 2.3.

Panel	SSA		PSO	
	Convergence time	Convergence ratio	Convergence time	Convergence ratio
1	1.78	98%	1.63	90%
2	1.82	98%	1.82	60%
3	1.77	84%	2.04	96%
4	1.77	100%	1.8	100%
5	1.81	96%	3.77	64%

Table 2.3 : Simulation results of 50 tests on each panel

As mentioned in Table 2.3, the proposed SSA_MPPt controller algorithm is faster and more accurate than the PSO_MPPT controller under various test conditions.
This proves not only its efficiency but also its robustness.

Discussion of results

From the conducted simulations, we can conclude that the SSA algorithm, similar to the PSO, manages to keep track of the MPP in all tested conditions.
Compared to the PSO, it consistently converges in a shorter time, which makes it more efficient overall.

Conclusion

In this chapter, we initially presented the Salp Swarm Algorithm, starting from its organic inspiration to its corresponding mathematical model. Subsequently, we explored its utilization in photovoltaic (PV) systems for effectively tracking the maximum power point. Finally, we executed the implementation and simulation of the controller in the Matlab environment, demonstrating its efficacy across diverse test scenarios and conducting a comparative analysis with the Particle Swarm Optimization (PSO) algorithm.

Moving forward to the subsequent chapter, having established the effectiveness of the SSA as a Maximum Power Point Tracker, we can now progress to the next phase of our work, which is the hardware implementation.

Chapter 3

Hardware Implementation of SSA_MPPT controller

Introduction

In the last decade, the field of microelectronics proposed new digital solutions such as Field Programmable Gate Array FPGA circuits for implementing control algorithms. Due to the inherent parallelism of FPGA circuits and their high computing capabilities, they can considerably reduce computation delays of the algorithm's implementation. Thus, embedding complex Bio-inspired algorithms into programmable devices such as FPGA circuits may play a major role in PV systems.
On the other hand, it remains complex for novice designers who do not have basic knowledge of Hardware Description Language (HDL) coding and difficult for expert designers who handwrite HDL code to get FPGA implementations in a short time (debugging, test, etc..). To shorten the design time and save money, a new design methodology called High-Level Synthesis emerged and gained popularity.
With this in mind, and as part of our work, we have targeted FPGA circuits and a High-Level synthesis approach for the hardware implementation of the proposed bio-inspired Salp Swarm algorithm. In this chapter, we will first introduce the Field Programmable Gate Array (FPGA) circuit, with its main characteristics and architecture. Then in the second step, we will discuss the HLS Design methodology for SSA_MPPT controller implementation, where we will target mainly the Vitis HLS method with the Vivado design tool.

3.1 Hardware design

3.1.1 Field Programmable Gate Arrays

FPGAs, or Field Programmable Gate Arrays, are integrated circuits which have programmable logic capabilities. Introduced by Xilinx in 1985 [42], their influence on so many fields is still rapidly growing 30 years later.

These circuits are composed of logic elements interconnected with one another by routing resources. The special feature of an FPGA is its reconfigurability. Logical elements, as well as the interconnect network, can be reconfigured to support various applications.

3.1.2 FPGA Architecture

As depicted in figure 3.1, the main architecture of an FPGA circuit contains three basic blocks that make up its functionality

- I/O Blocks.
- Programmable Interconnection Wires.
- Configurable Logic Blocks (CLBs).

Each CLB contains a Basic Logic Element (BLEs) used to implement the logical part of the circuit. The BLE consists of a set of transcoding tables (Look Up Table : (LUT) and a D-toggle to implement the basic functions with the memory block (SRAM), followed by a multiplexer.

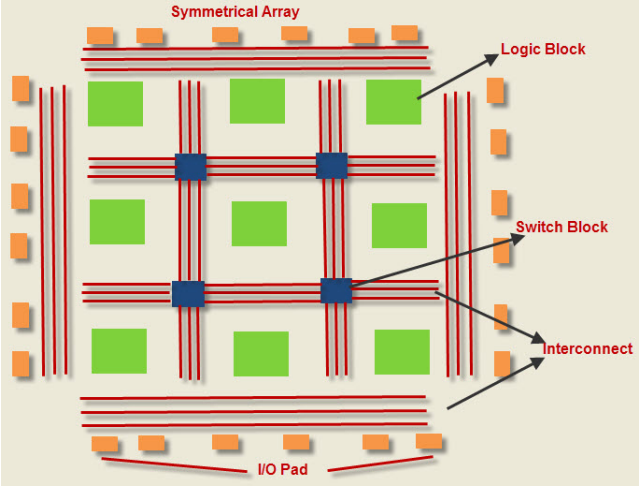


Figure 3.1 : FPGA Architecture [43]

Modern FPGAs also contain application-specific blocks (DSP blocks, for example) ; these blocks are also considered as CLBs, and used as such.

3.2 FPGA Design methodologies

Modern FPGA design tools offer a variety of ways to describe hardware behaviour. Based on very structured approaches, these methods allow a direct migration from a system modelling or a high-level algorithm to an HDL design.

Faced with the constraints of time and development complexity induced by the use of the classical design methodology (Handwriting HDL), many works have been carried out to raise the level of abstraction for the specification of the algorithm. The objective is to propose new tools for the automatic generation of RTL descriptions from algorithmic specifications using higher levels of abstraction, such as C, C++, and SystemC. We then speak of "High-Level Synthesis (HLS)"
Thus, the implementation on FPGA circuits then becomes more accessible, requiring less expertise in hardware design.

In our work, we are interested in the high-level synthesis approach with the exploitation of the Vitis HLS and Vivado design flow.

This approach will allow us to automatically generate hardware description block from our high-level specification (C algorithm, for example) using the Vitis HLS flow, which will then be integrated into the Vivado design flow along with other RTL components.

As can be seen in figure 3.2, the design methodology can be divided into two stages :

- Generating an IP from a high-level description (C algorithm) using the Vitis HLS tool.
- Integrating the generated IP, along with other HDL sources, and following the classical Vivado flow.

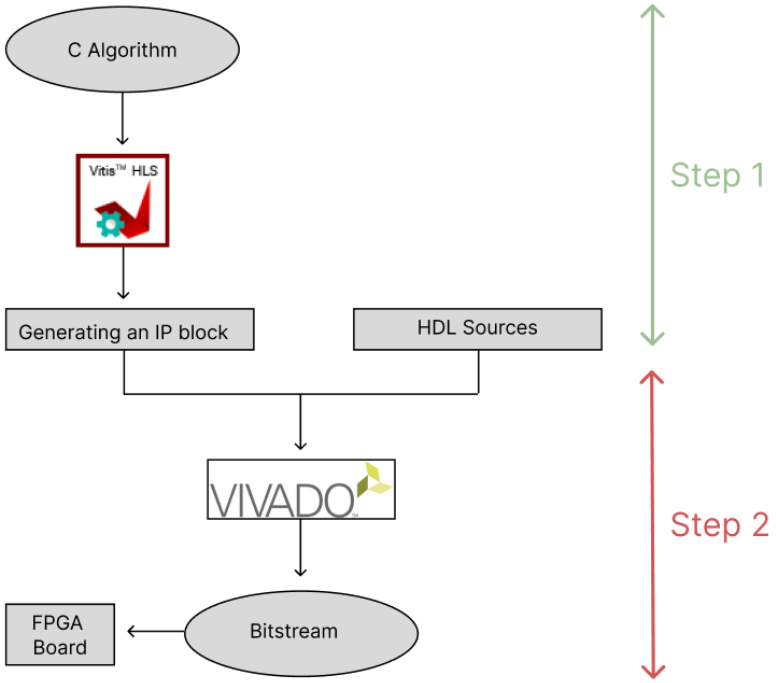


Figure 3.2 : HLS/Vivado design flow

3.2.1 Step 1 : Vitis HLS design flow

As mentioned previously, high-level synthesis (HLS) is a design methodology where we have the ability to generate production quality register transfer level (RTL) implementations from high-level specifications [44].

The development of a hardware design starts with specifications or a chart that the design needs to meet. Among other things, the specifications indicated the desired functionality of the final design.

As depicted in figure 3.3, HLS design flow starts from a software implementation; in this case (For Vitis_hls), we start with a C/C++ code containing a main function that we want to be implemented, and a C testbench, that's needed to be used to test the functionality of the generated RTL [45].
Along with the source files, HLS also takes as inputs hardware constraints : Chosen hardware target, clock frequency and other hardware parameters.

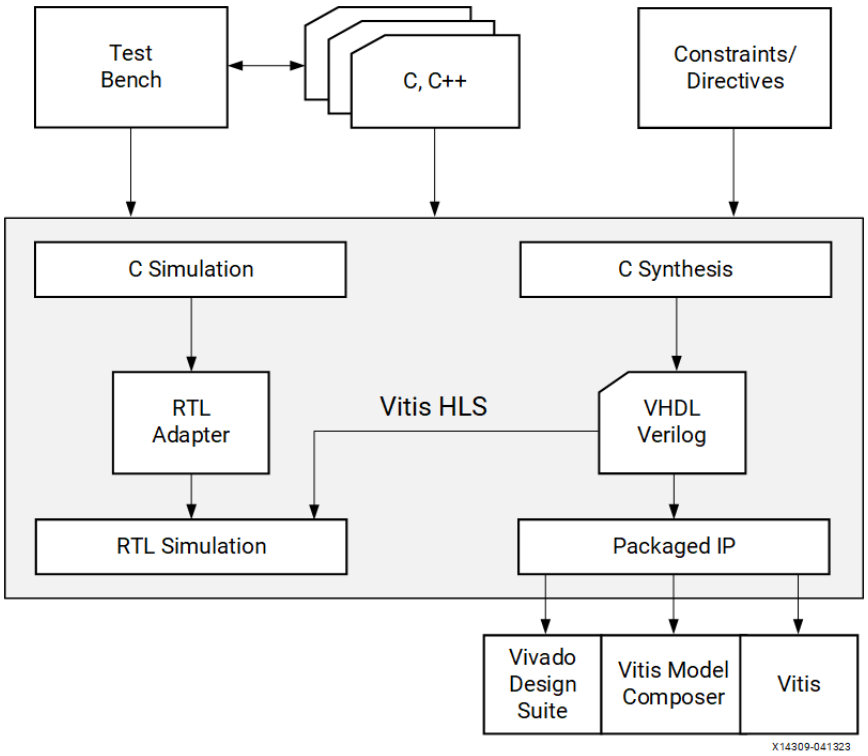


Figure 3.3 : HLS Design Flow
[46]

The first step is C simulation, where the source codes are taken, compiled and run. The goal is to verify the functionality.
After validating the results, we can move on to the next step, which is C synthesis, where the HLS tool converts the source code into RTL. In this step, reports are generated regarding the performance estimations of the generated architecture, which will be used to check for compliance with design requirements. If the requirements are not met, edits can be made to the initial source code or change optimisations through directives to the tool. Once satisfied with the results, we move to the next step.
The generated code is simulated using the same C testbench, and the results are compared between the RTL implementation and the C simulation. This step is called RTL/C co-simulation. The tool automatically compares results between the two simulations and validates the generated RTL code.
Once the co-simulation is passed, the output can be packaged in an IP that will be used in other designs and can be included in Vivado Projects.

3.2.2 Step 2 : Vivado design flow

The basic design flow for an FPGA design can be summarised in the diagram in figure 3.4.

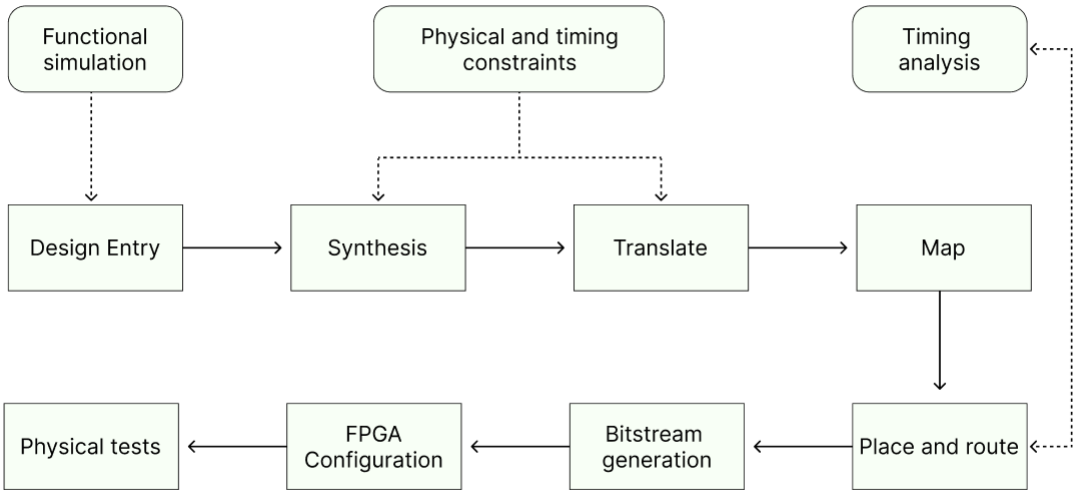


Figure 3.4 : FPGA Design Flow

The design entry, or source files, take the form of HDL code (VHDL or Verilog), IP (Intellectual property), or block design files. These source files can be generated or created in different ways, but mainly, they can be created manually through RTL coding or using an HLS approach.
After the creation of the main design, regardless of the approach, a test bench needs to be created. This file will be used to perform simulations in various steps.

Once the functionality of the design is verified, the next step is the synthesis. In this step, the synthesis tool takes the source files and translates them into a set of hardware components that are available in FPGA hardware(such as registers, LUTs, flip-flops...).

The next four steps can be considered as the design implementation phase; they take the synthesised design and convert and place it into the available hardware resources.

- Translation, it takes the synthesised netlists along with physical and timing constraints and assembles them into a single file;
- Mapping phase : the available device resources and timing delays are checked, and reports an error if the design violates any rule;
- Place and route : places the different symbols generated previously and assigns them to available FPGA resources.

After both the synthesis and implementation, a simulation needs to be performed (called post-synthesis and post-implementation simulation); these simulations take into account the timing delays generated by the different hardware components and operations.

Finally, once the design meets the desired requirements, the generated bitstream can be uploaded to the target device and tested on hardware.

3.3 SSA_MPPT Implementation : HLS mehtodology

As discussed previously, the first step will be implementing the algorithm using HLS to generate an IP bloc that will be integrated into our RTL design. We first implement our algorithm in a C program, in a similar way to our Matlab implementation, and we take it as a starting point, where we'll adapt it and optimise it for use in HLS. Then we will go through the HLS design flow and finally generate the desired IP block.

3.3.1 Generation of the IP SSA_MPPT controller

For the IP generation, we targeted the Basys 3 prototyping board with the "xc7a35tcpg236-1" FPGA circuit.

3.3.1.1 SSA C code Implementation

To have a starting C code that can later be adapted for hardware implementation, we write it and test it in a pure software environment.
The C implementation is very similar to the Matlab implementation, and similarly, it is divided into two main parts, the SSA function and a testbench.

SSA function C implementation :
The algorithm is implemented in the same way as the Matlab one and uses the same logic, the only difference being the random generation function. Unlike in Matlab, the "rand()" function in C doesn't generate a number between 0 and 1 but instead returns an integer between 0 and RAND_MAX. A function is implemented that would return a number between 0 and 1 and is based on the rand() function.

```
float r2()
{
    float a = (float)rand() / (float)RAND_MAX ;
    return a ;
}
```

Figure 3.5 : Random generator in C

Testbench C implementations
To be able to test and evaluate the performance of the SSA function, just like in Matlab, a testbench needs to be implemented. It is set up in the same way as the Matlab one; the Power_math function has also been translated.

In addition, to be able to use the panel characteristics, a Python script has been used to convert the panel characteristics stored in a ".mat" Matlab file m into arrays that are stored in a C header file as variables, which will be imported into the testbench C file to be used.

Preliminary C Simulation

Before proceeding, we need to check the basic functionality of our initial C implementation. To do so, we run our simulation for one of the characteristics under partial shading we used in our Matlab simulation (Results shown in figure 2.12) and check if we get the same result. Results are shown in figure 3.6.

```
mohamed@mohamed-HP-EliteBook-820-G3:~/SSA_MATLAB_MPPT/C$ ./out
power= 127.544075
power= 127.449310
power= 127.541214
power= 127.117165
power= 127.544846
power= 127.543854
power= 127.544899
power= 127.543854
power= 127.540421
power= 127.544968
```

Figure 3.6 : C testbench output

As we can see, the final power to which the model converges is the same as the one found using the Matlab simulation, which is 127W, and is very close to the actual maximum for the panel.

3.3.1.2 C code optimisations

As discussed previously, HLS presents some constraints; for example, not all C functions are synthesisable. In our previously coded program, only simple C functions were used, so there is no need to change it.

Even if the code we have is synthesisable, it is not optimal for hardware implementation, notably for resource usage. We'll need to use arbitrary precision data types and adapt our equations accordingly.

In addition, as we mentioned previously, the random number generator will be implemented on RTL. Knowing this, we need to adapt our SSA function to take a random number as input instead of calling an internal C function.

a-Arbitrary precision data types

Below, we discuss the choice of precision and data types for the different variables.

- **Inputs : Current and Voltage**

Starting from the function's main inputs, the Voltage and Current (V and I, respectively), both of which will be read through the integrated Digital to Analog converter (ADC). Given that the ADC has a precision of 12 bits [47], we limit the precision of these two values to 12 bits.

The maximum voltage for our photovoltaic system is around 100V (5 panels in series) ; we reserve 7 bits for the integer part, which would give us a maximum voltage reading of 128 volts, which is plenty enough for our application. This would leave us with 4 bits for the fractional part, giving us a precision (quantification step) of 0.0625V, which translates into an error of +/- 0.03125V. So the final data type used would be a fixed point with 7 bits in the integer part.

For the current, and to simplify the implementation of operations, we use the same type, which will give us the same range for the current, which is also enough for our application.

The final used type would be an unsigned fixed-point number of 12 bits, with 7 bits for the integer part ; noted "ap_ufixed<12,7>".

- **Function output : Duty cycle**

The used FPGA card (Basys 3) has an internal clock speed of 100MHz. For a generated PWM control signal of a frequency of 40KhZ, the maximum number of steps will be 2500. If we use 11 bits for the duty cycle, we get a maximum of 2048 steps, which will allow us to not lose much in terms of precision while still not wasting another bit, which will use more resources while not giving many benefits in terms of precision. We reserve another bit for the integer part (to have the value 1).

The final used type would be an unsigned fixed-point number of 12 bits, with 1 bit for the integer part ; note "ap_ufixed<12,1>".

- **Function input : Random number**

To simplify the implementation of operations, we use the same data type and precision for the random number as the output duty cycle.

The final SSA function prototype is given below :

```
ap_ufixed<12,1> SSA(ap_ufixed<12,7> V, ap_ufixed<12,7> I,ap_ufixed<12,1> r2) ;
```

- **Function internal variables**

- Power :
  The used solar panels have a maximum power output of 1000W, so for the integer part, 10 bits would give us a maximum value of 1024W, which would be enough. In addition, 4 bits for the fractional part gives us enough precision.
  The final used data type would be a 16-bit unsigned fixed point, with 12 bits for the

integer part.

- Other internal variables (for example, the state variable, loop indexes) are all unsigned integers, and for which the number of bits is determined according to their maximum values (log_ 2(Max value)).

Testbench
After editing the data types, the testbench needs to be adapted accordingly; the same goes for the header files.

Now that the source C++ code is ready and optimised for hardware implementation, we can start the generation of the IP block for our function.

3.3.1.3 Creating a Vitis HLS project :

We start by launching Vitis_ HLS and creating a new project. After choosing a project name and destination folder, we need to add our source file.

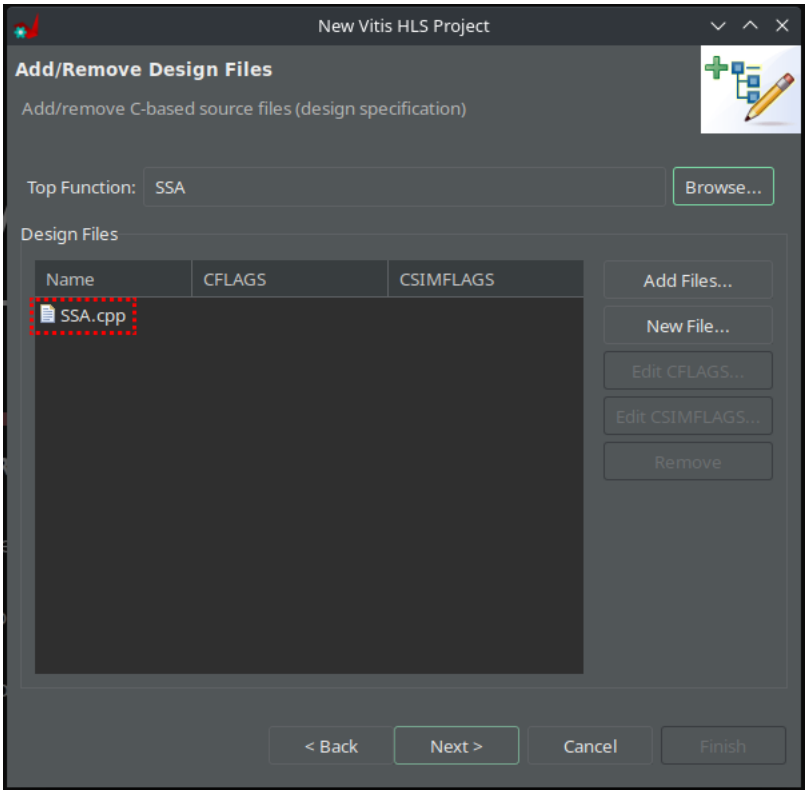


Figure 3.7 : Project creation : Adding source files.

After choosing our source file, we choose SSA as the top function from the list of available functions we can see when we click “browse” at the top of the window.
After clicking next, we get a similar window where we add our testbench files, which contains a “main” function that calls the SSA function, along with the corresponding header files and the

file defining the "Power_ math" function.
Finally, we get a window to choose the target device; we choose the "Basys 3" board from the list of available boards (which was added to the Vitis software earlier).

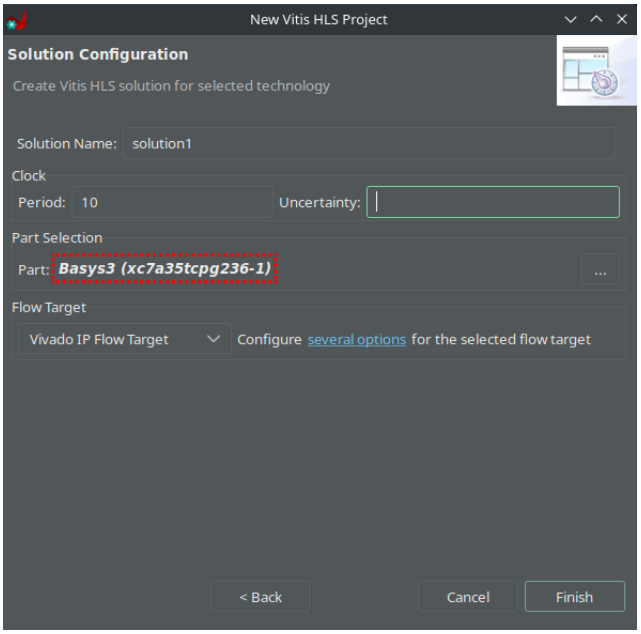


Figure 3.8 : Project creation : Target selection.

At this step, we also specify the target clock speed; in our case, the main clock has a frequency of 100MhZ, which translates to a period of 10ns.
Finally, we click finish to create our project.

3.3.1.4 C Simulation

Figure 3.9 shows the flow navigator, which summarises the different steps to be followed.

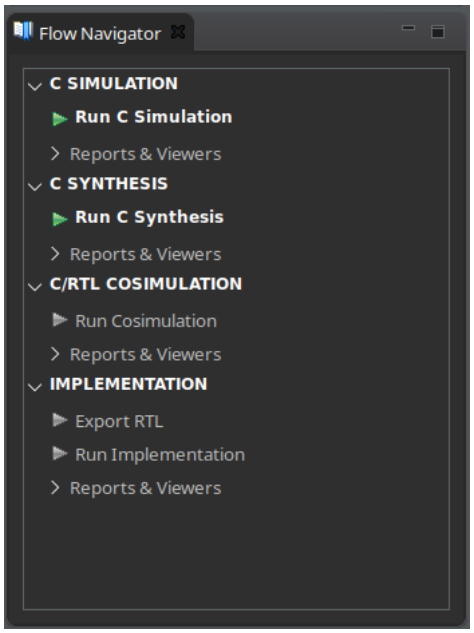


Figure 3.9 : Flow Navigator.

We start with the C simulation. Where the C source code is compiled and run, the output of the testbench function is checked to ensure the proper functionality of the code.
In this case, the main function runs the algorithm twice and prints the power after the system converges.
Figure 3.10 shows the output after running the simulation.

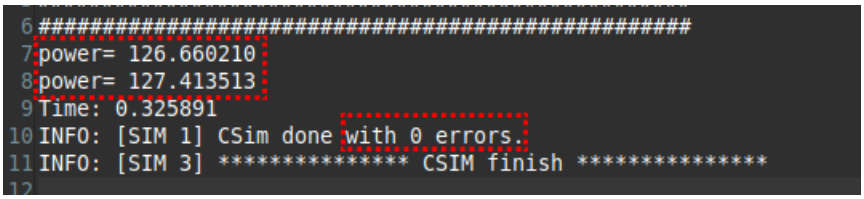


Figure 3.10 : C simulation output.

As depicted, the program compiles with no errors and the final power is indeed the maximum power of the tested panel (The same result found in Matlab shown in figure 2.12).

3.3.1.5 C synthesis

After successfully completing the simulation, we can move on to the next step, which is the C Synthesis.
This step takes the C code as input and converts it into HDL. We launch it from the flow navigator.
The synthesis completes with no errors, but we have some warnings :

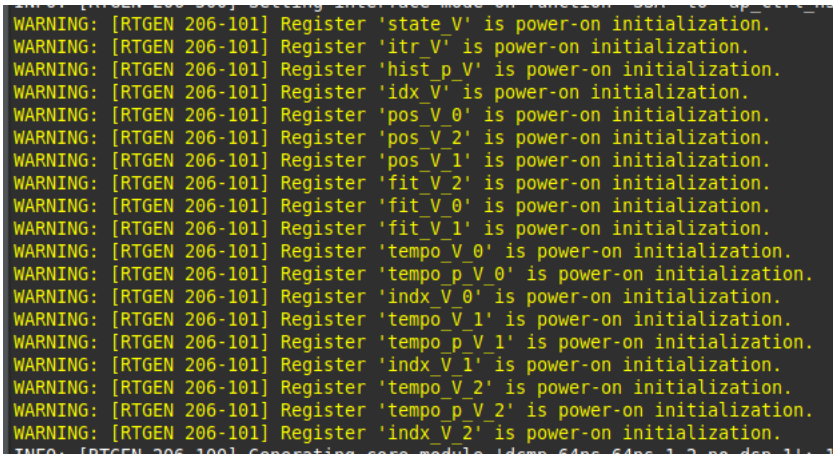


Figure 3.11 : C Synthesis Warnings

This warning concerns all persistent variables and means that these variables are only initialised at power-on and not after a reset.

To reset our algorithm, the only variable that needs to be initialised again is the “state” variable, and the algorithm will reset all other variables.

To solve this, we add a directive after the variable declaration, which would tell the synthesis tool to reset the “state” variable after the reset button is pressed.

The directive is “# pragma HLS reset variable=state”.

We run the synthesis again after changing the setting, and it completes with no warning for the “state” variable, which is the desired outcome.

At the end of the synthesis, a report is generated, where we can see various metrics about the generated RTL. Figure 3.12 depicts the timing estimation, which in this case, is within our design target (10ns).

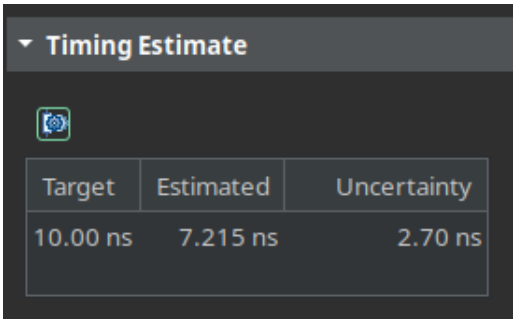
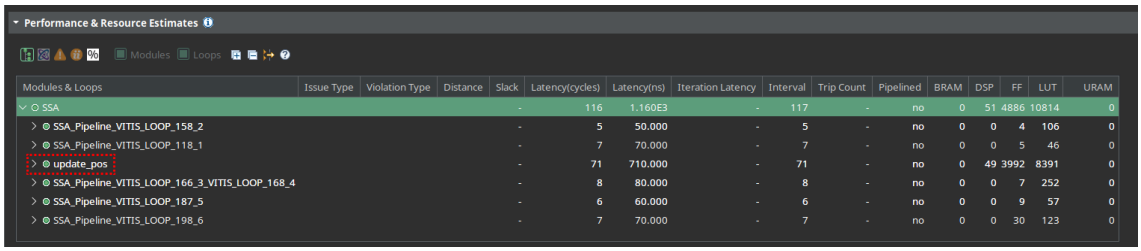


Figure 3.12 : C synthesis timing estimation

Another important metric is resource usage, and performance estimations, which are summarised in Figure 3.13



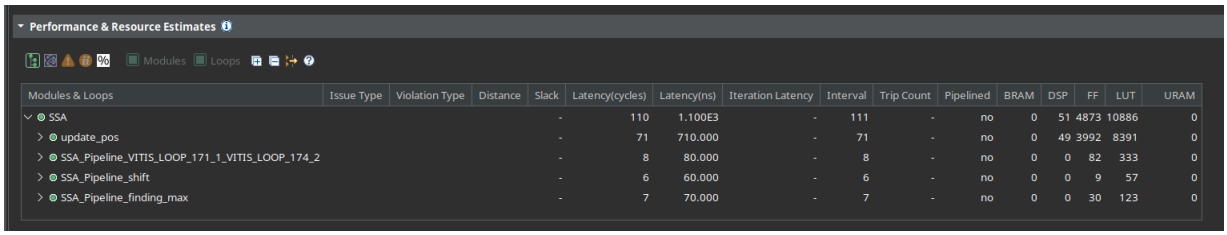
Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
SSA				-	116	1.160E3	-	117	-	no	0	51	4886	10814	0
SSA_Pipeline_VITIS_LOOP_158_2				-	5	50.000	-	5	-	no	0	0	4	106	0
SSA_Pipeline_VITIS_LOOP_118_1				-	7	70.000	-	7	-	no	0	0	5	46	0
update_pos				-	71	710.000	-	71	-	no	0	49	3992	8391	0
SSA_Pipeline_VITIS_LOOP_166_3_VITIS_LOOP_168_4				-	8	80.000	-	8	-	no	0	0	7	252	0
SSA_Pipeline_VITIS_LOOP_187_5				-	6	60.000	-	6	-	no	0	0	9	57	0
SSA_Pipeline_VITIS_LOOP_198_6				-	7	70.000	-	7	-	no	0	0	30	123	0

Figure 3.13 : C synthesis Usage estimation.

We can see the resources used by different parts and subfunctions of the design. As depicted, the "update_pos" function, which updates the position of the salps and is the function that contains all the used mathematical formulas, consumed the most resources in our design.
We can also see the performance estimation for loops present in the code, which have been pipelined, meaning that iterations are run one after the other in a pipeline. Directives can be used to flatten or unroll them, meaning iterations would be run in parallel, which would result in faster run time (fewer clock cycles to complete computation) but will also increase resource usage. This can be done in the case of independent loops (each iteration is independent of the others), which in our case, the loops are the reset loops. We unroll them by adding the directive "# pragma HLS UNROLL" inside the loop. After running the synthesis again, we review the resource usage shown in Figure 3.14.



Performance & Resource Estimates

Modules Loops

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
SSA				-	110	1.100E3	-	111	-	no	0	51	4873	10886	0
update_pos				-	71	710.000	-	71	-	no	0	49	3992	8391	0
SSA_Pipeline_VITIS_LOOP_171_1_VITIS_LOOP_174_2				-	8	80.000	-	8	-	no	0	0	82	333	0
SSA_Pipeline_shift				-	6	60.000	-	6	-	no	0	0	9	57	0
SSA_Pipeline_finding_max				-	7	70.000	-	7	-	no	0	0	30	123	0

Figure 3.14 : C synthesis Usage estimation.

The usage hasn't changed much, so we keep our loops unrolled. It didn't use up a lot of resources given that the unrolled loops are small (3 to 5 iterations), but in the case of long loops, the usage would increase significantly, in which case, the loop can be unrolled partially (run a given number of iteration in parallel instead of all of them).

In figure 3.15, we get a description of the different inputs and outputs of the generated IP, of which we have the C function inputs and outputs, along with automatically generated control signals, that will be used to control the block.

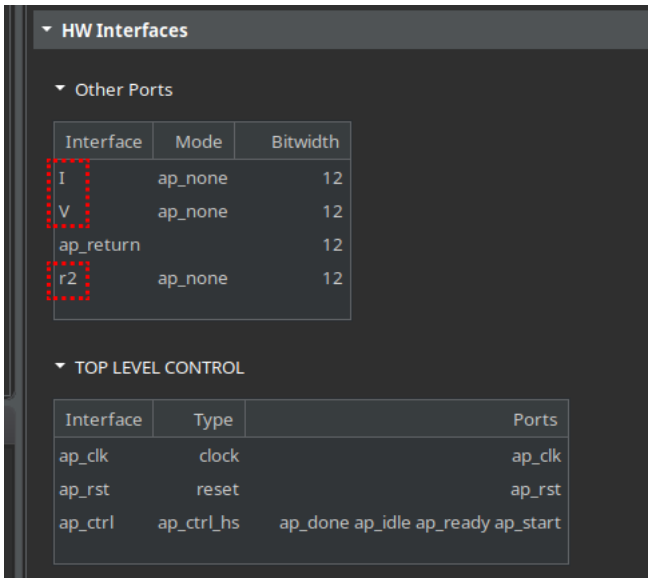


Figure 3.15 : IP Bloc Ports

The main inputs of the designed block are V, I and r2, which are, respectively, the voltage, current and a random number.

The output port is “ap_return”, which represents the return value of our C function and the main output of our block. The automatically generated control signals include the “ap_clk” and “ap_rst”, which represent the clock and reset input signals. “ap_ctrl” contains four signals :

- ” ap_done” : Output logic signal, high for one clock cycle when the output “ap_return” is ready.
- ” ap_idle” : Output logic signal, high when the bloc is idle.
- ” ap_ready” : Output logic signal indicates when the block is ready to take in new inputs.
- ” ap_start” : Input logic signal, an enable signal, the block runs when high, and is idle when low.

3.3.1.6 C/RTL cosimulation

Now that we are satisfied with our synthesis results, we can move on to the next step, which is the C/RTL CoSimulation. In this step, both the C and RTL implementations are simulated using the same C testbench, and the results are compared to check the good functionality of the generated HDL code. Figure 3.16 depicts the CoSimulation results.

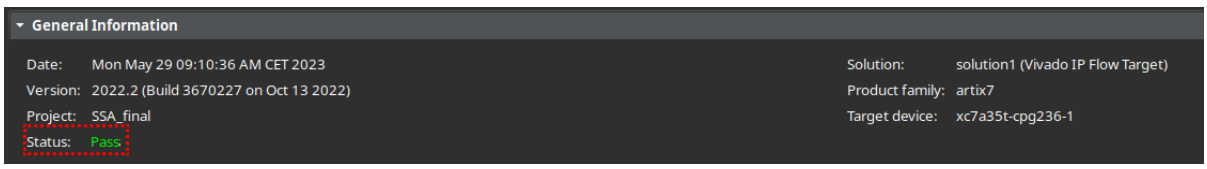


Figure 3.16 : Cosimulation result

At the end of this step, we get a report indicating that the implementation had passed the co-simulation, meaning that our generated HDL behaves as desired.

3.3.1.7 IP block generation

After successfully completing the co-simulation, we move on to the next step, which is exporting the design as an IP block. To do this, we click on "Export RTL" in the flow navigator. Figure 3.17 illustrates the generated SSA_MPPT IP block.

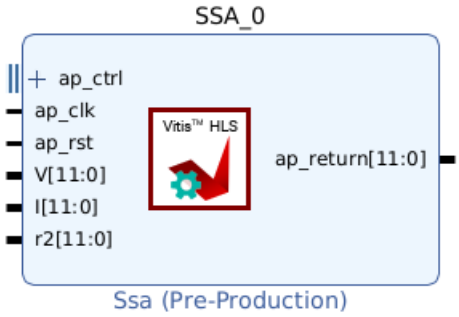


Figure 3.17 : The generated IP block

Along with the bloc, other files are also included in the package, including the generated HDL source files and the initialisation templates (used to include the IP in other RTL entities).

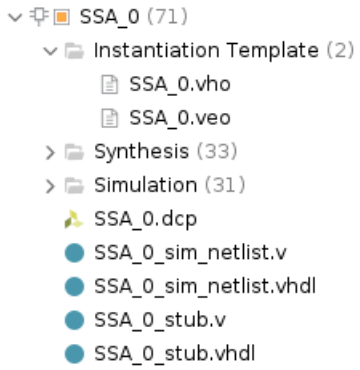


Figure 3.18 : IP Source files and initialisation templates

3.3.2 FPGA implementation of the IP SSA_MPPT

We will create a Vivado project where we'll integrate the generated IP_SSA_MPPT, and where we'll write HDL code to control its general behaviour and add functionalities that are not integrated into the IP, and follow the necessary steps to generate the final bitstream file. It is important to note that in this step, we target the same device selected previously during the IP generation phase, namely the "7a35tcpg236-1".

3.3.2.1 Creating project and integrating the generated IP

The first step in implementing the VIVADO design is creating a new empty RTL project, where we will need to specify the name and location of the project, as well as the target device, as

depicted in Figure 3.19.

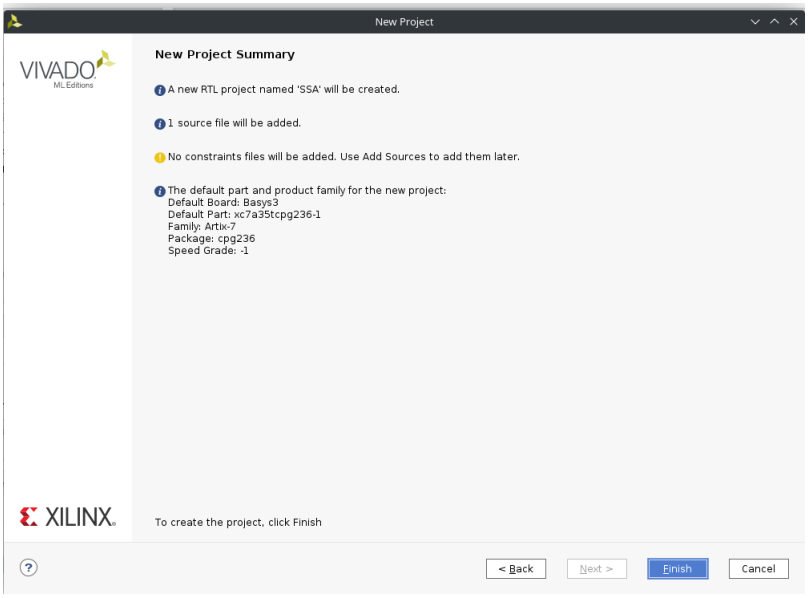


Figure 3.19 : New Vivado Project summary

After the project is created, we create a new VHDL source file, which will be our top-level design. Then, we add the generated IP to our project from the IP catalogue window and then add it to our top-level source file as a component.

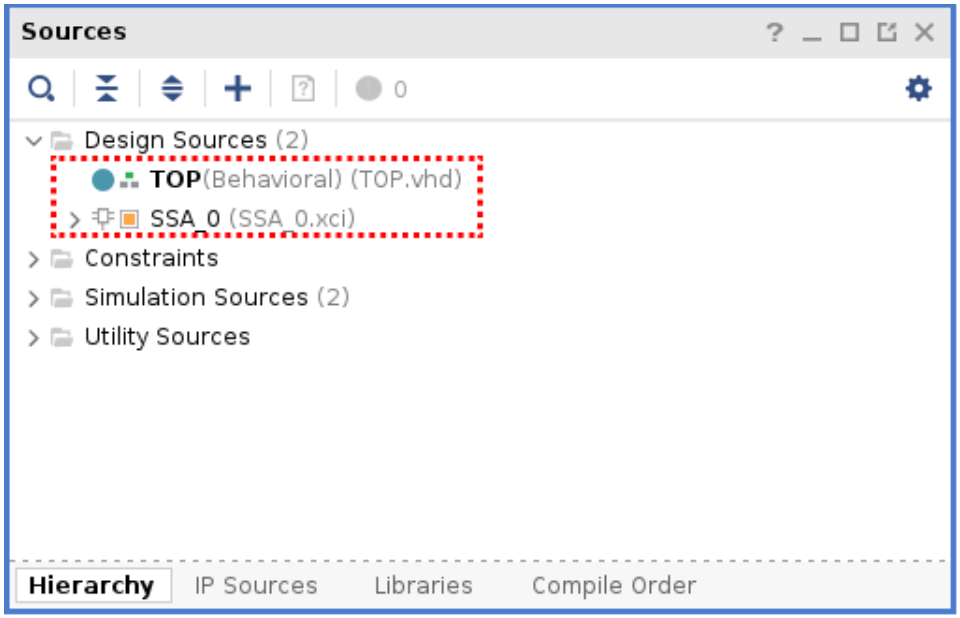


Figure 3.20 : Including the IP_SSA_MPPT in the Vivado project

-Random generator

As described previously, the random number needs to be generated and given as input to the

SSA bloc, so a random number generator needs to be implemented in RTL to provide them.
Our application is not a cryptographic problem where numbers need to be true random numbers; a pseudo-random number would suffice. So our goal is to find an easy-to-implement generator (or an already existing implementation).
Looking through the literature, we find one such random generator that is described in [48]. DAVID BLACKMAN and SEBASTIANO VIGNA based their generator on the Xorshift concept invented by George Marsaglia [49].
This generator has been implemented by the original authors in different software languages but not described in hardware.
Joris van Rantwijk implemented these generators in VHDL [50] and published them under an open-source licence.
This implementation is capable of outputting a number per clock cycle (after initialisation is done), which provides ample speed for our application.
It has been included in our project and modified to only output 12 bits.

-XADC
For the FPGA to be able to read the sensors' measurements, we use its integrated Analog to Digital Converter (XADC).

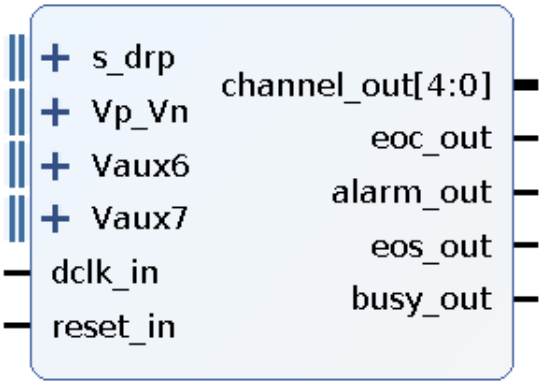


Figure 3.21 : XADC IP bloc

The XADC is provided as a reconfigurable IP core. It provides several input channels and different output protocols. The needed input channels are the ones wired to the XADC input of the Basys 3 board.
This IP bloc needs to be controlled, which we do through a finite state machine.
To reduce noise, for every value, we take the measure multiple times, and the average is outputted; this behaviour can be configured in the IP customisation window, where we can choose the number of samples to be averaged (16, 64 or 256), we choose 256.
The controller behaviour is summarised in the chart below :

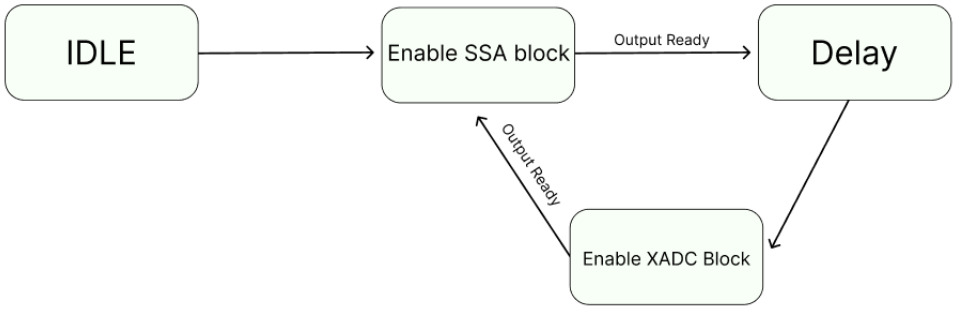


Figure 3.22 : XADC Controller flow chart

-PWM generator

To control the DC/DC converter, we need a PWM generator that takes as input the duty cycle and outputs a PWM signal with the same duty cycle. It also takes as generic inputs which specify the output PWM frequency and the number of bits for the duty cycle.
The module ports are given below :

```
ENTITY pwm IS
  GENERIC(
          sys_clk :INTEGER := 100_000_000 ;--default system clock frequency in Hz
          pwm_freq :INTEGER := 20_000 ;--PWM default switching frequency
          bits_resolution : INTEGER := 11--bits of resolution setting the duty cycle
         ) ;
  PORT(
          clk : IN  STD_LOGIC ;      --system clock
          reset_n : IN  STD_LOGIC ;     --asynchronous reset
          duty : IN  STD_LOGIC_VECTOR(bits_resolution-1 DOWNTO 0) ;--duty cycle
          pwm_out : OUT STD_LOGIC --pwm outputs
         ) ;
END pwm ;
```

Figure 3.23 : PWM Module ports

Based on the system and the desired output frequencies, the generator determines the number of output steps, in other words, the maximum number of distinct possible duty cycles.
The general behaviour of the algorithm can be summarised as follows :

The different iterations are performed at the system clock speed, which would result in the desired output frequency.

Control and top-level module

After seeing all the different parts of the design in the previous sections (SSA, Random generator, XADC), we now need to integrate them into a single top design, which will be our main design. We first include these different modules as components in the top module. These components

```
counter=0
half_duty= duty_cycle*number_of_steps
while (1)
     if (counter<half_duty)
         PWM='1'
         counter=counter+1
     else if counter<number_of_steps
          PWM='0'
         counter=counter+1
     else
         counter=0
```

Figure 3.24 : PWM generator behaviour

will be controlled through generated signals. The functionality of the controller is summarised in the chart below :

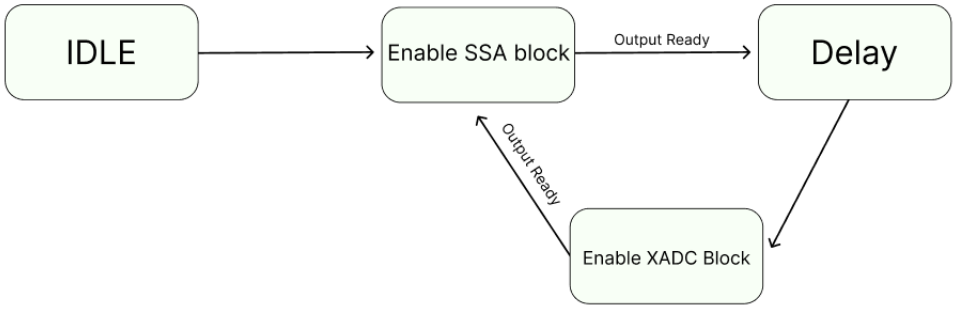


Figure 3.25 : Top module controller flow-chart

We can visualise the final source hierarchy of our design :

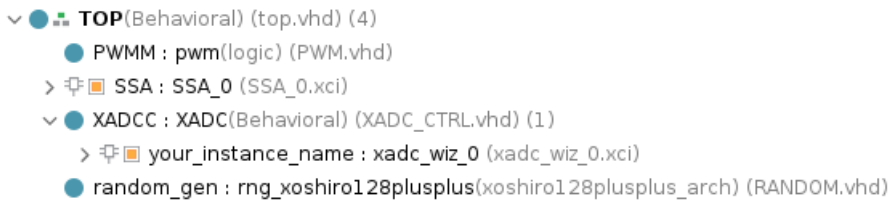


Figure 3.26 : Top module source hierarchy

The final ports of the top module are given below :

```
entity TOP is
        Port (
        clk : in std_logic ;
        rst : in std_logic ;
        vauxp6 : in std_logic ;
        vauxn6 : in std_logic ;
        vauxp7 : in std_logic ;
        vauxn7 : in std_logic ;
        vp_in : in std_logic ;
        vn_in : in std_logic ;
        OUTT : out STD_LOGIC_VECTOR ( 10 downto 0 ) ;
        pwm2 : out std_logic ;
        done : out std_logic
              ) ;
end TOP ;
```

Figure 3.27 : TOP Module ports

3.3.2.2 Simulation and Testing :

After finalising our design, we need to test its functionality and validate its behaviour and performance. We will use the simulator integrated into the Vivado design tool.

-Testbench :
To design our testbench, we follow the same logic as we did with the testbench in C language. The testbench would provide inputs to our design : in this case, the voltage and current, based on the current duty cycle.
This is done based on the previously used panel data and equations. We note that the panel data is converted to a usable format in VHDL (text file) using a Python script.

Behavioural Simulation
The first step is the behavioural simulation, where only the functionality of the design is evaluated. Figure 3.28 illustrates the simulation results.

The results of the simulation are shown below :

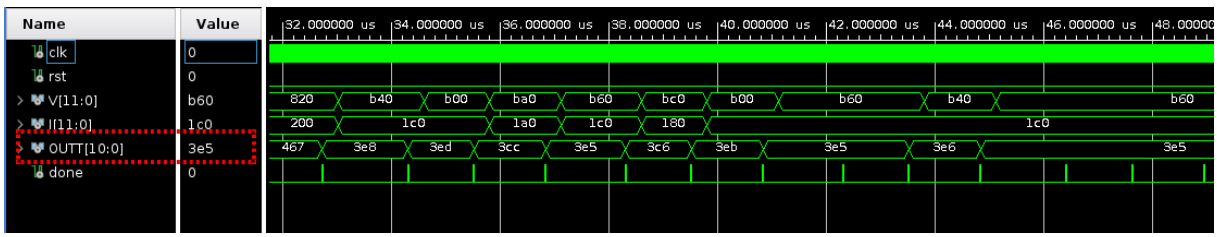


Figure 3.28 : Behavioural simulation results of the IP SSA_MPPT controller

To evaluate the good functionality, we compare the final result with our software simulation

results. We convert our hexadecimal duty cycle value to decimal by dividing the number by the maximum possible value with 11 bits. The duty cycle is represented by the signal "OUT", which converges to the hexadecimal value "3e5", which is 997; we divide this value by the maximum number we can represent in 11 bits, which would be 2047. The resulting number would be our output duty cycle; in this case, we find 0.48, or 48% .
By running our Matlab simulation for the same panel, we find the same final result, which is 48%, meaning our design behaves as expected.
Now we need to evaluate the behaviour of the system in other conditions; first of all, we check the reset behaviour : if the algorithm restarts properly after a reset.
To do this, we force our reset button to a high value and then back to a low value without changing the testbench and observe the behaviour.

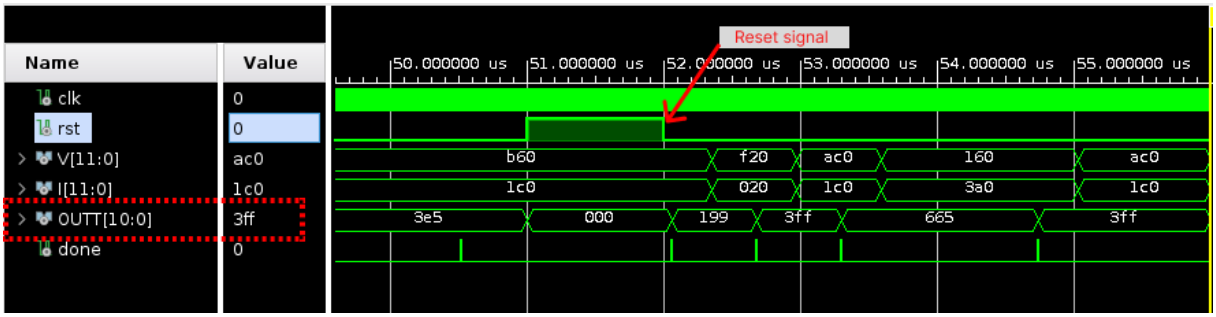


Figure 3.29 : Reset Behaviour simulation

As we can see, after the reset, the algorithm is properly restarted. We can see this by looking at the output values, which, in the first iteration, are uniformly distributed in the search space.

Another behaviour that needs to be checked is the change in weather conditions. To do this, after the program converges, we force one of the input values (current or voltage) to a different value and see if the algorithm restarts.

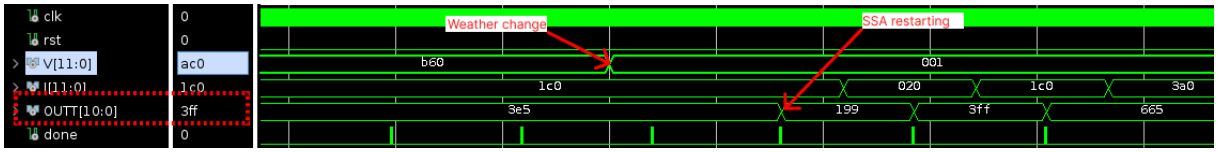


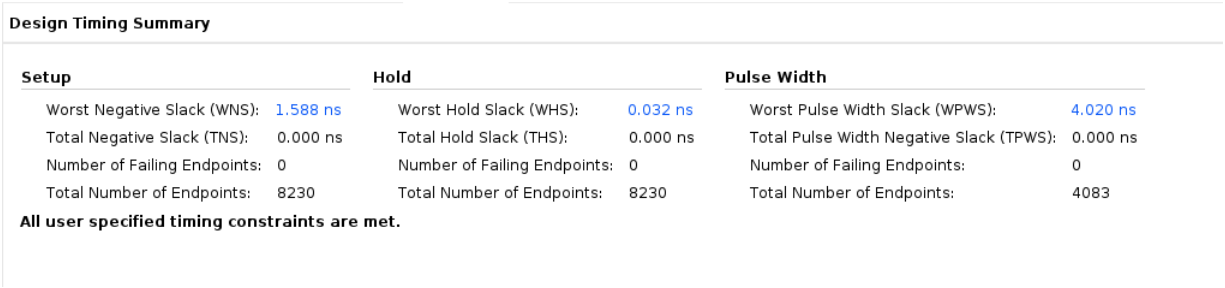
Figure 3.30 : Weather Change Behaviour simulation

As we can see if Figure 3.30, after the change in the power input (in this case, we forced the voltage to a different value), which means a change in weather conditions, the algorithm restarts to find the new maximum power point, which is the expected behaviour.

3.3.2.3 Synthesis & Implementation

Once we confirm the right behaviour of our design, the next steps in the design flow are synthesis and implementation, but first, we need to define our constraints.

To do so, we start by downloading the master constraints file for our board, which contains the definitions of all ports, and we assign our top-level design ports to their respective pins.
We start by running the synthesis from the flow navigator window in Vivado, after which we run the implementation.
After they are both completed successfully, we can evaluate the generated reports; we mainly focus on two metrics : usage and timing. Figures figs. 3.31 to 3.33 illustrate a summary of resources used and the Design timing.

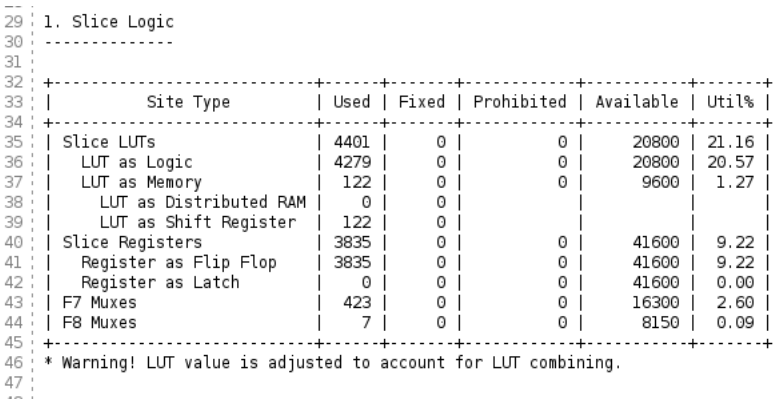


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.588 ns	Worst Hold Slack (WHS):	0.032 ns	Worst Pulse Width Slack (WPWS):	4.020 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	8230	Total Number of Endpoints:	8230	Total Number of Endpoints:	4083

All user specified timing constraints are met.

Figure 3.31 : Design Timing summary

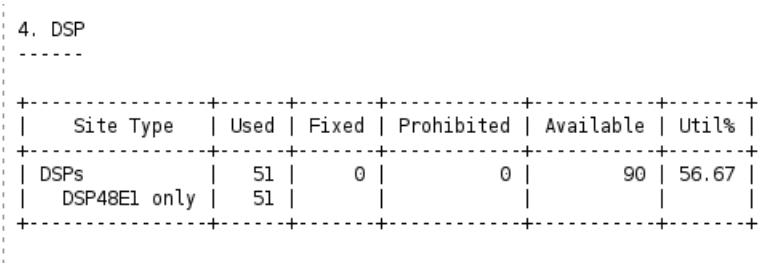


1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	4401	0	0	20800	21.16
LUT as Logic	4279	0	0	20800	20.57
LUT as Memory	122	0	0	9600	1.27
LUT as Distributed RAM	0	0			
LUT as Shift Register	122	0			
Slice Registers	3835	0	0	41600	9.22
Register as Flip Flop	3835	0	0	41600	9.22
Register as Latch	0	0	0	41600	0.00
F7 Muxes	423	0	0	16300	2.60
F8 Muxes	7	0	0	8150	0.09

* Warning! LUT value is adjusted to account for LUT combining.

Figure 3.32 : Design Logic Usage summary



4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	51	0	0	90	56.67
DSP48E1 only	51				

Figure 3.33 : Design DSP Usage summary

Our design falls within the physical limits of the selected prototyping board, not exceeding resource usage, and the timing performance is under our timing constraints to function with a 100Mhz clock.

Looking at a more detailed utilisation report shown in figure 3.34, we can see which part of the design uses the most resources.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)
N TOP	4401	3835	423	7	1580
PWMM (pwm)	50	40	0	0	25
random_gen (rng_xoshiro128plusplus)	133	158	0	0	38
SSA (SSA_0)	4174	3542	423	7	1494
U0 (SSA_0_SSA)	4174	3542	423	7	1494
dcmp_64ns_64ns_1_2_no_dsp_1_U62 (SSA_0_SSA_dcmp_64ns_64ns_1_2_no_dsp_1)	43	58	0	0	21
grp_SSA_Pipeline_finding_max_fu_596 (SSA_0_SSA_SSA_Pipeline_finding_max)	53	30	0	0	22
grp_SSA_Pipeline_shift_fu_590 (SSA_0_SSA_SSA_Pipeline_shift)	16	9	0	0	8
grp_SSA_Pipeline_VITIS_LOOP_171_1_VITIS_LOOP_174_2_fu_564 (SSA_0_SSA_SSA_Pipeline_VITIS_LOOP_171_1_VITIS_LOOP_174_2)	246	81	0	0	95
grp_update_pos_fu_540 (SSA_0_SSA_update_pos)	3514	2758	423	7	1275
dcmp_64ns_64ns_1_2_no_dsp_1_U11 (SSA_0_SSA_dcmp_64ns_64ns_1_2_no_dsp_1_0)	137	65	0	0	45
dexp_64ns_64ns_64_21_full_dsp_1_U13 (SSA_0_SSA_dexp_64ns_64ns_64_21_full_dsp_1)	1767	957	423	7	599
SSA_dexp_64ns_64ns_64_21_full_dsp_1_ip_u (SSA_0_SSA_dexp_64ns_64ns_64_21_full_dsp_1_ip)	1767	914	423	7	592
inst (SSA_0_floating_point_v7_1_15__parameterized5)	1767	914	423	7	592
i_synth (SSA_0_floating_point_v7_1_15_viv__parameterized11)	1767	914	423	7	592
EXP_OP.i_sp_or_dp.OP (SSA_0_flt_exp)	1767	914	423	7	592

Figure 3.34 : Design Detailed Usage

Most resources are used by the “update_pos” function and, more precisely, the calculation of the exponential.
We also get a detailed report regarding the power consumption estimation, shown in figure 3.35.

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.14 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.7°C**
Thermal Margin:	59.3°C (11.8 W)
Effective θJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

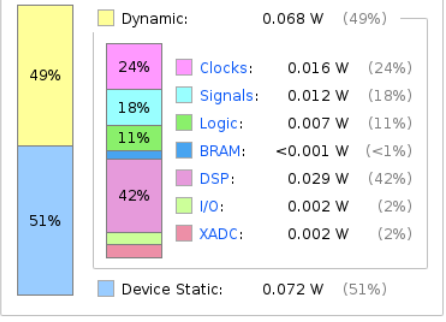


Figure 3.35 : Power usage estimation

As we can see, the power consumption is very low and way under our board limit (5W). This is mainly due to the fact that our design stays idle for most of the time in the delay between iterations. The actual calculations happen in a couple of clock cycles, while the delay takes a million clock cycles.

Figure 3.36 illustrates the implementation and occupation of the proposed MPPT controller on the selected FPGA circuit.

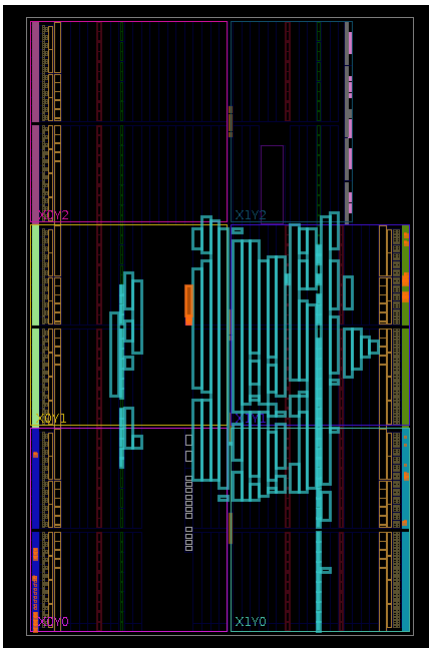


Figure 3.36 : Device implementation visualisation

Being satisfied with our results, we can generate the bitstream that will be loaded to our board.

Conclusion

In this chapter, we review the process of implementing the Salp Swarm Algorithm on an FPGA, using a workflow combining both Vitis HLS and Vivado design suits.
We simulated our design, and verified the proper functionality, then completed the different steps of the FPGA design flow. Finally, we evaluated the performance.
After completing these steps, being satisfied with our results, and for testing the functionality of the proposed controller in a real environment, we can generate the bitstream which will be loaded to our board.
In the next chapter, we'll go over the hardware testing in a real environment.

Chapter 4

Practical implementation and testing

Introduction

After implementing our Maximum Power Point Tracking algorithm, the Salp Swarm Algorithm, in both software (MATLAB) and hardware (FPGA) and validating its functionality using software environments and theoretical panel data. Now we need to test and validate the design using real-world data.

In this section, we will first extract solar panel data under different conditions while presenting the setup used.

The recovered data will then be used to test both the software and hardware implementations.

Finally, we will test the whole FPGA-implemented design in real-time, connecting it directly to the photovoltaic panels.

4.1 Presentation of the test setup

For this test, we need to gather all the elements constituting the whole architecture of the photovoltaic system under test.

4.1.1 DC/DC Converter

As explained in section 1.1.3, to be able to function at the Maximum Power Point, we need a variable impedance at the output of the panels.

Similarly, to extract panel characteristics, we need a way to vary the impedance at the output of our panels.

There are multiple ways of doing this, each presenting upsides and downsides. We will present in table 4.1 three possible approaches.

Method	Upsides	Downsides
Manual load variation	•Cheap and easy to implement	•Slow and laborious testing •Imprecise results and limited number of points •Requires high power variable load or various constant loads
Electronic variable load	•Fast,easy and precise testing	•Requires high power MOSFETs
Impedance adapter	•Fast,easy and precise testing	•Requires a high power load

Table 4.1 : Comparison of different panel testing methods

From these three methods, both the Electronic variable Load and Impedance adapter are good options; they present fast and precise testing. The choice between the two will be made depending on available hardware.
Having high-power resistors at hand, we chose to use an impedance adapter (a DC/DC converter) for our testing. To get the maximum range of testing, we decided to use a BUCK/BOOST converter.

4.1.1.1 BUCK/BOOST converter

Buckboost is a type of switching power converter. They can step up and down a voltage. The schematic for such a converter is given in figure 4.1

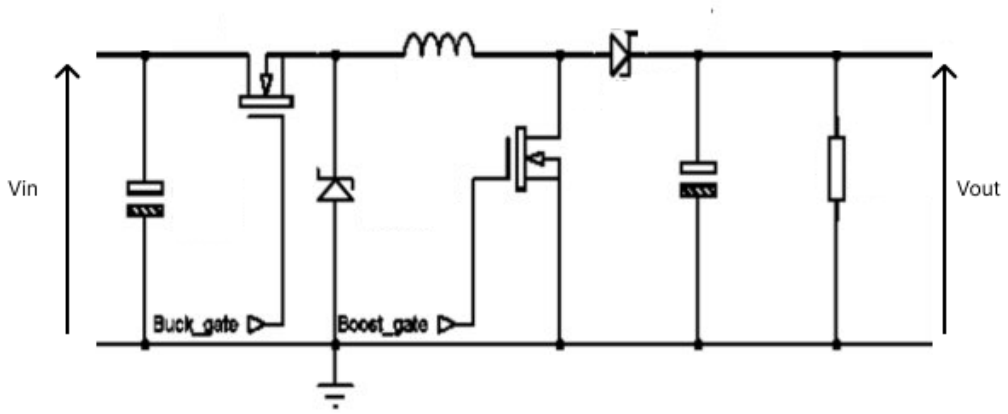


Figure 4.1 : Buck-Boost Converter Schematic

This converter can function in one of three modes depending on the control signals :

- Buck-Boost mode : Buck gate and Boost gate controls signals in sync.
- Buck mode : Boost gate always OFF, the control signal of the Buck gate.
- Boost mode : Buck gate always ON, the control signal of the Boost gate.

The converter is more efficient and presents fewer losses working in the Buck mode or Boost mode compared to the Buck-Boost mode.
Thus, for controlling the converter, we use the control signals detailed in table 4.2.

Buck-Boost Duty-Cycle (Dc)	Buck gate Duty-Cycle	Boost gate Duty-Cycle
Dc<50%	Dc.2	0%
Dc≥50%	100%	(Dc.2)-100%

Table 4.2 : Buck Boost converter control signals

4.1.2 Selected photovoltaic panels

The solar photovoltaic panel used consists of an array of 12 modules. The modules are from YINGLI SOLAR, model YL80.

The modules are connected in a series/parallel configuration : 2 panels in series, which will be connected in parallel with the other pairs (4.3).

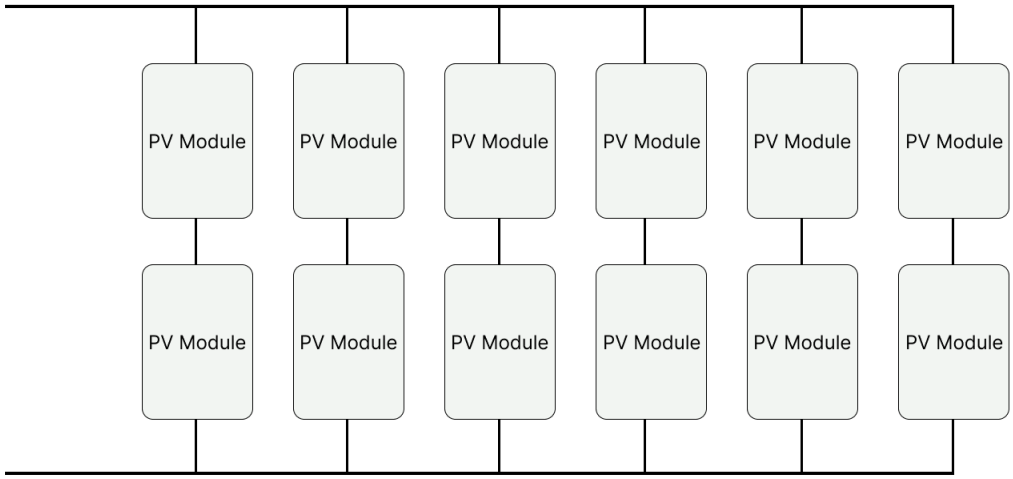


Figure 4.2 : Solar Panel Configuration

The modules have a max power output of 80W each; the whole array would give a maximum power output of 960W.
The modules being connected in pairs of 2 in series, would have a maximum output voltage of 44V.
Figure 4.3 shows the solar panels used.

Figure 4.3 : Chosen solar photovoltaic array

4.1.3 Sensors

To be able to collect data (Current and Voltage), we need to use some sensors.

4.1.3.1 Current sensor

The current sensor used is the LEM LA100-P (figure 4.4). It is based on the Hall effect. With a maximum current sensing capacity of 100A, it is enough for our application (the maximum current of the array is 30A).

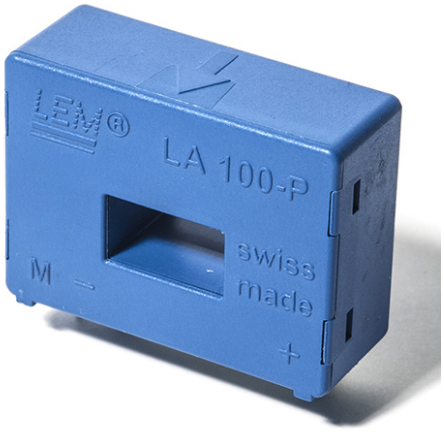


Figure 4.4 : LA100P Current sensor

The sensor has a factor of reduction of 2000 [51], meaning that for a sensed current of 1A, we'll have 0.5mA at the output.
To measure the output current, we use a shunt resistor of a value of 100 Ω. Figure 4.5 shows the final circuit used.

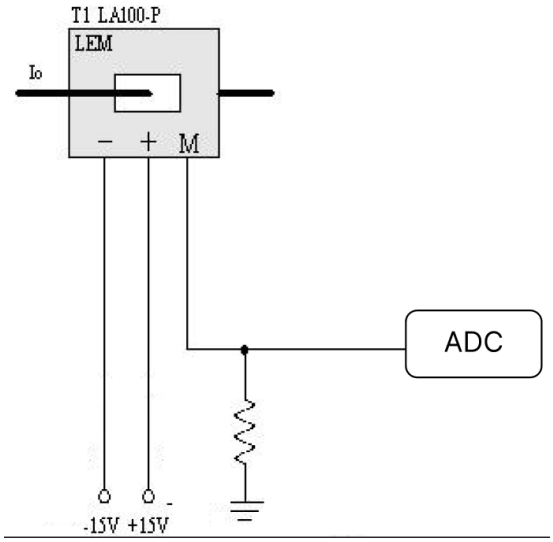


Figure 4.5 : Current sensor circuit

4.1.3.2 Voltage sensor

For voltage sensing, we use a simple voltage divider. Taking into account the maximum input voltage of our boards' ADCs (Microcontroller and FPGA), and the maximum output voltage of the panels, we determine the Resistors to be used. Figure 4.6 shows the circuit and values of the resistors used.

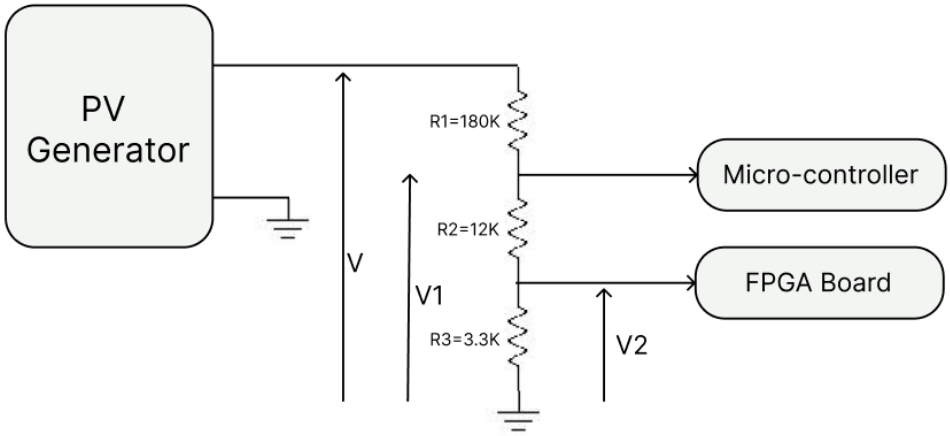


Figure 4.6 : Voltage sensor : Voltage divider

Using Ohm's law and Kirchhoff's Voltage Law, we can calculate $V1$ and $V2$ with respect to V ; the formulas used are given respectively in 4.1 4.2.

$$V_1 = \frac{R_2 + R_3}{R_2 + R_3 + R_4}.V = 0.078 * V \tag{4.1}$$

$$V_1 = \frac{R_3}{R_2 + R_3 + R_4}.V = 0.017 * V \tag{4.2}$$

The maximum voltage supported by the microcontroller is 3.3V, and the FPGA board is 1V. This is why we used three resistors to get two voltages suited for both boards.
For the maximum 44V output voltage, we'll get $V_1 = 3.12V$ and $V_2 = 0.74V$, which is under the maximum supported voltages.

4.2 Extracting panel characteristics under different shading conditions

The first practical test we will conduct is extracting panel characteristics.

4.2.1 Presenting the setup for collecting panel characteristics

To extract the panel characteristics, we use a microcontroller ; in this case, we used the STM32 BluePill 4.7.

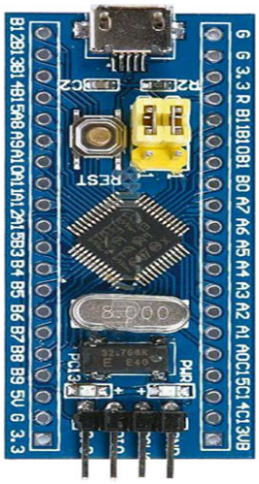


Figure 4.7 : STM32 BluePill

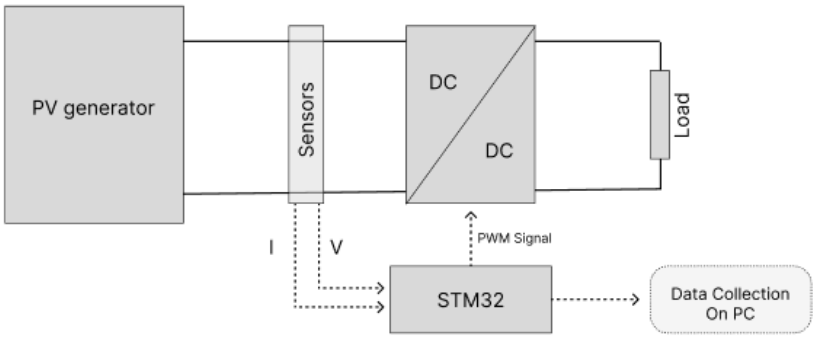
The choice was mainly based on the availability of a precise ADC converter (12 bits) and a high clock frequency (72 MHz) that would allow us to generate a high-frequency PWM control signal while not losing on precision.

We write a C program that loops through all possible PWM values for each one, measures both current and voltage, and sends them through a serial interface.
We set the PWM divider to 1800, meaning that, for the given clock frequency of 72MHz, the output PWM frequency would be 20MHz, and we would have 1800 steps (or measurement points).
The algorithm can be summarized in the pseudo-code shown in figure 4.8.

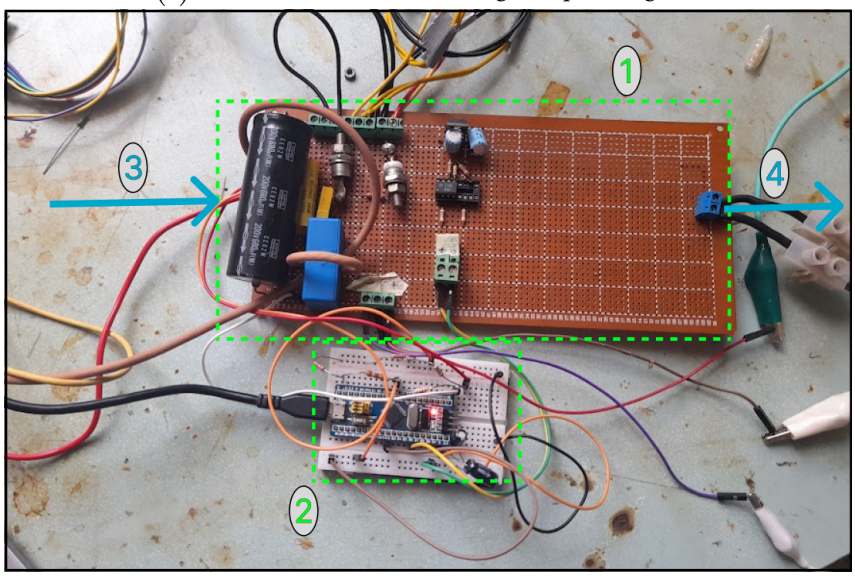
```
Duty_Cycle=0 ;
while Duty-Cycle<1800
        Set_PWM(Duty-Cycle) ;
        Duty_Cycle=Duty_Cycle+1 ;
        Delay ;
        Read(Voltage, Current) ;
        Serial_Print(Voltage, Current, Duty_Cycle) ;
end while
```

Figure 4.8 : Single objective SSA pseudo-code

The diagram of the test setup as a whole is shown in figure 4.9a, and the real test setup is in figure 4.9b.



(a) Panel characteristic testing setup : Diagram

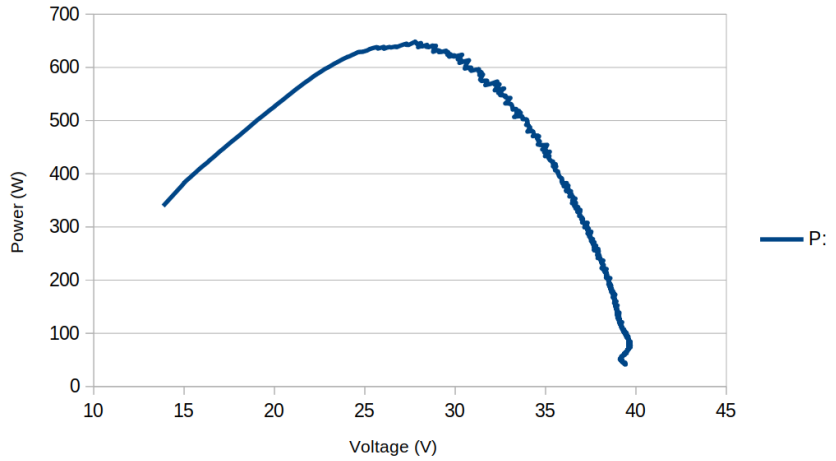


(b) Panel characteristic testing setup : Real image

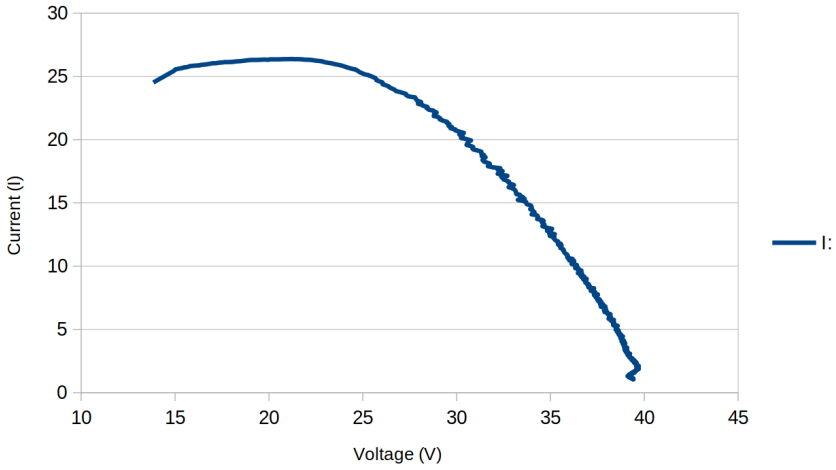
Figure 4.9 : Panel characteristic testing setup

4.2.2 Panel characteristics

From the microcontroller, we send raw data, meaning direct ADC values, which will be put in a spreadsheet in which we would calculate the real *voltage*, *Current* and *Power* Values, and then plotted.
To start, we test our panels in a sunny environment with no partial shading. The results are presented in figure 4.10.



(a) P-V Characterestic



(b) I-V Characteristic

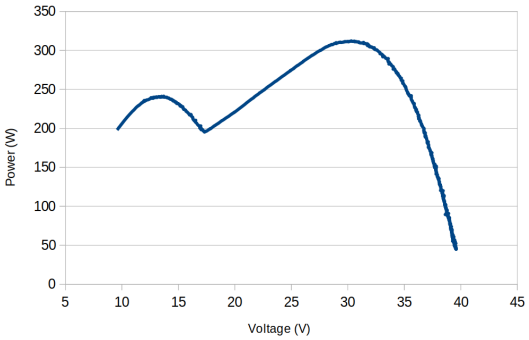
Figure 4.10 : Panel characteristics : No shading

Both the V-I and P-V characteristics of the panel are very similar to the theoretical ones, where we have a maximum power point, and the characteristic is linear below that point.
We can also notice a slight noise in the curve ; it is due to both the measurement and the converter circuit, where small noises are introduced to the system.
The maximum output power is 638W, which is below the maximum theoretical value of the panels, which is 960W.

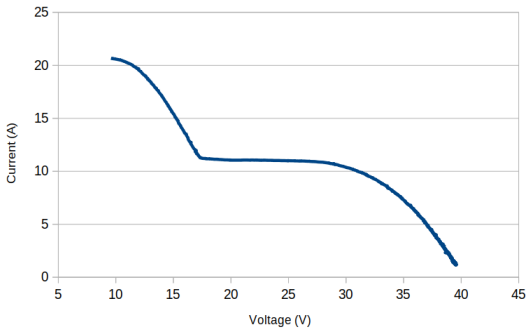
This is due to several reasons :

- Panels not being fully clean ;
- The ageing of the panels : reduces their efficiency ;
- Losses in the cables ;
- Not optimal lighting and temperature conditions.

Next, we proceed to test the panels under various partial-shading conditions ; we chose three different conditions we demonstrate in figures figs. 4.11 to 4.13.

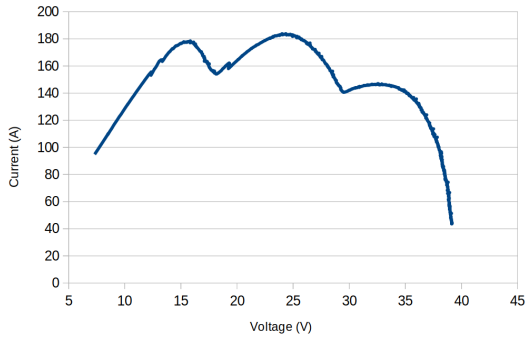


(a) P-V Characterestic

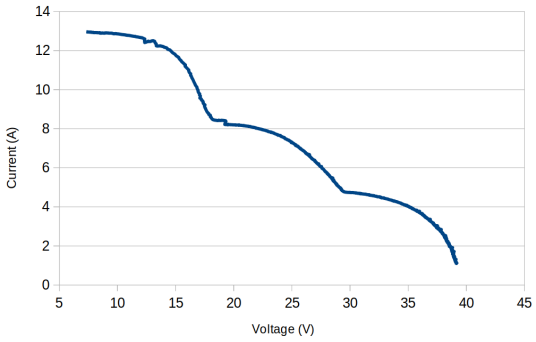


(b) I-V Characteristic

Figure 4.11 : Panel characteristics : Partial shading 1

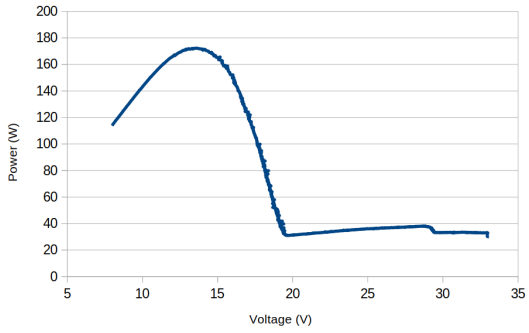


(a) P-V Characterestic

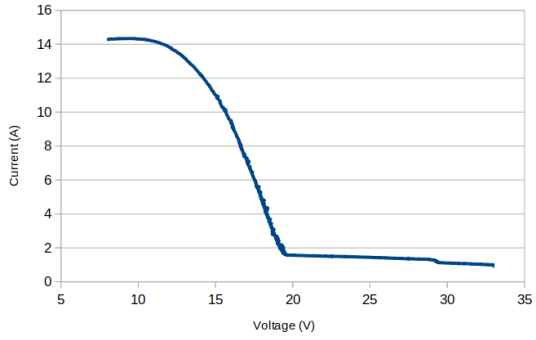


(b) I-V Characteristic

Figure 4.12 : Panel characteristics : Partial shading 2



(a) P-V Characterestic



(b) I-V Characteristic

Figure 4.13 : Panel characteristics : Partial shading 3

Just like the theoretical characteristics, under partial shading, the P-V characteristics show local maximas.

4.2.3 Preliminary testing of the design (off-grid testing)

Now that we have our practical panel data, we will use them to test our implemented Salp Swarm Algorithm, both in the Matlab environment and in Vivado.

4.2.3.1 Test under Matlab

We edit our testbench so that for each given duty cycle, we look through our recorded data and return the corresponding Voltage and Current values. (Instead of using the panel data and the equation of a converter).
First, we try the data with no shading; the results are shown in figure 4.14.

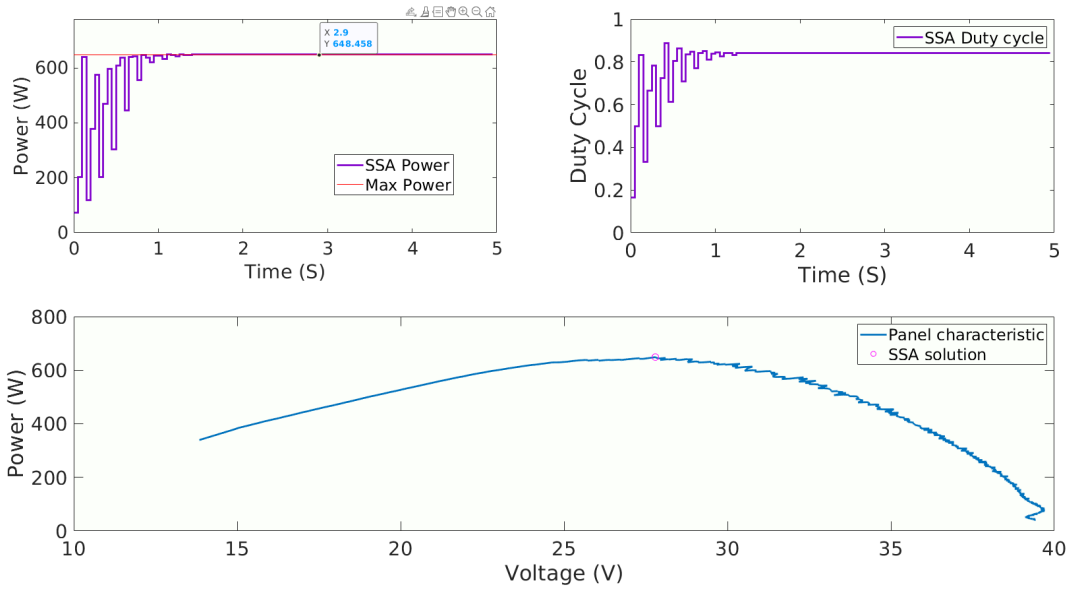
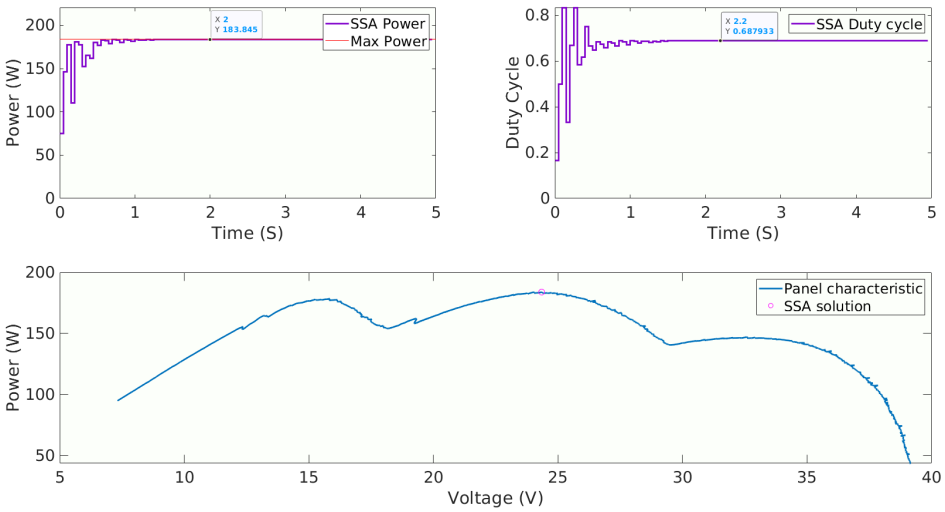
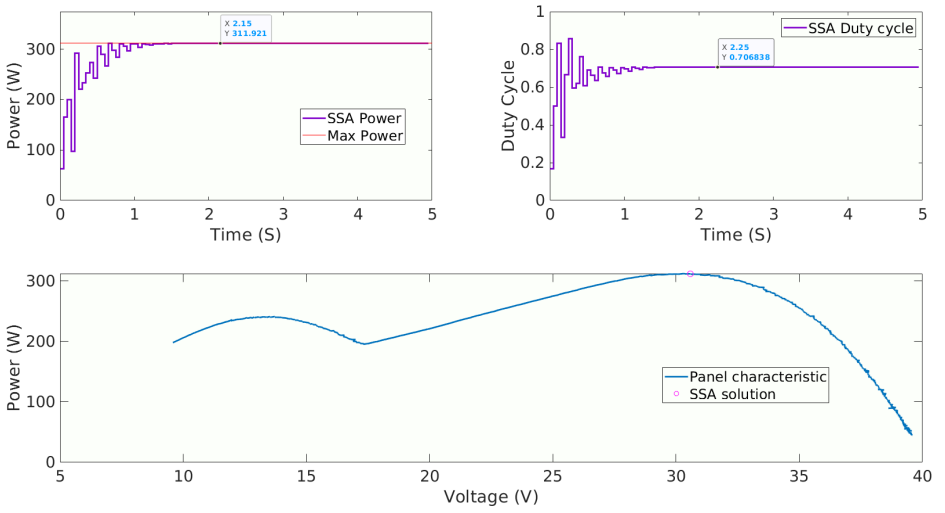


Figure 4.14 : SSA performance, no shading.

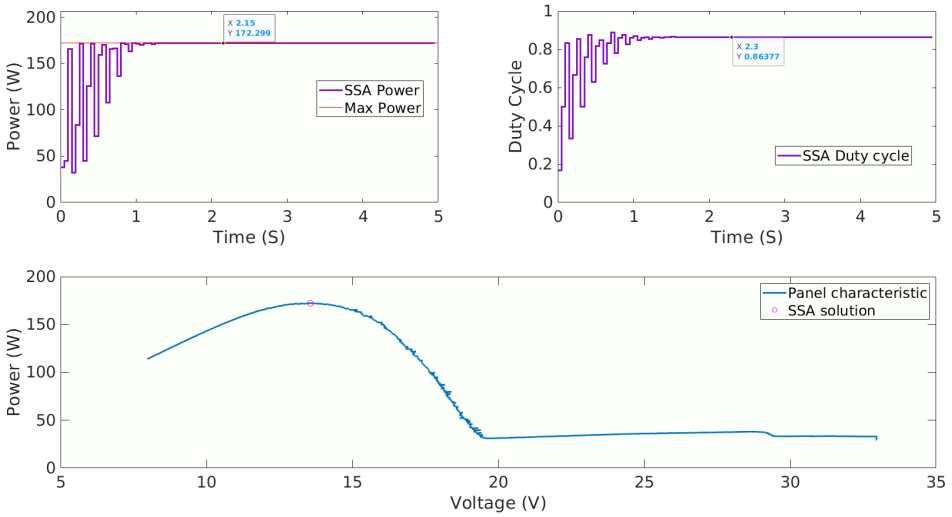
As we can see, our algorithm managed to converge to the MPP, with a final duty cycle value of 83%.
The second test is under more complicated conditions : Partial shading.
We run the three previously shown characteristics we got under partial shading; the results are shown in figure fig. 4.15.



(a) Partial shading 1



(b) Partial shading 2



(c) Partial shading 3

Figure 4.15 : SSA performance under partial shading conditions

In all three conducted tests, the algorithm managed to converge to the MPP.

4.2.3.2 Test under Vivado

Now that we validated our software implementation, we proceed to test our hardware design of the same algorithm.
We first start by adapting the VHDL testbench, and just like in Matlab, instead of using the converter equations, the testbench would look up the duty-cycle value in the panel characteristics and assign the corresponding voltage or current values.
The same Python script we used in Chapter 1 was used to convert the data into a text file that can be read in Vivado.
We can see the result of the simulation in figure 4.16.

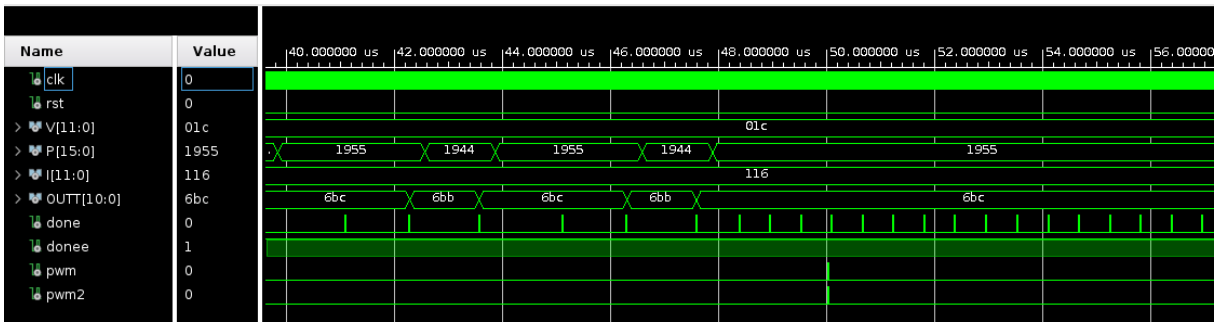
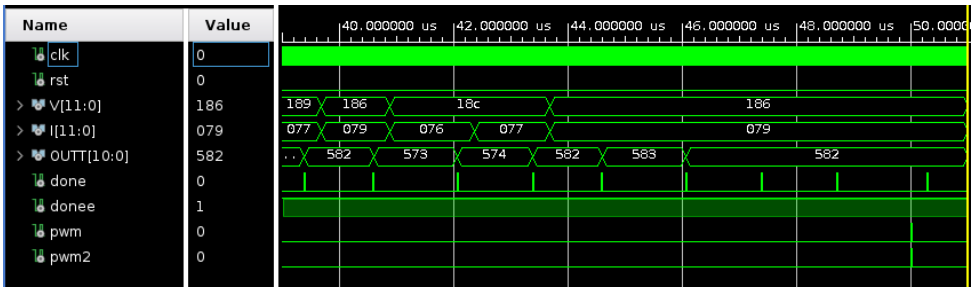


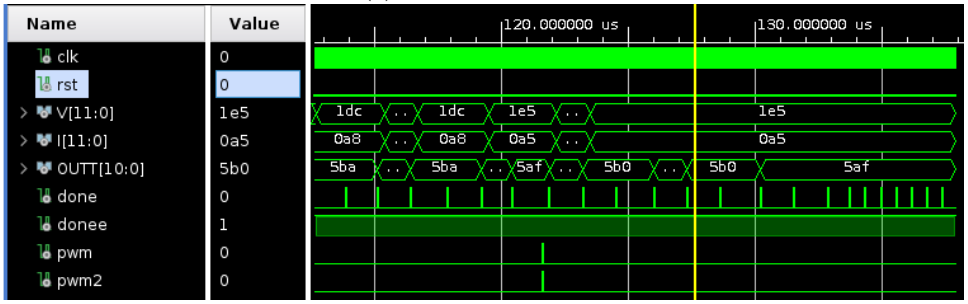
Figure 4.16 : SSA simulation, with no shading.

The final output duty cycle to which our design converged is "6bc", which equals 1724 in decimal. Dividing this by the maximum value (2048), we get a duty cycle of 0.84 or 84%, which is the same solution found on Matlab (figure 4.14) and which corresponds to the MPP.

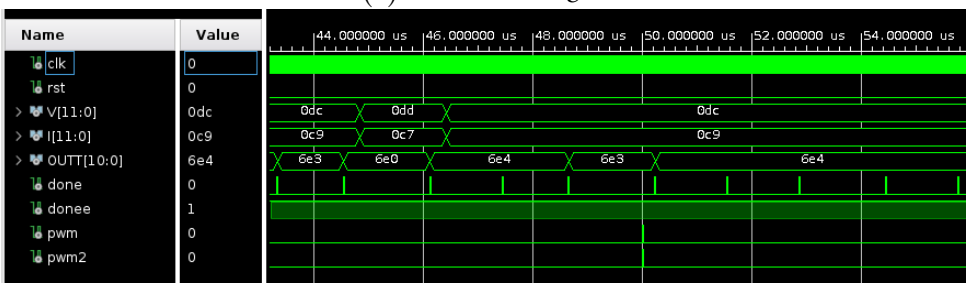
We test now with the remaining characteristics under partial shading conditions. Results are shown in figure 4.17.



(a) Partial shading 1



(b) Partial shading 2



(c) Partial shading 3

Figure 4.17 : SSA hardware implementation performance under partial shading

We summarize all our Matlab and Vivado results in table 4.3.

Conditions	Vivado			Matlab
	Hex Value	Decimal Equivalent	Duty-Cycle	Duty-cycle
No shading	6bc	1724	84%	83%
Partial shading 1	582	1410	68%	68%
Partial shading 2	5af	1455	71%	71%
Partial shading 3	6e4	1764	86%	86%

Table 4.3 : SSA simulation Results

As can be seen, the results we found on Matlab and the ones we found from our hardware design in Vivado are similar, and both converge to the MPP in both normal conditions and under partial shading.

4.3 Integrating the SSA MPPT

In this section, the aim is to integrate our FPGA-implemented SSA_MPPT controller into our test setup and test it in real-time.

4.3.1 Presenting the setup, including the implemented tracker on FPGA

In order to test the SSA controller, we need to first integrate the FPGA into our chain and implement a way to log data which would allow us to observe the controller's behaviour.

Our proposed approach is the use of a microcontroller to record data and log it to a computer, while the FPGA containing our SSA controller generates the control signals.
Figure 4.18 shows a diagram of our proposed set up.

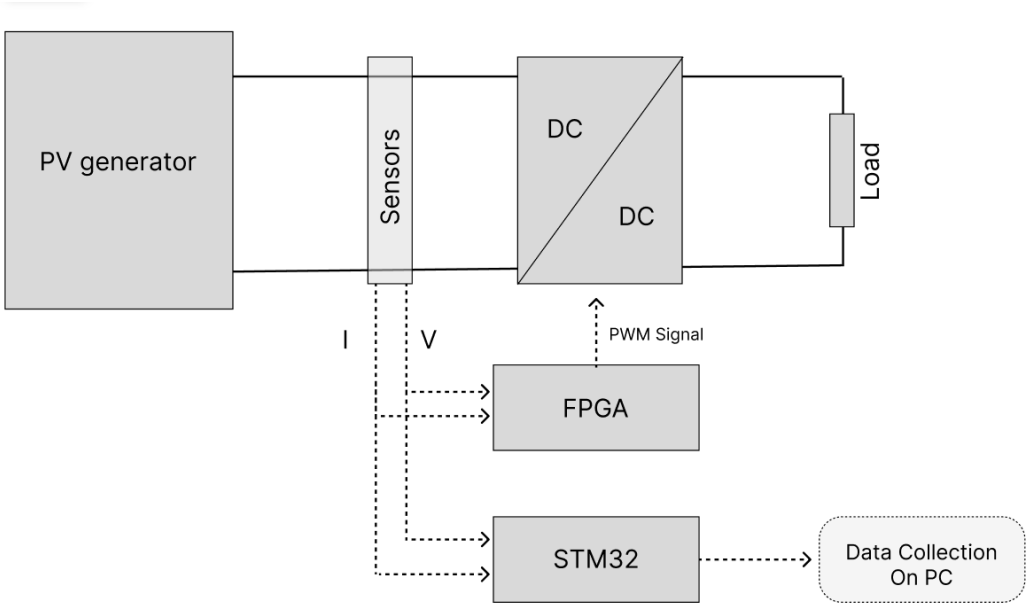


Figure 4.18 : Real-time testing setup diagram

Figure 4.19 presents the real used setup.

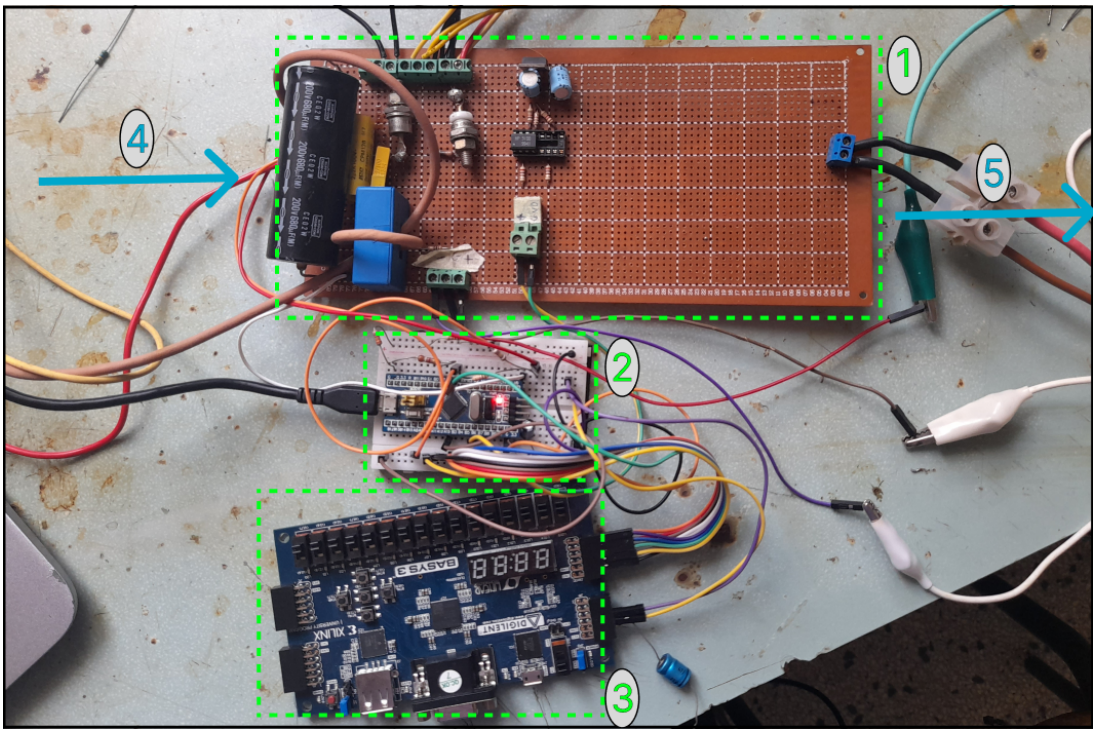


Figure 4.19 : Real-time testing setup

Where we have :

- 1 : Buck Boost Converter ;
- 2 : STM32 Microcontroller ;
- 3 : FPGA Basys 3 board ;
- 4 : DC Power input from PV panels ;
- 5 : DC Power output to Load.

4.3.2 SSA_MPPT live testing

Our testing methodology is compromised of 2 steps :

1. Extract panel characteristic using the microcontroller.
2. Run our MPPT controller while recording data thorugh the microcontroller

This methodology allows us to not only observe the behaviour of the controller through the collected data but also we will also be able to ensure that it's converging to the maximum power point by comparing the convergence point of the controller to the panel characteristic we extracted.

We tested our panel under a partial shading ; figure 4.20 shows the P(V) characteristic of the panel under the test conditions.

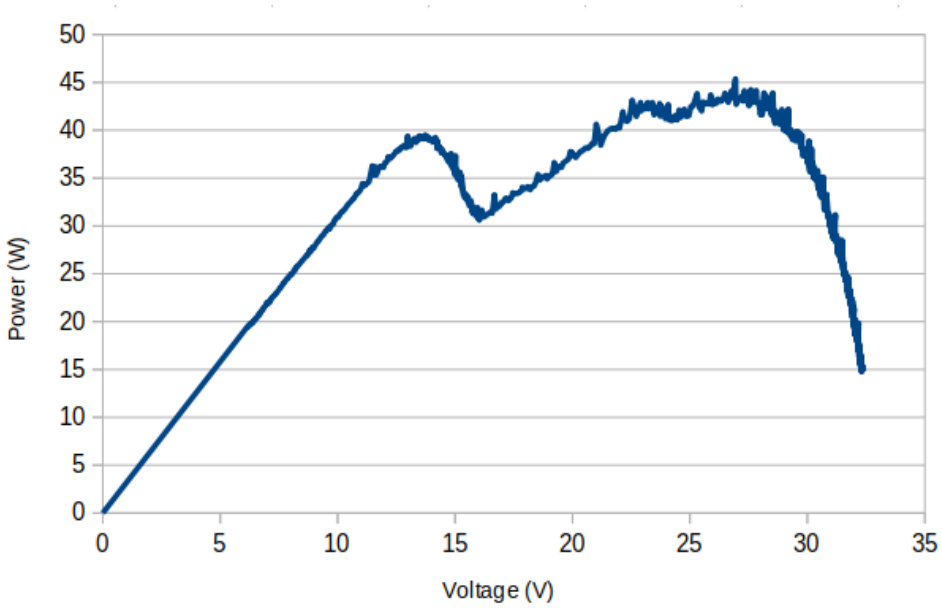


Figure 4.20 : Panel characteristic under partial shading

Right after recording the characteristic, we run our ssa_MPPT controller while monitoring and recording current and voltage values.
Figure 4.21 shows the variation of power output through time while the SSA controller is running.

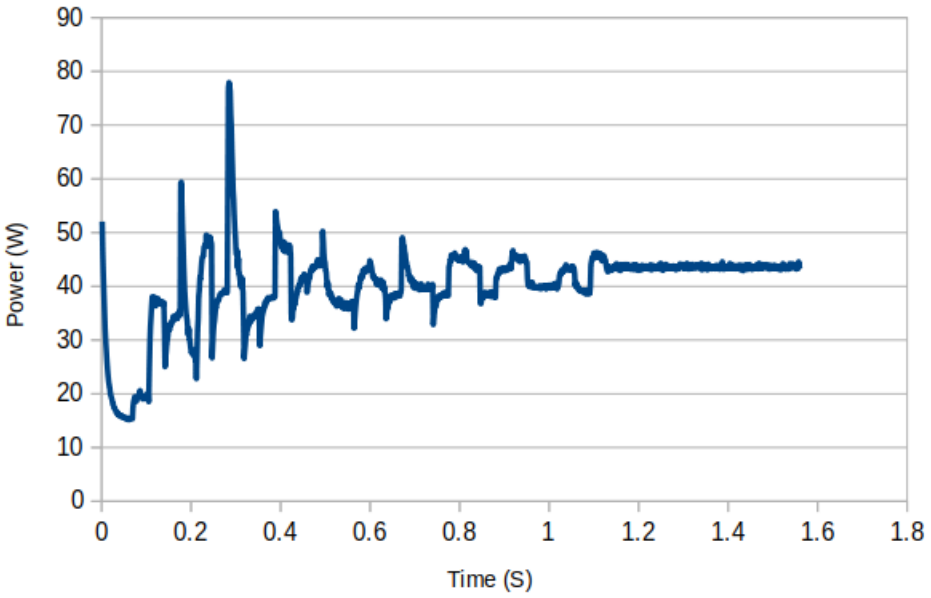


Figure 4.21 : SSA_MPPT controller behaviour : P(t)

As shown in figure 4.21, after 1.1 seconds, the algorithm converges and stabilises at 45W, which, by looking at the characteristic in figure 4.20, is the maximum power point.

We can also notice peaks in power exceeding the maximum power point; they are due to

measurement and circuit noise.

Conclusion

In this part, we realized a real test bench for testing solar panels and extracting their characteristics. We then used the panel data we collected under various conditions to validate our chosen method in a software environment (Matlab and Vivado). Finally, we tested our implemented controller on-grid and proved its effectiveness.

Conclusion and perspectives

This work aims to implement the SSA bio-inspired MPPT technique to control photovoltaic systems.
In this project, we implemented and tested the chosen algorithm in a software environment where we compared its performance to the PSO algorithm and where it showed its effectiveness in partial shading conditions.
We then used the Vitis HLS and Vivado design flows to create a hardware implementation and architecture for our controller. After this, we tested it and compared the results to the software simulation.
Finally, we developed a hardware test environment : From power circuitry to data collection, where we tested our designed controller in real time and validated its functionality.

In the end, not only did we successfully implement and validate the SSA algorithm, but the designs and test environments we developed can be used to test and implement any other MPPT algorithm quickly. Indeed, any other technique can be tested with the same tools and hardware, only needing to go through the HLS design flow.

The perspectives can be summarized in the following points :

- Further test the SSA controller under more varied conditions and using different panels ;
- A study about the large-scale deployment of the controller and the benefits it can give ;
- Using the test environments and hardware we developed to implement and test other techniques and do a comparative study.

Bibliography

1. ISHAQUE, Kashif ; SALAM, Zainal ; AMJAD, Muhammad ; MEKHILEF, Saad. An improved particle swarm optimization (PSO)–based MPPT for PV with reduced steady-state oscillation. *IEEE transactions on Power Electronics*. 2012, vol. 27, no. 8, pp. 3627–3638.
2. AZLI, Hadjer ; TITRI, Sabrina ; LARBES, Cherif. Modified particle swarm optimization based MPPT with adaptive inertia weight. In : *Smart Energy Empowerment in Smart and Resilient Cities* : *Renewable Energy for Smart and Sustainable Cities*. Springer, 2020, pp. 115–123.
3. ELTAMALY, Ali M ; AL-SAUD, MS ; ABOKHALIL, Ahmed G ; FARH, Hassan MH. Simulation and experimental validation of fast adaptive particle swarm optimization strategy for photovoltaic global peak tracker under dynamic partial shading. *Renewable and Sustainable Energy Reviews*. 2020, vol. 124, p. 109719.
4. GONZÁLEZ-CASTAÑO, Catalina ; RESTREPO, Carlos ; KOURO, Samir ; RODRIGUEZ, Jose. MPPT algorithm based on artificial bee colony for PV system. *IEEE Access*. 2021, vol. 9, pp. 43121–43133.
5. SUNDARESWARAN, Kinattingal ; PEDDAPATI, Sankar ; PALANI, Sankaran. MPPT of PV systems under partial shaded conditions through a colony of flashing fireflies. *IEEE transactions on energy conversion*. 2014, vol. 29, no. 2, pp. 463–472.
6. TITRI, Sabrina ; LARBES, Cherif ; AZLI, Hadjer ; BENKHIRA, Faycal ; DJEMAI, Narimane. A Powerful Bio-Inspired Fire Fly Algorithm Based MPPT Controller for PV Systems Under Partial Shading Conditions. In : *Artificial Intelligence and Heuristics for Smart Energy Efficiency in Smart Cities* : *Case Study* : *Tipasa, Algeria*. Springer, 2022, pp. 133–143.
7. DINCER, Ibrahim ; BICER, Yusuf. 3.17 Photonic Energy Production. In : DINCER, Ibrahim (ed.). *Comprehensive Energy Systems*. Oxford : Elsevier, 2018, pp. 707–754. isbn 978-0-12-814925-6. Available from doi : https://doi.org/10.1016/B978-0-12-809597-3.00336-9.
8. AL-KHAZZAR, Akram ; HASHIM, Emad. Temperature Effect on Power Drop of Different Photovoltaic Modules. 2015.
9. S.TITRI. *Circuits reconfigurables pour la gestion de l'énergie dans les systèmes photovoltaïques*. 2018. PhD thesis. Ecole Nationale Polytechnique.
10. AHMED, Jubaer ; SALAM, Zainal. An enhanced adaptive P&O MPPT for fast and efficient tracking under varying environmental conditions. *IEEE Transactions on Sustainable Energy*. 2018, vol. 9, no. 3, pp. 1487–1496.

11. VILLALVA, Marcelo G ; RUPPERT, Ernesto. Analysis and simulation of the P&O MPPT algorithm using a linearized PV array model. In : 2009 35*th Annual Conference of IEEE Industrial Electronics*. IEEE, 2009, pp. 231–236.

12. SAFARI, Azadeh ; MEKHILEF, Saad. Simulation and hardware implementation of incremental conductance MPPT with direct control method using cuk converter. *IEEE transactions on industrial electronics*. 2010, vol. 58, no. 4, pp. 1154–1161.

13. SHANG, Liqun ; GUO, Hangchen ; ZHU, Weiwei. An improved MPPT control strategy based on incremental conductance algorithm. *Protection and Control of Modern Power Systems*. 2020, vol. 5, pp. 1–8.

14. BAIMEL, Dmitry ; TAPUCHI, Saad ; LEVRON, Yoash ; BELIKOV, Juri. Improved Fractional Open Circuit Voltage MPPT Methods for PV Systems. *Electronics*. 2019, vol. 8, no. 3. issn 2079-9292. Available from doi : 10.3390/electronics8030321.

15. LEEDY, Aleck W ; GUO, Liping ; AGANAH, Kennedy A. A constant voltage MPPT method for a solar powered boost converter with DC motor load. In : 2012 *Proceedings of IEEE Southeastcon*. IEEE, 2012, pp. 1–6.

16. BOUAKKAZ, Mohammed Salah ; BOUKADOUM, Ahcene ; BOUDEBBOUZ, Omar ; BOURAIOU, Ahmed ; ATTOUI, Issam, et al. ANN based MPPT algorithm design using real operating climatic condition. In : 2020 2*nd international conference on mathematics and information technology (ICMIT)*. IEEE, 2020, pp. 159–163.

17. AL-MAJIDI, Sadeq D. ; ABBOD, Maysam F. ; AL-RAWESHIDY, Hamed S. Design of an intelligent MPPT based on ANN using a real photovoltaic system data. In : 2019 54*th International Universities Power Engineering Conference (UPEC)*. 2019, pp. 1–6. Available from doi : 10.1109/UPEC.2019.8893638.

18. DIVYASHARON, R ; BANU, R Narmatha ; DEVARAJ, D. Artificial neural network based MPPT with CUK converter topology for PV systems under varying climatic conditions. In : 2019 *IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*. IEEE, 2019, pp. 1–6.

19. ALI, Mahmoud N ; MAHMOUD, Karar ; LEHTONEN, Matti ; DARWISH, Mohamed MF. An efficient fuzzy-logic based variable-step incremental conductance MPPT method for grid-connected PV systems. *Ieee Access*. 2021, vol. 9, pp. 26420–26430.

20. REZK, Hegazy ; ALY, Mokhtar ; AL-DHAIFALLAH, Mujahed ; SHOYAMA, Masahito. Design and hardware implementation of new adaptive fuzzy logic-based MPPT control method for photovoltaic applications. *Ieee Access*. 2019, vol. 7, pp. 106427–106438.

21. HASSAN, Tehzeeb-ul ; ABBASSI, Rabeh ; JERBI, Houssem ; MEHMOOD, Kashif ; TAHIR, Muhammad Faizan ; CHEEMA, Khalid Mehmood ; ELAVARASAN, Rajvikram Madurai ; ALI, Farman ; KHAN, Irfan Ahmad. A novel algorithm for MPPT of an isolated PV system using push pull converter with fuzzy logic controller. *Energies*. 2020, vol. 13, no. 15, p. 4007.

22. ELAISSAOUI, Hayat ; ZEROUALI, Mohammed ; EL OUGLI, Abdelghani ; TIDHAF, Belkassem. MPPT algorithm based on fuzzy logic and artificial neural network (ANN) for a hybrid solar/wind power generation system. In : 2020 *Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*. IEEE, 2020, pp. 1–6.

23. GE, Xun ; AHMED, Faraedoon Waly ; REZVANI, Alireza ; ALJOJO, Nahla ; SAMAD, Sarminah ; FOONG, Loke Kok. Implementation of a novel hybrid BAT-Fuzzy controller based MPPT for grid-connected PV-battery system. *Control Engineering Practice*. 2020, vol. 98, p. 104380.

24. ALI, Mahmoud N ; MAHMOUD, Karar ; LEHTONEN, Matti ; DARWISH, Mohamed MF. Promising MPPT methods combining metaheuristic, fuzzy-logic and ANN techniques for grid-connected photovoltaic. *Sensors*. 2021, vol. 21, no. 4, p. 1244.

25. BABES, Badreddine ; BOUTAGHANE, Amar ; HAMOUDA, Noureddine. A novel nature-inspired maximum power point tracking (MPPT) controller based on ACO-ANN algorithm for photovoltaic (PV) system fed arc welding machines. *Neural Computing and Applications*. 2022, vol. 34, no. 1, pp. 299–317.

26. BENKHIRA FAYCAL, DJEMAI Narimane. *Contribution to the desing and implementation of a BioInspired algorithm on FPGA circuit*. 2021. MA thesis. Ecole Nationale Polytechnique.

27. HASSAN, Aakash ; BASS, Octavian ; MASOUM, Mohammad A.S. An improved genetic algorithm based fractional open circuit voltage MPPT for solar PV systems. *Energy Reports*. 2023, vol. 9, pp. 1535–1548. issn 2352-4847. Available from doi : https://doi.org/10.1016/j.egyr.2022.12.088.

28. DARABAN, Stefan ; PETREUS, Dorin ; MOREL, Cristina. A novel MPPT (maximum power point tracking) algorithm based on a modified genetic algorithm specialized on tracking the global maximum power point in photovoltaic systems affected by partial shading. *Energy*. 2014, vol. 74, pp. 374–388.

29. DARABAN, Stefan ; PETREUS, Dorin ; MOREL, Cristina. A novel global MPPT based on genetic algorithms for photovoltaic systems under the influence of partial shading. In : *IECON* 2013-39*th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2013, pp. 1490–1495.

30. KACED, Karim ; TITRI, Sabrina ; LARBES, Cherif. Enhancement of extracted power from photovoltaic systems through accelerated particle swarm optimisation based MPPT. In : *Smart Energy Empowerment in Smart and Resilient Cities : Renewable Energy for Smart and Sustainable Cities*. Springer, 2020, pp. 94–102.

31. KRISHNAN G, Satheesh ; KINATTINGAL, Sundareswaran ; SIMON, Sishaj P ; NAYAK, Panugothu Srinivasa Rao. MPPT in PV systems using ant colony optimisation with dwindling population. *IET Renewable Power Generation*. 2020, vol. 14, no. 7, pp. 1105–1112.

32. TITRI, Sabrina ; LARBES, Cherif ; TOUMI, Kamal Youcef ; BENATCHBA, Karima. A new MPPT controller based on the Ant colony optimization algorithm for Photovoltaic systems under partial shading conditions. *Applied Soft Computing*. 2017, vol. 58, pp. 465–479.

33. AZLI, Hadjer ; TITRI, Sabrina ; LARBES, Cherif. MPPT - Based Improved Salp Swarm Algorithm for Improving Performance and Efficiency of Photovoltaic System Under Partial Shading Condition. *International Conference in Artificial Intelligence in Renewable Energetic Systems*. 2020, pp 478–486. isbn 978-3-030-63845-0. Available from doi : 10.1007/978-3-030-63846-7_45.

34. YANG, Bo ; ZHONG, Linen ; ZHANG, Xiaoshun ; SHU, Hongchun ; YU, Tao ; LI, Haofei ; JIANG, Lin ; SUN, Liming. Novel bio-inspired memetic salp swarm algorithm and application to MPPT for PV systems considering partial shading condition. *Journal of cleaner production*. 2019, vol. 215, pp. 1203–1222.

35. KACED, Karim ; LARBES, Cherif ; RAMZAN, Naeem ; BOUNABI, Moussaab ; DAHMANE, Zine elabadine. Bat algorithm based maximum power point tracking for photovoltaic system under partial shading conditions. *Solar Energy*. 2017, vol. 158, pp. 490–503. issn 0038-092X. Available from doi : https://doi.org/10.1016/j.solener.2017.09.063.

36. TITRI, Sabrina ; KACED, Karim ; LARBES, Cherif. Maximum power point tracking based on the bio inspired BAT algorithm. In : *Smart Energy Empowerment in Smart and Resilient Cities : Renewable Energy for Smart and Sustainable Cities*. Springer, 2020, pp. 22–29.

37. LEI, Gang ; SONG, Heqing ; RODRIGUEZ, Dragan. Power generation cost minimization of the grid-connected hybrid renewable energy system through optimal sizing using the modified seagull optimization technique. *Energy Reports*. 2020, vol. 6, pp. 3365–3376.

38. MIRJALILI, Seyedali ; GANDOMI, Amir H. ; MIRJALILI, Seyedeh Zahra ; SAREMI, Shahrzad ; FARIS, Hossam ; MIRJALILI, Seyed Mohammad. Salp Swarm Algorithm : A bio-inspired optimizer for engineering design problems. *Advances in Engineering Software*. 2017, vol. 114, pp. 163–191. issn 0965-9978. Available from doi : https://doi.org/10.1016/j.advengsoft.2017.07.002.

40. ULLAH, Md Habib ; HAQUE, Md.Niaz ; RAHIMI, Md ; DHAR, Ripan. An Efficient Solar Pumping System for Rural Areas of Bangladesh. *International Journal of Scientific and Engineering Research*. 2013, vol. 4, p. 1765.

41. S.TITRI ; AL. A bio inspired maximum power point tracking controller for PV systems under partial shading conditions. *Indonesian Journal of Electrical Engineering and Computer Science*. 2022, vol. 28.

42. SKLYAROV, V. ; SKLIAROVA, I. ; BARKALOV, A. ; TITARENKO, L. *Synthesis and Optimization of FPGA-Based Systems*. Springer International Publishing, 2016. Lecture Notes in Electrical Engineering. isbn 9783319378626. Available also from : https://books.google.dz/books?id=2osPvgAACAAJ.

43. WESTE, N.H.E. ; HARRIS, D. *CMOS VLSI Design : A circuits and systems perspective*. Pearson India, 2015. isbn 9789332559042. Available also from : https://books.google.dz/books?id=0RAwDwAAQBAJ.

44. FINGEROFF, M. *High-Level Synthesis Blue Book*. Xlibris Corporation, 2010. isbn 9781453588147. Available also from : https://books.google.dz/books?id=pnGjcQAACAAJ.

45. XILINX. *Vitis High-Level Synthesis User Guide (UG1399)*. Xilinx, 2023. isbn 9781453588147.

46. YI, SU ; XIAO, HU ; YONGJIE, SUN. FPGA Accelerating Core Design Based on XNOR Neural Network Algorithm. *MATEC Web Conf*. 2018, vol. 173, p. 01024. Available from doi : 10.1051/matecconf/201817301024.

47. XILINX. *7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide (UG480)*. Xilinx, 2022.

48. BLACKMAN, David ; VIGNA, Sebastiano. Scrambled Linear Pseudorandom Number Generators. *ACM Transactions on Mathematical Software*. 2021, vol. 47, pp. 1–32. Available from doi : 10.1145/3460772.

49. MARSAGLIA, George. Xorshift RNGs. *Journal of Statistical Software*. 2003, vol. 8, no. 14, pp. 1–6. Available from doi : 10.18637/jss.v008.i14.

51. LEM. *LA\100-P*. 2022. Available also from : https://www.lem.com/sites/default/files/products_datasheets/la_100-p_v14.pdf.

Webography

39. *A salp chain*. wikimedia commons, [n.d.]. Available also from : https://bit.ly/3oHli62.

50. RANTWIJK, Joris van. *Vhdl prng*. [N.d.]. Available also from : https://github.com/jorisvr/vhdl_prng.