eta





**Tutorial TWIST-SPIN board**

**Photovoltaic Panels**

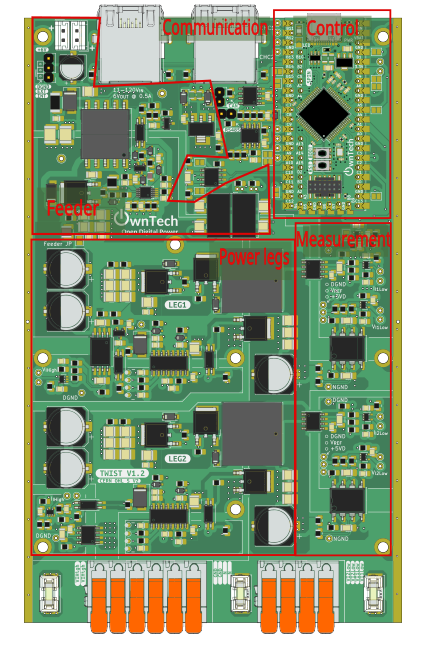


Table of contents

[Introduction 3](#_Toc200021937)

[Buck Converter: Solar Pumping with Fractional Open Circuit Voltage 4](#_Toc200021938)

[Objectif 4](#_Toc200021939)

[Required hardware 4](#_Toc200021940)

[Required Software 4](#_Toc200021941)

[Step-by-Step Implementation 4](#_Toc200021942)

[Code explanation 8](#_Toc200021943)

[Connect Hardware 8](#_Toc200021944)

[Buck Converter MPPT : Constant-Step Perturb and Observe 10](#_Toc200021945)

[Tutorial Objective 10](#_Toc200021946)

[Step-by-Step Implementation 10](#_Toc200021947)

[Code explanation 14](#_Toc200021948)

[Connect Hardware 16](#_Toc200021949)

# Introduction

The goal of this tutorial is to start using the OnwTech TWIST boards for Photovoltaic panels applications. Please refer to the “Tutorial\_Twist\_Board” for learning how to use the Twist board before starting this new series of tutorials.

# Buck Converter: Solar Pumping with Fractional Open Circuit Voltage

## Objectif

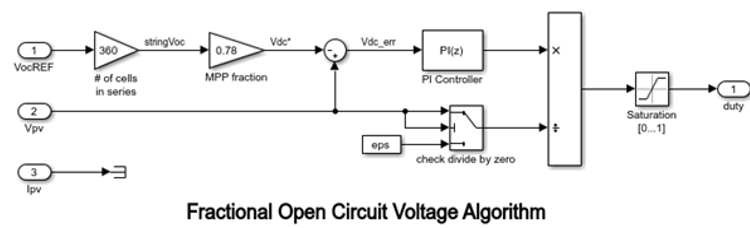
The goal of this tutorial is to retrieve the open-circuit voltage (Voc) of a photovoltaic panel when connected to a Buck converter in open-circuit mode and, then, calculate the maximum power point voltage (Vmp) using the relation:

**Vmp = 0.78 × Voc**

Finally, we will impose this voltage at the converter input using a PID control algorithm in closed-loop Buck mode.

The PID algorithm will be used to dynamically adjust the duty cycle in order to maintain the calculated Vmp. This allows the system to adapt to changes in the input voltage and operate optimally.

The algorithm below shows that the target voltage Vmp equals 78% of the input open-circuit voltage.



## Required hardware

* TWIST v\_1\_3 or later
* 64-bit PC (Windows or Linux)
* Photovoltaic panel
* Water pump (motopump)
* Oscilloscope or multimeter

## Required Software

* Git
* Visual studio code

## Step-by-Step Implementation

1. In the enum serial\_interface\_menu\_mode, we define MPPT\_A and MPPT\_I modes:

enum serial\_interface\_menu\_mode //LIST OF POSSIBLE MODES FOR THE OWNTECH CONVERTER

{

    IDLEMODE =0,

    MPPTMODE\_A,

    POWERMODE,

    MPPTMODE\_I,

};

Also set the control task period to 100 microseconds:

static uint32\_t control\_task\_period = 100; //[us] period of the control task

In the *User Variable Declarations* section, initialize the variables that will be used for automatic mode switching:

//--------------USER VARIABLES DECLARATIONS----------------------

static uint32\_t total\_time = 0; // Elapsed time static uint32\_t last\_mode\_change\_time = 0; // Last change mode time

static const uint32\_t mppt\_i\_duration = 100000; // Duration milliseconds (10s) static const uint32\_t mppt\_a\_duration = 300000; // Duration milliseconds (30s)

We also initialize variables Vco and Vmp corresponding to the Open -circuit voltage and the Maximum power point voltage

static float32\_t Vco;

static float32\_t Vmp;

Add the MPPTMODE\_A and MPPTMODE\_I modes handling to the loop\_communication\_task:

   case 'h':

            //----------SERIAL INTERFACE MENU-----------------------

            printk(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

            printk("|     ------- MENU ---------             |\n");

            printk("|     press i : idle mode                |\n");

            printk("|     press m : inactive mppt mode       |\n");

            printk("|     press p : power mode               |\n");

            printk("|     press u : duty cycle UP            |\n");

            printk("|     press d : duty cycle DOWN          |\n");

            printk("|     press a : active mppt mode         |\n");

            printk("|     press s : save values to file      |\n");

            printk("|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n\n");

            //------------------------------------------------------

            break;

case 'i':

            printk("idle mode\n");

            mode = IDLEMODE;

            break;

        case 'm':

            printk("inactive mode mppt\n");

            mode = MPPTMODE\_I;

            break;

        case 'p':

            printk("power mode\n");

            mode = POWERMODE;

            break;

        case 'u':

            printk("duty cycle UP: %f\n", duty\_cycle);

            duty\_cycle = duty\_cycle + duty\_cycle\_step;

            if(duty\_cycle>1.0) duty\_cycle = 1.0;

            break;

        case 'd':

            printk("duty cycle DOWN: %f\n", duty\_cycle);

            duty\_cycle = duty\_cycle - duty\_cycle\_step;

            if(duty\_cycle<0.0) duty\_cycle = 0.0;

            break;

        case 'a':

            printk("active mode mppt\n");

            mode = MPPTMODE\_A;

            break;

In loop\_application\_task we allow mode switching between MPPT\_A and MPPT\_I and compute Vco and Vmp:

        if (mode == MPPTMODE\_I && (total\_time - last\_mode\_change\_time) >= mppt\_i\_duration) {

         mode = MPPTMODE\_A;

printk("MPPT mode actif\n");

         last\_mode\_change\_time = total\_time;

        } else if (mode == MPPTMODE\_A && (total\_time - last\_mode\_change\_time) >= mppt\_a\_duration) {

            mode = MPPTMODE\_I;

            printk("MPPT mode inactif\n");

            last\_mode\_change\_time = total\_time;

        }

    if(mode==IDLEMODE) {

        Vco=V\_high\_value;

        spin.led.turnOff();

    }else if(mode==MPPTMODE\_I) {

        Vco=V\_high\_value;

        spin.led.turnOff();

    }else if(mode==POWERMODE) {

        spin.led.toggle();

    }else if(mode==MPPTMODE\_A) {

        spin.led.turnOn();

        cpt\_step ++;

    }

    printk("%f:", Vco);

    printk("%f\n", Vmp);

    Vmp= 0.78\*Vco;

}

In loop\_control\_task function, we assign the elapsed time since the beginning of the program to variable total\_time. Finally, we enforce Vmp by changing the duty cycle with a PID controller.

void loop\_control\_task()

{

static uint32\_t tick\_count = 0;

    tick\_count++;

    // Increment elapsed time every 100 microsecond (assuming loop\_control\_task runs every 100 microsecond)

    if (tick\_count >= control\_task\_period) {

        total\_time += tick\_count;

        tick\_count = 0;

    }

if(mode==IDLEMODE || mode==MPPTMODE\_I) {

         pwm\_enable = false;

         twist.stopAll();

    }else if(mode==POWERMODE || mode==MPPTMODE\_A) {

        if(!pwm\_enable) {

            pwm\_enable = true;

            twist.startAll();

        }

  if(mode==MPPTMODE\_A){

  duty\_cycle = opalib\_control\_interleaved\_pid\_calculation(Vmp, V\_high\_value);

    }

    //Sends the PWM to the switches

    twist.setAllDutyCycle(duty\_cycle);

    }

}

## Code explanation

1. Voc Retrieval: We retrieve the open-circuit voltage (Voc) from the photovoltaic panel by reading V\_high when the converter is in MPPTMODE\_I (mppt inactive), i.e., when the output behaves as an open-circuit.
2. Vmp Calculation: We compute the maximum power voltage (Vmp) using the formula Vmp = 0.78 × Voc.
3. PID Algorithm: The PID controller dynamically adjusts the duty cycle to maintain Vmp at the buck converter input according to the difference between the real measured voltage and the desired voltage (Vmp). The PID algorithm is only active in MPPTMODE\_A (mppt active).
4. The code switches automatically between active and inactive MPPT modes:

* 30 seconds in active MPPT mode
* 10 seconds in inactive MPPT mode

This allows the system to frequently update the Voc value, which varies throughout

the day due to changing irradiance and solar panel position.

## Connect Hardware

1. For hardware setup, connect your TWIST board to the power supply and the PC via USB-C.

* Connect Vhigh and GND of the TWIST to the photovoltaic panel (set a 2 A current limit).
* Connect VLow1, VLow2, and GND of the TWIST to a motor pump.
* Connect the USB-C port of the SPIN to your PC via a USB cable.

Une image contenant circuit, Ingénierie électronique, Composant électronique, Composant de circuit passif

Le contenu généré par l’IA peut être incorrect.

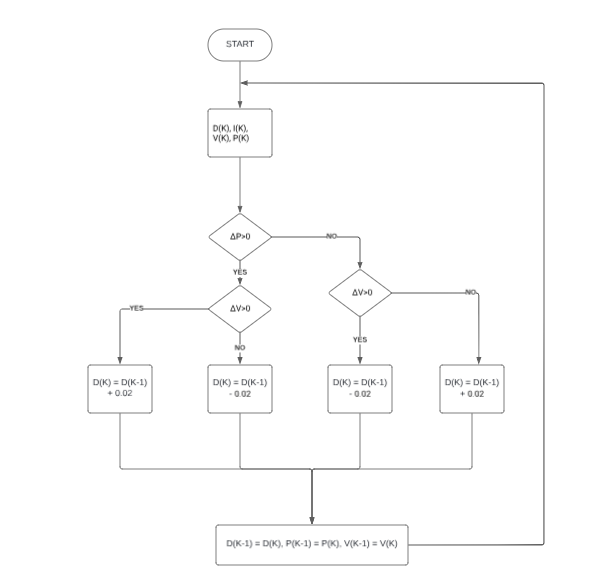
1. Build  build and Upload upload
2. Use OwnPlot or the Serial Monitor to visualize your data and control the converter.

Voilà !

# Buck Converter MPPT : Constant-Step Perturb and Observe

## Tutorial Objective

The goal of this tutorial is to implement a Perturb and Observe (P&O) algorithm for MPPT. This algorithm is designed to track and maintain the optimal operating point of a photovoltaic (PV) panel based on the connected load, such as a pump. By periodically perturbing the duty cycle and observing the resulting power changes, the algorithm adjusts the operating point to maximize the solar system’s energy efficiency, even in the presence of variations in irradiance, temperature, and load demand.



## Step-by-Step Implementation

In the user variable declaration section, we initialize the following variables:

//--------------USER VARIABLES DECLARATIONS-------------------

#define BUFFER\_SIZE 20

static float32\_t bufferI[BUFFER\_SIZE];

static float32\_t Ifinal;

static float32\_t bufferV[BUFFER\_SIZE];

static float32\_t Vfinal=0;

static float32\_t Vprev=0;

static int secuV = 0;

static int bufferp;

static float32\_t Pfinal =0;

static float32\_t Pprev =0;

static float32\_t deltaP;

static float32\_t deltaV;

We define two operating modes in the enum serial\_interface\_menu\_mode: IDLEMODE and POWERMODE. Then, we initialize the converter legs in buck mode.

enum serial\_interface\_menu\_mode // LIST OF POSSIBLE MODES FOR THE OWNTECH CONVERTER

{

    IDLEMODE = 0,

    POWERMODE

};

uint8\_t mode = IDLEMODE;

/\* buck voltage mode \*/

    twist.initAllBuck();

In void loop\_application\_task(), we implement a protection mechanism to prevent overvoltage on the converter output. The pump's nominal voltage is 24 V, so we set an upper limit of 23.9 V. The output voltage must also be no less than 3.1 V.

 case 'u':

            if (V1\_low\_value >= 23.9)  //security for not overvoltage

            {

                duty\_cycle -= 0.02;

                printk("La tension est déjà au maximum\n");

                printk("tension cible : %f\n\n", voltage\_reference);

            }

            else

            {

              printk("duty cycle UP: %f\n", duty\_cycle);

              duty\_cycle = duty\_cycle + duty\_cycle\_step;

              if(duty\_cycle>0.9) duty\_cycle = 0.9;

            }

            break;

        case 'd':

            if (V1\_low\_value <= 3.1)   //security for not undervoltage

                {

                  duty\_cycle += 0.02;

                  printk("La tension est déjà au minimum\n");

                  printk("tension cible : %f\n\n", voltage\_reference);

               }

               else

                {

                  printk("duty cycle DOWN: %f\n", duty\_cycle);

                  duty\_cycle = duty\_cycle - duty\_cycle\_step;

                  if(duty\_cycle<0.1) duty\_cycle = 0.1;

                }

            break;

Add the following code in void loop\_application\_task() to implement the P&O MPPT algorithm. It adjusts the converter’s duty cycle based on power and voltage variations to maximize PV system efficiency.

bufferI[bufferp] = I1\_low\_value;

bufferV[bufferp] = (V1\_low\_value <= 0) ? 0 : V1\_low\_value;

bufferp += 1;

if (bufferp == BUFFER\_SIZE)

{

   bufferp = 0;

    secuV = 0;

    for (int bufferp = 0; bufferp < BUFFER\_SIZE; bufferp++) {

    Ifinal += bufferI[bufferp];

    Vfinal += bufferV[bufferp];

    if (bufferV[bufferp] == 0) {

        secuV++;

    }

}

    Ifinal = Ifinal / BUFFER\_SIZE;

    Vfinal = Vfinal / BUFFER\_SIZE;

    Pfinal = Ifinal \* Vfinal;

    deltaP = Pfinal - Pprev;

    deltaV = Vfinal - Vprev;

    if(secuV >= 5)     // Si trop d'activité, évite comparaison et reporte au buffer suivant

    {

        printk("Alimentation trop peu stable / trop basse\n");

    }

  else if (deltaP >= 0  )

    {

      if (V1\_low\_value >= 23.9)      // Sécurité pour éviter la surtension

        {

            duty\_cycle = duty\_cycle\_prev – 0.02;

            if(duty\_cycle<0.15) duty\_cycle = 0.15;

            printk("La tension est déjà au maximum\n");

            printk("La rapport cyclique passe à : %f\n",duty\_cycle);

        }

      else if (deltaV> 0)

        {

            duty\_cycle = duty\_cycle\_prev + 0.02;

            if(duty\_cycle>0.85) duty\_cycle = 0.85;

            printk("Le rapport cyclique a augmenté de %f \n", deltaD);

        }

        else

        {

            duty\_cycle = duty\_cycle\_prev – 0.02 ;

            if(duty\_cycle<0.15) duty\_cycle = 0.15;

            printk("Le rapport cyclique a diminué de %f \n", deltaD);

         }

     }

    else if(deltaP < 0)

     {

        if (V1\_low\_value <= 3.1)       // Sécurité pour éviter la sous-tension

        {

            duty\_cycle =  duty\_cycle\_prev + 0.02;

            if(duty\_cycle>0.85) duty\_cycle = 0.85;

            printk("La tension est déjà au minimum\n");

            printk("La rapport cyclique passe à : %f\n",duty\_cycle);

        }

         else if (deltaV > 0)

        {

           duty\_cycle  = duty\_cycle\_prev – 0.02 ;

            if(duty\_cycle>0.85) duty\_cycle = 0.85;

            printk("Le rapport cyclique a diminué de %f \n", deltaD);

        }

        else {

            duty\_cycle = duty\_cycle\_prev + 0.02 ;

            if(duty\_cycle<0.15) duty\_cycle = 0.15;

            printk("Le rapport cyclique a augmenté de %f \n", deltaD);

        }

      }

    printk("Is : %fA   ||   Vs : %fV   ||   Ve : %fV   ||   α : %f\n", I1\_low\_value, V1\_low\_value, V\_high, duty\_cycle);

    printk("Tension cible : %fV  ||  Pfinal : %fW\n", voltage\_reference, Pfinal);

    printk("Vprev : %fV  ||  deltaP : %fW  ||  deltaV : %f\n", Pprev, deltaP, deltaV);

    Pprev = Pfinal;

    duty\_cycle\_prev = duty\_cycle;

    Vprev = Vfinal;

    printk("===============================================================================\n\n");

  }

}

In function void loop critical task (), add this code to control the pwm activation:

 if (mode == IDLEMODE)

    {

        if (pwm\_enable == true)

        {

            twist.stopAll();

        }

        pwm\_enable = false;

    }

    else if (mode == POWERMODE)

    {

        if(!pwm\_enable) {

            pwm\_enable = true;

            twist.startAll();

        }

        //Sends the PWM to the switches

    twist.setAllDutyCycle(duty\_cycle);

    }

}

## Code explanation

* bufferI and bufferV: Arrays that store current and voltage readings over a time window defined by BUFFER\_SIZE.
* bufferp: Index used to fill bufferI and bufferV arrays, incremented after each sample.
* I1\_low\_value and V1\_low\_value: Measured output current and voltage.
* If V1\_low\_value <= 0, we clamp it to 0 to avoid negative or invalid values.
* When bufferp reaches BUFFER\_SIZE, this means that the buffer is complete and ready to be treated. It is reinitialized at 0 and starts to stock new values. secuV is also reinitialized at 0.

A loop iterates through the buffers to calculate the sum of the current values (stored in Ifinal) and the voltage values (stored in Vfinal). During this process, if any voltage value in the buffer is equal to zero, the secuV counter is incremented in order to keep track of the number of zeros.

Once the loop has processed all values, the accumulated sums for current and voltage are divided by the constant BUFFER\_SIZE in order to compute the average current and average voltage over the buffer period. Also:

* Pfinal is calculated as the product of the average current (Ifinal) and the average voltage (Vfinal).
* deltaP is computed as the difference between the current power value (Pfinal) and the previous iteration’s power value (Pprev).
* deltaV is calculated as the difference between the current average voltage (Vfinal) and the voltage value from the previous iteration (Vprev).
* If the secuV counter is greater than or equal to 5—meaning that at least five zero voltage readings were encountered in the buffer—a warning message is printed to indicate that the input voltage is too unstable or too low, and the MPPT comparison and duty cycle adjustment are skipped for this iteration.

If deltaP is positive or zero (indicating that the power has increased or remained constant since the last iteration), the algorithm then checks the current output voltage:

* If the voltage is greater than or equal to 23.9 V (the upper safety threshold), the duty cycle is decreased to prevent overvoltage.
* If the voltage is below 23.9 V and deltaV is positive (i.e., the voltage has increased since the last iteration), the duty cycle is increased to try to reach a higher power point.
* If the voltage is below 23.9 V but deltaV is negative (i.e., voltage has decreased), the duty cycle is decreased to search in the opposite direction on the power curve.

On the other hand, if deltaP is negative (indicating a drop in power compared to the previous iteration):

* If the output voltage is less than or equal to 3.1 V (the lower safety limit), the duty cycle is increased to avoid undervoltage and ensure the load continues to operate.
* If the voltage is above 3.1 V and deltaV is positive (voltage increased), the duty cycle is decreased to move away from a less efficient operating point.
* If the voltage is above 3.1 V and deltaV is negative (voltage decreased), the duty cycle is increased to explore a potentially better operating point.

At each iteration, the current values of the measured current, voltage, calculated power, and duty cycle are printed to the serial interface for monitoring and debugging. Additionally, the current values of power (Pfinal), duty cycle (duty\_cycle), and voltage (Vfinal) are stored so they can be used in the next iteration to compute the new deltaP and deltaV. This allows the Perturb and Observe algorithm to dynamically adjust the converter operation to maximize power extraction from the photovoltaic source.

## Connect Hardware

1. For hardware setup, connect your TWIST board to the power supply and the PC via USB-C.

* Connect Vhigh and GND of the TWIST to the photovoltaic panel (set a 2 A current limit).
* Connect VLow1, VLow2, and GND of the TWIST to a motor pump.
* Connect the USB-C port of the SPIN to your PC via a USB cable.

Une image contenant circuit, Ingénierie électronique, Composant électronique, Composant de circuit passif

Le contenu généré par l’IA peut être incorrect.

1. Build  build and Upload upload
2. Use OwnPlot or the Serial Monitor to visualize your data and control the converter.

Voilà !

