

**American University of Sharjah**

**ELE546: ADVANCED POWER ELECTRONICS**

FALL 2017/2018

**MPPT Battery Solar Charger**



**Students Name:** Abdulla Eyad Lutfi & Al Baraa El-Hag

**Student ID #:** B00045951 / B00047451

***Instructor: Dr. Habib ur-Rehman***

|  |  |
| --- | --- |
| **Table of contents** | |
| Abstract | Pg1 |
| Introduction | Pg1 |
| Objectives | Pg |
| Project components | Pg |
| Simulation | Pg |
| Results and Discussion | Pg |
| Challenges and Difficulties | Pg |
| Conclusion | Pg |

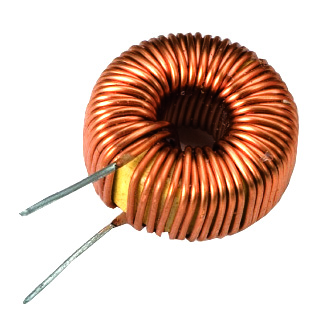
|  |  |
| --- | --- |
| **List of Figures** | |
|  | Pg |
|  | Pg |
|  | Pg |
|  | Pg |
|  | Pg |
|  | Pg |
|  | Pg |
|  | Pg |

* **Abstract**Power Electronic converters are one of the main building blocks for most solar energy applications. It can step up or step down the DC voltage produced by the solar panel. In this project, a150W solar panel is used to charge a 12V battery andturn on a 55W street light using an Arduino based buck converter as an interface. A simulation using PSIM has been done to study the characteristics of the output and compare it to the practical output. Furthermore, the solar charge controller is designed based on the Maximum Power PointTracking (MPPT) algorithm where the controller tries to follow the maximum point in the PV-curve using hill climb method.

1. **Introduction**
2. **Objectives**The objectives of this project are summarized as follows:
3. Design and Simulate the buck converter circuit using PSIM software.
4. Implement the buck converter circuit in the lab.
5. Implement the concept of Maximum Power Point Tracker (MPPT).
6. Test the circuit by charging a 12V battery using a solar panel and check if the MPPT algorathim is working.
7. Compare the theoretical simulation results with the experimental results.
8. **Project Components**
9. **Sizing of the Inductor and Capacitor**
10. **Components used**



Capacitor



3mH Inductor



Arduino Uno R3



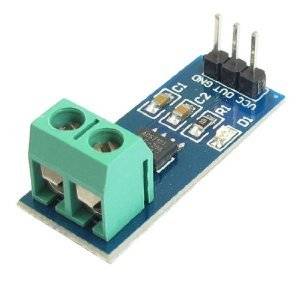


IR2110 Gate Drive



NOT Gate

MOSFETs x2  
(2nd one acting as a diode)



Current Sensor

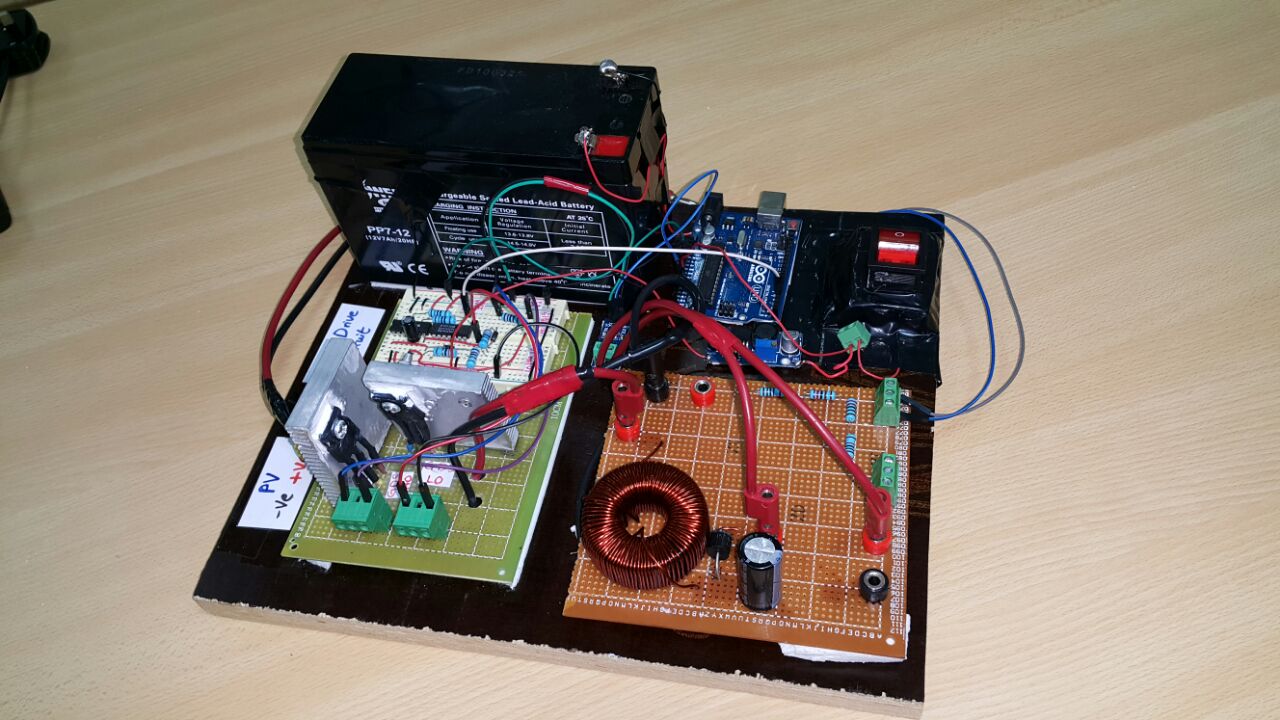
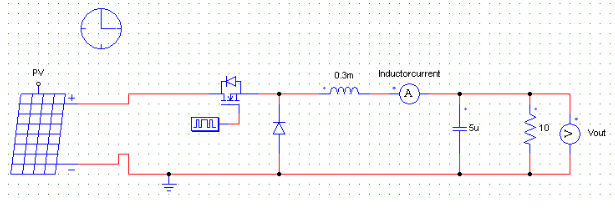
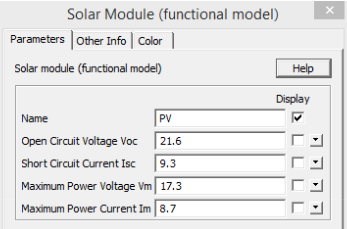


Figure The Buck Converter when put together.

1. **Simulation**

****

****

1. **Results and Discussion**

* PWM Pin #3 in the Arduino was declared as an output and a PWM signal was generated from it at 30KHZ frequency and 50% Duty Cycle. This was connected to an oscilloscope and the result is shown below.

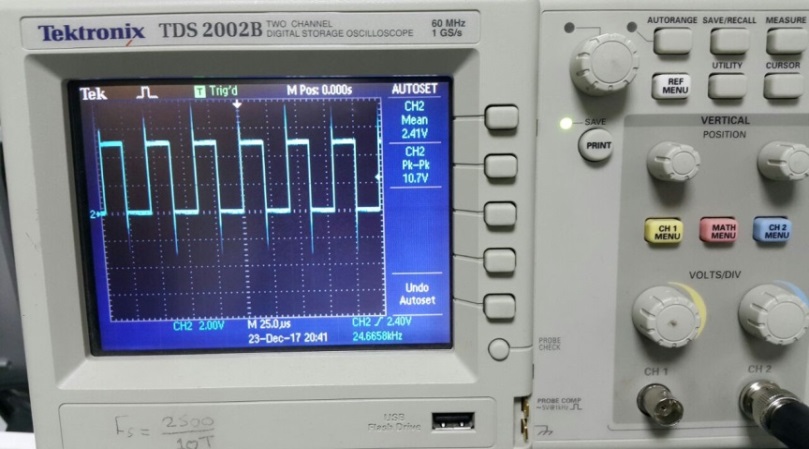


Figure 1 PWM Arduino Output

* Figure 2 shows the same Arduino PWM output after adding a 330 ohm resistor. The signal looks much smoother and the overshoots where taken care of.

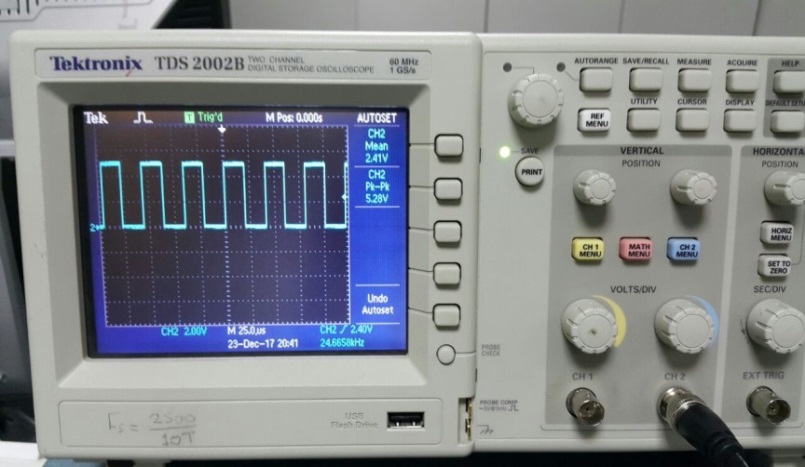


Figure 2 Arduino Output after adding 330 ohm Resistor

* The Arduino PWM signal was then inverted using an inverting Op-amp. Both these signals were fed to the High Input and Low input of the gate drive respectively. The 2 signals are shown below.

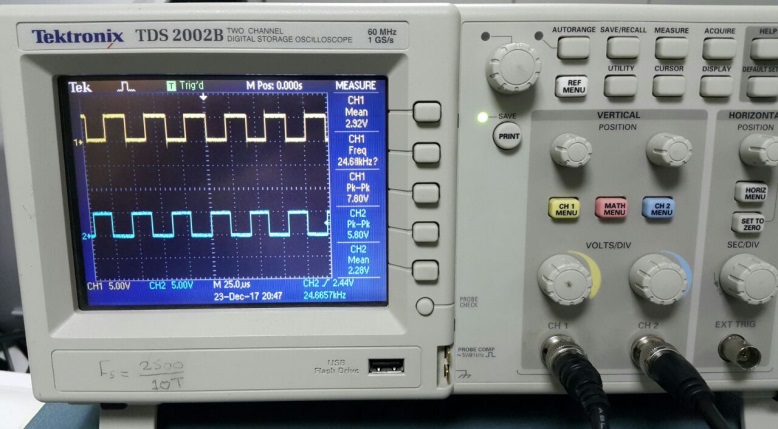


Figure 3

* Fig 4 shows the gate drive circuit that has been used for the project. 2 opposite signals are entered into the HIN and LIN pins of the gate drive which can drive 2 MOSFETS that work in a complementary manner. The first MOSFET is acting as a switch for the buck while the second one is acting as a “Free Wheeling Diode”.

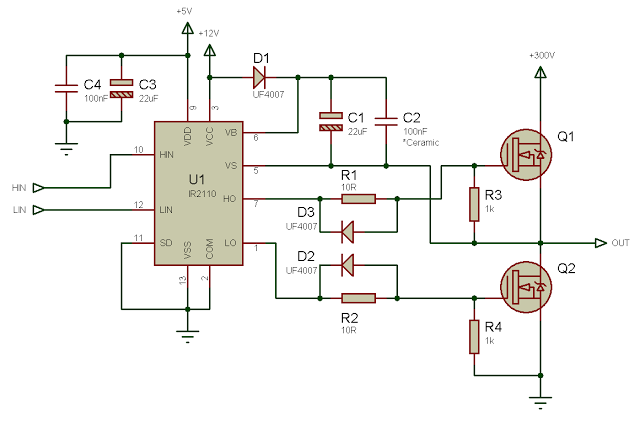


Figure 4

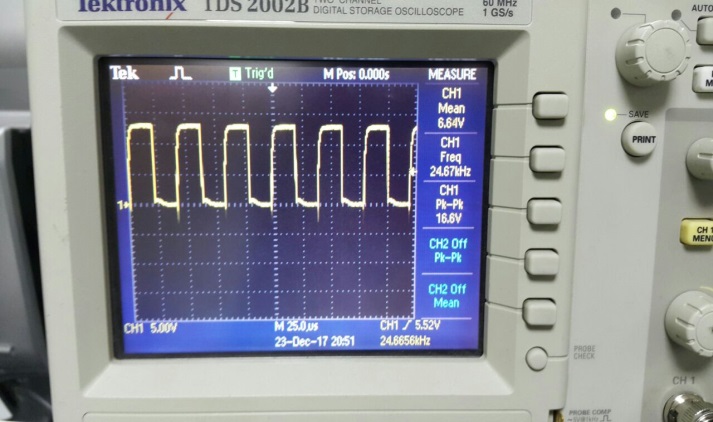


Figure 6

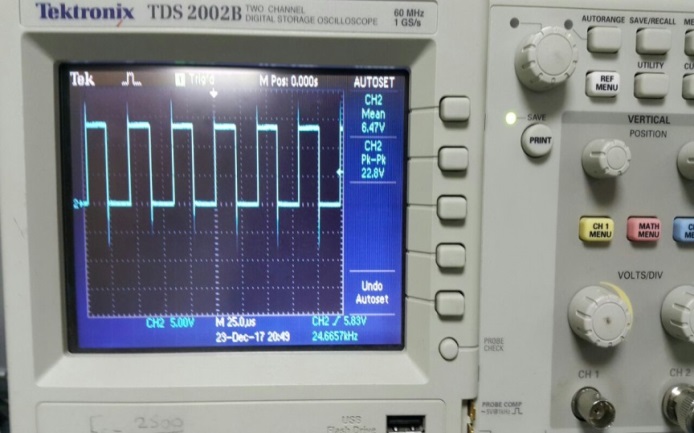


Figure 5

* Figures 5 and 6 shows the output of the LO and HI of the gate drive respectively that will be fed into the gate terminals of the 2 MOSFETS.
* Using a 15V source input at a 0.5 Duty cycle the following was the output of the buck as read by the oscilloscope.

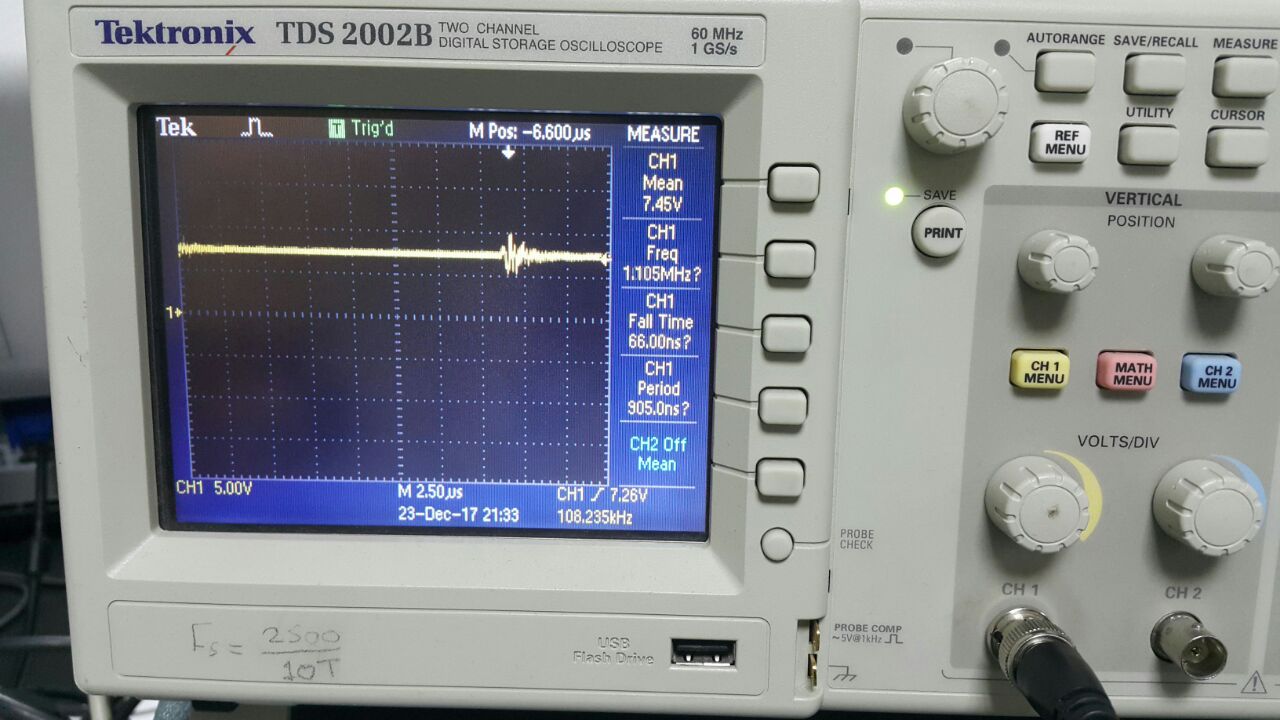


Figure. 7

1. **Challenges and Difficulties**
2. **Conclusion**
3. **Appendix**

/////////////////////////////////////////////////////////////////////////////////////////

// Constant Global Variables //

/////////////////////////////////////////////////////////////////////////////////////////

const int Pin\_Iout = A4;

const int Pin\_Vout = A1;

const int Pin\_Vdd = A2;

const int Pin\_Ctrl = A5;

const double Frequency = 200;

/////////////////////////////////////////////////////////////////////////////////////////

// Global Variables //

/////////////////////////////////////////////////////////////////////////////////////////

double D; double D\_Old; double D\_New;

double Vdd = 4.96;

double Vout\_Old; double Vout\_New; double Iout\_Old; double Iout\_New; double Pout\_Old; double Pout\_New;

double Jump = 0.05; // How much we change the duty cycle every iteration

double Delta\_Compare = 0.001;

double Jump2 = 0.015;

/////////////////////////////////////////////////////////////////////////////////////////

// Setup //

/////////////////////////////////////////////////////////////////////////////////////////

void setup()

{

Serial.begin(9600);

pinMode(Pin\_Iout, INPUT);

pinMode(Pin\_Vout, INPUT);

pinMode(Pin\_Vdd, INPUT);

pinMode(Pin\_Ctrl, INPUT);

pinMode(3, OUTPUT);

D = 0.1;

TCCR2A = \_BV(COM2B1) | \_BV(WGM21) | \_BV(WGM20); // Just enable output on Pin 3 and disable it on Pin 11

TCCR2B = \_BV(WGM22) | \_BV(CS22);

OCR2A = Frequency; // defines the frequency 51 = 38.4 KHz, 54 = 36.2 KHz, 58 = 34 KHz, 62 = 32 KHz

OCR2B = Frequency \* D ; // defines the duty cycle - Half the OCR2A value for 50%

TCCR2B = TCCR2B & 0b00111000 | 0x2; // select a prescale value of 8:1 of the system clock

Do\_Nudge();

Get\_VddV2();

}

void loop()

{

Get\_VddV2();

Do\_MPPT();

Serial.print("D = ");

Serial.println(D);

Serial.print("Vout = ");

Serial.println(Vout\_New);

Serial.print("Iout = ");

Serial.println(Iout\_New);

Serial.print("Pout = ");

Serial.println(Pout\_New);

Serial.println(" ");

delay(1000);

}

/////////////////////////////////////////////////////////////////////////////////////////

// Finding Vdd //

/////////////////////////////////////////////////////////////////////////////////////////

void Get\_Vdd() {

long result;

// Read 1.1V reference against AVdd

ADMUX = \_BV(REFS0) | \_BV(MUX3) | \_BV(MUX2) | \_BV(MUX1);

delay(2); // Wait for Vref to settle

ADCSRA |= \_BV(ADSC); // Convert

while (bit\_is\_set(ADCSRA, ADSC));

result = ADCL;

result |= ADCH << 8;

result = 1125300L / result; // Back-calculate AVdd in mV

Vdd = result / 1000.0;

}

double Get\_VddV2()

{

double Error;

Error = analogRead(Pin\_Vdd);

Vdd = (Error + 826) / (4.04 \* 1023);

Vdd = Vdd - 0.2;

Vdd = 1 / Vdd;

return Error;

}

/////////////////////////////////////////////////////////////////////////////////////////

// Here is the Gets! //

/////////////////////////////////////////////////////////////////////////////////////////

double Get\_Analog(int Pin)

{

double ADC;

double ADC2;

double V\_Pin1 = 0;

double V\_Pin;

for (int i = 1; i <= 90; i++)

{

ADC = analogRead(Pin);

ADC2 = (ADC) \* (Vdd / 1024.0);

V\_Pin1 = V\_Pin1 + ADC2;

// delay(1);

}

V\_Pin = V\_Pin1 / 90.0;

return V\_Pin;

}

double Get\_Iout()

{

double Iout1;

double Iout2;

Iout1 = Get\_Analog(Pin\_Iout);

Iout2 = 1.025 \* 10.427 \* (Iout1 - (Vdd / 2));

if (Iout2 < 0.05)

{

Iout2 = 0;

}

return Iout2;

}

double Get\_Vout()

{

double Vout;

Vout = Get\_Analog(Pin\_Vout);

Vout = 1.15 \* ((2.45 + 9.99) / 2.45) \* Vout;

if (Vout < 0.05)

{

Vout = 0;

}

return Vout;

}

double Get\_Pout(double Vout, double Iout)

{

double Pout;

Pout = Vout \* Iout;

return Pout;

}

/////////////////////////////////////////////////////////////////////////////////////////

// Control Algorithms //

/////////////////////////////////////////////////////////////////////////////////////////

void Do\_Duty()

{

if (D > 0.95)

{

D = 0.95;

}

if (D < 0.05)

{

D = 0.05;

}

OCR2B = Frequency \* (1 - D); // defines the duty cycle - Half the OCR2A value for 50%

}

void Do\_Nudge ()

{

D\_Old = D;

Vout\_Old = Get\_Vout();

Iout\_Old = Get\_Iout();

Pout\_Old = Get\_Pout(Vout\_Old, Iout\_Old);

D = 0.10;

Do\_Duty();

delay(1000);

D\_New = D;

Vout\_New = Get\_Vout();

Iout\_New = Get\_Iout();

Pout\_New = Get\_Pout(Vout\_New, Iout\_New);

}

void Do\_Voltage (double V\_desired)

{

double Vout;

Vout = Get\_Vout();

if (V\_desired > Vout)

{

D = D + Jump;

}

if (V\_desired < Vout)

{

D = D - Jump;

}

}

void Do\_Current (double I\_desired)

{

double Iout;

Iout = Get\_Iout();

if (I\_desired > Iout)

{

D = D + Jump;

}

if (I\_desired < Iout)

{

D = D - Jump;

}

}

void Do\_MPPT ()

{

double Vout\_Delta;

Vout\_Delta = Vout\_New - Vout\_Old;

double Pout\_Delta;

Pout\_Delta = Pout\_New - Pout\_Old;

double Delta;

Delta = Vout\_Delta \* Pout\_Delta;

D\_Old = D;

Vout\_Old = Get\_Vout();

Iout\_Old = Get\_Iout();

Pout\_Old = Get\_Pout(Vout\_Old, Iout\_Old);

if (Delta > Delta\_Compare)

{

D = D + Jump;

if (D\_New > D\_Old && Vout\_New < Vout\_Old)

{

D = D - 2 \* Jump;

}

}

else if (Delta < -Delta\_Compare)

{

D = D - Jump;

}

else

{

if (Delta > 0)

{

D = D + Jump2;

if (D\_New > D\_Old && Vout\_New < Vout\_Old)

{

D = D - 2 \* Jump2;

}

}

if (Delta < 0)

{

D = D - Jump2;

}

}

Do\_Duty();

D\_New = D;

Vout\_New = Get\_Vout();

Iout\_New = Get\_Iout();

Pout\_New = Get\_Pout(Vout\_New, Iout\_New);

}