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

Abstract

In today's world, power electronics circuits can be seen everywhere. From the 10W rectifiers used in charging cell phones to the massive inverters used in MW for power generation. Our project will consist of a small scale replica of a solar energy system, in which we will connect a step down DC-DC converter to a 150 W panel and have it extract the maximum power. We first perform a simulation on our DC-DC circuit, then design a chopper circuit and after that extend it to a full buck converter by adding a capacitor and an inductor. Lastly, we design a Maximum Power Point Tracking (MPPT) using the hill-climbing methodology on an Arduino such that we will always be at the most optimal point on the PV curve.

Table of Contents

Table of Figures

1. Objectives

a) Plan and simulate a buck converter
b) Implement a chopper circuit
c) Design a buck converter
d) Write and test an MPPT control algorithm

2. Project Components

In this section, the project materials and components will be figured out and sized. This is a very important step since acquisition of some of these components take quite a while.

2.1 Sizing

The specifications of the project are that it should have less than 5% ripple for the both the current and voltage, utilizing a 20 kHz switching frequency. It is required that the system stay in the continuous conduction mode for any power input above 30 watts. Lastly, the power requirements of all the components is that it should be able to handle at the minimum 20 V and 10 A.

$V_o(\max) = 20\ V$ $\qquad$ $\Delta i_L(min) = 5A * 0.05 = 0.25\ A$

$f_s = 20 * 10^3$ Hz $\qquad$ $D\ (min) = \frac{Bat\ V(\min)}{PV\ V(max)} = \frac{12}{21} = 0.57$

$$L = \frac{V_o}{\Delta i_L * f_s} * (1 - D) = \frac{20}{0.25 * 20 * 10^3} * (1 - 0.57) > 1.72\ mH$$

L = 3mH was chosen for the project.

$\frac{V_o}{\Delta V_o} = \frac{1}{0.05} = 20$ $\qquad$ $L = 3 * 10^{-3} H$

$f_s = 20 * 10^3$ Hz $\qquad$ $D\ (min) = \frac{Bat\ V(\min)}{PV\ V(max)} = \frac{12}{21} = 0.57$

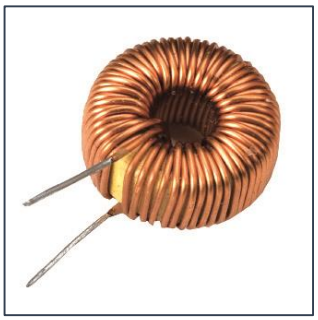
$$C = \frac{1}{8} * \frac{V_o}{\Delta V_o} * \frac{(1 - D)}{f_s^2 * L} = \frac{1}{8} * 20 * \frac{1 - 0.57}{(20 * 10^3)^2 * 3 * 10^{-3}} > 0.9\ \mu F$$

C = 2200uF was chosen for the project (the price difference between a small capacitor and a larger one is almost negligible)

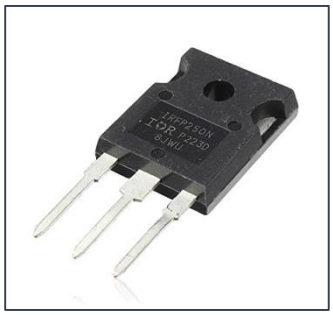
2.2 Components

Arduino Uno R3

Capacitor = 2200uF

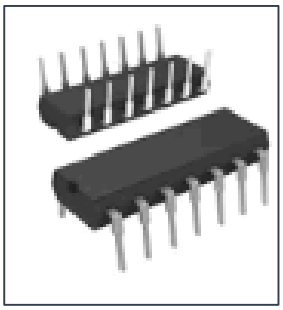


Inductor = 3mH



MOSFETs x2 (2nd one acting as a diode)

IR2110 Gate Drive



NOT Gate

Current Sensor

3. Simulation

Given below in Figure 1is a simulation of the buck converter that is planned to be built with a PI control scheme implemented. The values of the inductor and capacitor were the ones previously calculated in section 2.22.2; 3mH and 2200uF respectively. In addition the load was set to 6 ohms, the PV voltage to 20 V and Vo Desired to 14.3 V.

An important thing to note is that the lower diode is replaced with another MOSFET. The original is labeled “High MOSFET”and the second “LOW MOSFET”. To drive the gates it is needed to give them both the exact opposite gating signal, and this is implemented through the use of a NOT gate.

The purpose of replacing the diode with another MOSFET is because the gate drive “IR2110” is only capable of getting a low out or a low out and high out. It cannot only give a High Out.

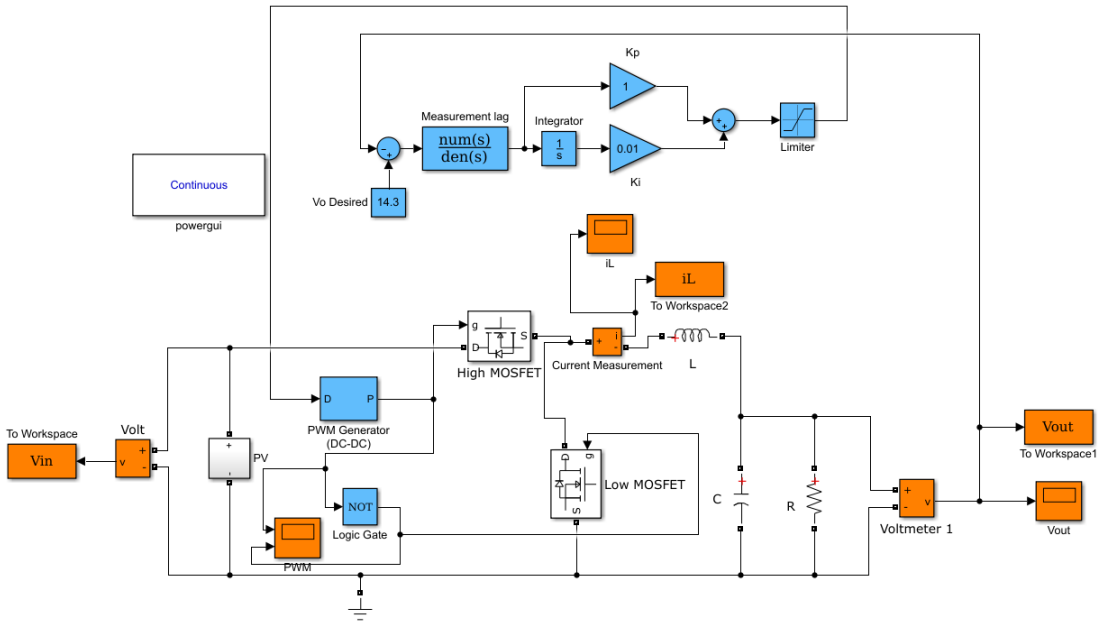


Figure 1: Buck Model with PI Control in Simulink

Given below in Figure 2 and Figure 3 are the output voltage and current ripples respectively.

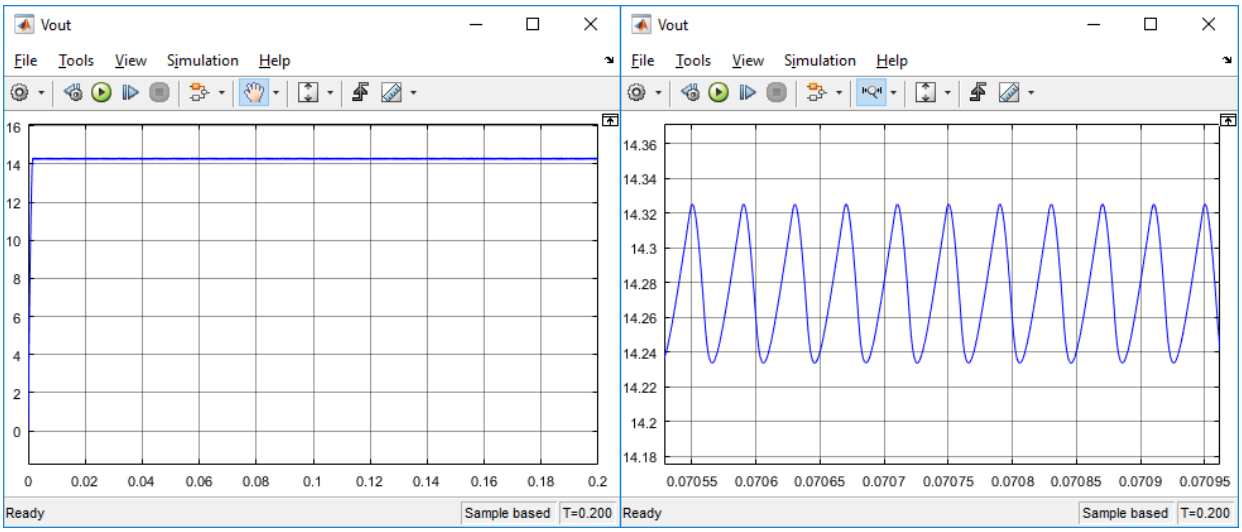


Figure 2: Output Voltage and Ripple

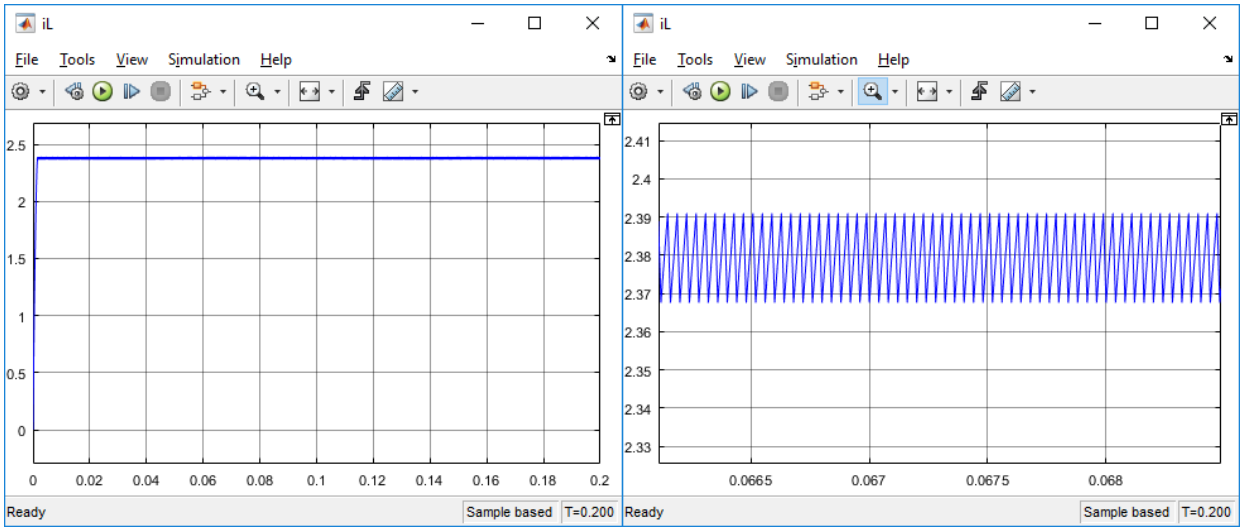


Figure 3: Inductor Current and Ripple

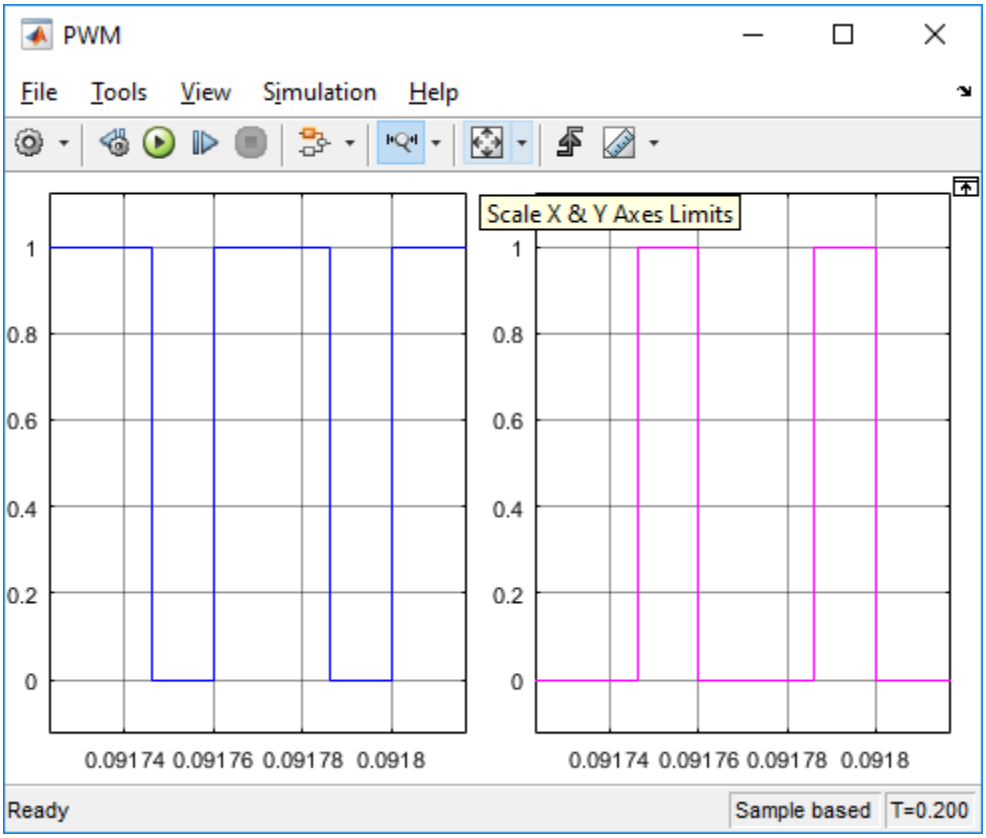


Figure 4: Gating Signals

Lastly given above in Figure 4 are the gatings signals sent to the MOSFETs. The left is for the "High MOSFET" and the inverted output is for the "Low MOSFET".

4. Implementation

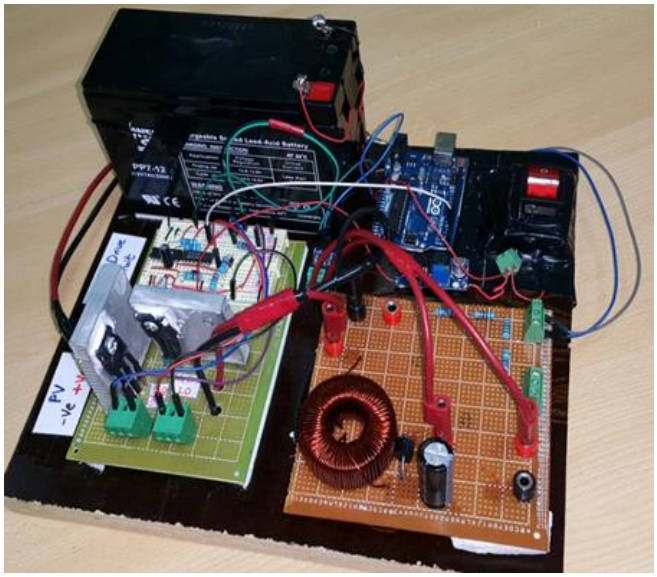


Figure 5: Final Circuit of Buck

In this section, the physical implementation of the system will be described. Given above in Figure 5 is the final circuit of the buck converter with an Arduino acting as the controller.

4.1 PWM Arduino

```
Frequency = 80;
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
```

The given code above utilizes the Arduino timers to give us a PWM output. The output of the Arduino measured between the output pin and ground is given on the next page in Figure 6.

Figure 6: Arduino PWM Output

As can be seen the output is not clean, there is a very high and under shoots. To fix this a 330 ohm resistor was added to the output to give a much cleaner (but albeit slower) PWM output to give to the gate drive. The output is given below in Figure 7.

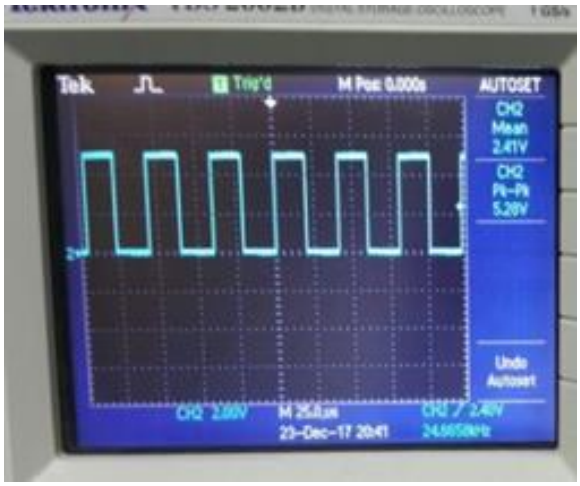


Figure 7: Arduino PWM Output with R=330

4.2 NOT Gate

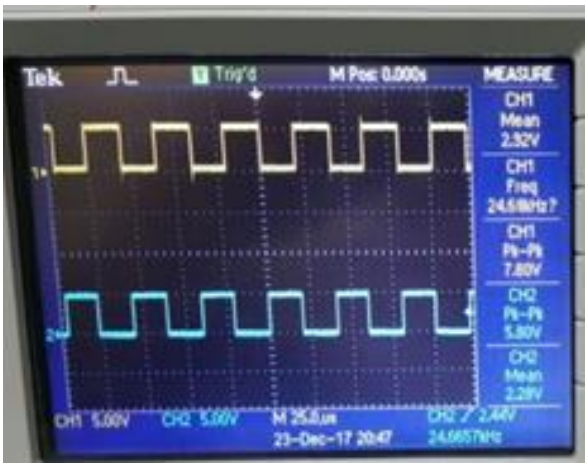


Figure 8: Inverter and Arduino Output

Since the gate drive needs both this signal and an inverted copy of it to drive both the High and Low MOSFET's, it is needed to find a methodology to do just that. This was realized through the use of a NOT gate. Above, in Figure 8, both the Arduino output and NOT gate output are given. It can be easily seen that they are exactly the opposite of each other.

4.3 Gate Drive

The circuit topology suggested by Tahmid [1] is given below in Figure 9.

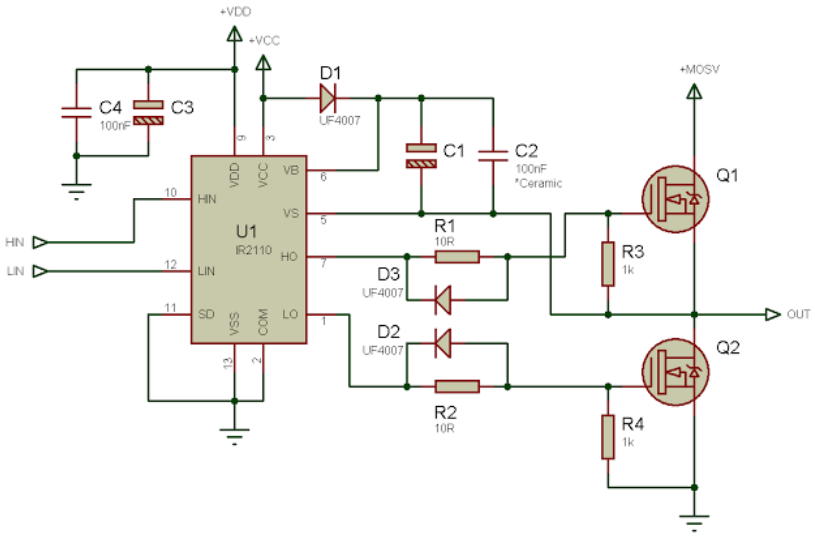


Figure 9: Tahmid's HO and LO Gate Drive Circuit [1]

Starting from the left, VDD was taken from the Arduino +5V pin and a capacitor was added to try and smooth the output. The IR2110 utilizes this voltage to determine the high and low values of the inputted PWM Signal and the fact the +5 V pin on the Arduino is perfectly in match with it's output max the circuit completely in tune.

The Hin and Lin pins were the inputs from the NOT gate and the Arduino out respectively (the Hin should have got the Arduino out and not the NOT gate out, but if we were to feed 1-Duty_Cycle in the code it is easily fixed).

Also as a note, only the capacitors of C4 and C1 were utilized.

Now on the right, R1 and R2 were set to 33 ohms, R3 and R4 to 1k ohms and D1, D2, D3 connected. The LO pin was given to the gate of Q2 (Low MOSFET) while it's source was grounded. On the other hand, the HO pin was given to gate of Q1 (High MOSFET) and the pin VS was given to the source of Q1.

The drain of Q1 is the PV input and the source of Q1 (which is also connected to the drain of Q2) is the output of our chopper. This is to be connected to the inductor when wiring the buck. Given below in Figure 10, is both the LO and HO gating signals given to the MOSFETs. Disregarding the overshoot, the peak value is what was feed into them from Vcc. In out circuit, we connected a boost converter on a 12 V battery to give the IR2110 a constant 15V.

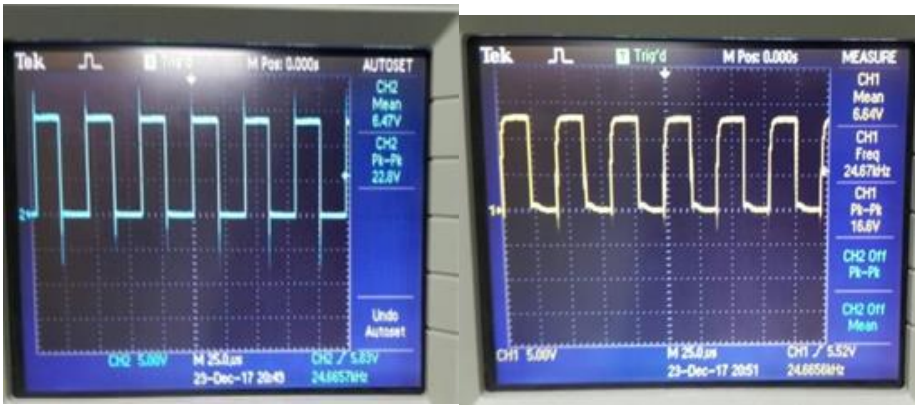


Figure 10: LO and HO of IR2110

The IR 2110 chip requires a high current to drive our MOSFETs. Experimentally it was seen that the higher switching frequency of the system, the higher the current draw. For example; at 10 kHz the gate drive draws 1A; at 20 kHz it draws 2.5A and at 40 kHz it draws greater than 3.5 A (the limit of our source). In addition, if the input voltage of the buck is higher it needs to draw more gating current.

5. Arduino Code

In section 4.1, the paper touched on the Arduino code. The following section will delve deeper into the Arduino algorithm. The Arduino code is made up of independent functions which will be described individually. Given in Appendix I is the complete Arduino code.

5.1 Get_VDDV2

```
double Get_VddV2()
{
  double Error;
  Error = analogRead(Pin_Vdd);
  Vdd = (Error + 826) / (4.04 * 1023);
  Vdd = Vdd - 0.2;
  Vdd = 1 / Vdd;
  return Error;
}
```

The problem with the 5 V pin of the Arduino is that it is not exactly 5 V, experimentally it was found that it fluctuated between 4.5 – 5.09 V. This is not an issue with our gate drive since the PWM output is exactly the same Vdd entering into it. But when activating the analog to digital converter of the Arduino, it compares the inputting signal to what is supposed to be 5V and then inputs. The code given above fixes this error in analogRead, making it accurate around to the 2.5 mV.

5.2 Get_Analog

```
double Get_Analog(int Pin)
{
  double ADC;
  double ADC2;
  double V_Pin1 = 0;
  double V_Pin;

  for (int i = 1; i <= 1000; i++)
  {
    ADC = analogRead(Pin);
    ADC2 = (ADC) * (Vdd / 1024.0);
    V_Pin1 = V_Pin1 + ADC2;
    //      delay(1);
  }
  V_Pin = V_Pin1 / 1000.0;
  return V_Pin;
}
```

Another issue faced with the Arduino, is the noise of our measuring devices. To take care of the problem, the Arduino was set to take 1000 measurements and then average them out.

5.3 Get_Iout, Get_Vout

```
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;
}
```

The given two codes of Get_Vout and Get_Iout utilizes the analog read pins of the Arduino to read the output voltage and output current respectively. For the output voltage, a simple voltage divider was used and for the current a current sensor.

5.4 Get_Pout

```
double Get_Pout(double Vout, double Iout)
{
  double Pout;
  Pout = Vout * Iout;
  return Pout;
}
```

A simple function to get the output power by multiplying Vout and Iout

5.5 Do_Duty

```
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%
}
```

This code is used to change the duty cycle of the output PWM signal but also keeping the frequency constant. As mentioned before, (1-D) was used instead of D since the Hin pin was connected to the inverted output and not to the Arduino output.

5.6 Do_Voltage

```
void Do_Voltage (double V_desired)
{
  double Vout;
  Vout = Get_Vout();

  if (V_desired > Vout)
  {
    D = D + Jump2;
  }

  if (V_desired < Vout)
  {
    D = D - Jump2;
  }

  Do_Duty();
  Iout_New = Get_Iout();
  Vout_New = Get_Vout();
  Pout_New = Get_Pout(Vout_New, Iout_New);
}
```

This is a simple code that takes in a desired output voltage (from the user) and tries to maintain that constant output utilizing if statements. “Jump2” is a global variable that is used to change the duty cycle each iteration. It was found that a value of 0.015 was optimal.

5.7 Do_Current

```
void Do_Current (double I_desired)
{
  double Iout;
  Iout = Get_Iout();

  if (I_desired > Iout)
  {
    D = D + Jump2;
  }

  if (I_desired < Iout)
  {
    D = D - Jump2;
  }


  Do_Duty();
  Iout_New = Get_Iout();
  Vout_New = Get_Vout();
  Pout_New = Get_Pout(Vout_New, Iout_New);

}
```

The same is Do_Voltage except that it tries to establish a desired current.

5.8 Do_MPPT

```
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 > 0)
  {
    D = D + Jump;
 }

  if (Delta < 0)
  {
    D = D - Jump;
  }

  Do_Duty();

  D_New = D;
  Vout_New = Get_Vout();
  Iout_New = Get_Iout();
  Pout_New = Get_Pout(Vout_New, Iout_New);

}
```

A simple implementation of the hill-climbing algorithm. It calculates the variable “Delta” by first calculating “Pout_Delta” and “Vout_Delta” (the new reading of the variable minus the old reading of the variable). Then it multiplies Pout_Delta and Vout_Delta and checks if it is greater or less than zero. If it is greater than zero that in increases the duty cycle and vice versa.

6. Conclusion

Via simulation, a buck converter can be done in less than 15 minutes but when building the actual circuit from discrete components it takes much longer. The first issue is the acquiring of the required power components, finding an inductor that can handle 1 W isn't too difficult but one that can handle 50 W is another matter. In addition, there are many components that are very easily damaged. For example, it was found that the gate drive would burn if it wasn't given enough supply current. As a class we went through about 20 until we managed to understand all the bugs and quirks of the IR2110. Lastly, the documentation on the internet was misleading. In Tahmid's blog it was suggested that the IR2110 could be used in a HO only mode and that is simply not true.

As a group, we managed to implement first a buck converter and then an MPPT control algorithm. In the lab, the buck was tested with the maximum values the bench power supply can give (which is 30V at 3A) and our buck managed to handle the 90 watts going through it. We didn't have a high power supply to test, but we are sure that our circuit should probably be able to handle at least 150 – 200 watts. While running the circuit for 10 minutes, the heat sinks barely heated up. For the MPPT control algorithm, we managed to find the maximum point and stay at the point. Unfortunately, due to the time spent on the project, we couldn't document all of our results.

In conclusion, the project has given invaluable experience, not only in the field of power electronics but in electrical engineering in general.

References

[1] Tahmid, “Tahmid's blog,” *Using the high-low side driver IR2110 - explanation and plenty of example circuits*, 01-Jan-2010. [Online]. Available: http://tahmidmc.blogspot.ae/2013/01/using-high-low-side-driver-ir2110-with.html. [Accessed: 24-Dec-2017].

Appendix I

```
////////////////////////////////////////////////////////////////////////////
/////////////
// 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 = 0.0002;
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_MPPTV2();

  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(1);
}

////////////////////////////////////////////////////////////////////////////
////////////
// 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 <= 1000; i++)
  {
    ADC = analogRead(Pin);
    ADC2 = (ADC) * (Vdd / 1024.0);
    V_Pin1 = V_Pin1 + ADC2;
    //      delay(1);
  }
  V_Pin = V_Pin1 / 1000.0;
  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 + Jump2;
  }

  if (V_desired < Vout)
  {
    D = D - Jump2;
  }

  Do_Duty();
```

```
  Iout_New = Get_Iout();
  Vout_New = Get_Vout();
  Pout_New = Get_Pout(Vout_New, Iout_New);
}


void Do_Current (double I_desired)
{
  double Iout;
  Iout = Get_Iout();

  if (I_desired > Iout)
  {
    D = D + Jump2;
  }

  if (I_desired < Iout)
  {
    D = D - Jump2;
  }


  Do_Duty();
  Iout_New = Get_Iout();
  Vout_New = Get_Vout();
  Pout_New = Get_Pout(Vout_New, Iout_New);

}

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);

}
```