Experiment 3-2
Introduction to Photovoltaic Systems and Power Electronics

ECEN 4517

Team Members:
Ali Abu AlSaud Hassan AlAhmed

Tuesday’s Lab - Bench 2

Date Performed: March 10, 2017
Instructor: Professor Khurram Afridi

Table of Contents

1. Objectives

- Design, construct, test, and demonstrate the maximum power point tracking, to charge the battery at a faster rate than would be attained with direct energy transfer.

This report is organized in the same order as Experiment 3-2's procedure document.

2. Battery Current and Voltage Sensing

In this section, a sensing circuitry will be designed and implemented. The sensing circuitry consists of a current and voltage sensors. To do that, the following considerations are taken into account:

- The voltages at the MSP430 ADC input channels must not exceed 3.3V. Therefore, 3.3V zener diodes will be placed in both ADC inputs of the MSP430 to ensure that if the readings from the sensing circuitry exceeds 3.3V, no signal goes to the Microcontroller.
- The current reading is very noisy. Thus, RC filtering is required at both the input and the output of the INA194 to clean up the signal.

The following diagrams show the entire sensing circuitry and the INA194 input filter that are going to be designed and implemented in this section.

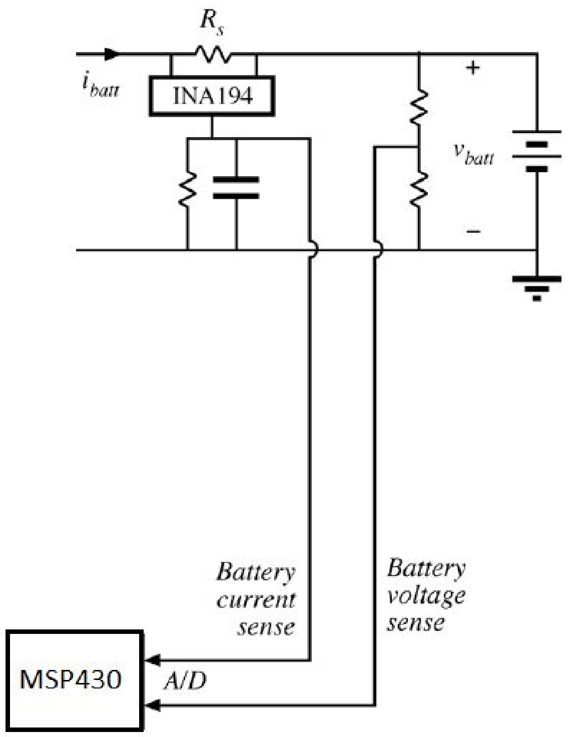

Figure 1: Sensing Circuitry

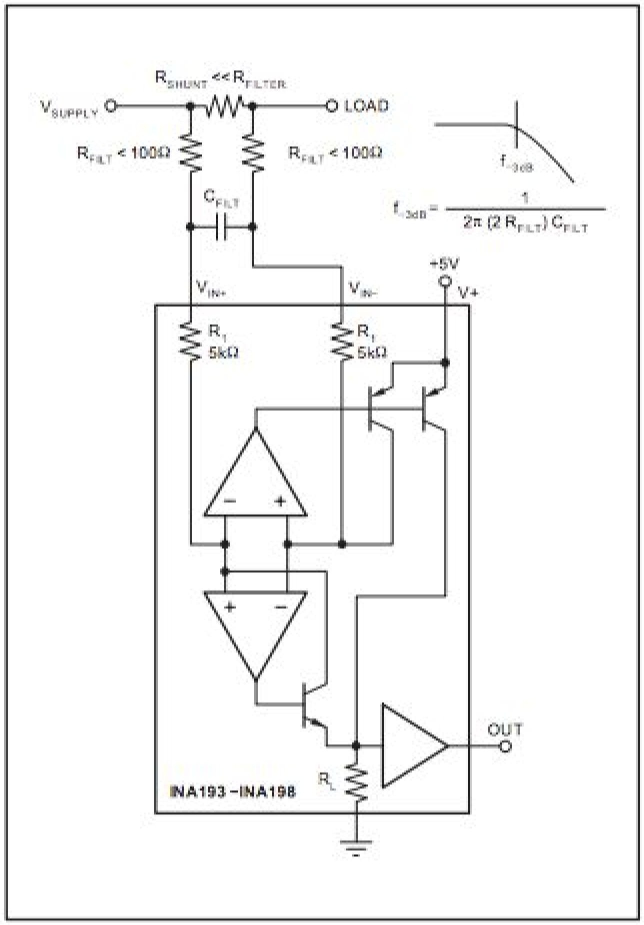

Figure 2: INA194 Input Filter

2.1. Voltage Sensing

In order to implement the voltage sensing circuitry, the following points were taken into consideration:

- MSP430 ADC maximum input voltage is 3.3V.
    - Will make the maximum 3.1V to count for any errors.
- The maximum voltage of the battery is 13V.
    - Will make the maximum 13.5V to have a margin for any errors.

From the circuitry shown above, the sensing voltage and the battery voltage are related as the following:

$$V_{sense} = V_{Battery} \frac{R_2}{R_2 + R_1}$$

The values for the resistors need to be large enough to make the current through these resistors very small; which will make the current sensing a lot easier. As a result, the following resistor values were chosen:

$$R_2 = 1.2k\Omega \qquad R_1 = 10k\Omega$$

The overall current sensing circuitry looks like figure 3 below.

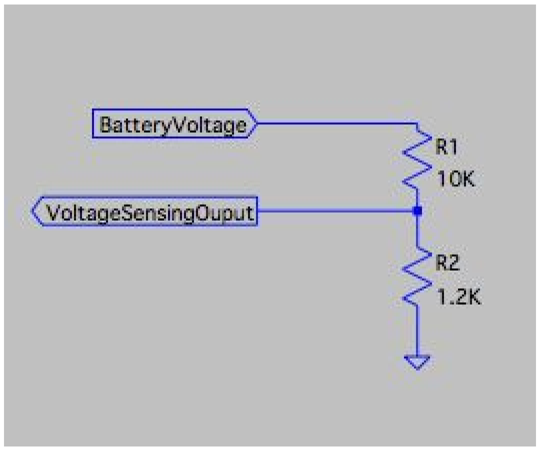

Figure 3: Overall Voltage sensing circuit

In the ADC pin, the following expression shows the relationship between the reading voltage and the battery voltage:

$$V_{ADC} = V_{Battery}\frac{R_2}{R_2 + R_1} \times \frac{1}{1023.0}$$

Where 1023.0 is the maximum reading that the ADC pin can read.

2.2. Current Sensing

The INA194 integrated chip will be used to sense the current across a sensing resistor. Note that an RC filter will be used in both the input and the output in order to make the readings clearer. Given that the current through the voltage sensing resistors is very small, the sensing current can be written as:

$$I_{sense} = I_{Battery}$$

From the switching average, the following relationship is drawn:

$$I_{Battery} = \frac{I_{in}}{D} \Rightarrow I_{Battery} = 8.25A$$

Note that the previous value is the maximum battery current when the duty cycle is 60%. From the datasheet of the INA194, the voltage drop across the sensing resistor has to be less than 500mV. As a result, the sensing resistor can be found as:

$$V = IR \implies 0.5 = 8.25\, R_s \implies R_s = 60\ m\Omega$$

Also, from the datasheet, the resistor in the RC filter has to be less than 100Ω. Therefore, the following values were chosen for the current sensing circuitry.

Parameter	**Value**
R_s	60 mΩ
R_{filter}	56 Ω
C_{filter}	1 µF

The overall current sensing circuitry looks like figure 4 below.

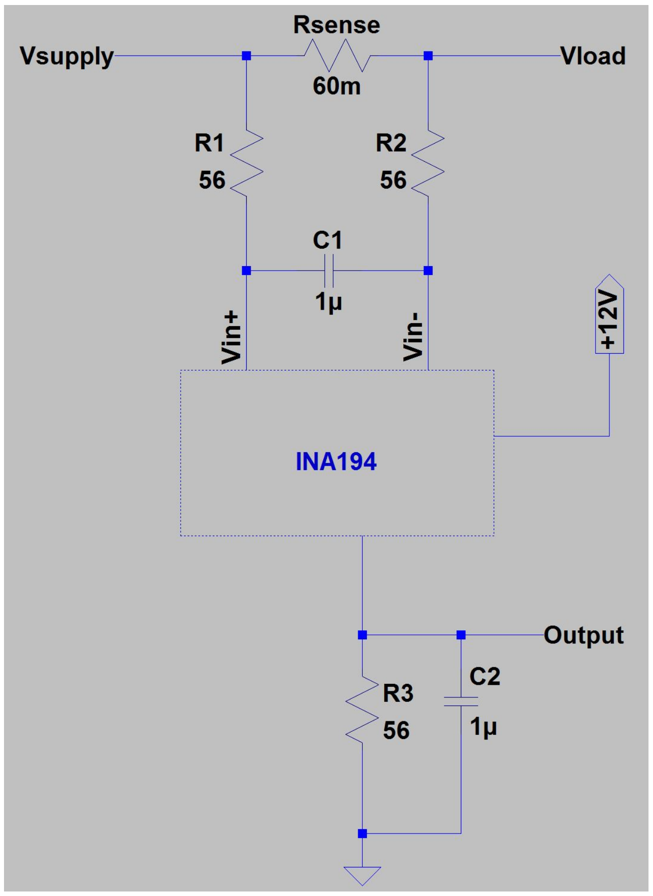

Figure 4: Overall current sensing circuit

In the ADC pin, the following expression shows the relationship between the battery current and the ADC pin voltage reading.

$$V_{ADC} = I_{Battery} \times \frac{1}{1023.0}$$

2.3. Importing the Sensing Circuit to the Buck Converter

After implementing the sensing circuitry, the sensing circuitry was added to the buck converter. The following diagram shows the entire circuitry that consists of the power stage, control stage, and sensing stage.

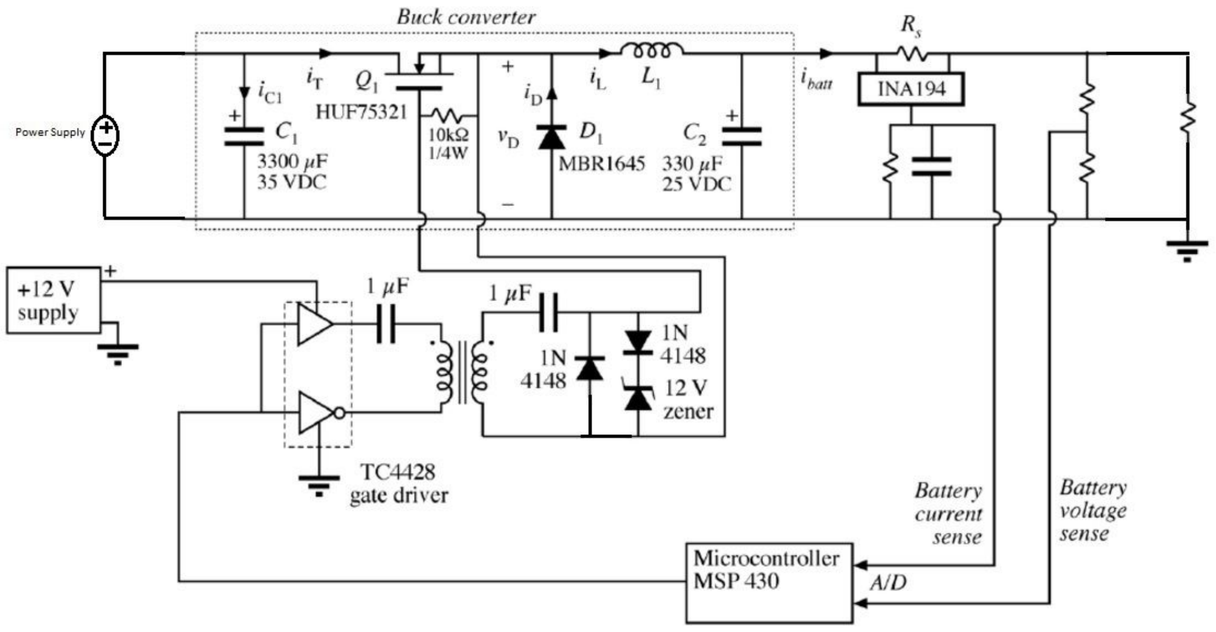

Figure 5: Sensing Circuitry

As previously mentioned, 3.3V zener diodes were connected directly to the MSP430 Microcontroller to prevent the signal from exceeding 3.3V; which damages the MSP430 Microcontroller. Note the the voltage and current sensing are connected to the following pins on the MSP430 microcontroller:

- Current sensing is connected to pin 6, which represents A1 ADC input.
- Voltage sensing is connected to pin 7, which represents A2 ADC input.

Finally, a code that finds the maximum power point by changing the duty cycle was implemented. The code works as the following:

1. Initial power reading.
2. Delay for some time to count for transition time.
3. Second power reading.

4. Finding whether the power changes or not.
    a. If power increases: change the duty cycle in the same direction.
    b. If the power decreases: change the duty cycle in the opposite direction.
5. Store the second power reading into the initial power reading.
6. Go to step 3.

    The complete code that does this can be found in section 5.1.

2.4. Testing the Buck Converter Inside the lab

After implementing the buck converter and writing the code, the following steps were performed to test the system and ensure that it works as expected.

Step 1: The first step, and before connecting the sensing circuitry to the MSP430 Microcontroller, is to test both voltage and current readings, and find the maximum readings. If the maximum reading is more than 1.5V, some modifications in the hardware will be made. The result of this step, and after changing the filter resistors in the current sensing circuitry, was ensuring that the maximum voltage reading at 13V in the output is 1.42V, and the maximum current reading at 8.5A in the output is 1.87V. Note that the 13V and 8.5A are the maximum voltage and current of the battery, respectively.

Step 2: The voltage reading and the current reading were connected to the MSP430 Microcontroller individually. For each reading, a code was written to change the duty cycle with respect to the change in either the voltage or the current. Then, the voltage was changed and the duty cycle was observed to see whether is changes as the voltage changes or not. Same thing happened with the current sensor. This step allows ensuring that both the current and the voltage can be read on the MSP430 Microcontroller without damaging the board.

Step 3: Both input, the voltage and the current, were read at the same time. In addition, the duty cycle was modified by the sum of both the voltage reading and the current reading. After that, the input voltage or the output resistance were changed to observe how the duty cycle changes as either the voltage or the current changes. This step ensures that the MSP430 Microcontroller can read both inputs and change the duty cycle as either one changes.

Step 4: A simple code that changes the duty cycle by 1% each run was written. This step is essential because the code needs to have an acceptable delay for the reading, and to ensure that the duty cycle doesn't change very fast.

Step 5: Finally, the complete code that finds the maximum power point was tested in the system. Note that the final code was a combination of the codes from the previous steps and some other things.

2.5. MPPT Results Inside the Lab

After ensuring that the code is working as anticipated, the buck was driven in conditions that are similar to the PV panel conditions. That is, the input voltage is 17.2V and the input current can goes up to 6.45A. Note that a resistive load was used in this step. The code ran successfully and the output was the expected output. The following plot shows the output voltage, current, and power at the MPPT inside the lab.

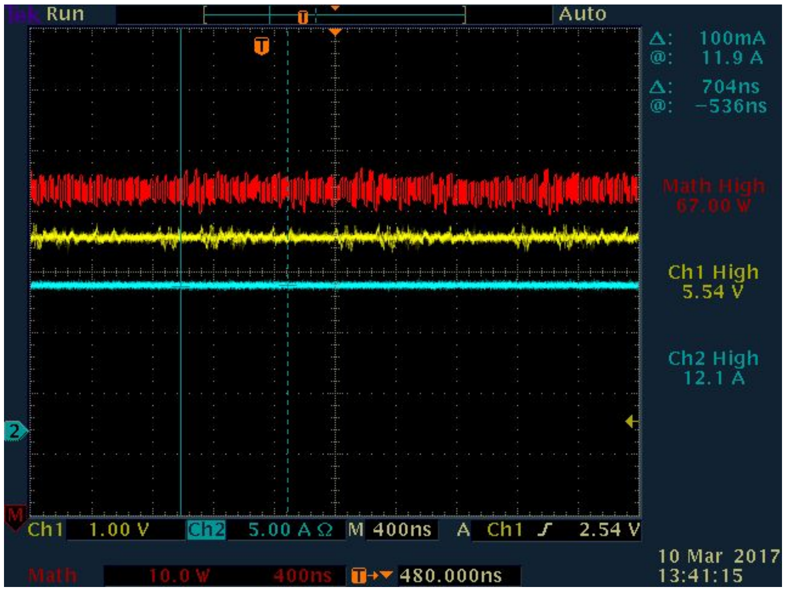

Figure 6: MPPT point inside the Lab

The maximum power read was 67.00W. The code increased the duty cycle to its maximum point since the greater the duty cycle is, the greater the output power is when using a resistive load. Note that the input voltage and current are: 10.02V and 3.54A. In addition, the following plots shows the current reading and the voltage reading in the MSP430 Microcontroller pins.

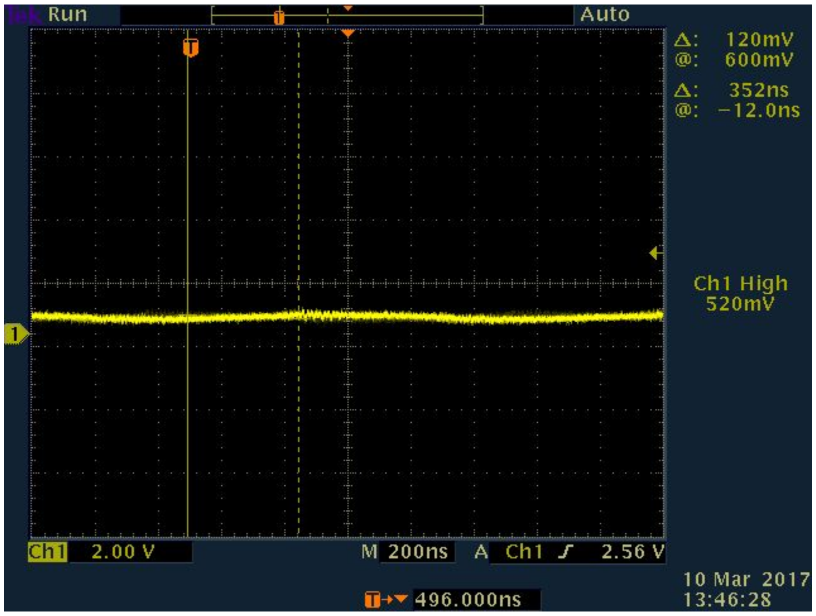

Figure 7: Voltage Sensing

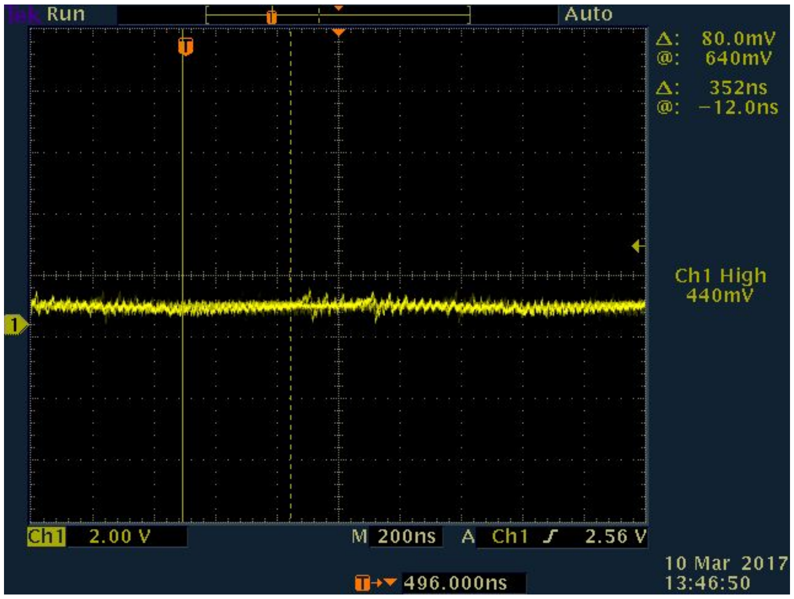

Figure 8: Current Sensing

Observation: As mentioned above, the current sensing and voltage sensing don't exceed the maximum voltage rate in the MSP430 Microcontroller.

3. Peak Power Tracking with the PV System Cart

After testing and debugging the Buck converter and the MPPT code inside the lab, the Buck converter was taken outside and tested with the PV panel. Noting that at the day when the data was taken (Friday March 10th, 2017), the weather was cloudy and there was partial sun. In addition, several hours spent trying to catch a good sun. As a result of that, the readings are not very accurate and they fluctuate rapidly. The following diagram shows the schematic of the system including the power stage, control stage, sensing stage, and the Microcontroller.

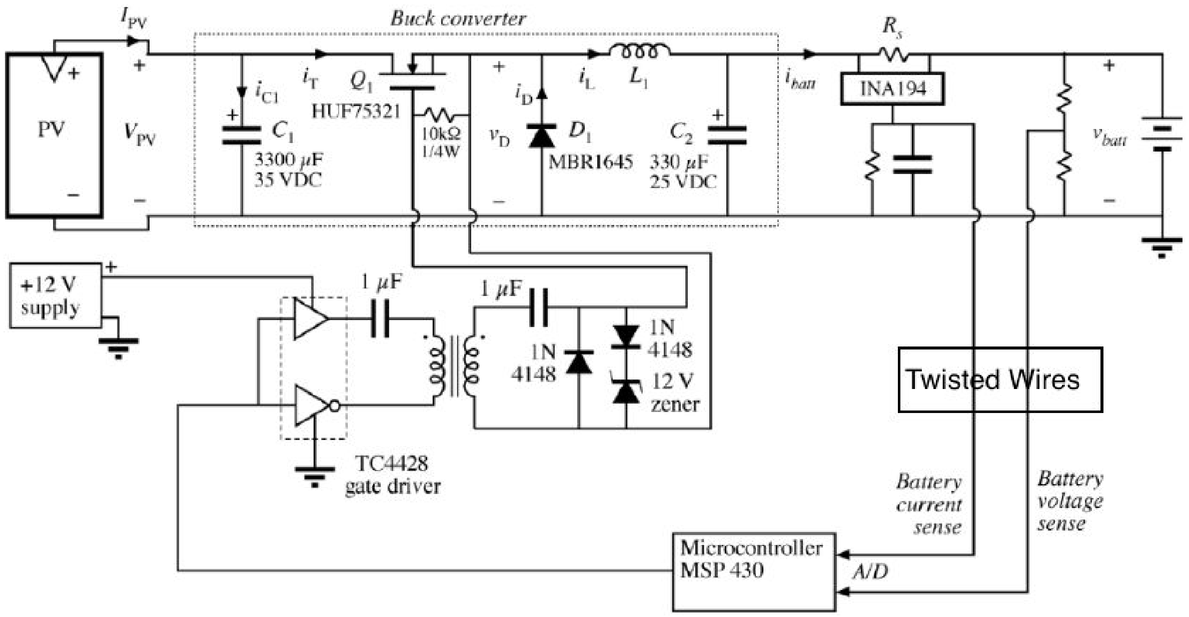

Figure 9: The full circuit (Buck with sensing circuitry)

3.1. Testing, Debugging, and Evaluation Procedure

In order to test, debug, and evaluating the overall system, the following steps were followed:

Step 1: The first step was testing the buck converter with a constant duty cycle. This step ensures that the system (excluding the code) is working even when connecting the sensing circuit. In addition, this tells that if there is an error while running the code, then the problem is in the code itself and not the hardware of the system.

Step 2: Running the code and observing whether the duty cycle changes or not when the energy of the sun changes, which was determined by observing when a cloud passes. This step ensures that the code is working, and there is a voltage and current reading from the sense circuitry. In addition, this ensures that the code makes the system stable, and the delay is enough to read accurate readings from the sensing circuitry to a good precision.

Step 3: Ensuring that the buck converter is operating in the maximum power point. This was tested by three ways: The first one is by seeing how the duty cycle changes as the sun energy changes. Second, by observing how the duty cycle goes back and forth between a point, which represents the maximum power point. Third, by comparing the duty cycle at the maximum power point with the modeled duty cycle that gives the maximum power point. After this test, the system works as anticipated and the code works perfectly.

Step 4: The last step when testing and debugging the system is by covering some PV cells and observing the change in the duty cycle. This happened multiple times with different number of shaded cells. In fact, while testing, covering all the PV panel was done and the duty cycle went to zero since there is no readings from the PV panel. After this test, it was concluded that the buck converter and the maximum power point works perfectly with the PV panel, and that the overall system is communicating with itself the expected way.

3.2. Data collected at Maximum Power Point

Unfortunately, and because the weather was cloudy at the time the data was collected, the power is a lot lower than usual. However, when testing the system, there was sun and a conclusion was drew that the system in general works fine. The following plots show the PV panel voltage and the MSP430 PWM output signal of the duty cycle at different times. Please take into consideration that the weather was cloudy, which made taking the data a bit difficult. In addition, an oscilloscope was connected to the output of the battery, which made the battery voltage less than 13V.

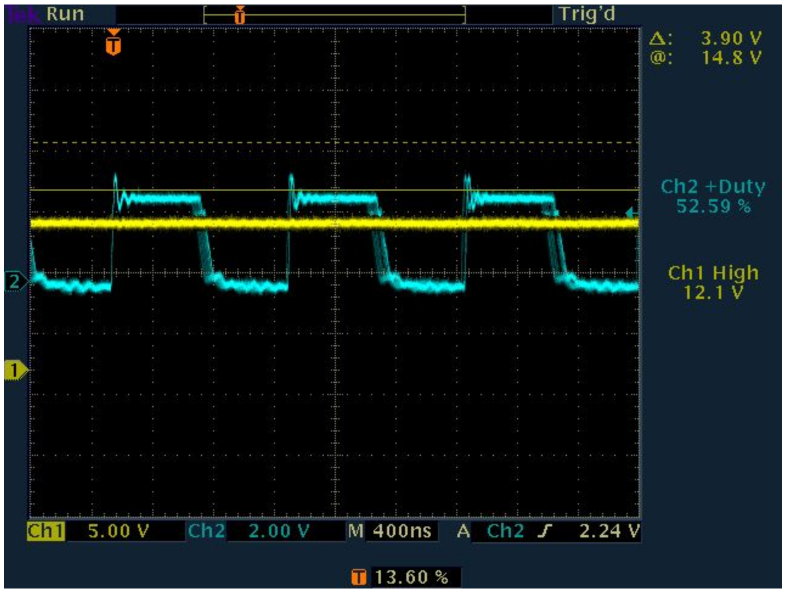

Figure 10: The duty cycle with some sun

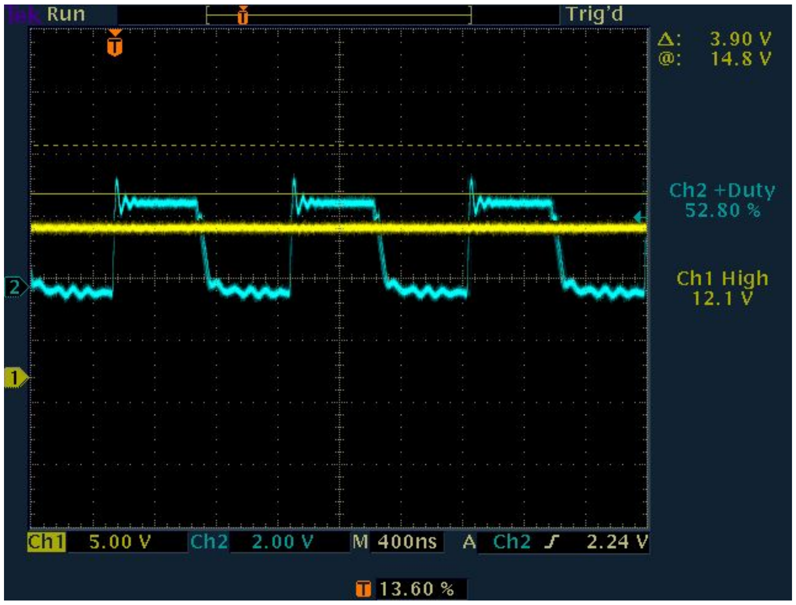

Figure 11: Another reading showing that the duty cycle is stable for the MPPT

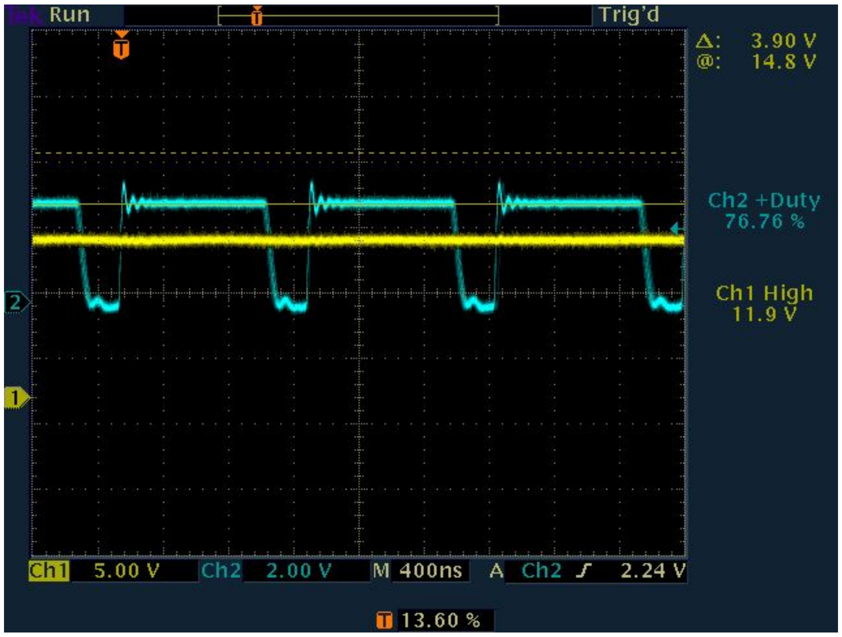

Figure 12: The duty cycle changes with the sun situation

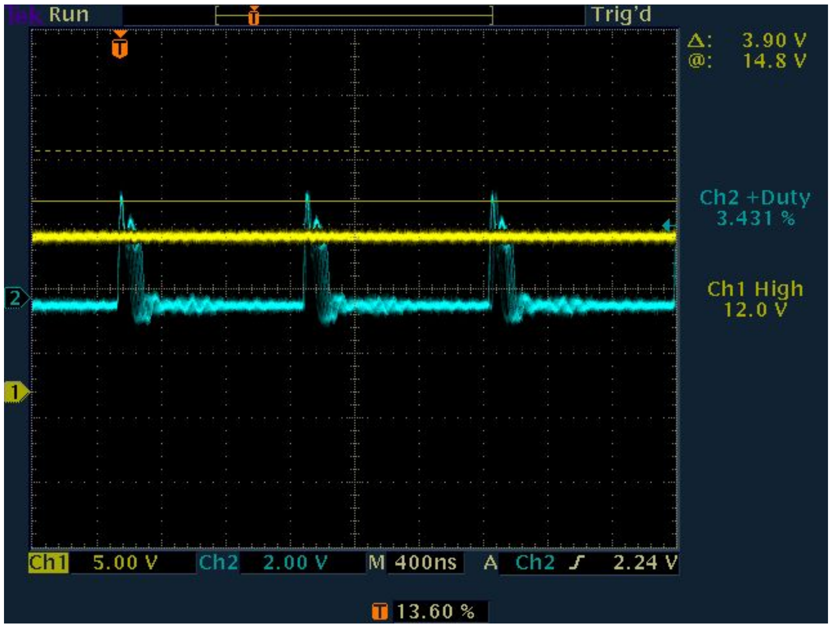

Figure 13: Reading with a little bit of a sun and a lot of cloud

In addition to observing the duty cycle, the output power was observed using an oscilloscope with both probes; one for the output voltage and one for the output current. After observing and ensuring that this is the maximum power point, the following plots were taken to show the output power at the maximum power point.

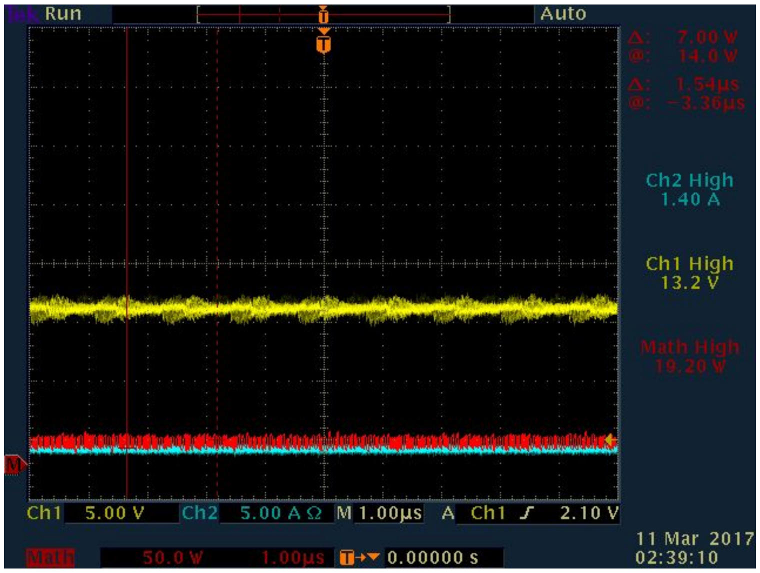

Figure 14: First reading at the maximum power point

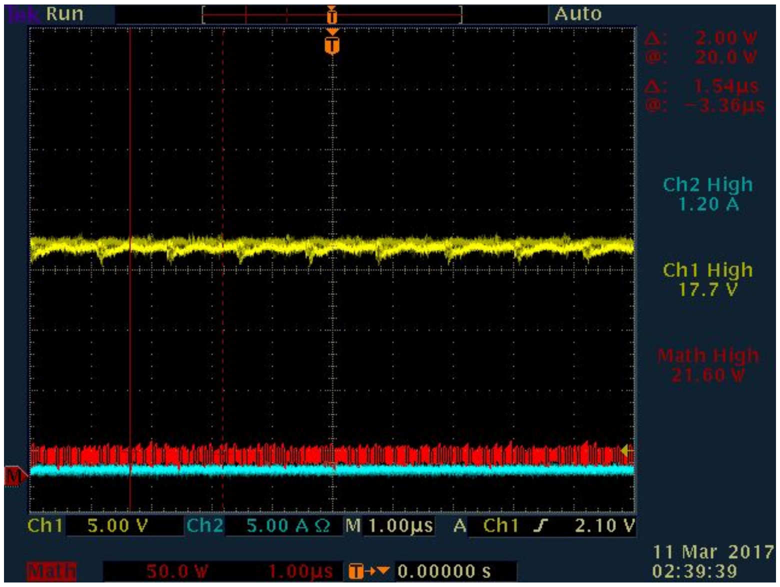

Figure 15: Second reading at the maximum power point

Unfortunately, and due to the weather, the data collected are limited, but the observation was very rigorous. A lot of time was spent waiting for the sun, and the above readings are the clearest. In addition, the data collected shows a relatively low power due to the weather. The following table shows the input and output readings of the buck converter and the PV panel.

V_{panel}	I_{panel}	P_{panel}	$V_{battery}$	$I_{battery}$	$P_{battery}$
13.2V	1.4A	19.2W	12.34V	1.54A	18.9W
17.7V	1.2A	21.6W	12.31V	1.64A	20.2W

The efficiency of the converter can be calculated by dividing the output power over the input power. The efficiency is calculated and is equal to approximately 97%. This efficiency is better than the efficiency calculated without the maximum power point code.

3.3. Comparison between Battery Power With the Buck and the Direct Conversion

From the previous lab, the energy converted with the buck converter is greater than the energy converted with direct conversion. That is because the direct conversion operates at 100% duty cycle, while the maximum power point is with a duty cycle that is usually less than 100%.

This was concluded from the power curve found on the battery. Subsequently, when using the maximum power point code, the output power is greater than the power from direct power conversion. Unfortunately, because of the cloudy weather, the output voltage recorded using the maximum power point is relatively small. However, while observing, an output power of around 65W was seen but not recorded.

3.4. The PV panel with 4 Cells Shaded

When shading 4 of the PV cells, the energy converted went down and the following plot was recorded. At the time in which the measurements were taken, the weather was very cloudy; which caused the very small current (shown as negative) and hence zero output power (also shown as negative).

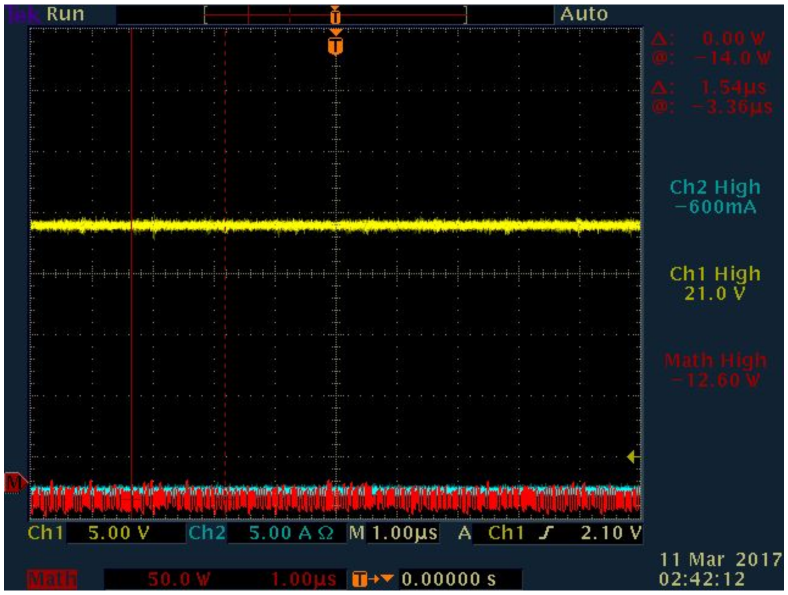

Figure 16: Output power when 4 cells are shaded

Theoretically, the output power using the buck converter is greater than that using direct conversion even with shaded cells. That is because the direct conversion operates at 100% duty cycle, while the maximum power point is with a duty cycle that is usually less than 100%. This was concluded from the power curve found on the battery. Unfortunately, and because of the cloudy weather, there were no clear experimental data showing that. However, this was observed when testing the functionality of the buck converter before recording any readings and with more sun.

Improving the system under shading condition: One way to improve the performance of the system when shading is occurring is by modifying the code to count for any slight change. In addition, one might consider adding a photovoltaic source in front of the PV panel in order to count for any change in the shading condition. By doing that, even when shading, the readings will be better and the power conversion will be greater.

4. Conclusion

To sum up, a voltage and current sensing circuitry was designed, implemented, and tested. In addition, a maximum power point code that changes the duty cycle of the gate driver was written. This code ensures that the buck converter is operating in the maximum power point, and the greatest possible energy conversion is occurring. The maximum power point was added to the MSP430. In addition, the maximum power point code was tested initially inside the lab. The maximum power point code was successfully working with the PV panel. Finally, the buck converter works in the maximum power point all the time to ensure highest output power with a great efficiency.

5. Appendix

5.1. MPPT Code

```
#include <msp430.h>

void SetVcoreUp (unsigned int level);

/*
 * main.c
 */

// Declaring variables that are needed to fins the maximum power point
float vbat = 0.0;
float ibat = 0.0;
float pout1 = 1000000.0;
float pout2 = 0.0;        // Initial reading so that the initial change is increasing the duty cycle
float dutyCycle = 134.0;
int direction = 1;
int counter = 0;

void main(void) {
   volatile unsigned long i;      // Declare counter variable
   WDTCTL = WDTPW | WDTHOLD;           // Stop watchdog timer

   P1SEL |= BIT6;                              // Set P1.6 to output direction (Timer D0.0 output)
```

```
    P1DIR |= BIT6;
    P1SEL |= BIT7;                              // Set P1.7 to output direction (Timer D0.1 output)
    P1DIR |= BIT7;
    P2SEL |= BIT0;                              // Set P2.0 to output direction (Timer D0.2 output)
    P2DIR |= BIT0;
    P1DIR |= 0x01;                              // Set P1.0 to output direction (to drive LED)
    P1OUT |= 0x01;                              // Set P1.0  - turn LED on
    __delay_cycles(500000);
    P1OUT ^= 0x01;                              // Toggle P1.0 using exclusive-or function  - turn
LED off

    // Increase Vcore setting to level3 to support fsystem=25MHz
    // NOTE: Change core voltage one level at a time..
    SetVcoreUp (0x01);
    SetVcoreUp (0x02);
    SetVcoreUp (0x03);

    // Initialize DCO to 25MHz
    __bis_SR_register(SCG0);                 // Disable the FLL control loop
    UCSCTL0 = 0x0000;                      // Set lowest possible DCOx, MODx
    UCSCTL1 = DCORSEL_6;                     // Select DCO range 4.6MHz-88MHz operation
    UCSCTL2 = FLLD_1 + 763;                 // Set DCO Multiplier for 25MHz
                                    // (N + 1) * FLLRef = Fdco
                                    // (762 + 1) * 32768 = 25MHz
                                    // Set FLL Div = fDCOCLK/2
    __bic_SR_register(SCG0);                 // Enable the FLL control loop

    // Worst-case settling time for the DCO when the DCO range bits have been
    // changed is n x 32 x 32 x f_MCLK / f_FLL_reference. See UCS chapter in 5xx
    // User Guide for optimization.
    // 32 x 32 x 25 MHz / 32,768 Hz = 782000 = MCLK cycles for DCO to settle
    __delay_cycles(782000);

     // Configure TimerD in Hi-Res Regulated Mode
    TD0CTL0 = TDSSEL_2;                     // TDCLK=SMCLK=25MHz=Hi-Res input clk select
    TD0CTL1 |= TDCLKM_1;                    // Select Hi-res local clock
    TD0HCTL1 |= TDHCLKCR;                                       // High-res clock input
>15MHz
    TD0HCTL0 = TDHM_0 +                                   // Hi-res clock 8x TDCLK = 200MHz
                    TDHREGEN +                                  // Regulated mode, locked
to input clock
                    TDHEN;                                // Hi-res enable
```

```
   // Wait some, allow hi-res clock to lock
   P1OUT ^= 0x01;                                              // Toggle P1.0 using
exclusive-OR, turn LED on
   // __delay_cycles(500000);



   while(REFCTL0 & REFGENBUSY);
   REFCTL0 |= REFVSEL_0 + REFON;
   _delay_cycles(75);

   P1OUT ^= 0x01;                                              // Toggle P1.0 using
exclusive-OR, turn LED off

   P1OUT ^= 0x01;                    // Toggle P1.0 output

   while(!TDHLKIFG);                                  // Wait until hi-res clock is locked


   // Configure the CCRx blocks
   TD0CCR0 = 224;                    // PWM Period. So sw freq = 200MHz/2500 = 80 kHz

   while(1){

        _delay_cycles(250000);

        readVoltage();
        readCurrent();

        pout2= ibat * vbat;
        /********************************/
        /*    Checking Power Change     */
                /********************************/
                // Each run, change the duty cycle by almost +/-0.1%

                // If the power increases and the direction is clockwise
                // Don't change the duty cycle and increase the duty cycle
                if( pout2 > pout1 && direction == 1 && dutyCycle != 224.0)
                {
                        dutyCycle = dutyCycle + 2;
                }
                // If the power is decreasing and the direction is clockwise
                // Change the direction and decrease the duty cycle
                else if( pout1 > pout2 && direction == 1 && dutyCycle != 0.0)
                {
                        dutyCycle = dutyCycle - 2;
                        direction = 0;
                }
                // If the power increases and the direction is counterclockwise
                // Don't change the duty cycle and decrease the duty cycle
                else if( pout2 > pout1 && direction == 0 && dutyCycle != 0.0)
                {
                        dutyCycle = dutyCycle - 2;
                }
                // If the power is decreasing and the direction is counterclockwise
                // Change the direction and increase the duty cycle
                else if( pout1 > pout2 && direction == 0 && dutyCycle != 224.0)
                {
                        dutyCycle = dutyCycle + 2;
                        direction = 1;
                }

        TD0CCR1 = dutyCycle;
        pout1 = pout2;
                TD0CCTL2 = OUTMOD_7 + CLLD_1;            // CCR2 reset/set
                TD0CCR2 = 134;                           // CCR2 PWM duty cycle of 500/2000 = 25%
                TD0CTL0 |= MC_1 + TDCLR;                 // up-mode, clear TDR, Start timer
   }
}

  void SetVcoreUp (unsigned int level)
  {
        // Subroutine to change core voltage
   // Open PMM registers for write
   PMMCTL0_H = PMMPW_H;
   // Set SVS/SVM high side new level
   SVSMHCTL = SVSHE + SVSHRVL0 * level + SVMHE + SVSMHRRL0 * level;
   // Set SVM low side to new level
   SVSMLCTL = SVSLE + SVMLE + SVSMLRRL0 * level;
   // Wait till SVM is settled
   while ((PMMIFG & SVSMLDLYIFG) == 0);
   // Clear already set flags
   PMMIFG &= ~(SVMLVLRIFG + SVMLIFG);
   // Set VCore to new level
   PMMCTL0_L = PMMCOREV0 * level;
   // Wait till new level reached
```

```
    if ((PMMIFG & SVMLIFG))
      while ((PMMIFG & SVMLVLRIFG) == 0);
    // Set SVS/SVM low side to new level
    SVSMLCTL = SVSLE + SVSLRVL0 * level + SVMLE + SVSMLRRL0 * level;
    // Lock PMM registers for write access
    PMMCTL0_H = 0x00;
}

  void readVoltage()
  {
      // A2 ADC input (Voltage Sensing) - Pin 7
       ADC10CTL0 = ADC10SHT_2+ADC10ON;

       ADC10CTL1 = ADC10SHP + ADC10CONSEQ_0;

       ADC10CTL2 = ADC10RES;
       ADC10MCTL0 = ADC10SREF_1 + ADC10INCH_2;

                    ADC10CTL0 |= ADC10ENC + ADC10SC;
                    while(ADC10CTL1 & ADC10BUSY);
                    TD0CCTL1 = OUTMOD_7 + CLLD_1;           // CCR1 reset/set
                    vbat = ADC10MEM0;
  }

  void readCurrent()
  {
                    // A1 ADC input (Current Sensing) - Pin 6
                    ADC10CTL0 = ADC10SHT_2+ADC10ON;

                    ADC10CTL1 = ADC10SHP + ADC10CONSEQ_0;

                    ADC10CTL2 = ADC10RES;
                    ADC10MCTL0 = ADC10SREF_1 + ADC10INCH_1;

                    ADC10CTL0 |= ADC10ENC + ADC10SC;
                    while(ADC10CTL1 & ADC10BUSY);
                    TD0CCTL1 = OUTMOD_7 + CLLD_1;           // CCR1 reset/set

                    ibat = ADC10MEM0;
  }
```