Construction of a functional solar panel inverter
Power Electronics Project

B. Ersayin
b.ersayin@student.utwente.nl
(**s3256472**)

P. Geronimo
p.d.d.geronimo@student.utwente.nl
(**s2522373**)

S.L. Geerdink
s.l.geerdink@student.utwente.nl
(**s2820234**)

M.F. Mallant
m.f.mallant@student.utwente.nl
(**s2705168**)

J.A. Grefte
j.a.grefte@student.utwente.nl
(**s2840804**)

J.S. Boersma
j.s.boersma@student.utwente.nl
(**s2629089**)

January 2024

1 Introduction

The global conventional energy system consisting of fossil fuel-powered electricity has many flaws. First of all, fossil fuel sources are not infinite and are likely to be depleted in the near future. The fossil fuel that is still available is also needed for the fabrication of plastics and rubber. These are materials that have become irreplaceable in modern society. Furthermore, burning fossil fuel for the production of electricity is polluting the environment due to the emission of carbon dioxide (CO_2). This phenomenon is widely acknowledged as the main reason for climate change. To tackle climate change, global agreements have been concluded (Paris 2015) that state that the increase in average global temperature should not exceed 2 $^{\circ}C$.
Governments from all participating countries should invest in solutions for green renewable energy. Photovoltaic (PV) solar energy is one of such renewable energy sources. It has a continuing growing industry that has the potential for private individuals to produce clean renewable energy on the roof of their own houses with the use of solar panels. Since these solar panels produce DC power, it has to be transformed into the grid's AC power using an inverter. The design and prototyping of such an inverter is what this project is all about.

The requirements for the solar inverter articulate distinct goals, breaking down its functions into manageable components. These components include implementing Maximum Power Point Tracking (MPPT) to optimize power transfer from solar panels, converting DC to AC to align with the grid's requirements, and ensuring synchronization with the grid's phase and frequency. These goals collectively enhance the inverter's performance and facilitate easier testing and evaluation.

So one of the goals is to transfer the maximum amount of power from the solar panels to the grid. The maximum power point (MPP) is the operating point of voltage and current output where the solar panel generates the maximum amount of power for a given set of environmental conditions. Therefore, an MPP Tracker (MPPT) needs to be implemented.
As the grid is alternating current, the DC current that is obtained from the solar panel must be converted to AC current. The phase from the current of the solar panel must match the phase of the grid. An out-of-phase source on the grid could lead to large, possibly destructive currents. Also, the voltage outputted by the inverter must be slightly higher than the grid voltage to supply current to the grid.

The first step that has to be taken, is the inverting of the DC voltage to an AC voltage. To do this, MOSFETs can be used as switches to make a so-called H-bridge. This H-bridge can be used to swap the polarity of the DC-voltage, which creates a square wave. By opening and closing the switches every 20 ms, a square wave with a frequency of 50 Hz is generated.

An MPPT algorithm will continuously monitor the solar panel's output and adjust the electrical operating point to ensure that the system operates at or near the maximizing energy harvest. One way to make an MPPT is by using a Buck-Boost converter.
A low-pass filter can be added after the H-bridge's square wave output in order to convert it into a sinusoidal signal.

In order to adapt to the grid, we could have the controller of the H-bridge sense the grid. The switches could then be opened and closed according to the frequency and phase of the grid. Hence, the resulting wave will have the same frequency and phase, so that the electrical power after filtering can be provided to the grid.

In this report, Section 2 contains the literature review and Section 3 contains the top-level design, while Section 4 will include the detailed design which looks into the theory required for the realization of the project. Section 5 will illustrate the complete experimental setups and the gathered results. Section 6 will contain the Discussion of the experiment's outcome and results, particularly discussing error analysis and possible improvements of the system. Lastly, the conclusion will be asserted in Section 7. The Appendix will include any extra results and/or code used for the project.

2 Literature review

2.1 Key techniques in solar power inversion

The conversion of DC power from the solar panels to AC power provided to the utility consists of multiple stages. The literature available on the various stages of solar power inversion encompasses a broad spectrum of research and development efforts aimed at enhancing the efficiency, reliability, and performance of solar energy conversion systems. This section delves into the critical aspects of solar power inversion, including Maximum Power Point Tracking (MPPT), DC-AC conversion, and the grid injection process, with a specific focus on phasor synchronization with the grid. As discussed before, we need to collect maximum power from the solar panels all the time through an MPPT. How, and at what stage in the inverter this happens, is not prescribed. A commonly used approach is implementing MPPT in a DC to DC converter connected to the output of the solar panels directly [1].

2.2 Maximum power point tracking (MPPT)

Every solar panel/cell has a characteristic I-V curve. This curve determines the relation between the voltage and the current that the panel delivers. Its shape is dependent on several factors such as the amount of sunlight, ambient temperature, and the solar panel itself [2]. A typical looking I-V curve for a solar panel can be seen in Figure 1.

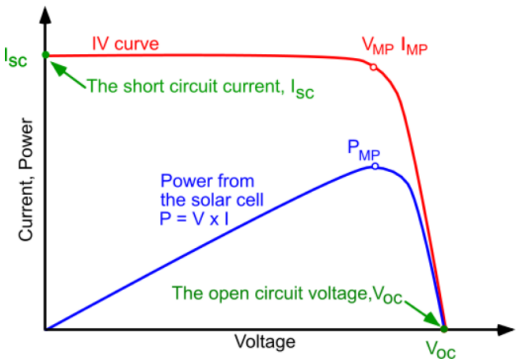

Figure 1: Typical voltage current graph of a solar cell

The goal of an MPPT system is to keep the voltage and current at the optimal point of the I-V curve for maximum power output of the solar panels. This can be accomplished by changing the equivalent resistance as seen from the solar panel output. When looking at figure 1, such an equivalent resistance can be thought of as a straight line through the origin intersecting the IV curve at some point. If that point would be the MPP, this equivalent resistance is matched perfectly yielding the highest power from the solar panels. The equivalent resistance can be tuned using the duty cycle of a DC to DC converter[3], in particular a boost converter. The resistance is controlled by the duty cycle of the buck/boost converter that is placed in between the solar panel and the load, the relationship between the duty cycle and the resistance is given by [3]

$$R_{equivalent} = (1 - D)^2 R_{load} \tag{1}$$

The challenge lies in the fact that the boost converter not only determines the equivalent resistance that will yield the highest power but also should boost the voltage enough such that the power from the solar panels can be injected into the grid. As it does not matter that much if the output voltage of the boost converter is a little higher than the grid, the duty cycle that boosts the input voltage to just the grid voltage is considered the minimum needed duty cycle. The maximum voltage would yield the maximum duty cycle. There thus exists a range in which the duty cycle can be varied to reach the MPP. The duty cycle is then controlled by algorithms that will seek the highest power, a number of these algorithms are explained in the next paragraph.

Algorithms for maximum power point tracking
Different techniques for reaching the maximum power point exist, these can be split up into three categories; indirect methods, direct methods, and soft computing [4]. Indirect methods use models of the solar panel to get the I-V curve based on parameters such as temperature, irradiance, etc. Direct methods measure the current and voltage and use a search algorithm to find the maximum power point. Lastly, soft computing uses algorithms for high-level problems, an example of such an algorithm is neural networks. The disadvantage of an indirect method that uses a model of the solar panel compared to the direct method, is that using an indirect method

requires sensors for parameters on which that model is based. These could be for example temperature meters or irradiance meters. Whereas direct methods look at the current and voltage. Furthermore, soft computing requires lots of data to "train" the model. As using large models for indirect methods and soft computing is out of the scope of this report, the focus is on direct methods for MPPT. Hereafter an overview of different algorithms for direct MPPT that are used is given.

Perturb&Observe
The perturb and observe (P&O) algorithm works by incrementing over a voltage range and observing how the power changes [5]. If the power increases the algorithm keeps incrementing the voltage. If the maximum power point has been reached the algorithm stops incrementing the voltage. One disadvantage of this algorithm is that the power output keeps oscillating around the MPP, as the choice of whether to keep incrementing is dependent on the previous increment [6]. The algorithm can be implemented using a controller. In advance of the algorithm, a variable named Maximum power (PMAX) is set to an initial value, usually zero. From then on, the power is constantly measured and at each increment, it checks whether this power is greater than PMAX, if so, it is set as the new PMAX and the next increment will start where the duty cycle of the boost converter is again updated in the same direction until it oscillates through the MPPT [7].

Incremental Conductance
The Incremental Conductance algorithm is similar to Perturb & Observe in the sense that it uses an algorithm that checks the voltage and current every iteration and alters the operation point accordingly. However, the Incremental Conductance algorithm uses the incremental conductance change $\frac{dI}{dV}$ and compares it with the array's conductance $\frac{I}{V}$. If those two are equal, the MPP is reached. Although Incremental Conductance has less oscillatory behaviour around the MPP, research has shown that the performance of Perturb & Observe and Incremental Conductance is generally quite equal [8].

2.3 DC-AC conversion

H-bridge functionality
In order to convert DC current into sine wave AC current, An H-bridge circuit is commonly used. Firstly, to achieve this, the polarity of the current flowing through the load has to change periodically and that is where an H-bridge comes into play. H-bridge circuits allow the polarity of the current going though the load to flip by using switches to change the direction of the current periodically between two possible paths. [9]

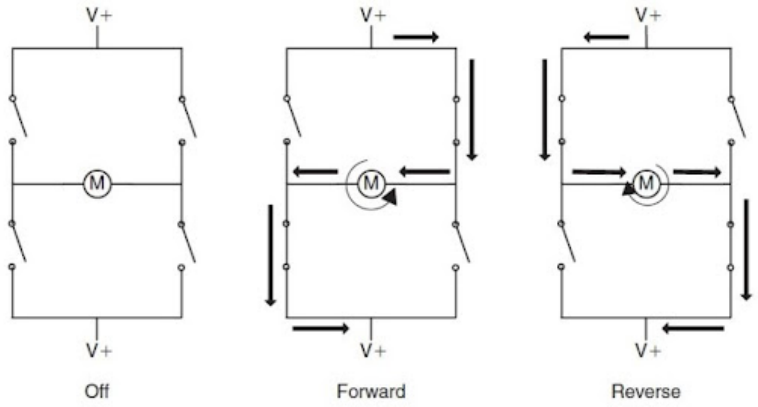

Figure 2: Circuit model representation of H-bridge circuit design

Considering the forward case given above in figure 2. By closing the top right and bottom left switches only, the current will flow through the load from its right to its left. On the other hand, considering the reverse case in Figure 2, by closing the top left and bottom right MOSFET, the current will only flow through the load from its left to its right. When the forward case and reverse case are done repetitively in a particular period, an AC square wave voltage is obtained across the load. [9]

In practice, because switching physical switches manually is impossible for high frequent applications, semiconductors such as transistors, IGBT's or MOSFETS are used to switch computationally and accurate. The application with the use of N-channel MOSFETS is shown in Figure 3.

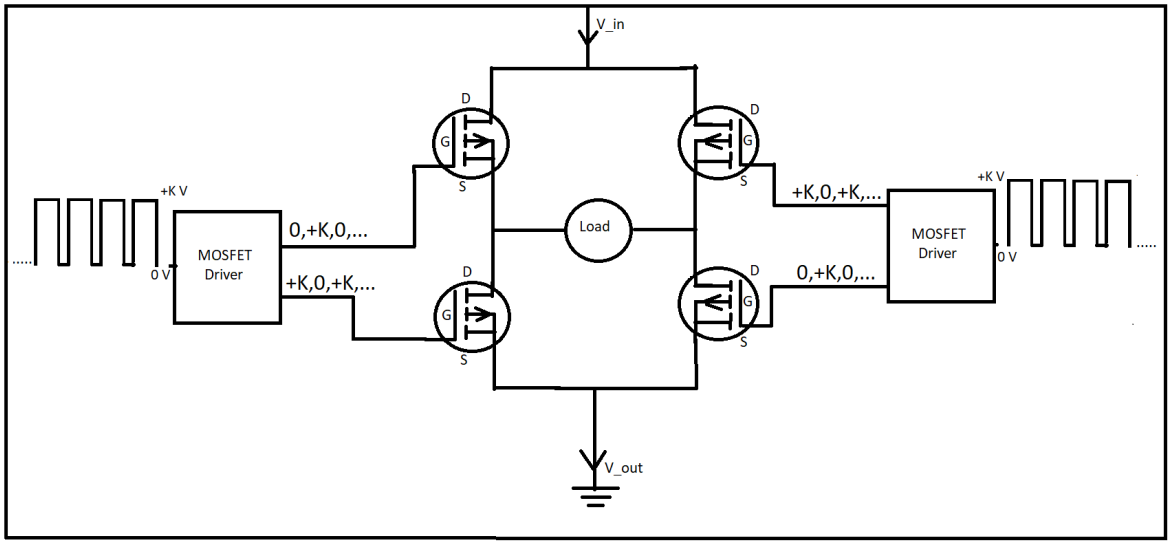

Figure 3: H-bridge circuit sketching with N-channel MOSFETs and MOSFET drivers

When the voltage applied to the gate pin of the N-channel MOSFET exceeds the threshold voltage, the path from the drain to the source behaves like a short circuit. In the forward case, as depicted in Figure 3, the gate pins of the top right and bottom left MOSFETs receive a positive gate voltage from the driver, sufficient to close the gate of the MOSFET. In the reverse case, the gate pins of the top left and bottom right MOSFETs receive this voltage. By repeating the forward and reverse cases consecutively, DC is converted into an AC square wave. [10]

To convert the AC square wave into an AC sine wave, the first step involves implementing sPWM to control the MOSFETs. In sPWM, the goal is to modulate the duty cycle in a way that the resulting waveform resembles a sine wave. This modulation is achieved by adjusting the duty cycle during each cycle of the square wave to match the amplitude of a corresponding point on a sine wave. In this case, the width of each specific square wave pulse in the AC square wave supplied by the MOSFET drivers will match the amplitude of a corresponding point on a sine wave.

The implementation of sPWM will be carried out using an ESP32 microcontroller. This microcontroller will be programmed to regulate the width of the square wave pulses supplied through the MOSFET drivers.

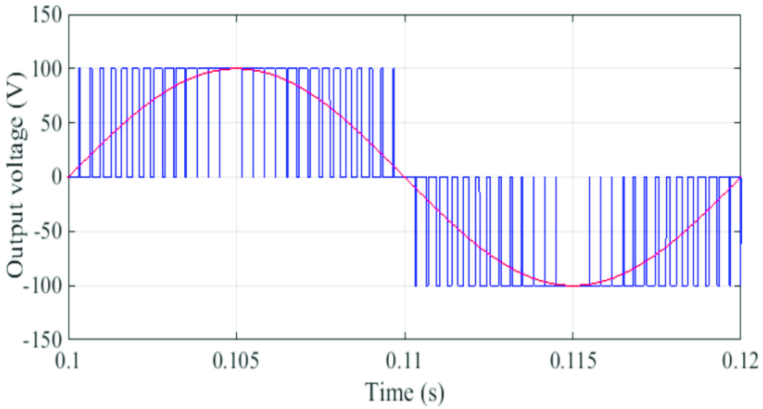

Figure 4: PWM signal(blue) and the underlying sine [11]

As a result of the implementation of sPWM, the AC square wave will be converted into a modified AC square wave as given in Figure 4. To convert the modified square wave into a sine wave, as illustrated in Figure **??**, low pass filtering across the load is necessary.

Filter

A low-pass filter is decided to be used to convert the modified AC square wave into an AC sine wave. The reason is a low-pass filter allows the lower frequencies to pass through while attenuating the higher frequencies of the PWM for example. This is essential if to filter out unwanted frequency components and retain only the

desired part of the signal. That being said, in the application converting the modified AC square wave into an AC sine wave, a low-pass filter will smooth out the sharp edges and harmonics present in the square wave.

In order to decide on the components used for a second order low-pass filter, we must use equation 2 to determine the cutoff frequency w_c of the inverter output.

$$f_c = \frac{1}{2\pi\sqrt{LC}} \tag{2}$$

2.4 Grid injection

To connect the AC voltage generated by the DC-AC converter and filtered by the filter to the grid, the MOSFETs of the H-bridge need to be synchronized with the grid [12]. This means that the voltage of the solar panel should be equal to the voltage of the grid. In addition, both must have the same frequency and the phase between them should be zero. The voltage is achieved by for example a Boost converter [12]. The frequency and phase are determined by the H-bridge. Depending on when the MOSFETs open and close, this will give the AC signal its frequency and phase with respect to the grid. Therefore, it is necessary to measure or sense the grid's voltage signal, such that it can be used to control the MOSFETs in the H-bridge. One way to do this is by the use of a microcontroller. It will measure the grid's voltage, and based on that and a triangular wave it can generate a PWM signal to open or close the MOSFETs [11]. When the grid's voltage changes from negative to positive, it will modulate 2 of the MOSFETs to generate the positive part of the signal. Once it notices that the voltage changes from positive to negative, it will start modulating the second pair of MOSFETs and it will open the first 2 MOSFETs. When the grid's voltage becomes positive again, the closed MOSFETs will be opened, and the others will be closed. By using this method, the generated signal has the same frequency as the grid, and there is no phase shift. If the microcontroller only has one PWM output signal, an inverter is necessary to invert the signal to properly control the MOSFETs. The resulting output from the modulation is shown in Figure 4 [11].

The same output can also be achieved using a duty cycle between the positive maximum and the negative maximum by inversely switching the diagonal pairs in one period. However, a dead time should be introduced to prevent a short circuit in the same 'leg' of the H-bridge, making it harder to control [11].

Both the grid and the Photo-Voltaic system are voltage sources. Thus, no current will flow when the amplitudes and frequencies are the same while the phase is zero, as described before. To let current flow from the generated AC signal to the grid, we consider a series impedance Z between the AC signal and the grid, where current will flow through according to [12]:

$$I = \frac{V_{generated\ signal} - V_{grid}}{Z} \tag{3}$$

Therefore, there needs to be a voltage difference between the generated signal and the grid. The frequency cannot be changed, as this will not lead to a consistent current flow. Things that can be changed are the amplitude and the phase. For example, when the amplitude is the same but the phase is different, the voltages can be expressed as [12]:

$$V_{grid} = V\sqrt{2}\sin(\omega t) \tag{4}$$

$$V_{generated\ signal} = V\sqrt{2}\sin(\omega t + \theta) \tag{5}$$

This gives a power flow from the generated signal to the grid according to Equation 6 [12].

$$P = \frac{V^2}{Z}\sin(\theta) \tag{6}$$

As can be seen, a phase of $\theta = 0$ would give no current flow so no power flow. But when there is a slight phase shift, there is a current and power flow into the grid. To achieve such a phase shift, the micro-controller only has to delay its signal by a short amount of time. This creates a delay in the generation of the AC signal. If this delay is slightly less than one period of the grid, i.e., it is slightly less than $\frac{1}{50} = 0.02$ s, the generated voltage will be slightly ahead of the grid, causing the necessary voltage difference [12].

Filtering

The resulting sine from the Pulse Width Modulation is unlike the grid sine not smooth. Instead, it has discrete steps with the width corresponding to the period of the PWM carrier signal and the height of the step corresponding to the step in the duty cycle. To achieve the desired smooth sine, the higher harmonics of the sharp steps must be filtered out employing a low-pass filter, as mentioned in the previous section [11].

3 Top-level design

The literature review in Section 2 gives an overview of the possible methods for MPPT, DC to AC conversion and the injection to the grid, sometimes with advantages and disadvantages given for each method. In this section, we will discuss the methods that we have chosen, including the motivation of our choices. First, the whole system is shown in a block diagram in Figure 5, after which we dive into each component.

The blocks in the block diagram consist of the descriptions given in the literature parts. The choice for first boosting the voltage and then inverting was made to allow for the use of a boost converter. This is a fairly efficient and easy-to-make solution for boosting the voltage. The two controllers represent two different algorithms to generate PWM signals for MOSFET drivers.

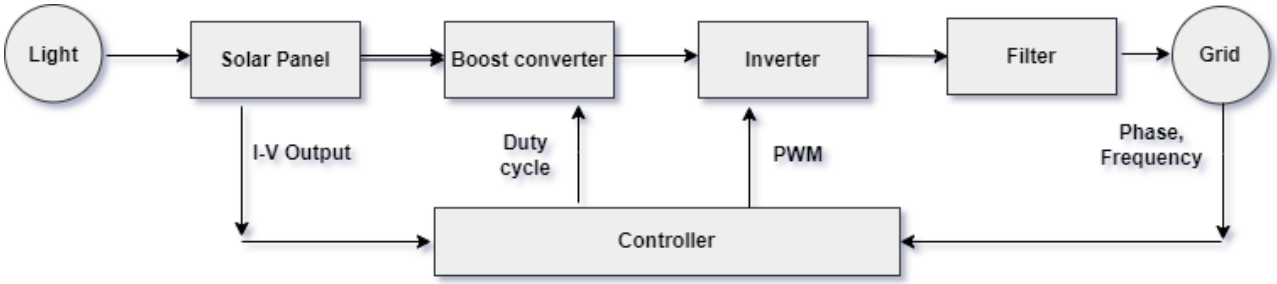

Figure 5: Block Diagram of top-level design

In Subsection 2, two direct methods for MPPT were described: Perturb&Observe and Incremental conductance. For the design of the solar power inverter, the Perturb&Observe algorithm is used. The Perturb&Observe algorithm has a similar performance to the Incremental conductance algorithm [8] but is easier to implement. In order to not boost the output voltage too much, a voltage meter is added that measures the output voltage and sends the data to a microcontroller, in which the duty cycle is decreased accordingly.

For the injection to the grid, as described in Section 2, it was chosen to control the H-bridge using a microcontroller. This microcontroller senses the grid's voltage, and based on that, it opens or closes the MOSFETs. This sensing of the grid's voltage creates a zero phase difference between the generated signal and the grid, but the microcontroller will delay its signals a little bit to create a small phase shift, to let current flow from the generated signal to the grid. The H-bridge will be controlled by a PWM modulated signal, as described in Section 2. This creates a sine wave of 50 Hz with higher-order harmonics, which have to be filtered out.

The filter will be an LC filter. This is a second-order filter, so it has -40 dB per decade of attenuation above the cut-off frequency, and it has a (rather small) resonance peak near the cut-off frequency. Therefore, the cut-off frequency should be a little bit higher than 50 Hz, otherwise, the core harmonic will be amplified, which is not desired. But the cut-off frequency should also not be too high, because this will reduce the attenuation of the higher harmonics. Therefore, it is important to know at what frequencies the higher harmonics are. For the PWM modulation, 10 kHz is used as the frequency. So with this, if a cut-off frequency in the range of 500-1000 Hz is chosen, the attenuation of 10 kHz by a second-order filter is $\leq$ -40 dB, or $\leq$ 0.01. This is enough attenuation. Also, a cut-off frequency of 500 Hz is high enough so that the resonance is not a problem. Furthermore, we should make sure that the capacitance values are not too high because otherwise, it may take too long to charge it up with a rough sine wave as a result.

To invert DC to AC, the H-bridge circuit receives DC voltage from the Boost converter. As detailed in Section 2, the DC current undergoes inversion into a square wave AC with a varying duty cycle. Subsequently, it is received by the filter to be smoothed out, transforming it into a sinusoidal AC waveform. The H-bridge comprises four equivalent MOSFETs driven by two MOSFET drivers to ensure well-regulated voltage is applied to the gate.

4 Design

4.1 Design of MPPT & Boost converter

Range of Dutycycles

In Figure 6, the boost circuit that is used to boost the voltage is given. This boost circuit is not only used to boost the input voltage to the desired output voltage but also for maximum power point tracking. Equation 7 and 8 determine the output voltage and the equivalent load of the boost circuit. The equivalent load determines

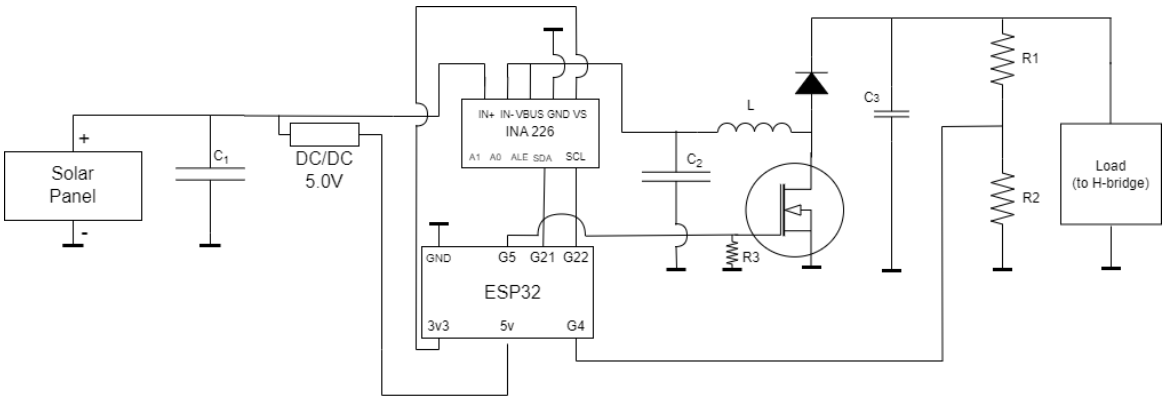

Figure 6: Boost circuit

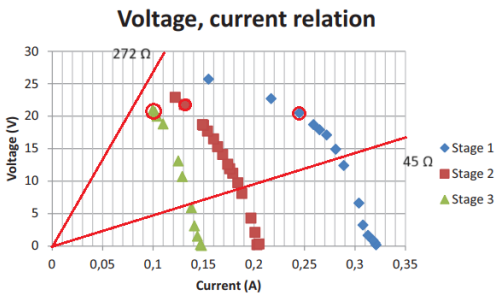

Figure 7:
I-V curves of the solar panel

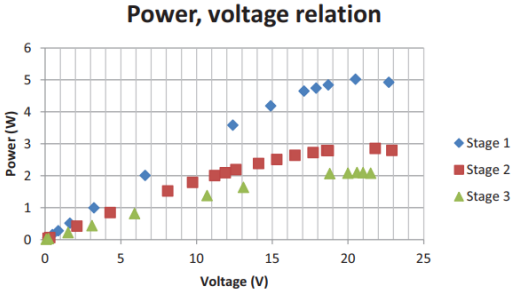

Figure 8:
P-V curve of the solar panel

where on the I-V curve of the solar panel the inverter is operating.

$$V_{out} = \frac{V_{in}}{(1-D)} \tag{7}$$

$$R_{equiv} = (1-D)^2 R_{load} \tag{8}$$

Therefore a range of duty cycles needs to be defined in which the input voltage is still boosted enough to achieve the required output voltage but the duty cycle can be varied to track the maximum power point. This range will be deducted using the I-V curves of the solar panel.

In Figure 7, the I-V curve of the solar panel is depicted. The maximum power points are encircled, and two lines are drawn from the origin with a certain slope corresponding to the equivalent resistance seen from the solar panels. As can be seen, the maximum load that is needed to reach the maximum power points for all stages is around 272Ω. For experimenting purposes, a resistance of around that value is used. When in the end the inverter is connected to the grid, this grid is also part of the load. One of the difficulties of this load being the grid is that the resistance is not known and that it is not adjustable. Therefore, it can not be known beforehand if the maximum power points can be met for all weather conditions.

Judging from Figure 7, for the solar panel at stage 1, the MPP should be achievable with a load resistance of around 200-300 , an experiment will be done and the experimentally found maximum power point will be compared to Figure 7, see also Subsection 5.1.

To obtain the minimal value of the duty cycle, first it must be known to which value the input voltage must be boosted. To inject the generated power into the grid a transformer is used that steps the voltage up from 12 V to 230 V. However this transformer has a no-load factor of 1.2 [13]. Furthermore, the 230 V that the transformer boosts to is the RMS value, this means that to make an AC signal the value should be multiplied by a factor of $\sqrt{2}$. Lastly the rest of the circuit in between the boost circuit and the transformer also has voltage drop, this is accounted for using a safety factor of $SF = 1.15$. Concluding the output voltage of the boost converter should be:

$$V_{out,boost} = 12\ [V] \cdot \sqrt{2} \cdot 1.2 \cdot 1.2 \cdot SF = 28[V] \tag{9}$$

As said before, the minimum output voltage from the boost converter should be 28 V. From the I-V curve that can be seen in Figure 7 the maximum output voltage is around 25 V[14]. Hence, the minimum duty cycle

is:

$$D_{min} = 1 - \frac{V_{in,max}}{V_{out,min}} = 1 - \frac{25}{28} \approx 0.107 \tag{10}$$

Furthermore, also an upper limit is chosen for the duty cycle, this is done to protect the MOSFET that is used in the circuit. This upper limit is set to $D_{max} = 0.850$. Therefore, the range in which the MPPT algorithm operates is between 0.107 and 0.850.

Capacitance and Inductance

A couple of other parameters, such as the capacitance and the inductance also need to be determined. To obtain the values for the capacitance and the inductance, Equation 11 and Equation 12 are used. Choosing the right value for the capacitance will ensure minimal voltage ripple, ΔV. Choosing the right value for the inductance will ensure proper conductance.

$$C_2 = \frac{I_{out}}{\Delta V \cdot f} = \frac{(1-D)I_{in}}{\Delta V \cdot f} \tag{11}$$

$$L > \frac{V_{in}}{2I_{in}f} \tag{12}$$

To determine the inductance, in Equation 12, the maximum value for V_{in} must be used and the minimum value for I_{in}. An acceptable value for the voltage ripple is chosen as $\Delta V = 0.25V$. Then, the value of f is determined by a reasonable high PWM frequency the ESP32 can provide. For f, a value of 100000 is chosen. Therefore the inductance and the capacitance are given by

$$C_2 = \frac{(1-0.5)\cdot 0.2}{0.25 \cdot 100000} = 4\mu F \tag{13}$$

$$L > \frac{25}{2 \cdot 0.1 \cdot 100000} = 1.25mH \tag{14}$$

Just after the solar panel, a capacitor is placed, this capacitor acts as a low-pass filter to filter out fluctuations in the input voltage. These fluctuations in the input voltage are present due to fluctuations in the light source that is shining on the solar panel. By placing the capacitor after the panel, a smoother signal is obtained. The value of the capacitance is $C_1 = 4700\mu F$.

Wiring of ESP32 and INA226

The MOSFET that is used to switch in the boost circuit is being controlled by the ESP32 microcontroller. The current and voltage that the solar panel delivers are measured by the INA226 power sense meter, these values are sent to the ESP32 by the I^2C port of the INA226. The ESP32 is powered by the solar panel. The INA226 measures the voltage drop over a shunt resistor that is placed between pin IN+ and pin IN-. The voltage drop divided by the resistor value of the shunt resistor gives the current that the panel outputs. With the current and the voltage, the power can be calculated. As the ESP32 is powered using 5 V and the solar panel gives more than 5 V a buck converted is used to reduce the voltage to 5 V. The INA226 is powered using the 3.3 V output port of the ESP32.

MPPT Algorithm

The algorithm that is used to track the maximum power point is the perturb&observe algorithm, the workings of this algorithm are explained in Subsection 2.2. Two important parameters that need to be determined for the perturb&observe algorithm are the initial duty cycle and the increment size. The initial duty cycle is the algorithm's first guess to get to the maximum power point. The increment size is important as it determines how fast the maximum power point can be reached and how fast the algorithm adapts to changes in the maximum power point. A higher increment size leads to a faster adapting algorithm, however, choosing a too-high increment size is also not beneficial as the algorithm oscillates around the maximum power point. The increment size is chosen to be 1% of the maximum duty cycle. The initial duty cycle is chosen to be the minimum duty cycle of 0.1. This is chosen because then the algorithm starts with an equivalent resistance being as high as possible. From then, the algorithm will increase the duty cycle until it oscillates around MPP.

4.2 Design of the H-bridge

The H-bridge circuit receives the input DC voltage from the DC-DC boost converter and signals of sPWM from the ESP32. In the H-bridge circuit, IR2109 MOSFET drivers are going to be used.

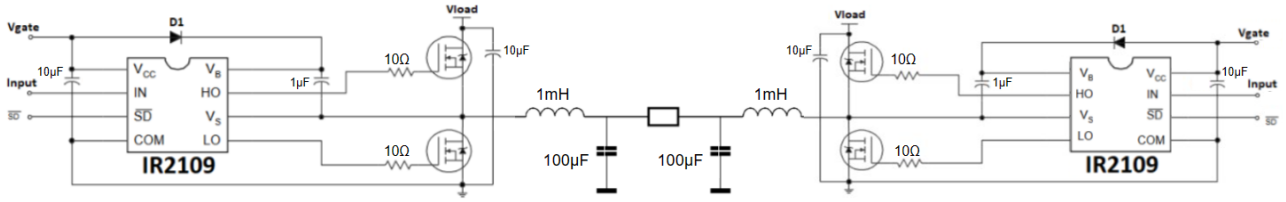

Figure 9: H-bridge Circuit Model including the filter

However, as a backup we have the schematic of another half bridge driver that has the same functionality as the IR2109 which is the IR2184, with the only difference being a different pin-out shown in Figure 10.

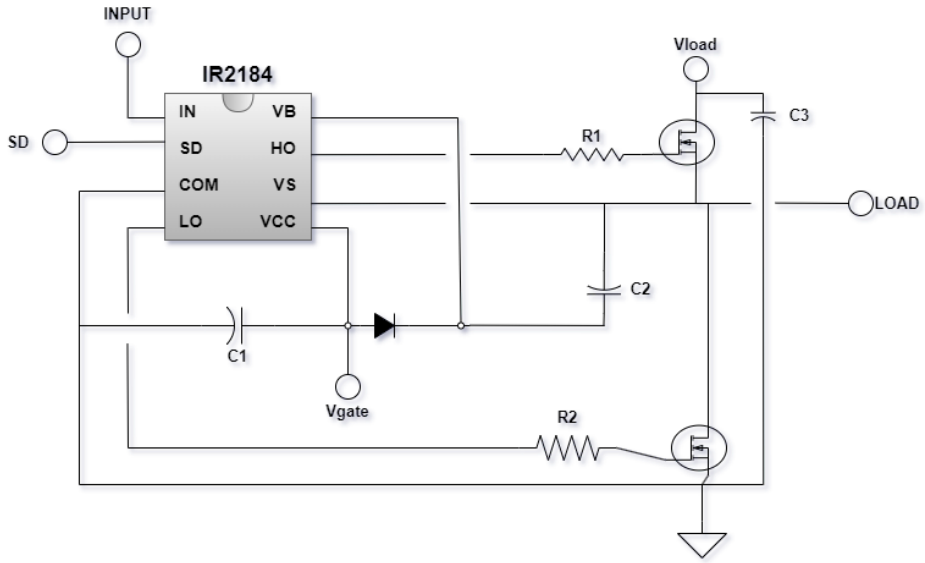

Figure 10: IR2184 Half bridge driver connections schematic

Where R1=R2=10Ω, C1=C3=10µF, C2=1µF.

4.3 Design of the filter

The filter applied to the square wave will filter out the higher frequencies, thereby obtaining only lower frequencies. The low-pass filter needs to cut off all frequencies that are higher than the grid frequency. In this way, only the first sine component of the block wave is left over. We will choose a second-order low pass filter because of its sharper cutoff due to the -40 dB/decade attenuation.

Taking the cutoff frequency Equation 2 in mind with the desired frequency of approximately $500 - 1000\ Hz$, a capacitor value of 6.8 µF and an inductor value 4.7 mH was chosen. This cutoff frequency should be enough to filter out the 18 kHz frequency component received from the PWM.

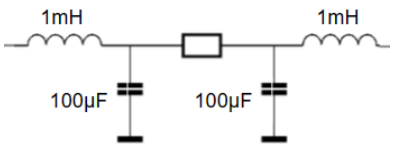

Figure 11: LC band-pass filter. Source: Adapted from [15].

The transfer function of this band-pass filter is shown in Equation 15.

$$H(j\omega) = \frac{1}{1 + (j\frac{\omega}{\omega_0})^2} \tag{15}$$

4.4 Design of the phase matching

In order to get the inverter synchronized with the grid, we first need to be able to sense the grid's zero crossings. With these zero crossings, we can determine when to turn on the H-bridge. A microcontroller will take care of this task. Measuring the grid cannot be directly done using the ADC on the ESP32 as it can only handle very low voltages. Also continuously measuring the voltage of the grid would be very computationally intensive. Therefore having an external comparing device that provides a trigger to the ESP32 when it detects a zero crossing. At first, the idea was to implement an optocoupler with a built-in Schmitt trigger as this would use the least amount of Integrated Circuits (IC). However, detecting a zero-crossing proved difficult as the LED inside the optocoupler turned off and on at around 0.6V instead of 0V, meaning we would have an inaccurate estimate of the actual crossing. Also with the actual grid the forward and reverse voltage on the optocoupler would be too high. Therefore, it was chosen to use an op-amp in differential mode to convert the sine around zero to a sine oscillating around 6V with a lower amplitude. After that, the sine gets fed into a comparator that compares the sine to 6VDC. The output is a logic signal between 0V and 3.3V. It was attempted to feed the signal into a Schmitt trigger to prevent undesired zero-crossing detections by the microcontroller. A Schmitt trigger is a comparator with hysteresis, meaning it needs a certain threshold to turn on and another to turn off. Between these thresholds, the device maintains its current value. However, the used Schmitt trigger had too little hysteresis voltage to be effective. It was also attempted to make a Schmitt trigger from an LM324N operational amplifier. This, however, is not a rail-to-rail op-amp and therefore does not output 0V low and 3.3V. Instead, it outputs a little over 500mV low and around 2V high due to the way the output is pulled high or low, which is too weak to reach the V+ and V- inputs of the Op-amp. Therefore, it was decided to reduce false zero crossings in the software by having the ESP32 ignore any falling edge after the first one measured.

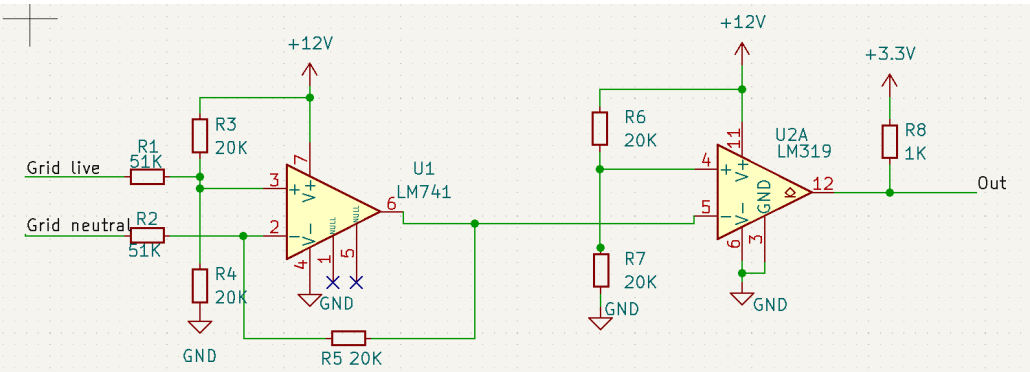

Figure 12: Zero Crossing detector

In Figure 12 the circuit for the zero crossing is shown. The comparator has an active low driver output. Meaning when it is active it will pull a normally pulled-up output to the ground, thus inverting the logic. Therefore a zero crossing from negative to positive will result in a falling edge from the comparator. The ESP32 sees this falling edge on one of its pins as an interrupt.

5 Experiment and Results

Data gathering of the following experiments were conducted through the use of the following components:

Oscilloscope	Rohde & Schwarz RTB2002
Frequency Generator	Agilent/Keysight 33120A
LCR Meterbridge	Rohde & Schwarz/Hameg HM8118
Power supply	E030-1 Delta Elektronika

Table 1: Equipment used in experiments

5.1 Testing Boost Converter with MPPT

Research question

In this experiment, the boost converter is tested together with the MPPT algorithm to investigate the performance of the first stage of the inverter. The circuit of Figure 6 is made and the performance is tested. The performance of the boost converter with MPPT is assessed on the following criteria:

1. To what voltage does the boost converter boost, and does it reach 28 V.
2. How big is the voltage ripple on the output voltage
3. What is the efficiency of the boost circuit
4. What maximum power is found and does it correspond to the data on the solar panel.

Hypothesis
We hypothesize that the input voltage is boosted by a factor of $\frac{1}{(1-D)^2}$ where D is the duty cycle that is received from the ESP-32. We expect the voltage ripple to not exceed the value that can be calculated with Equation 11 which is 0.25 V at max at an output current of 0.15 A. Because of the use of more efficient inductors with lower DC-resistance, the efficiency is expected to be quite good The efficiency of the boost circuit that can be calculated before measuring the output power is given by

$$eff = \frac{P_{out}}{P_{in}} = \frac{P_{in} - loss}{P_{in}} \tag{16}$$

The biggest losses take place at the inductor, as it has the highest internal series resistance among the circuit components. The loss through the inductor is given by

$$P_L = R_L \cdot (I_{in})^2 \tag{17}$$

To get a rough estimation of the efficiency only the power loss of the inductor is accounted for, the efficiency is given in Equation 18 [16].

$$eff = 1 - \frac{R_L(I_{in})^2}{I_{in} \cdot V_{in}} \tag{18}$$

To get a rough estimate, a point on the I-V curve of the solar panel is picked for stage 1; $V_{in} = 15$, $I_{in} = 0.28$. This gives

$$eff = 1 - \frac{1.8 \cdot (0.28)^2}{15 \cdot 0.28} = 96\% \tag{19}$$

Therefore it is hypothesized that the efficiency of the boost converter circuit will be around 96%. The major challenge of the experiment lies in obtaining maximum power from the solar panels using the MPPT algorithm of Perturb & Observe. It is hypothesized that this algorithm works and that the drawn power from the solar panel is at its maximum.

Experimental setup
These criteria will be tested for stage 1. The boost circuit and the MPPT algorithm will be tested separately from the rest of the inverter. To simulate the load resistance of the rest of the circuit a resistance of 220 Ω will be placed in parallel with C_2. The first and second criteria will be tested by measuring the output voltage of the boost converter with a probe connected to an oscilloscope. The third criterion will be tested by measuring the input voltage and current, the output voltage and current, and dividing output power by input power. The fourth criterion will be tested by comparing the found maximum power point to the maximum power point in Figure 7. The components that are used in the boost circuit are given in Table 5.1

Component	**Name/value**	**Amount**
Microcontroller	esp-wroom-32	1
Capacitor 1&2	NICHICON LLS1H472MELB	2
Capacitor 3	6.8 mF	1
Current and voltage sensor	INA226	1
Doide	1N5819 schotkey rectifier	1
Inductor	ROXBURGH EMC SF1120	1
MOSFET	STP16NF06	1
Resistor 1	60 kΩ	1
Resistor 2	3 kΩ	1
Resistor 3	1 k Ω	1
Buck converter 5V	RECOM DC-DC 5V	1

Figure 13: The output voltage of the boost converter

```
Bus voltage:   23.66125 V
Shunt voltage: 0.01691 V
Power: 3.99993 W
Duty cycle:  0.235
Boosted output voltage: 29.89764 V
```

Figure 14: Boost converter algorithm

Presentation

Figure 13 and 14 shows the output voltage of the boost converter and the parameters that are calculated by the algorithm, both figures are for stage 1. The Bus voltage is the voltage that the solar panel outputs and the shunt voltage is the voltage over the shunt resistor that in the current and voltage sensor, and is used to get the current of the solar panel. With the Bus voltage and the current, the power can be calculated. Furthermore, the algorithm calculates the duty cycle, and the boosted output voltage is calculated and monitored for the safety of the components after the boost converter.

Observation

With these values and the resistance of the load, the sub-research questions can be answered. The output voltage can be seen in Figure 14, and the voltage ripple can be seen in Figure 13. The efficiency is given by

$$eff = \frac{P_{out}}{P_{in}} = \frac{(\frac{V_{load}^2}{R_{load}})}{P_{in}} = \frac{(\frac{28.85^2}{220})}{3.99} = 95\% \tag{20}$$

Note that the voltage that is used to calculate the efficiency is the voltage that is measured by the oscilloscope and not the microcontroller as the oscilloscope is more accurate than the microcontroller. Furthermore, the load resistance that is used in the experiment is 220Ω. To assess whether the maximum power point can be met, the point at which the algorithm does not change the duty cycle anymore is compared to the maximum power point in Figure 8. In Figure 8 the maximum power point is found at 20.5 V and the maximum power is 5 W. The maximum power point the algorithm finds is at 23.6 V and the maximum power it gets is 4 W.

1. The output voltage is boosted to 28.85 V, therefore the boost converter can boost to 28 V.
2. The voltage ripple seems to be around 0.5 to 1 V
3. The efficiency of the boost converter is 95%.
4. The maximum power point is not found according to figure 8.

Discussion
As can be seen from Figure 13 and 14, there is a difference between the boosted voltage output the oscilloscope measures versus the boosted output voltage the ESP32 measures. The most likely reason for this difference is the inaccuracy in the voltage divider resistor ratio. Nonetheless, the boost converter still boosts to 28 V. Furthermore, the voltage ripple visible in Figure 13 is higher than the expected 0.25 V. One reason for this could be the fluctuations in the light source that powers the solar panel. The fluctuations of the light source would result in fluctuations in the input voltage, which would give some oscillations in the output voltage. The hypothesis on the efficiency seems to match the realized efficiency, the hypothesized efficiency is somewhat higher, however in the hypothesis losses in the MOSFET and diode were neglected. Lastly, when comparing the algorithm's found MPP to the MPP in the provided I-V curve of the solar panel, a large difference can be seen between the voltage at which the MPP is reached and what power is achieved at the MPP. The most likely reason for the large difference is that the measurements that were done to get the provided I-V curve were performed on a different solar panel than the solar panel that was used for our experiment. As every solar panel has its own I-V curve, the chances that the I-V curve of the measured data perfectly resembles the I-V curve of the solar panel that is used for the experiment is rather small.

5.2 Testing the DC to AC converter

Research question
What does the waveform of the H-bridge look like?

Hypothesis
We expect the waveform to be a Modified Sine Wave as shown in the blue graph in Figure 4 due to the on and off switching of the H-bridge circuit along with the implementation of sPMW signals.

Experimental setup
The input and output signals will be measured using an oscilloscope. We set $V_{gate} = 12$ V and $V_{load} = 15$ V.

Component	Name	Amount
N-channel MOSFET	STP16NF06	4
Diode, fast switching	1n4148	2
Half-Bridge driver	IR2109	2
Controller	ESP32	1

Table 2: Equipment used in experiments

Presentation
A screenshot of the oscilloscope output of the DC to AC H-bridge circuit in the form of the modified square wave is given in Figure 15 below.

Figure 15: The modified square wave output from the H-bridge circuit.

In addition, the resultant square wave output from only low duty cycle and high duty cycle are given in Figure 16 and Figure 17 respectively.

Figure 16: Square wave output from H-bridge circuit with low duty cycle.

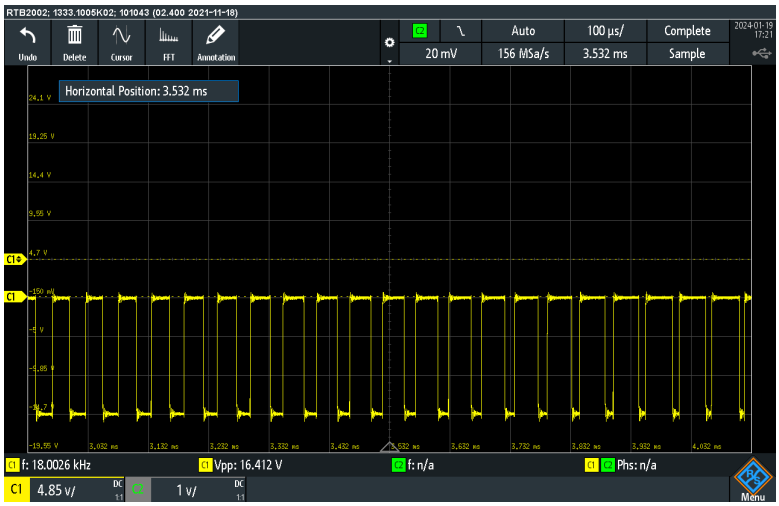

Figure 17: Square wave output from H-bridge circuit with high duty cycle.

Observation
Firstly starting with the impact of the duty cycle on the output square wave, it can be observed that as the duty cycle increases, the duration of the voltage peaks is increased. In other words, the ratio of voltage peaks of the output square wave compared to its minimum for a period is increased by increasing the duty cycle. This can easily be observed by comparing Figure 16 and Figure 17. In Figure 15 it can also clearly be seen that the high side and low side mosfets are turned on and off at the correct time, creating the oscillatory behaviour at roughly 50 Hz.

Discussion

By considering the information gathered in the observation, the resultant modified square wave output in Figure 15 represents an sPWM signal shown in Figure 4 as so desired. This means that the H-bridge functions as intended.

5.3 Testing the filter

To test the characteristics of the applied filter we expect it to filter the 18kHz frequency from the PWM and therefore provide a pure sinewave output similar to the red curve in Figure 4.

Research Question
Does the waveform that is outputted by the filter represent a sine wave?

Hypothesis
We hypothesize that the high-frequency component from the PWM (18kHz) will be sufficiently attenuated with a cutoff frequency of approximately 500 Hz, while the low-frequency component of 50 Hz will be let through.

Experimental Setup
The circuit of the LC low-pass filter as well as the corresponding values for the capacitor and inductor are shown in Figure 11. Moreover, a Fast Fourier Transform (FFT) was also conducted while the filter was connected to the H-bridge to determine the frequency components present. This was done using the oscilloscope. Again, we set $V_{gate} = 12$ V and $V_{load} = 15$ V. The components used for this experiment are listed in table 3.

Component	Value	Measured value	ESR	Amount
Inductor	1mH	$875.33\mu H$	11.61Ω	2
Capacitor	$100\mu F$	$87.29\mu F$	$418.35m\Omega$	2

Table 3: H-Bridge filter values

Presentation
The output of the H-bridge with the filter disconnected is shown in Figure 15. The output of the H-bridge with the filters properly connected is shown in Figure 18. In addition, the FFT of the signal in Figure 18 is shown in Figure 19.

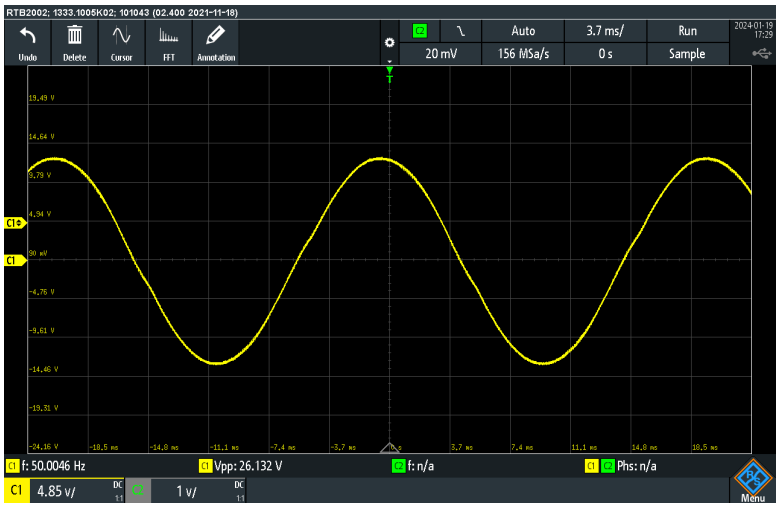

Figure 18: H-Bridge output with filter connected.

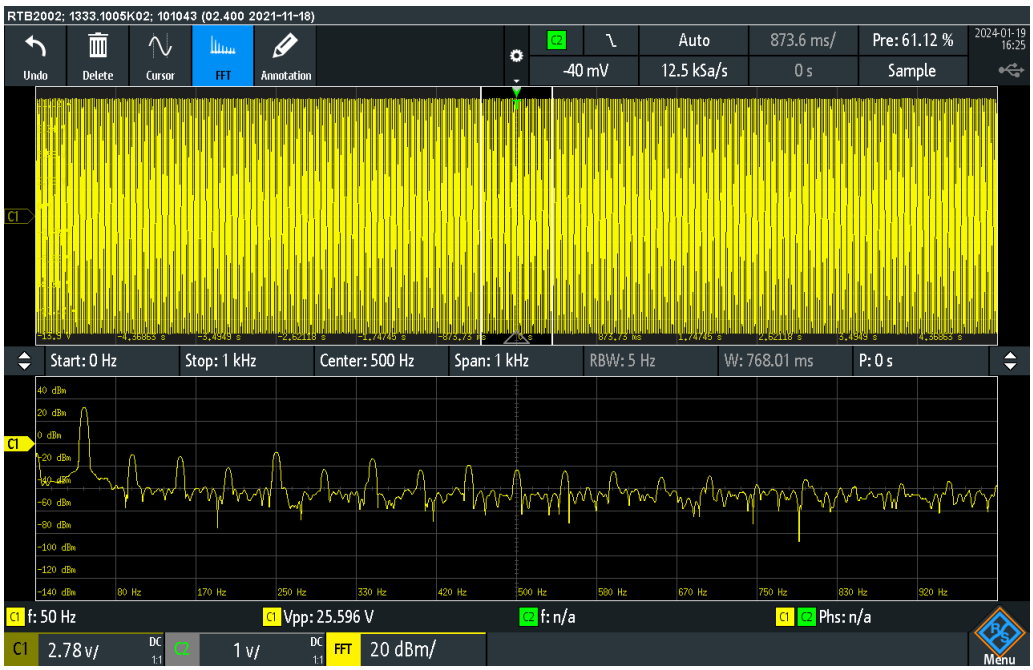

Figure 19: H-Bridge output with filter connected and subjected to a 0Hz to 1kHz Fast Fourier Transform (FFT) sweep using the oscilloscope.

Observation
From Figure 18 we see a pure sine wave when the filter is connected to the output of the H-Bridge circuit. We do not see the 18 kHz component of the PWM signal back into this sine wave. Figure 19 shows the contributing frequencies to the output of the H-Bridge with the filter connected in dBm. The FFT at a frequency of approximately 50 Hz shows a large spike, indicating the big 50 Hz component of the signal.

Discussion
As mentioned in the observation, the 50 Hz component of the signal is let through while the 18 kHz component is attenuated. Therefore, a cutoff frequency of about 500 Hz is enough to successfully attenuate the 18 kHz frequency from the PWM which results in a sine wave with an output frequency of 50 Hz. This matches the hypothesis.

5.4 Testing the phase matching

Research Question
Can we match the frequency of the output with the grid by turning the MOSFET drivers on and off at the right time?

Hypothesis
If the MOSFET drivers are shut down before the grid crosses zero and turned on when the grid has a zero crossing, the phase will be matched at least at the start of the period.

Experimental Setup
The circuit as shown in Figure 12 is added to the H-bridge and filter. The inputs 'grid live' and 'grid neutral' are connected to the output and ground of the function generator respectively. The function generator outputs a $50Hz$ sine wave at $1V_{pp}$ amplitude. Determining when the drivers should shut down and turn on is done by the ESP32. The components used for this experiment are listed in Table 4.

Component	Name	Amount
Operational amplifier	LM324N	1
Comparator	LM319N	1
Controller	ESP32	1
Half-Bridge driver	IR2109	1

Table 4: Equipment used in experiments

Presentation

The output of the differential amplifier compared to the input of the 'grid' is shown in Figure 20. The output of the comparator compared to the input of the 'grid' is shown in Figure 21. When the grid crosses from negative to positive, the comparator has a falling edge. Figure 22 shows the output of the H-bridge with filter after phase synchronisation.

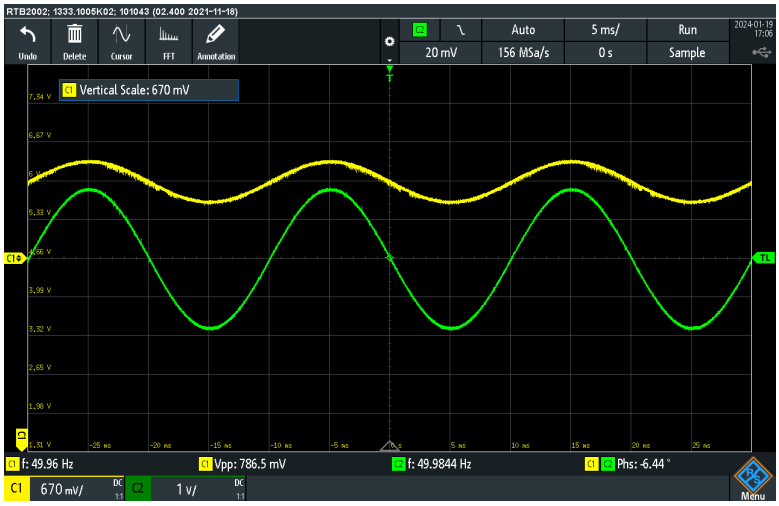

Figure 20: The output of the differential opamp (yellow) compared to the grid (green).

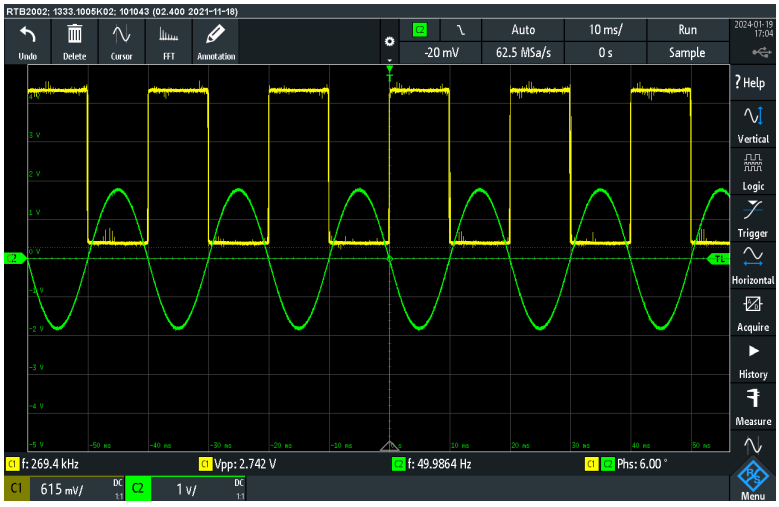

Figure 21: Output of the comparator (yellow) compared to the grid (green).

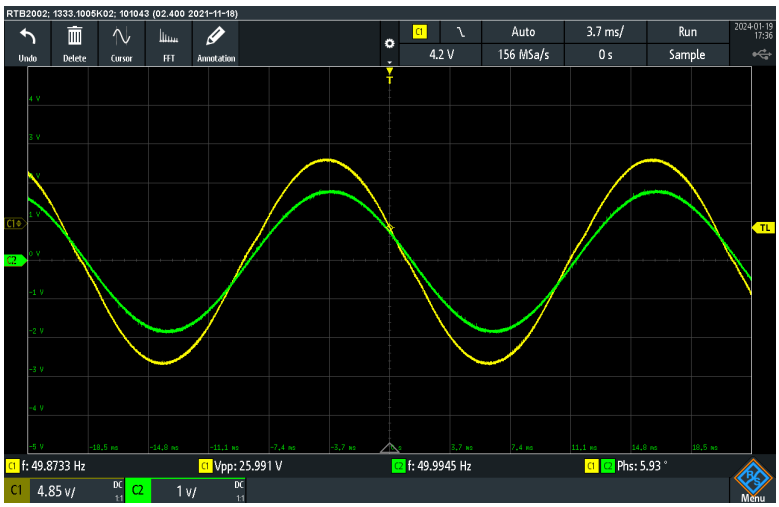

Figure 22: Output of the H-bridge with filter after synchronisation with the grid (yellow) compared to the grid (green). As can be seen in the bottom right corner, the generated sine has a phase lead of 5.93°.

Observation
As can be seen from Figure 22, the generated sine wave has a phase lead of 5.93°. However, in Figure 20 it can be observed that the output of the differential amplifier already contains quite some noise. This translates to undesired falling and rising edges on the output of the comparator. It was attempted to reduce this noise by using decoupling capacitors on every IC's power supply, which helped a little bit.

Discussion
According to Equation 6, the phase lead of 5.93° would give a power flow from the solar panel into the grid. The hypothesis is therefore matched.

6 Discussion of Results

Looking back at the design of the power inverter and the results paragraphs of the experiments, one can see that the individual parts of the solar inverter work. Starting with the boost converter, the boost converter was able to boost to the desired 28 V, and, judging from the algorithm printed on screen, was able to reach the maximum power point, with an efficiency of 95 % and a voltage ripple between 0.5 and 1 V. Secondly, the H-bridge was functional and successful in generating sPWM. Thirdly, the LC filter implemented to filter out frequencies higher than 50 Hz, is successful. What can be seen from the FFT that is made of the filtered signal is that only a high peak is present at 50 Hz. Lastly, a phase lead of 5.93° was achieved with phase matching.

To improve on the phase matching and to make the system more adaptable to frequency changes, an attempt was made to construct a software phase-locked-loop. This was to be achieved by comparing the time when the 360 steps of the internal sine generation were done with the detected falling edge from the comparator. These time measurements were then compared and the PWM frequency was changed accordingly for a lagging or leading phase. A leading phase means the H-bridge runs through the 360 sine steps too quickly and the modulating frequency should be reduced. For a lagging phase, the modulating frequency should be increased.

The way the time was measured in the interrupt produced by the completion of the internal sine or the falling edge was by using the Arduino 'micros();' function. The time difference was then plotted in the serial monitor of the Arduino IDE. The resulting time differences were not as to be expected and thus the modulating frequency was not correctly adapted. We are unsure of what caused this, but it might be due to the way interrupts interact with the 'get_time' function of the ESP. Altogether we were not able to implement the full solar power inverter, although the individual parts of the solar power inverter were functional, linking each part together opposed problems. The reason for this was a soldering mistake in the H-bridge that was not discovered before, which created a short somewhere, and thus resulted in burnt components. This mistake was discovered at such short notice before the end of the project that it could not be resolved anymore. Points for improvement regarding the MPPT would be optimizing the increment size and using a driver to drive the MOSFET of the boost converter instead of feeding the gate directly from the output pins of the microcontroller.

7 Conclusion

Due to its many flaws, the world is slowly moving away from fossil fuel-powered electricity. A good alternative is the use of PV solar energy through the use of solar panels. These solar panels give a DC voltage, while the grid uses AC voltage. Therefore, the DC voltage output of a solar panel has to be converted to an AC voltage to supply it to the grid. This project considered the design, building and testing of such an AC-to-DC converter.

The AC-to-DC converter had certain requirements. First of all, MPPT had to be implemented to optimize the power transfer from the solar panels. Secondly, the actual inverting of the DC signal to an AC signal had to be realised. After that, filtering of the signal was necessary before it could be supplied to the grid. Lastly, it was important to synchronize the generated signal with the grid.

Literature showed various methods to achieve each of the requirements. For the inverting of the DC-to-AC signal, an H-bridge could be used. One could either switch the MOSFETs in the H-bridge at a frequency of 50 Hz to generate a block wave which could then be filtered, or use a PWM signal. Both options would use the same low-pass filter, but the latter option would be harder to implement because it would require a PWM signal of which the duty cycle could be changed. This is however not a problem when a controller would be used to generate such a signal, and literature showed that the use of such a controller would also allow a fast and easy way of synchronizing the generated AC voltage with the grid.

In the literature on MPPT, several algorithms were presented for tracking. The two most important direct algorithms written in the literature review are Preturb&Observe and Incremental inductance. For the solar inverter in this report, the Perturb&Observe algorithm is chosen as it was easier to implement. This is done by using a boost converter. It was calculated that this boost converter had to boost the voltage to 28 V. The values for the capacitors and inductors in the boost converter were chosen from the desired voltage ripple and the frequency of the PWM signal. In addition, a code was written for the ESP32 to control the boost converter, which needs measurements from the INA226 power meter.

For our design, we have chosen to use an H-bridge with a low-pass LC filter to generate the AC voltage. In addition, we have chosen to use a controller to sense the grid voltage, which would generate a PWM signal with a changing duty cycle such that the AC voltage generated by the H-bridge is synchronous with the grid. By using a differential Op-amp and a comparator, the grid's zero crossing could be measured, which would give an interrupt to the ESP32. Upon this interrupt, the ESP32 would turn on the H-bridge to generate one period of a sine wave, after which the H-bridge would be turned off again to measure the grid's zero crossing.

Several experiments were done to test all the components. The boost converter was able to boost to a voltage of 28 V and track the MPP, but the voltage rippler was higher than the expected 0.25 V. Also, the location of the MPP was different than indicated in the datasheet of the solar panel, however the efficiency was higher than expected. The H-bridge and filter were working as expected, and the output was a nice sine wave at a frequency of 50 Hz. In addition, phase matching of the generated sine wave with the grid was achieved, with a slight phase lead of 5.93°. This was unfortunately not reached with phase-locked-loop, because the time differences were not what was to be expected. The reason behind this was unfortunately not found.

In conclusion, the individual components of the power inverter work, but the inverter as a whole did not function. The boost converter boosted the voltage from the solar panel to the required voltage and was also able to track the MPP. In addition, the H-bridge with filter on a breadboard generated a nice sine wave at 50 Hz, and the controller was able to match the phase of this signal with the grid. However, adding the boost converter to the soldered DC/AC converter did not give any output because of mistakes in the soldering. Because of time limitations, this could not be resolved in time.

References

[1] S. B. Kjaer, J. K. Pedersen, and F. Blaabjerg, "A review of single-phase grid-connected inverters for photovoltaic modules," *IEEE Transactions on Industry Applications*, vol. 41, no. 5, p. 1292 – 1306, 2005. Cited by: 3122.

[2] T. Huld, R. Gottschalg, H. G. Beyer, and M. Topič, "Mapping the performance of pv modules, effects of module type and data averaging," *Solar Energy*, vol. 84, no. 2, pp. 324–338, 2010.

[3] K. Zhang, T. Ye, Z. Yan, B. Song, and A. P. Hu, "Obtaining maximum efficiency of inductive power-transfer system by impedance matching based on boost converter," *IEEE Transactions on Transportation Electrification*, vol. 6, no. 2, pp. 488–496, 2020.

[4] S. Motahhir, A. El Hammoumi, and A. El Ghzizal, "The most used mppt algorithms: Review and the suitable low-cost embedded board for each algorithm," *Journal of cleaner production*, vol. 246, p. 118983, 2020.

[5] M. A. Eltawil and Z. Zhao, "Mppt techniques for photovoltaic applications," *Renewable and sustainable energy reviews*, vol. 25, pp. 793–813, 2013.

[6] I. Yahyaoui, *Advances in renewable energies and power technologies: volume 1: solar and wind energies*. Elsevier, 2018.

[7] J. J. Nedumgatt, K. B. Jayakrishnan, S. Umashankar, D. Vijayakumar, and D. P. Kothari, "Perturb and observe mppt algorithm for solar pv systems-modeling and simulation," in *2011 Annual IEEE India Conference*, pp. 1–6, 2011.

[8] D. Sera, L. Mathe, T. Kerekes, S. V. Spataru, and R. Teodorescu, "On the perturb-and-observe and incremental conductance mppt methods for pv systems," *IEEE Journal of Photovoltaics*, vol. 3, no. 3, pp. 1070–1078, 2013.

[9] B. Schulz, "H-bridge basics." `https://www.youtube.com/watch?v=l4Ty92JittY&t=76s`, 05/06/2015.

[10] J. Bone, "Pv inverter: The h bridge." `https://www.youtube.com/watch?v=7Rl7LgiZpqs&t=17s/`, 13/12/2014.

[11] M. Vujacic, M. Hammami, M. Srndović, and G. Grandi, "Theoretical and experimental investigation of switching ripple in the dc-link voltage of single-phase h-bridge pwm inverters," *Energies*, vol. 10, p. 1189, 08 2017.

[12] Y. Beck, B. Bishara, and D. Medini, "Connecting an alternative energy source to the power grid by a dsp controlled dc/ac inverter," in *2005 IEEE Power Engineering Society Inaugural Conference and Exposition in Africa*, pp. 120–124, IEEE, 2005.

[13] "FL 24/12 - Safety isolating transformer — block.eu." `https://www.block.eu/en_EN/productversion/fl-2412/`. [Accessed 15-01-2024].

[14] University of Twente, *Solar cell analysis*, 1 2024.

[15] S. L. V. P. Hartman T., Laanstra G.J., *Pre-Prejoct Manual Module 2*, November 2023.

[16] Z. Ivanovic, B. Blanusa, and M. Knezic, "Power loss model for efficiency improvement of boost converter," in *2011 XXIII International Symposium on Information, Communication and Automation Technologies*, pp. 1–6, 2011.

A Code

The following code was used to control the H-bridge and to attempt phase matching.

```
/**
 * @file      esp32-self-triggered-irq-pwm-18KHz-potentiometer-pushpull.ino
 * @author    cybernetic-research
 * @brief     A simple open loop SPWM generator intended to be used to
 *            generate ac at 50 or 60Hz. This program outputs SPWM at 18KHz for
 *            50hz, or 21600Hz for 60Hz. the PWM output is wired back into the
 *            chip as a rising edge interrupt allowing a new PWM value to be
 *            output every edge transition. An ADC is used to read a
 *            potentiometer allowing the PWM output voltage to be raised or
 *            lowered (because we scale the sine lookup table).
 *            This program is meant to be used on ESP32 boards
 *
 *            This version is for a push pull configuration with centre tapped
 *            transformers
 *
 * @version   0.1
 * @date      2020-09-19
 *
 * @copyright Copyright (c) 2020
 *
 */
#include <stdint.h>
//pinouts here: https://www.bing.com/images/search?view=detailV2&ccid=L%2bau9S29&id=5019621DC7350272217DF7C36EE9AD869B1CE64D&thid=OIP.L-au9S29i6JYHjCKPQ5WzQHaEk&mediaurl=https%3a%2f%2flastminuteengineers.com%2fwp-content%2fuploads%2f2018%2f08%2fESP32-Development-Board-Pinout.jpg&exph=468&expw=758&q=esp32+arduino+pinout&simid=607997902785809138&ck=FCAD5061950FE300E30F4A7416EE3687&selectedIndex=1&FORM=IRPRST&ajaxhist=0

int32_t     potValue    =   0;  // variable for storing the potentiometer value
int32_t     brightness  =   0;  // how bright the LED is
uint8_t     pwm         =   1;

#define PUSHPULL_DRIVE_1    33
#define PUSHPULL_DRIVE_2    25
#define SELF_TRIGGERING_IRQ 0
#define DRIVE1_PWM          1
#define DRIVE2_PWM          2

const byte  led_gpio    =   32;       // the PWM pin the LED is attached to
const int   potPin      =   27;       // Potentiometer is connected to GPIO 27 (Analog ADC1_CH6)
int32_t PWM_FEEDBACK_PIN =  18;       // connect a wire from GPIO 18 to GPIO 32 !!!!!!

//Add the input pin for the measurement of the zero-crossing of the grid using an optocoupler.

int32_t OPTOCOUPLER     = 14;         // Connect the optocoupler to this pin (you can change the pin number in the code if you want).

//Add the output pin to shut down the mosfets drivers (you can change the pin number in the code if you want).

#define SD          27

//8-bit representation of a sinewave scaled to 0->255
```

```
48
49 #define FULL_BRIDGE             1       // change this to "1" if you want to
       drive a full bridge
50
51 //360 step
52 #define MODULATING_FREQ         18000   // <-- 18KHz is 360 x 50 Hz for American
        60Hz use number 21600 instead
53 #define MAX_STEPS               360     //  one pwm update per degree of mains
       sine
54 #define TRANSISTOR_SWITCH_STEP  179     // when to use other transistor bank
55
56
57
58 #if FULL_BRIDGE
59 ///
60 /// Full Bridge Sine Wave Table
61 ///
62 int32_t sinetable[]=
63 {
64 0 ,
65 4 ,
66 8 ,
67 13  ,
68 17  ,
69 22  ,
70 26  ,
71 31  ,
72 35  ,
73 39  ,
74 44  ,
75 48  ,
76 53  ,
77 57  ,
78 61  ,
79 65  ,
80 70  ,
81 74  ,
82 78  ,
83 83  ,
84 87  ,
85 91  ,
86 95  ,
87 99  ,
88 103 ,
89 107 ,
90 111 ,
91 115 ,
92 119 ,
93 123 ,
94 127 ,
95 131 ,
96 135 ,
97 138 ,
98 142 ,
99 146 ,
100 149 ,
101 153 ,
102 156 ,
103 160 ,
104 163 ,
105 167 ,
```

```
106 170 ,
107 173 ,
108 177 ,
109 180 ,
110 183 ,
111 186 ,
112 189 ,
113 192 ,
114 195 ,
115 198 ,
116 200 ,
117 203 ,
118 206 ,
119 208 ,
120 211 ,
121 213 ,
122 216 ,
123 218 ,
124 220 ,
125 223 ,
126 225 ,
127 227 ,
128 229 ,
129 231 ,
130 232 ,
131 234 ,
132 236 ,
133 238 ,
134 239 ,
135 241 ,
136 242 ,
137 243 ,
138 245 ,
139 246 ,
140 247 ,
141 248 ,
142 249 ,
143 250 ,
144 251 ,
145 251 ,
146 252 ,
147 253 ,
148 253 ,
149 254 ,
150 254 ,
151 254 ,
152 254 ,
153 254 ,
154 255 ,
155 254 ,
156 254 ,
157 254 ,
158 254 ,
159 254 ,
160 253 ,
161 253 ,
162 252 ,
163 251 ,
164 251 ,
165 250 ,
166 249 ,
```

```
248 ,
247 ,
246 ,
245 ,
243 ,
242 ,
241 ,
239 ,
238 ,
236 ,
234 ,
232 ,
231 ,
229 ,
227 ,
225 ,
223 ,
220 ,
218 ,
216 ,
213 ,
211 ,
208 ,
206 ,
203 ,
200 ,
198 ,
195 ,
192 ,
189 ,
186 ,
183 ,
180 ,
177 ,
173 ,
170 ,
167 ,
163 ,
160 ,
156 ,
153 ,
149 ,
146 ,
142 ,
138 ,
135 ,
131 ,
127 ,
123 ,
119 ,
115 ,
111 ,
107 ,
103 ,
99  ,
95  ,
91  ,
87  ,
83  ,
78  ,
74  ,
```

```
228 70  ,
229 65  ,
230 61  ,
231 57  ,
232 53  ,
233 48  ,
234 44  ,
235 39  ,
236 35  ,
237 31  ,
238 26  ,
239 22  ,
240 17  ,
241 13  ,
242 8 ,
243 4 ,
244 0 ,
245 5 ,
246 9 ,
247 14  ,
248 18  ,
249 23  ,
250 27  ,
251 32  ,
252 36  ,
253 40  ,
254 45  ,
255 49  ,
256 54  ,
257 58  ,
258 62  ,
259 66  ,
260 71  ,
261 75  ,
262 79  ,
263 84  ,
264 88  ,
265 92  ,
266 96  ,
267 100 ,
268 104 ,
269 108 ,
270 112 ,
271 116 ,
272 120 ,
273 124 ,
274 128 ,
275 132 ,
276 136 ,
277 139 ,
278 143 ,
279 147 ,
280 150 ,
281 154 ,
282 157 ,
283 161 ,
284 164 ,
285 168 ,
286 171 ,
287 174 ,
288 178 ,
```

```
289 181 ,
290 184 ,
291 187 ,
292 190 ,
293 193 ,
294 196 ,
295 199 ,
296 201 ,
297 204 ,
298 207 ,
299 209 ,
300 212 ,
301 214 ,
302 217 ,
303 219 ,
304 221 ,
305 224 ,
306 226 ,
307 228 ,
308 230 ,
309 232 ,
310 233 ,
311 235 ,
312 237 ,
313 239 ,
314 240 ,
315 242 ,
316 243 ,
317 244 ,
318 246 ,
319 247 ,
320 248 ,
321 249 ,
322 250 ,
323 251 ,
324 252 ,
325 252 ,
326 253 ,
327 254 ,
328 254 ,
329 255 ,
330 255 ,
331 255 ,
332 255 ,
333 255 ,
334 255 ,
335 255 ,
336 255 ,
337 255 ,
338 255 ,
339 255 ,
340 254 ,
341 254 ,
342 253 ,
343 252 ,
344 252 ,
345 251 ,
346 250 ,
347 249 ,
348 248 ,
349 247 ,
```

```
246 ,
244 ,
243 ,
242 ,
240 ,
239 ,
237 ,
235 ,
233 ,
232 ,
230 ,
228 ,
226 ,
224 ,
221 ,
219 ,
217 ,
214 ,
212 ,
209 ,
207 ,
204 ,
201 ,
199 ,
196 ,
193 ,
190 ,
187 ,
184 ,
181 ,
178 ,
174 ,
171 ,
168 ,
164 ,
161 ,
157 ,
154 ,
150 ,
147 ,
143 ,
139 ,
136 ,
132 ,
128 ,
124 ,
120 ,
116 ,
112 ,
108 ,
104 ,
100 ,
96  ,
92  ,
88  ,
84  ,
79  ,
75  ,
71  ,
66  ,
62  ,
```

```
411 58  ,
412 54  ,
413 49  ,
414 45  ,
415 40  ,
416 36  ,
417 32  ,
418 27  ,
419 23  ,
420 18  ,
421 14  ,
422 9 ,
423 5 ,
424 1
425 };
426
427 #else
428 ///
429 /// Half Bridge Sine Wave Table
430 ///
431 int32_t sinetable[]=
432 {
433 128 ,
434 129 ,
435 131 ,
436 133 ,
437 135 ,
438 137 ,
439 139 ,
440 141 ,
441 143 ,
442 145 ,
443 147 ,
444 149 ,
445 151 ,
446 153 ,
447 155 ,
448 157 ,
449 159 ,
450 161 ,
451 163 ,
452 165 ,
453 167 ,
454 168 ,
455 170 ,
456 172 ,
457 174 ,
458 176 ,
459 177 ,
460 179 ,
461 181 ,
462 183 ,
463 185 ,
464 186 ,
465 188 ,
466 190 ,
467 191 ,
468 193 ,
469 195 ,
470 196 ,
471 198 ,
```

```
200 ,
201 ,
203 ,
204 ,
205 ,
207 ,
209 ,
210 ,
211 ,
213 ,
214 ,
215 ,
217 ,
218 ,
219 ,
220 ,
221 ,
222 ,
223 ,
225 ,
226 ,
227 ,
228 ,
229 ,
230 ,
231 ,
231 ,
232 ,
233 ,
234 ,
235 ,
235 ,
236 ,
236 ,
237 ,
238 ,
238 ,
239 ,
239 ,
240 ,
240 ,
240 ,
240 ,
241 ,
241 ,
241 ,
242 ,
242 ,
242 ,
242 ,
242 ,
242 ,
242 ,
242 ,
242 ,
242 ,
241 ,
241 ,
241 ,
240 ,
```

```
240 ,
240 ,
240 ,
239 ,
239 ,
238 ,
238 ,
237 ,
236 ,
236 ,
235 ,
235 ,
234 ,
233 ,
232 ,
231 ,
231 ,
230 ,
229 ,
228 ,
227 ,
226 ,
225 ,
223 ,
222 ,
221 ,
220 ,
219 ,
218 ,
217 ,
215 ,
214 ,
213 ,
211 ,
210 ,
209 ,
207 ,
205 ,
204 ,
203 ,
201 ,
200 ,
198 ,
196 ,
195 ,
193 ,
191 ,
190 ,
188 ,
186 ,
185 ,
183 ,
181 ,
179 ,
177 ,
176 ,
174 ,
172 ,
170 ,
168 ,
167 ,
```

```
165 ,
163 ,
161 ,
159 ,
157 ,
155 ,
153 ,
151 ,
149 ,
147 ,
145 ,
143 ,
141 ,
139 ,
137 ,
135 ,
133 ,
131 ,
129 ,
128 ,
125 ,
123 ,
121 ,
119 ,
117 ,
115 ,
113 ,
111 ,
110 ,
107 ,
105 ,
103 ,
101 ,
100 ,
98  ,
96  ,
94  ,
92  ,
90  ,
88  ,
86  ,
84  ,
83  ,
81  ,
79  ,
77  ,
75  ,
74  ,
72  ,
70  ,
68  ,
66  ,
65  ,
63  ,
61  ,
60  ,
58  ,
57  ,
55  ,
54  ,
52  ,
```

```
655 51  ,
656 49  ,
657 47  ,
658 46  ,
659 45  ,
660 43  ,
661 42  ,
662 41  ,
663 39  ,
664 38  ,
665 37  ,
666 36  ,
667 34  ,
668 33  ,
669 32  ,
670 31  ,
671 30  ,
672 29  ,
673 28  ,
674 27  ,
675 26  ,
676 25  ,
677 24  ,
678 23  ,
679 23  ,
680 22  ,
681 21  ,
682 20  ,
683 20  ,
684 19  ,
685 18  ,
686 18  ,
687 17  ,
688 16  ,
689 16  ,
690 15  ,
691 15  ,
692 15  ,
693 14  ,
694 14  ,
695 14  ,
696 13  ,
697 13  ,
698 13  ,
699 13  ,
700 13  ,
701 13  ,
702 13  ,
703 13  ,
704 13  ,
705 13  ,
706 13  ,
707 13  ,
708 13  ,
709 13  ,
710 13  ,
711 14  ,
712 14  ,
713 14  ,
714 15  ,
715 15  ,
```

```
15  ,
16  ,
16  ,
17  ,
18  ,
18  ,
19  ,
20  ,
20  ,
21  ,
22  ,
23  ,
23  ,
24  ,
25  ,
26  ,
27  ,
28  ,
29  ,
30  ,
31  ,
32  ,
33  ,
34  ,
36  ,
37  ,
38  ,
39  ,
41  ,
42  ,
43  ,
45  ,
46  ,
47  ,
49  ,
51  ,
52  ,
54  ,
55  ,
57  ,
58  ,
60  ,
61  ,
63  ,
65  ,
66  ,
68  ,
70  ,
72  ,
74  ,
75  ,
77  ,
79  ,
81  ,
83  ,
84  ,
86  ,
88  ,
90  ,
92  ,
94  ,
```

```
777 96  ,
778 98  ,
779 100 ,
780 101 ,
781 103 ,
782 105 ,
783 107 ,
784 110 ,
785 111 ,
786 113 ,
787 115 ,
788 117 ,
789 119 ,
790 121 ,
791 123 ,
792 125 ,
793 127
794 };
795 #endif
796
797 uint8_t readPot =0;
798 int32_t ctr=0;
799 int32_t ctr2=180;
800 int32_t val2 = 0;
801
802 // Define the on-off button, which will turn the sine wave generator on or off.
       Off = 0, on = 1.
803 int32_t on_off = 0;
804
805 /**
806  * @brief IRAM_ATTR
807  * An interrupt service routine that runs whenever a rising edge is detected on
808  * the feedback I/O pin. We generate a PWM pulse at 18KHz for 50Hz, or 21.6Khz
809  * for a 60Hz pulse. the PWM output pin is fed back into the chip as an
810  * interrupt and we retrigger from it.
811  * Every time an interrupt is received we load in the next PWM value from the
812  * sine table. this allows each degree of the 50 or 60Hz waveform to have a
813  * separate pulse associated with it
814  */
815 void IRAM_ATTR isr()
816 {
817
818   // Only generate a signal if we may generate a signal
819   if(on_off)
820   {
821
822   ///
823   /// 1. Compute next PWM register value
824   ///
825   int32_t val = sinetable[ctr] ; //scale the sine tablkee
826
827 #if FULL_BRIDGE
828   ///
829   /// Full Bridge
830   /// 2. select which transitor bank to use
831   ///
832   if(ctr>TRANSISTOR_SWITCH_STEP)
833   {
834     ledcWrite(DRIVE1_PWM,   val);                          //signal to mosfet
          gate
```

```
835     ledcWrite(DRIVE2_PWM,   0);                                 //signal to mosfet
            gate
836   }
837   else
838   {
839     ledcWrite(DRIVE1_PWM,   0);                                 //signal to mosfet
            gate
840     ledcWrite(DRIVE2_PWM,   val);                               //signal to mosfet
            gate
841   }
842 #else
843   ///
844   /// Half Bridge
845   ///
846   ledcWrite(DRIVE1_PWM,   val);                                 //signal to mosfet
          gate
847   ledcWrite(DRIVE2_PWM,   0);                                   //signal to mosfet
          gate
848 #endif
849
850   ///
851   /// 3. increment counter
852   ///
853   ctr += 1;
854   if(ctr==MAX_STEPS)
855   {
856     ctr     = 0;
857     readPot = 1;  //allow amplitude to change
858   }
859
860   //If the counter ctr reaches a certain value, we want to switch the H-bridge
          off, so that we can measure the grid again.
861   //This time, we start at a counter value of 10, and continue until a maximum
          of 359.
862   //(360 is not possible because the if-statement before sets the counter to 0
          if it is 360, so it will never enter this if-statement)
863   if(ctr==20)
864   {
865     ctr     = 30;
866     val     = 0;
867     on_off  = 0;
868     digitalWrite(SD, LOW);
869   }
870
871
872   ///
873   /// 4. self triggering interrupt (mandatory, it just has to have some non-zero
          value)
874   ///
875   ledcWrite(SELF_TRIGGERING_IRQ, 1); // set the brightness of the LED
876
877   }
878
879 }
880
881
882 // Define the function that does something on the optocoupler interrupt.
883 void IRAM_ATTR isr_optocoupler()
884 {
885   // When we get an interrupt, so when the optocoupler has a rising edge, we
          want to turn on our system.
```

```
886   on_off = 1;
887   digitalWrite(SD, HIGH);
888
889 }
890
891
892
893
894
895
896 int32_t t = 0;
897 int32_t oldPotValue = -1;
898
899 const int numReadings = 10;
900
901 int readings[numReadings];      // the readings from the analog input
902 int readIndex = 0;              // the index of the current reading
903 int total = 0;                  // the running total
904 int average = 0;                // the average
905 int pV = 0;
906
907
908 /**
909  * @brief setup
910  * the setup routine runs once when you press reset:
911  */
912 void setup() {
913
914
915   // Initialize channels
916   // channels 0-15, resolution 1-16 bits, freq limits depend on resolution
917   // ledcSetup(uint8_t channel, uint32_t freq, uint8_t resolution_bits);
918   ledcSetup(SELF_TRIGGERING_IRQ,  MODULATING_FREQ, 8);                 // 18 kHz
          PWM, 8-bit resolution
919   ledcSetup(DRIVE1_PWM,           MODULATING_FREQ, 8);                 // 18 kHz
          PWM, 8-bit resolution
920   ledcSetup(DRIVE2_PWM,           MODULATING_FREQ, 8);                 // 18 kHz
          PWM, 8-bit resolution
921
922
923   ledcAttachPin(led_gpio,         SELF_TRIGGERING_IRQ); // assign a led pins to
         a channel
924   ledcAttachPin(PUSHPULL_DRIVE_1, DRIVE1_PWM);
925   ledcAttachPin(PUSHPULL_DRIVE_2, DRIVE2_PWM);
926
927
928   pinMode(PWM_FEEDBACK_PIN, INPUT_PULLDOWN);
929   attachInterrupt(PWM_FEEDBACK_PIN, isr, RISING);           //this is wired to
         the PWM output
930
931   ledcWrite(SELF_TRIGGERING_IRQ,  pwm);
         //signal to mosfet gate
932   ledcWrite(DRIVE1_PWM,           127);                              //signal to
         mosfet gate
933   ledcWrite(DRIVE2_PWM,           127);                              //signal to
         mosfet gate
934   Serial.begin(115200);
935
936   for (int thisReading = 0; thisReading < numReadings; thisReading++) {
937     readings[thisReading] = 0;
938   }
```

```
939
940   // Add the interupt for the optocoupler. The interrupt will be when the
          optocoupler has a rising edge.
941   pinMode(OPTOCOUPLER, INPUT_PULLDOWN);
942   attachInterrupt(OPTOCOUPLER, isr_optocoupler, FALLING);
943
944   // Add the shutdown pin
945   pinMode(SD, OUTPUT);
946   digitalWrite(SD, LOW);
947
948 }
949
950
951
952 const int CAP_ADC = 255;///176;
953
954 /**
955  * @brief loop
956  * Shall read ADC value and allow user to modify PWM amplitude by means of
957  * potentiometer. We scale the SPWM in fact, allowing a synthesised lower
          voltage
958  */
959 void loop() {
960
961   // Only run this loop if we may generate a signal
962   if(on_off)
963   {
964
965   if(readPot)
966   {
967
968     t+=1;
969     if(20==t)
970     {
971        readPot = 0;
972     int p = analogRead(potPin);  //https://randomnerdtutorials.com/esp32-adc-
           analog-read-arduino-ide/
973       pV = p/16;
974 // subtract the last reading:
975   total = total - readings[readIndex];
976   // read from the sensor:
977   readings[readIndex] = pV;
978   // add the reading to the total:
979   total = total + readings[readIndex];
980   // advance to the next position in the array:
981   readIndex = readIndex + 1;
982
983   // if we're at the end of the array...
984   if (readIndex >= numReadings) {
985     // ...wrap around to the beginning:
986     readIndex = 0;
987   }
988
989   // calculate the average:
990   average = total / numReadings;
991
992
993       potValue = average;
994       potValue = 255;
995       if(potValue>CAP_ADC)
996       {
```

```
997           potValue=CAP_ADC;
998         }
999         if(oldPotValue!=potValue)
1000        {
1001          Serial.printf("ADCM␣%u\n", potValue);
1002          oldPotValue=potValue;
1003        }
1004       t=0;
1005
1006      if(potValue<1)
1007      {
1008        potValue = 0;
1009      }
1010      if(potValue>255)
1011      {
1012        potValue = 255;
1013      }
1014      }
1015
1016    }
1017
1018    }
1019
1020 }
```

The code hereafter was used to control the boost converter for maximum power point tracking. The used algorithm is 'perturb and observe'.

```
1  const int ledchannel = 0;
2  const int resolution = 8;
3  const int output_voltage_pin = 4;
4
5  const float initial_guess = 0.1;
6  double Previous_D;
7  double Previous_Power = 0;
8  double current_BusVoltage = 0;
9  double current_ShuntVoltage = 0;
10 const double increment = 0.005;
11 double action;
12
13 const int pwm_pin = 5;
14 const double ShuntResistance = 0.1;
15 const double pwm_frequency = 100000.0; //Hz
16 double dt = 1/pwm_frequency;
17 double D; //Duty cycle initial guess
18 double delay_on;
19 double delay_off;
20 int calc_counter = 0;
21
22 /*
23     INA226 Bi-directional Current/Power Monitor. Simple Example.
24     Read more: http://www.jarzebski.pl/arduino/czujniki-i-sensory/cyfrowy-
       czujnik-pradu-mocy-ina226.html
25     GIT: https://github.com/jarzebski/Arduino-INA226
26     Web: http://www.jarzebski.pl
27     (c) 2014 by Korneliusz Jarzebski
28 */
29
30 #include <Wire.h>
31 #include <INA226.h>
32
33 INA226 ina;
```

```
34
35 void checkConfig()
36 {
37   Serial.print("Mode:                  ");
38   switch (ina.getMode())
39   {
40     case INA226_MODE_POWER_DOWN:      Serial.println("Power-Down"); break;
41     case INA226_MODE_SHUNT_TRIG:      Serial.println("Shunt Voltage, Triggered")
         ; break;
42     case INA226_MODE_BUS_TRIG:        Serial.println("Bus Voltage, Triggered");
        break;
43     case INA226_MODE_SHUNT_BUS_TRIG:  Serial.println("Shunt and Bus, Triggered")
         ; break;
44     case INA226_MODE_ADC_OFF:         Serial.println("ADC Off"); break;
45     case INA226_MODE_SHUNT_CONT:      Serial.println("Shunt Voltage, Continuous"
         ); break;
46     case INA226_MODE_BUS_CONT:        Serial.println("Bus Voltage, Continuous");
         break;
47     case INA226_MODE_SHUNT_BUS_CONT:  Serial.println("Shunt and Bus, Continuous"
         ); break;
48     default: Serial.println("unknown");
49   }
50
51   Serial.print("Samples average:       ");
52   switch (ina.getAverages())
53   {
54     case INA226_AVERAGES_1:           Serial.println("1 sample"); break;
55     case INA226_AVERAGES_4:           Serial.println("4 samples"); break;
56     case INA226_AVERAGES_16:          Serial.println("16 samples"); break;
57     case INA226_AVERAGES_64:          Serial.println("64 samples"); break;
58     case INA226_AVERAGES_128:         Serial.println("128 samples"); break;
59     case INA226_AVERAGES_256:         Serial.println("256 samples"); break;
60     case INA226_AVERAGES_512:         Serial.println("512 samples"); break;
61     case INA226_AVERAGES_1024:        Serial.println("1024 samples"); break;
62     default: Serial.println("unknown");
63   }
64
65   Serial.print("Bus conversion time:   ");
66   switch (ina.getBusConversionTime())
67   {
68     case INA226_BUS_CONV_TIME_140US:  Serial.println("140uS"); break;
69     case INA226_BUS_CONV_TIME_204US:  Serial.println("204uS"); break;
70     case INA226_BUS_CONV_TIME_332US:  Serial.println("332uS"); break;
71     case INA226_BUS_CONV_TIME_588US:  Serial.println("558uS"); break;
72     case INA226_BUS_CONV_TIME_1100US: Serial.println("1.100ms"); break;
73     case INA226_BUS_CONV_TIME_2116US: Serial.println("2.116ms"); break;
74     case INA226_BUS_CONV_TIME_4156US: Serial.println("4.156ms"); break;
75     case INA226_BUS_CONV_TIME_8244US: Serial.println("8.244ms"); break;
76     default: Serial.println("unknown");
77   }
78
79   Serial.print("Shunt conversion time: ");
80   switch (ina.getShuntConversionTime())
81   {
82     case INA226_SHUNT_CONV_TIME_140US:  Serial.println("140uS"); break;
83     case INA226_SHUNT_CONV_TIME_204US:  Serial.println("204uS"); break;
84     case INA226_SHUNT_CONV_TIME_332US:  Serial.println("332uS"); break;
85     case INA226_SHUNT_CONV_TIME_588US:  Serial.println("558uS"); break;
86     case INA226_SHUNT_CONV_TIME_1100US: Serial.println("1.100ms"); break;
87     case INA226_SHUNT_CONV_TIME_2116US: Serial.println("2.116ms"); break;
88     case INA226_SHUNT_CONV_TIME_4156US: Serial.println("4.156ms"); break;
```

```
89      case INA226_SHUNT_CONV_TIME_8244US: Serial.println("8.244ms"); break;
90      default: Serial.println("unknown");
91    }
92
93    Serial.print("Max possible current:  ");
94    Serial.print(ina.getMaxPossibleCurrent());
95    Serial.println(" A");
96
97    Serial.print("Max current:           ");
98    Serial.print(ina.getMaxCurrent());
99    Serial.println(" A");
100
101   Serial.print("Max shunt voltage:     ");
102   Serial.print(ina.getMaxShuntVoltage());
103   Serial.println(" V");
104
105   Serial.print("Max power:             ");
106   Serial.print(ina.getMaxPower());
107   Serial.println(" W");
108 }
109
110 void setup()
111 {
112   pinMode(pwm_pin, OUTPUT);
113   Previous_D = initial_guess;
114   Serial.begin(115200);
115
116   Serial.println("Initialize INA226");
117   Serial.println("-----------------------------------------------");
118
119   // Default INA226 address is 0x40
120   ina.begin(69);
121
122   // Configure INA226
123   ina.configure(INA226_AVERAGES_1, INA226_BUS_CONV_TIME_1100US,
          INA226_SHUNT_CONV_TIME_1100US, INA226_MODE_SHUNT_BUS_CONT);
124
125   // Calibrate INA226. Rshunt = 0.01 ohm, Max excepted current = 4A
126   ina.calibrate(0.1, 1);
127
128   // Display configuration
129   checkConfig();
130
131   Serial.println("-----------------------------------------------");
132
133   ledcSetup(ledchannel,pwm_frequency, resolution);
134   ledcAttachPin(pwm_pin,ledchannel);
135 }
136
137 void loop()
138 {
139
140
141   if(calc_counter != 80000){
142     calc_counter  = calc_counter +1;
143   }
144   else{
145     calc_counter = 0;
146   }
147
148
```

```
149  if(calc_counter == 0){
150    double BusVoltage = ina.readBusVoltage();
151    double ShuntVoltage = ina.readShuntVoltage();
152    double BusCurrent = ShuntVoltage/ShuntResistance;
153    double Power = BusCurrent*BusVoltage;
154    double output_voltage;
155    double output_voltage_boosted;
156
157    output_voltage_boosted = analogRead(output_voltage_pin)*(3.3/4095)
           *(1/0.047619)*1.177;
158
159    if(output_voltage_boosted<28){
160      D = Previous_D + increment;
161      Previous_D = D;
162    }
163
164    if(BusVoltage<15){
165      D = 0.1;
166    }
167    if(output_voltage_boosted>30){
168      Serial.print("Boosted␣output␣voltage␣higher␣than␣30␣V");
169      Serial.println();
170      Serial.print("␣The␣boosted␣output␣voltage␣is:␣");
171      Serial.print(output_voltage_boosted);
172      Serial.print("␣V");
173      Serial.println();
174      D = Previous_D - increment;
175      Previous_D = D;
176    }
177    else{
178      Serial.print("Bus␣voltage:␣␣␣");
179      Serial.print(BusVoltage, 5);
180      Serial.println("␣V");
181
182
183      Serial.print("Shunt␣voltage:␣");
184      Serial.print(ShuntVoltage, 5); //*22.26
185      Serial.println("␣V");
186
187      Serial.print("Power:␣");
188      Serial.print(Power, 5);
189      Serial.print("␣W");
190
191      Serial.println();
192      Serial.print("Duty␣cycle:␣␣");
193      Serial.print(D,3);
194
195      Serial.println();
196      Serial.print("Boosted␣output␣voltage:␣");
197      Serial.print(output_voltage_boosted,5);
198      Serial.print("␣V");
199
200      Serial.println();
201
202      Serial.println("");
203
204        if(Power > Previous_Power && action > 0){
205          D = Previous_D + increment;
206          action = increment;
207        }
208        else if (Power > Previous_Power && action < 0) {
```

```
209           D = Previous_D - increment;
210           action = -increment;
211         }
212         else if (Power < Previous_Power && action > 0) {
213           D = Previous_D - increment;
214           action = -increment;
215         }
216         else {
217           D = Previous_D + increment;
218           action = increment;
219         }
220         Previous_Power = Power ;
221         if (D>0.85) D = 0.85;
222         else if (D<0.107) D = 0.107;
223         Previous_D = D;
224
225   }
226     }
227
228   ledcWrite(ledchannel,D*225);
229 }
```