Peak Power Tracker Circuit Description (from www.timnolan.com)

V1.00 2/14/03 First pass at it.

Circuit Description:

The Peak Power Tracker is a microprocessor controlled DC/DC step down converter used by a solar power system to charge a 12v battery. It steps the higher solar panel voltage down to the charging voltage of the battery. The microprocessor tries to maximize the watts input from the solar panel by controlling the step down ratio to keep the solar panel operating at its Maximum Power Point.

The PPT circuit is divided into two basic parts: the DC/DC converter and the microprocessor control. The buck mode DC/DC converter is made up of the synchronous MOSFET switches Q2 and Q3 and the energy storage devices inductor L1 and capacitors C1, C2, C9 and C10. Both the high side and low side MOSFETs are IRFZ44 N-fets. N-fets were chosen for their low Rdson to reduce resistance losses in the DC/DC converter. The input(C9, C10) and output capacitor (C2) are low ESR capacitors to handle the large current pulses from the switching DC/DC converter. The inductor value is 33uH and it is sized to handle 11 amps. D1 is a fast recovery diode used to conduct the circulating current while the low side MOSFET is turning on. C15 and R8 are a snubber network used to cut down on some of the electrical noise generated by the switching current in the inductor. R25 drains the current out of the gate on the low side switch to make sure it is off when the power is shut off. The input power connector (to the solar panels) is the screw terminal J4. J3 is the output screw terminal connector (to the battery). F1 is the 10A safety fuse.

The third MOSFET Q1 is added to allow the system to be turned off when connected to a battery. In the PPT Q2 and Q3 are shut off at night when there is not enough solar power to charge the battery. The microprocessor also should shut off when there is no longer any solar power to run it. However, the body diode in the high side MOSFET Q2 will conduct power in the reverse direction from the battery to the rest of the PPT and the system can never be shut off. Q1 is installed so its body diode blocks the reverse current flow. Q1 is driven by the top gate drive of the LT1185 MOSFET driver the same as the high side switch Q2. D2 keeps the gate on Q1 charged so Q1 is on all the time instead switching like Q2. This is more efficient because Q1 will not be switching so it dissipates less power. R24 drains the charge on the gate when the driver is shut off so the system will shut off.

The synchronous MOSFET gate driver U2 is an LT1158. It drives the high and the low side MOSFETs using the PWM signal from the microprocessor (on IN pin 6). D4 and C3 are part of the bootstrap circuit that generates the high side gate drive voltage for Q1 and Q2. When a N-fet MOSFET is used as the high side switch the gate voltage to drive it must be at 10v greater than the source voltage. Since the source voltage is the input voltage on the high side MOSFET the gate driver chip must generate a higher voltage that the input voltage to turn on the high side switch. The bootstrap circuit doubles the input voltage so the high switch can be driven on. However this bootstrap circuit only works when the MOSFETs are switching. In some cases the microprocessor

turns the high side switch full on to make a direct connection between the solar panel and the battery. The LT1158 has a special charge pump that kicks in to continue to provide the higher than input voltage to keep the the high side MOSFET on even when the MOSFETs are no longer switching. D3 is to keep the battery current from flowing into the LT1158 when the system off. This keeps the MOSFETs from turning on with the battery connected even when the system is off. Please refer to the LT1158 data sheet for more information on the MOSFET driver (LT1158.pdf).

The DC/DC converter is controlled by the PIC16F876 microprocessor U5 (see data sheet for more complete information, link to data sheet). The microprocessor is clocked at 20Mhz by the crystal XTAL1. The 5v to power the PIC16F876 is generated by a 78M05 (U3) linear voltage regulator. C4 and C7 are power supply smoothing capacitors. The PWM output of the PIC on pin 13 is used to control the duty cycle of the DC/DC converter which sets its voltage conversion ratio. The frequency of the PWM is set to 100Khz by the PIC software. The PWM duty cycle is controlled by the PIC software to optimize the power output from the solar panels. The enable pin (pin 4) on the MOSFET driver (LT1158 U2) is controlled by the output pin 14 of the PIC.

The PIC calculates the solar watts generated by reading the voltage and current of the solar panels through the A/D converter on pins 2 and 3. R3 and R4 make a voltage divider used to drop the input voltage into the 5v range that can be read by the PIC. R1 is the current sense resistor. The small voltage generated by R1 as the input current flows is amplified and buffered by an LM358 op-amp (U6). The op-amp is configured as a difference amplifier with a gain of 10 set by R10, R11, R18 and R20.

The PIC calculates the battery watts with the exact same circuit. R5 and R6 make up the voltage divider used to read the output voltage. R2 is current sense resistor. U1 (LM358) is the difference amplifier with a gain of 10 set by R13, R14, R21 and R22.

The PIC uses the LEDs D5-7 as indicators to show the software state. The jumpers (U4) allow the user to set software configurations. The RJ11 connector (J2) is the debugging interface for the PIC. The Microchip development environment MPLAB can communicate through this interface to download and program code into the PIC and run simple debugging commands. This allows the developer to write and debug code in circuit without ever removing the processor.

The PIC has a build in serial port that is used to output data from the PPT so it can be logged externally. The serial port can also be used for simple control of the PPT during debugging. The serial port is pins 17 and 18 of the processor. The MAX232A (U8) is used to convert the TTL level serial signals from the processor to RS232 signals for external interfacing. The MAX232 has built in charge pump to generate RS232 compatible voltages using capacitors C18-22. The connector J1 is a DB9 for the serial port.

Theory of Operation:

The Peak Power Tracker is basically a microprocessor controlled DC/DC converter. The PPT is used in solar panel battery charging systems to increase the efficiency of the system by closing matching the input voltage of the solar panel to the output voltage of the battery.

Solar panels designed for charging 12v batteries actually generate the most watts running at around 17v. When connected directly the battery pulls the solar panel operating voltage down to 12v of the charging battery. By using the DC/DC converter to connect the solar panel to the battery it allows the solar panel to run at the higher voltage that is its maximum power output (maximum power point, MPP) than when connected directly to the battery (see my article).

If the MPP of the solar panel was fixed then it would be simple to design a DC/DC converter with a fixed conversion ratio to convert the MPP voltage down to the battery voltage. However, the MPP changes depending on the amount of light and the temperature of the solar panel. The PIC microprocessor takes care of this by measuring the input watts from the solar panel and changing the conversion ratio to keep the solar panel at its MPP.

The DC/DC converter is a buck converter which means it takes a higher input voltage and converts it to a lower output voltage. Since this a switching converter topology it doesn't dissipate any power internally (except for some small resistive losses). That means the output power is equal to the input power, watts in = watts out. So if the watts stays the same and the voltage drops then output current must be greater than the input current.

This is a synchronous switching converter, meaning it has a low side and a high side MOSFET switch. This is slightly more efficient than a converter with just a high side switch. N-fets are used for all the MOSFET switches because they have a lower Rdson which means there is less resistive losses in the switches. However, this causes problems with driving the high side MOSFET. To fully turn on an N-fet must have a gate to source voltage (Vgs) of roughly 8v or greater. The high side switch in the DC/DC converter has its source pin tied to the input voltage so we have to generate a gate drive voltage of at least 8v higher than the input voltage. This is taken care of by the bootstrap capacitor connected to the MOSFET driver LT1158. When the low side switch is on the bootstrap capacitor is charged to the input voltage. Then when the high side switch is on the cap adds this voltage to the input voltage so the doubled input voltage now drives the high side switch. This works as long as the converter is switching but occasionally the high side switch is turned on full time to directly connect the solar panel to the battery. In this case the LT1158 has an internal charge pump that kicks in to keep the Vgs high enough to keep the high side switch on.

The high side gate drive voltage is also used to turn on the third MOSFET (Q1) that is added to block reverse current flow from the battery. The body diode in the high side MOSFET allows current to flow from the battery back to the rest of the system even when the MOSFET is off. The third MOSFET is connected so its body diode blocks the current from the battery. It turns on when the high side switch is on so the resistive losses are minimized.

The microprocessor, a PIC16F876, controls the conversion ratio of the of the DC/DC converter. The PIC generates a 100kHz PWM signal with its internal PWM circuit. The duty cycle of the PWM signal sets the ratio of the on time for the high side MOSFET switch versus the on time of the low side MOSFET switch. The ratio of the on time of the switches sets the conversion ratio of the input to the output voltage of the DC/DC converter (see article xxx for the mathematical explanation).

The PIC tries to set the conversion ratio of the DC/DC converter to allow the solar panels to operate at their Maximum Power Point (MPP, see my article for an explanation of MPP). The microprocessor does this using an iterative algorithm to maximize the input watts of the solar panels (see Software Description for explanation of this algorithm). The input watts from the solar panel are calculated by measuring the voltage and current with the PIC's A/D inputs and multiplying internally to get the watts. The solar panel voltage runs through a resistor divider network to get it down in the 5v range of the PIC's A/D converter. The solar panel current is measured with a current sense resistor and difference amplifier to condition the signal before it is read by the PIC's A/D.

The PIC can also read the voltage of the battery with another voltage divider and the current flowing into the battery with another sense resistor and difference amplifier. The battery voltage is used by the PIC to tell when the battery is fully charged. If the battery is charged the microprocessor rolls back the charging current to keep from overcharging the battery.

The microprocessor transmits out the serial port all the values it has measured and calculated (volts, amps and watts) once a second to be logged by an external computer. Every 20 seconds the PIC set the conversion ratio of the DC/DC converter to 100% or full on. This simulates a direct connection between the solar panels and the battery. The solar panel input watts are measured and compared the solar panel watts at MPP to get the gain or boost of the Peak Power Tracker.

Problem Areas and Future Improvements:

I've test the DC/DC converter up to 100 watts, 20 volts and 5 amps input. The inductor is rated at 11 amps and the MOSFETs up 50 amps so the converter should handle up to 200 watts. The MOSFETs need adequate cooling, the PCB is designed so that they can be mounted up against the side of a metal box or else they should be mounted to heat sinks. If MOSFETs with metal tabs are mounted to the same heatsink they should be mounted with insulating pads because the tabs are at different voltages.

It should be possible to use the PPT with 24v solar panels (or two 12v panels in series) but I have not tested it. You might want to do this if you have a long cable run from the panels to cut down on the wire losses. The input capacitors are rated for 25 volts for 12v panels, use 50v capacitors for two 12v panels in series because they can generate up to 40v. You will also have to limit the gate drive voltage generated by the bootstrap capacitor connected to the MOSFET driver. Most MOSFETs have a Vgs limit of 20v. The bootstrap circuit uses the input voltage to generate the gate drive, when the input voltage is greater than 20v then the gate drive voltage will exceed the 20v Vgs limit. I suggest using a 78M15 15v linear voltage regulator connected to the diode D4 instead of the input voltage to limit the gate drive voltage to 15v.

The solar panel current sense resistor is on the ground side of the solar panel power input. This makes it easier to use a difference amp to measure the current. This could cause problems however in measuring the current. If the solar panel and the battery share a ground or earth connection that is not through the PPT then some or all of the input current may flow along that alternative path and not through the sense resistor. This will give false current readings and therefore false wattage readings so the PPT will not be able to find the MPP. The first solution to this problem is to make sure the there is on

ly one ground path through the PPT in your solar power system. The other solution to this problem is to use a high side current sensor that puts the current sense resistor in the positive side of the solar power input. I've had good luck with the MAX4173 High Side Current Sensor (datasheet link) but it only comes in a surface mount version and I wanted to keep this version of the PPT with all through hole components. The next circuit board version I layout will use the surface mount MAX4173 high side current sensor.

Peak Power Tracker Software Description (from www.timnolan.com)

V1.00 2/14/03 First pass at it.

This document will describe the software used to operate the Peak Power Tracker designed by Tim Nolan. The code is written in C and the file name is PPT.C. Please see my website www.timnolan.com to find this software. This software was written under MPLAB for Windows, v4.12.00 from Microchip, www.microchip.com. The software was debugged and the chip programmed by Microchip ICD development tools, www.microchip.com/1010/pline/tools/picmicro/icds/mplabicd/index.htm. The C code was compiled in MPLAB with the CCS PCM C Compiler, Version 2.703, www.ccsinfo.com/picc.shtml.

The Peak Power Tracker is a microprocessor controlled DC/DC converter. It is used to match the Maximum Power Point voltage of the solar panels to the charging voltage of the battery. Take a look at my article, Peak Power Tracker, for a more in depth discussion about the theory behind peak power tracking. Also take a look at the Peak Power Tracker hardware description document, (ppt_hardware.doc). Both these documents are at my website www.timnolan.com.

Microprocessor Development Environment:

The microprocessor used to control the DC/DC converter is a Microchip PIC16F876. The PIC was chosen because of the excellent onboard peripherals and easy to use development environment. The onboard peripherals include A/Ds for reading the volts and amps, PWM to set the duty cycle of the DC/DC converter, a serial port to output data values for logging, and flash program memory to simplify software development (see PIC16F87x data sheet, 30292c.pdf).

The development environment is MPLAB from Microchip, I'm using v4.12.00 for Windows. MPLAB is free, download it from Microchip's website, www.microchip.com. With MPLAB you can write PIC assembly language code with the free download package. However, it's much easier and quicker to write code in a higher level language like C so I bought a C compiler from CCS, www.ccsinfo.com/picc.shtml. I'm using CCS PCM C Compiler, Version 2.703, currently CCS is selling v3.00 for $125. The CCS C compiler has a lot of nice built in functions to access the PIC hardware which make it easy to quickly write useful programs. It is not however strictly ANSI C compatible. A college who uses gcc (a very widely used C compiler) was somewhat disappointed with the CCS C compiler calling it "baby C" but for PIC programs, which are usually very simple, I think it works fine. I especially like the floating point math support which makes it easy to output and log actual volt and amp values.

Once the C code is compiled it still has to be programmed in to the microprocessor. The PIC16F876 has onboard program flash and an in circuit program feature. This allows you to reprogram the PIC by downloading the new software into flash program memory using three pins on the chip. So you don't have to remove the chip from your circuit if you provide a connector for these three pins. I'm using the ICD development system from Microchip,

www.microchip.com/1010/pline/tools/picmicro/icds/mplabicd/index.htm. The ICD connects to the serial port of your PC and works with the MPLAB development environment. Besides allowing you to download and program your PIC in circuit the ICD also lets you do limiting debugging. You can look at memory and registers, set limited breakpoints, single step and do C source level debugging. The ICD is available from Digikey, www.digikey.com. I'm using the older version of the ICD, the new ICD2 is now available from Digikey for $159. I find the combination of MPLAB, CCS C compiler and the ICD debugger to be a very easy and effective way of developing PIC code.

Software Description:

The software for the Peak Power Tracker is in the file ppt.c on my website, www.timnolan.com. This section of this document will describe the theory of operation for the PPT software, please refer to the source file ppt.c.

Starting in main() section of the C code, the first we do is define the constants and variables. Notice that the CCS C compiler supports floating constants and variables. After the variables are defined the next section of software is used to set up the hardware. The code uses the built in CCS compiler functions to access the PIC hardware. This code sets up the A/D ports for reading and the timer1 as a PWM port with a frequency of 100kHz.

```
output_low(PIN_C3);                    // turn off FETs by clearing EN pin on FET driver
setup_port_a(ALL_ANALOG);              // set up A/D channels
setup_adc(adc_clock_div_32);           // start A/D clocks
setup_timer_2(T2_DIV_BY_1, 49, 1);     // pwm frequency 100kHz
setup_ccp1(CCP_PWM);                   // Configure CCP1 as a PWM
```

Please see the compiler manual for more information on the hardware specific functions.

These three instructions set the duty cycle of the DC/DC converter to 85% to start the program. This initial condition is chosen because it is in the range of duty cycles that the DC/DC converter is normally running at.

```
pwm_max = 200;                         // maximum value for PWM based on frequency
pwm100 = 85;                           // set pwm to 85% to start
pwm1_duty100(pwm100, pwm_max);         // set pwm duty cycle to pwm100 value
```

The actual duty cycle is set in the function "pmw1_duty100" which we'll describe more in depth in the function section. The duty cycle is stored as a 0-100 or percentage value in the variable "pwm100".

The "while(TRUE)" statement starts a continuous loop where the main work of this program is done. The first thing checked is the battery voltage to see if it has gone over it's fully charged value set by the constant CHARGED.

```
if (batvolts > CHARGED) {              // if batvolts is > charged then battery is charged so
   delta = NDELTA;                     // start rolling back current until batvolts < charged
   output_high(PIN_C5);                // turn on red led if battery at maximum voltage
}
```

If the battery voltage is greater than its fully charged voltage then set the variable "delta" to the constant -1 and turn on the red LED to show that the battery is charged. "Delta" will be added to "pwm100" later on so in effect the software is decreasing the duty cycle of the DC/DC converter which lowers the charging current of battery which lowers the battery voltage to keep the battery from overcharging. This is a very simple charge control algorithm that only looks at limiting the maximum voltage of the battery. There are many more sophisticated charge control algorithms that could be implemented.

If the battery is not yet fully charged then go on and implement the "hill climbing" peak power tracking algorithm. Please see my article on Peak Power Tracking for a more general description of the "hill climbing" algorithm.

```
else if (old_watts > watts) {          // else if battery not charged yet change the value of
  delta = -delta;                      // delta to make pwm increase of decrease to maximize watts
  output_low(PIN_C5);                  // turn off red led if battery not at maximum voltage
}
```

The value in "delta" gets added to "pwm100" in the next few lines of code and has the effect of increasing or decreasing the duty cycle of the DC/DC converter. The value "old_watts" is the amount of watts the solar panel is generating last time through the main program loop. If that value is greater than the current measured value of watts then that means that the last time the duty cycle was changed it was in the wrong direction, the software was going "down the hill" of maximum watts. If this is true then flip the sign of "delta" so that the direction of duty cycle change is now "going up the hill" of maximum watts. Also turn off the red if it was on to show the battery is no longer at maximum voltage. If the opposite is true, the latest value of watts from the solar panel is greater than the previous value then don't change the sign of "delta", the software is changing the duty cycle of the DC/DC in the correct direction to maximize the watts generated.

Once the software decides whether or not to change the sign of "delta" then "pwm100" has to be adjusted by that value and "old_watts" has to be set with the current value of "watts" for the next pass through the loop.

```
old_watts = watts;                     // load old_watts with current watts value for next time
pwm100 += delta;                       // add delta to change PWM duty cycle for PPT algorythm
if (pwm100 > 100) pwm100 = 100;        // check limits of PWM duty cyle and set range to 0-100
  else if (pwm100 < 0) pwm100 = 0;
pwm1_duty100(pwm100, pwm_max);         // call function to set PWM duty cyle in PIC hardware
```

Also "pwm100" has to be range checked to make sure it does go below zero or above 100.

The next section of code will read in the amps and volts flowing in from the solar panel and the amps and volts flowing out to the battery. The solar panel amps and volts are used to calculate "watts" for the "hill climbing" algorithm. The battery volts is used to see if the battery is fully charged. Right now the battery amps and battery watts (calculated from battery volts and amps) is not used for control functions, they are just reported out the serial port.

```
current = average_ad(AMP_CHAN)/40.885;     // read current and voltage from solar panel and going to
voltage = average_ad(VOLT_CHAN)/18.2887;   // battery, use average_ad function return average value
batamps = average_ad(BAMP_CHAN)/41.3428;   // as a float, divide by constant to scale into actual
batvolts = average_ad(BVOLT_CHAN)/20.4831; // volts and amps. These constants were measured in the
                                           // actual prototype hardware.
```

The amps and volts are read using the function "average_ad" which will be explained in the function descriptions. This function is called with one of four constants that selects which of four values to read, solar amps (current), solar voltage (voltage), battery amps (batamps) or battery voltage (batvolts). These values are averaged by the function and returned as floating point variable. Each value is divided by a constant scale factor to give actual volts and amps in the floating point variable. The scale constant can be derived by either actual measurement or calculated from the circuit hardware components. I've included constants measured on my actual hardware and a set of calculated constants.

To use measured constants first run the program with no scale constants and note the actual A/D value output from the serial port. Then measure the actual value of the parameter with an external meter. Then calculate the scale factor using these two values. For example with my prototype hardware I measured 17.44 volts on the solar panel power input with my meter. The output for solar volts with no scaling constant in the software was 318.95. So to calculate the scaling constant divide the A/D output by the

actual voltage, 318.95 / 17.44 = 18.2887. Copy this procedure for the other three constants.

To use calculated constants first find the value of the voltage divider in the hardware. In my prototype hardware the voltage divider is 1k and a 10k resistor which gives a 1/11 voltage divisor ratio. So for a solar panel voltage of 17.44 after the voltage divider would be 1.5854 read into the A/D. The PIC has a 10 bit A/D and a 5 volt reference so it would read 0-1023 for a 0-5v input. For our 1.5854 input voltage to calculate the A/D output as follows 1.5854/5.0 = .31709 then .31709 * 1023 = 324.384 is the output of the A/D. Then divide the A/D output by the actual voltage to get the scale factor 324.384 / 17.44 = 18.6. Use the scale factor to divide the A/D output to get actual voltage output.

You can calculate the scale constant for current measurement in a similar manner. If 1.1A is coming from the solar panels it is flowing through a 0.02 ohm resistor. This gives a voltage across the resistor of 0.022 volts. The difference amp has a gain factor of 10 because of the 10k and 1k resistors so its output would be 0.022 * 10 = 0.22 volts. The 10 bit A/D has a 0-1023 range for a 0-5v reference. So 0.22 / 5 = 0.044 and 0.044 * 1023 = 45.012 is the value that the A/D would read for 1.1A. To get the scale factor divide the A/D output by the actual amps so 45.012 / 1.1 = 40.92. This is the scale factor in the code that is commented out because I'm using the actual measured values from my prototype hardware. As you can see the calculated constants match pretty closely to the constants that I measured on my hardware.

```
watts = current * voltage;              // do solar watts calculations using input current and volts
batwatts = batamps * batvolts;          // do battery watts calculation with batamps and batvolts
```

This code calculates the watts generated by the solar panel and the watts going out to the battery. The solar panel watts are used in the peak power tracking algorithm, the battery watts are just reported with the other parameters out the serial port. Since the current and voltage are scaled to give actual volts and amps values the watts are also in the correctly scaled units.

The following section of code is called once every 20 times and used to try and calculate the boost or gain of the peak power tracking algorithm. It also checks to see if no more current is coming from the solar panel (night time) and shuts the system down.

```
if ((++boost_counter % 20) == 0) {    // This section calculates the boost or gain of the PPT
  pwm1_duty100(100, pwm_max);                 // set PWM duty cycle to 100% to give direct
  delay_ms(20);                               // connection between solar panel and battery
  amp100 = average_ad(AMP_CHAN)/40.885; // read solar amps and volts using measured constants
  volt100 = average_ad(VOLT_CHAN)/18.2887;
  if (amp100 < 0.05) {                        // while PWM is set to 100% duty check to see if no more sun
    output_high(PIN_C4);              // if current is < .05 then turn off PPT turn on yellow led
    output_low(PIN_C3);                       // turn off FETs by clearing EN pin on FET driver
    delay_ms(1000);                           // delay to shut down
  }                                           // this will shut system down if no more solar current
  watts100 = amp100 * volt100;                // if not shutting system down calculate watts at 100% PWM
  temp_watts = watts / watts100;              // divide peak power watts by 100% watts, this is gain or boost
  boost = (long)((temp_watts * 100) - 100); // turn boost value into a percentage for output
}
```

Once every 20 times through the main loop set up to calculate the peak power tracking boost. First set the duty cycle of the DC/DC converter to 100%. This simulates the solar panel being directly connect to the battery. Then measure the volts and amps from the solar and calculate the watts at 100% PWM duty cycle. This gives you the watts as if you didn't have peak power tracking, as if the solar panel was just connected to the battery. Then divide the watts measured while the peak power tracker was running by the direct

connection watts. This is the gain or boost of the Peak Power Tracker. Convert that number to a percentage and store in the variable boost to output from the serial port later.

Also in this section of code check to see if the amps from the solar panel are less than 0.05A. If the solar panel amps are this low then you can assume that it is night time so set low the PIC output pin that shuts off the enable signal to the MOSFET driver. This shuts off the MOSFET switches which shuts off the power to the microprocessor so the system shuts down.

The next line of C code sends all the measured Peak Power Tracker values out the serial port.

```
                                                    //send all values out the serial port
    printf(" %lu %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %i %6.2f\r\n",
      pwm100, current, voltage, watts, batamps, batvolts, batwatts, boost, watts100);
```

The CCS C compiler uses the built in printf function to format and write variables directly out the serial port as ASCII text. The PPT uses this function to output the system variables for debugging and data logging. This includes the PWM duty cycle, current, voltage and watts measured for the solar panel and the battery and the calculated PPT boost and watts measured at 100% PWM duty cycle.

The last two lines in the main loop blink the green LED to show that the software is running and delay until the next loop.

```
    output_bit(PIN_C0, (++led_count & 0x01));      //blink green LED to show software running
    delay_ms(500);                                 //delay before next loop
```

Back to the top of the listing are two functions used in this program. The first function reads and averages a value from the A/D channel.

```
//-------------------------------------------------------------------------------
// This function read the value from the PIC A/D channel. It reads the value
// 32 times and averages it. The function is called with an A/D channel number and
// returns a float value that is average of the A/D channel value.
//-------------------------------------------------------------------------------
float average_ad(byte chan) {

  int i,avg_num = 32;
  float avg = 0, temp_avg = 0;

  set_adc_channel(chan);
  delay_us(100);
  for (i=1; i<=avg_num; ++i) {
    avg += read_adc();
    delay_us(100);
  }
  return(avg/avg_num);
}
```

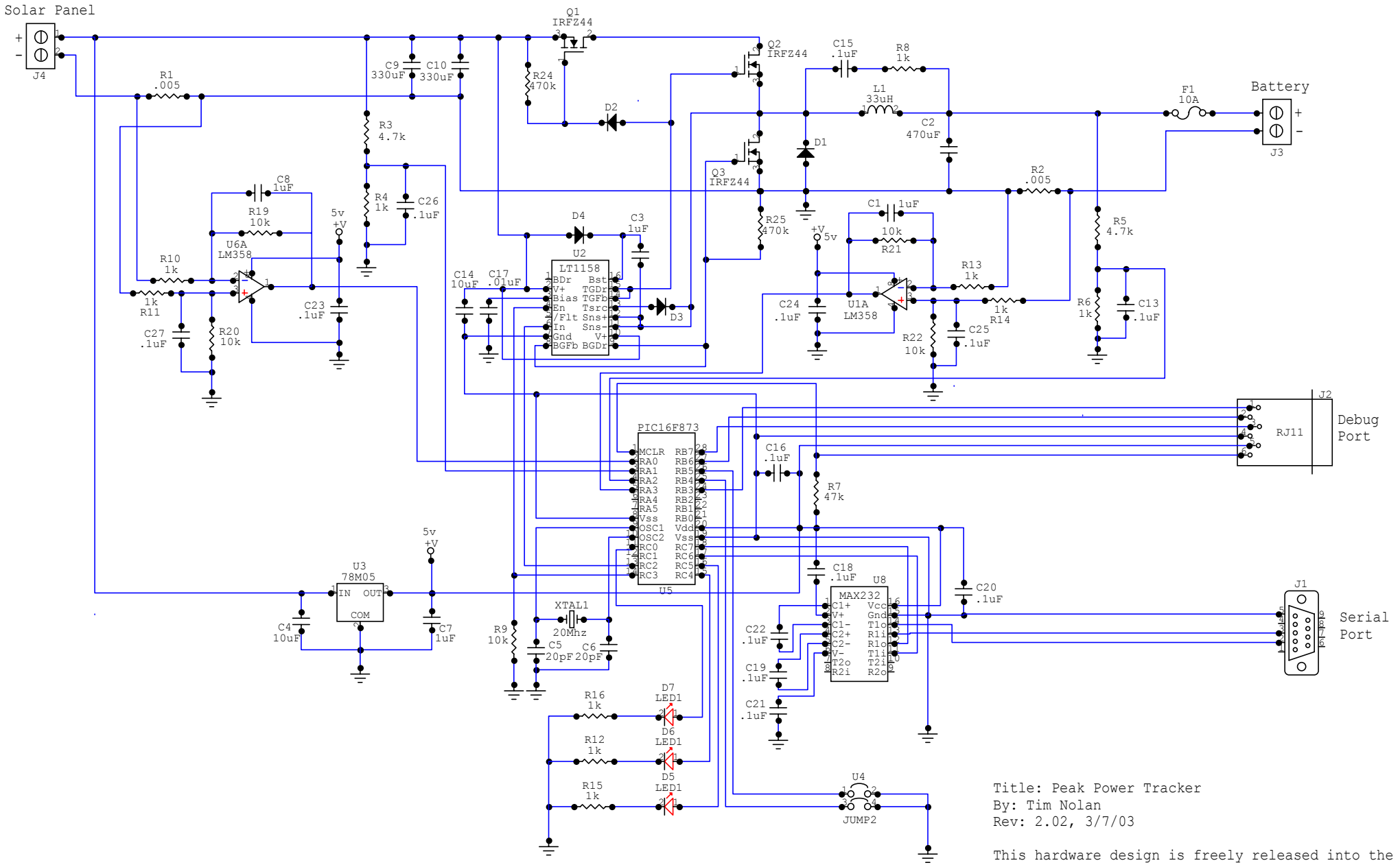
This function is called with a constant that is used to set the A/D channel or which A/D pin on the PIC to read from. Then the A/D is read and summed in the variable avg 32 times. The function returns the average as a floating point average by dividing the sum by 32.

The second function sets the PWM duty cycle.

```
//-------------------------------------------------------------------------------
// This function is used to set the duty cycle of the PWM output. It uses the built
// in C compiler function set_pwm1_duty to set the duty cycle. The function is called
// with a pwm duty cycle percentage (0-100) and also with the maximum pwm value which
// is function of the pwm frequency, see PIC data sheet for more information.
//-------------------------------------------------------------------------------
void pwm1_duty100(long pwm_percent, long pwm_maximum) {

  set_pwm1_duty((pwm_maximum/100)*pwm_percent);       // set PWM #1 duty cycle
}
```

The function is called with a percentage value (0-100) to set the PWM duty cycle. It is also called with the PWM maximum value which is the actual value that the hardware sets as the maximum duty cycle. The function scales this maximum by the PWM percentage that it is sent and then sets the hardware with the built in C function. For more information on the PIC PWM hardware see the 16F876 data sheet (30292c.pdf).

For more information on the CCS C compiler see the C compiler manual on the CCS website (www.ccsinfo.com/download.shtml). For information on the PIC16F876 see the chip data sheet at the Microchip website (www.microchip.com/1010/pline/picmicro/category/embctrl/14kbytes/devices/16f877/index.htm). For more information on the Peak Power Tracker see my Home Power article on my website (www.timnolan.com).

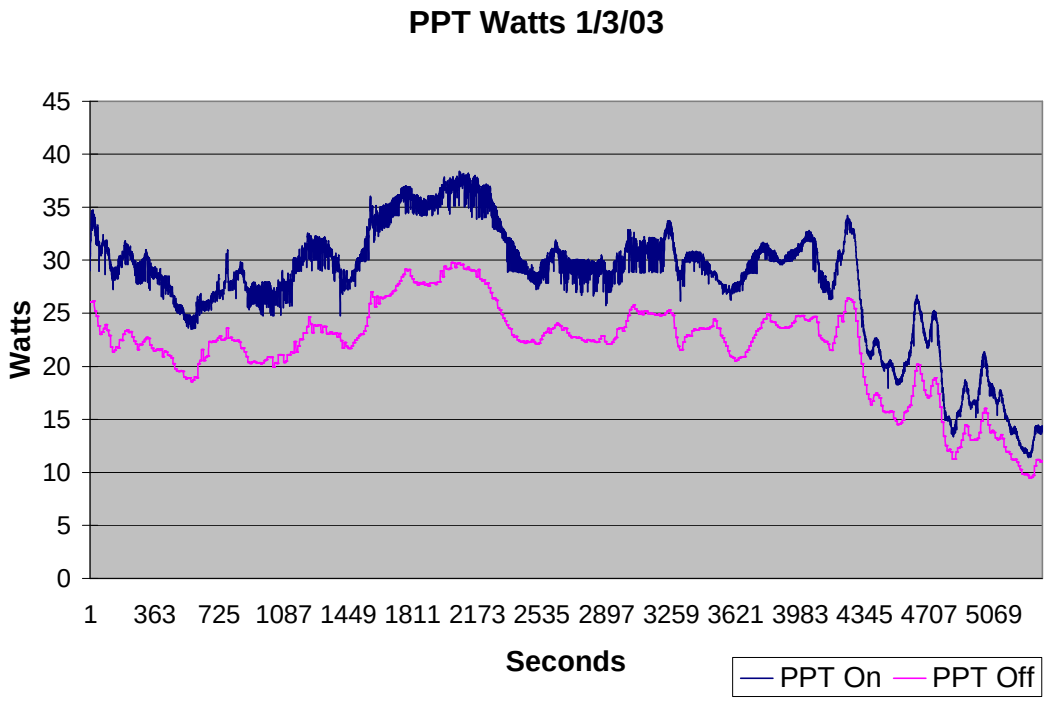


Title: Peak Power Tracker
By: Tim Nolan
Rev: 2.02, 3/7/03

```
////////////////////////////////////////////////////////////////////////
// Peak Power Tracker   copyright March 7, 2003 by Tim Nolan
//
// from www.timnolan.com
//
//    This code was written by me, Tim Nolan, to run the Peak Power
// hardware that I designed. It was compiled with the
// CCS PCM C Compiler, Version 2.703, http://www.ccsinfo.com/picc.shtml,
// running under Microchip's MPLAB development system, www.microchip.com.
// I used the ICD development system from Microchip for debugging and and programming,
// http://www.microchip.com/1010/pline/tools/picmicro/icds/mplabicd/index.htm.
// This code is more fully explained in my Software Description document at my
// my website www.timnolan.com. For more information on the chip PIC16F876 see
// the data sheet 30292c.pdf at www.microchip.com.
//    I have decided to release the software and hardware for my Peak Power
// Tracking system into the public domain with no restrictions. I hope you use
// it and improve it. I only ask that you give me credit with my name, Tim Nolan
// and my website www.timnolan.com somewhere in your documentation. I also
// would like hear about how you're using it, send me an email!
//
//
// v1.00 3/7/03              Software cleaned up and commented for release.
//
////////////////////////////////////////////////////////////////////////

#include <16f876.h>
#DEVICE *=16 ICD=TRUE
#use Delay(Clock=20000000)
#use RS232(Baud=9600,Xmit=PIN_C6,Rcv=PIN_C7,brgh1ok)

//-----------------------------------------------------------------------------
// This function read the value from the PIC A/D channel. It reads the value
// 32 times and averages it. The function is called with an A/D channel number and
// returns a float value that is average of the A/D channel value.
//-----------------------------------------------------------------------------
float average_ad(byte chan) {

  int i,avg_num = 32;
  float avg = 0, temp_avg = 0;

  set_adc_channel(chan);
  delay_us(100);
  for (i=1; i<=avg_num; ++i) {
    avg += read_adc();
    delay_us(100);
  }
  return(avg/avg_num);
}
```

```
//-------------------------------------------------------------------------------
// This function is used to set the duty cycle of the PWM output. It uses the built
// in C compiler function set_pwm1_duty to set the duty cycle. The function is called
// with a pwm duty cycle percentage (0-100) and also with the maximum pwm value which
// is function of the pwm frequency, see PIC data sheet for more information.
//-------------------------------------------------------------------------------
void pwm1_duty100(long pwm_percent, long pwm_maximum) {

  set_pwm1_duty((pwm_maximum/100)*pwm_percent);          // set PWM #1 duty cycle
}
//-------------------------------------------------------------------------------
// Main code section.
// For more information on the chip PIC16F876 see the data sheet 30292c.pdf.
//-------------------------------------------------------------------------------
main()
{
  CONST float CHARGED = 14.4;                    // initialize constants and variables
  CONST long NDELTA = -1;
  CONST byte AMP_CHAN = 0, BAMP_CHAN = 3, BVOLT_CHAN = 2, VOLT_CHAN = 1;
  float current, voltage, watts = 0.0, old_watts = 0.0;
  float batamps, batvolts, batwatts;
  float temp_watts, watts100, amp100, volt100;
  long delta = 1, pwm_max, pwm100;
  long boost = 0, boost_counter;
  byte led_count = 0;
                                                 // initialize hardware
  output_low(PIN_C3);                            // turn off FETs by clearing EN pin on FET driver
  setup_port_a(ALL_ANALOG);                      // set up A/D channels
  setup_adc(adc_clock_div_32);                   // start A/D clocks
  setup_timer_2(T2_DIV_BY_1, 49, 1);             // pwm frequency 100kHz
  setup_ccp1(CCP_PWM);                           // Configure CCP1 as a PWM
  pwm_max = 200;                                 // maximum value for PWM based on frequency
                                                 // set pwm to 85% to start
  pwm100 = 85;
  pwm1_duty100(pwm100, pwm_max);                 // set pwm duty cycle to pwm100 value
  output_high(PIN_C3);                           // set enable pin on FET driver
  output_high(PIN_C0);                           // turn on green led
  delay_ms(1000);                                // delay to get started

  while(TRUE) {                                  // loop forever
                                                 // check for max battery voltage, very simple battery
                                                 // charge control algorithm
    if (batvolts > CHARGED) {                    // if batvolts is > charged then battery is charged so
      delta = NDELTA;                  // start rolling back current until batvolts < charged
      output_high(PIN_C5);                       // turn on red led if battery at maximum voltage
    }                                            // start hill climbing algorithm for peak power tracking
    else if (old_watts > watts) {    // else if battery not charged yet change the value of
      delta = -delta;                // delta to make pwm increase of decrease to maximize watts
      output_low(PIN_C5);                        // turn off red led if battery not at maximum voltage
```

```
    }
    old_watts = watts;                          // load old_watts with current watts value for next time
    pwm100 += delta;                                    // add delta to change PWM duty cycle for PPT algorithm
    if (pwm100 > 100) pwm100 = 100;                     // check limits of PWM duty cycle and set range to 0-100
      else if (pwm100 < 0) pwm100 = 0;
    pwm1_duty100(pwm100, pwm_max);                      // call function to set PWM duty cycle in PIC hardware

    current = average_ad(AMP_CHAN)/40.885;              // read current and voltage from solar panel and going to
    voltage = average_ad(VOLT_CHAN)/18.2887;            // battery, use average_ad function return average value
    batamps = average_ad(BAMP_CHAN)/41.3428;    // as a float, divide by constant to scale into actual
    batvolts = average_ad(BVOLT_CHAN)/20.4831;// volts and amps. These constants were measured in the
                                                        // actual prototype hardware.
//    current = average_ad(AMP_CHAN)/40.96;             // These constants are calculated by the gain of the
//    voltage = average_ad(VOLT_CHAN)/18.618;           // hardware not actually measured.
                                                                    // They are commented out
//    batamps = average_ad(BAMP_CHAN)/40.96;    // if not used. If volt and amp constants not measured
//    batvolts = average_ad(BVOLT_CHAN)/18.618;// then use this code

    watts = current * voltage;                  // do solar watts calculations using input current
                                                        // and volts
    batwatts = batamps * batvolts;              // do battery watts calculation with batamps and batvolts

    if ((++boost_counter % 20) == 0) {          // This section calculates the boost or gain of the PPT
      pwm1_duty100(100, pwm_max);                       // set PWM duty cycle to 100% to give direct
      delay_ms(20);                                     // connection between solar panel and battery
      amp100 = average_ad(AMP_CHAN)/40.885;     // read solar amps and volts using measured constants
      volt100 = average_ad(VOLT_CHAN)/18.2887;
      if (amp100 < 0.05) {                              // while PWM is set to 100% duty check to see if no
                                                        // more sun
        output_high(PIN_C4);                    // if current is < .05 then turn off PPT turn on yellow led
        output_low(PIN_C3);                             // turn off FETs by clearing EN pin on FET driver
        delay_ms(1000);                                 // delay to shut down
      }                                                 // this will shut system down if no more solar current
      watts100 = amp100 * volt100;                      // if not shutting system down calculate watts at 100% PWM
      temp_watts = watts / watts100;                    // divide peak power watts by 100% watts, this is
                                                        // gain or boost
      boost = (long)((temp_watts * 100) - 100); // turn boost value into a percentage for output
    }
                                                        //send all values out the serial port
    printf(" %lu %6.2f %6.2f %6.2f %6.2f %6.2f %6.2f %li %6.2f\r\n",
      pwm100, current, voltage, watts, batamps, batvolts, batwatts, boost, watts100);

    output_bit(PIN_C0, (++led_count & 0x01));           //blink green LED to show software running
    delay_ms(500);                                      //delay before next loop

  }     //end while forever
}   //end main
```


PPT Watts 1/3/03
Watts
45
40
35
30
25
20
15
10
5
0
1
363
725
1087
1449
1811
2173
2535
2897
3259
3621
3983
4345
4707
5069
Seconds
PPT On
PPT Off