

## Digital Electronic Speed Control

When I first began building RC models I was attracted to electric flight, probably because of my electronics background. So when I needed an ESC, I said to myself “looks like an interesting little project, I’ll just whip one out and save a few bucks”. Well, we won’t say who whipped who, but after innumerable revisions, a nice little design has evolved.

One of the realities of microprocessor-based projects is the high cost of the development tools. That is the first of the many beauties of this design. The development tools are free! The project is based on a Microchip Technology PIC12C508 in a very space conscious 8-pin package. Simply log on to [www.microchip.com](http://www.microchip.com), click on Development Tools and download MPLAB IDE, a complete development environment with code assembler and a very nice simulator. Being based on a microprocessor, the usual handful of capacitors, transistors and resistors are reduced to 5 easily assembled components. Changing the function is easy too, for example, the unit can be an onboard on/off switch with simple code changes.

The second advantage of this design is that it is “scalable”, allowing you to choose the current capacity, weight and features that you want to load it up with. I have built units with 100 Amp capacity and units that weigh as little as 5 grams. The components I have chosen for this article will allow operation up to 14 cells and current up to 17 Amps. If you want to increase these specs, all you have to do is select different components. I will show you how to make the simple calculations to select components for your application.

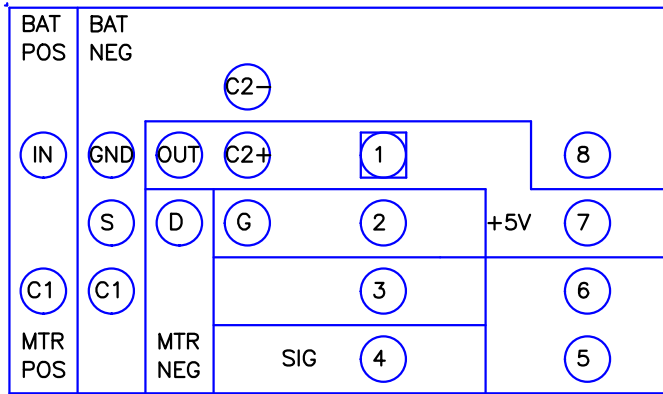
Third, with only 5 components, you have several options when it comes to wiring. You can, of course, lay out and pay for a commercially etched PWB (Printed Wiring Board). But only a few minutes of effort with an Xacto knife, and you can chop out your own using the template shown. Or you can forego the PWB entirely; wire the entire project in free space and glob on a bit of hot melt glue to keep the components in place for the ultimate in miniaturization.

Lastly, it is kinda cool when someone sees your ESC and asks, “What brand is that?” and you say “Mine!”

### Build Options

First of all, you do not have to make a Printed Wiring Board (PWB) to assemble this project. Because the microprocessor handles all of the signal processing, there are very few parts to interconnect. You have several wiring options; you can use a PWB which makes a cleaner, more professional looking result, you can use predrilled “perf board” and solder the interconnections, you can use a solid foil covered piece of copper clad board and cut the foil patterns with a hobby knife or Dremel, or lastly, you can simply wire all of the components together. With as few as 5 components, these prototyping methods will result in a small, easily constructed ESC. If you elect to build the ESC using either of the latter techniques, you probably want to get the discrete, not SMT (Surface Mount Technology) parts for easier assembly. Many microprocessor projects admonish you not to attempt to build using these prototype techniques due to the high frequencies involved, however, on the PIC12C508, the high frequency clock circuits are all self-contained on the chip – it doesn’t even need an external crystal.

Here is a top view illustration (foil side up) of the 20 A ESC showing how to scribe a small piece of copper clad material .6” wide x .8” long.



I lightly scribed a .1" grid, marked the cuts with a "Sharpie" and then ground away the copper with a bit on my Dremel. I have found that it is also easy to cut the copper with the Xacto knife, then use the backside of the blade to widen the cut. The entire process takes less than 10 minutes. The circles show where the component pads go, you do not have to drill the holes. Components are soldered directly to the copper.

## Microprocessor

The microprocessor, a PIC12C508, costs less than \$2 in single quantities from Digikey. Speaking of Digikey ([www.digikey.com](http://www.digikey.com)), get a catalog and keep it handy – it is an excellent reference for components.

The code provided is quite straightforward. It was written with simplicity in mind so that you can easily understand and modify it. It is all in-line, not even a subroutine call is used. The system will measure the incoming signal from the RC receiver and translate this to an output pulse rate to the MOSFET motor drivers. The MOSFETs are turned on and off at a 3 kHz pulse frequency, more efficient for lower inductance motors. You can choose whatever pulse frequency you desire. Motor braking, arming switch and low voltage cutoff are supported in the code but not implemented in this physical PWB layout.

The incoming signal will range from .95 mS to 1.95 mS from a normal RC transmitter set to 100% travel. This range is divided into four regions:

Signal in mS	Mode
< 1.16	Brake
< 1.26	Off
< 1.77	PWM
> 1.77	100% on

The onboard 8 bit timer is set to increment at a 4 uS/bit rate making pulse measurement easy.

Features such as soft start, automatic max stick accommodation, indicator LEDs and non-linear translation are all doable, but, after doing all of these, I have found that the simpler the better – it is embarrassing to tear apart the model wondering why the darn thing is not working when the

reason is because I forgot to hold max stick for 3 seconds, then turn on the arming switch and spin around three times. It only adds to the confusion when you have several commercial ESCs, each one with its own startup sequence. "Keep it simple" always works for me.

Copious comments have been included to make maintenance and modifications easy.

## **MOSFETS**

The MOSFETs are the heart of the ESC. They determine how much current your ESC will handle and how many batteries you can stack together.

The maximum voltage "Vds" will determine the maximum number of cells that the ESC will support. The International Rectifier IRL3502 that I use is rated to 20 volts. If you put more than 20 volts, or about 14 NiCd cells, across it, it will self-destruct.

Generally speaking, heat is the killer of MOSFETs and we must reduce the heat generated by selecting a MOSFET with low "on resistance" or get rid of the heat with a heatsink, or both. The IRL3502S is rated to a maximum of 110 A continuous, 420 Amps intermittent – wow! But wait – at 10 amps the on resistance of .007 Ohms produces .7 Watts of heat ( $10A \times 10A \times .007 \text{ Ohm}$ ). The thermal resistance of the junction to ambient is about 60 degrees C / watt, meaning each watt of power dissipated by the MOSFET will raise the temperature of the device 60 degrees. If the temperature goes over 150 C, the MOSFET will die, probably while making a crackling sound and emitting the diagnostic smoke that is contained inside. So, we can say our MOSFET is limited to 17 Amps ( $17 A \times 17 A \times .007 = 2 \text{ W}$ ,  $2 \text{ W} \times 60 \text{ deg/W} = 120 \text{ deg}$  above ambient, ambient is 25C). If you hang the tab of the transistor in the air stream or add an external heatsink, the circuit will be able to handle more current. Adding more MOSFETs in parallel will also increase the current rating.

The voltage drop across a MOSFET while conducting 5 Amps as measured with an oscilloscope is 25 mV, proving that the MOSFETs are indeed exhibiting an on resistance of about .005 Ohms ( $.025 \text{ V} / 5 \text{ A}$ ).

## **The Schottky Diode**

Remember the "points" of an old ignition system? The three key components were the points (our MOSFET), the coil (our motor) and the "condenser" (our Schottky Diode) The points closed to send current through the coil (our motor is an inductor) and when the points opened, hundreds of volts were generated as a spark across the points. Exactly the same thing would happen in our circuit without the Schottky diode (including the spark!). When the MOSFET attempts to shut down the current, the inductive windings of the motor will reverse polarity and the voltage will rise until the current can continue to flow, even if it means breaking down the Vds rating of the MOSFET. The Schottky diode provides a path for this "flyback" current. Each time the MOSFET turns off, the Schottky diode will conduct the stored energy of the motor back into itself. So when the MOSFET turns off, the motor is not coasting, the re-circulating flyback current through the diode is still driving the motor forward. This is why a high rate ESC is more efficient than the old low rate designs based on the 20 mS frame rate of the transmitter. If the pulse frequency is high enough so that the motor is always being driven even when the MOSFET is off, the motor never imposes a load on the propeller.

If you want a very small circuit, you don't have to put the diode on the PWB. You can solder it directly to the motor as many ESC manufacturers do. But don't forget it or you will blow out your transistor.

Schottky diodes conduct very quickly, fast enough to prevent the inductive voltage spike from exceeding the power supply voltage. Fast recovery diodes can be substituted, general purpose

rectifier diodes will not work. The reverse voltage rating must be greater than the power supply voltage.

### **MOSFET Driver**

The IRL3502S is a “logic level” device, meaning that it is designed to be driven with +5 volt gate drive. “Great!” you say, “No gate driver needed.” Back when MOSFET gates needed +12 volts, a gate driver was always needed, so now, with +5 Volt specified parts, it is tempting to simply omit it. Well, not so fast. The gate capacitance of the MOSFET is .005 microfarads. The maximum current out of the microprocessor is 25 mA. With this amount of current available, it will take about a microsecond for the gate to charge up. During this time, the power dissipated is extremely high – approximately half of the power supply voltage times half of the maximum current or  $20/2 \text{ Volts} \times 15/2 \text{ Amps} = 75 \text{ Watts}$ . It is vital that the MOSFET transition through this region as quickly as possible. At 3 kHz repetition rate, the transitions happen twice every 333 uS or about .7% of the time. So the heat generated is about a half watt (.7% x 75 W). This is called the switching losses and is the downside to increasing the pulse frequency beyond 3 kHz. Decreasing the pulse frequency will reduce the switching losses. In general you can omit the gate driver for currents of 10 Amps or less through a single MOSFET. However, if you parallel MOSFETs for increased current capacity, you may have to use a gate driver in order to charge the increased gate capacitance. I have found the Texas Instruments TPS2829DPV easy to use.

Driving the gate to a voltage greater than 5 volts has an incremental advantage of further reducing the on resistance, but not by much. A 15% reduction in resistance can be gained at the expense of additional circuit complexity.

### **Motor Brake Circuit**

Motor braking is a handy thing in sailplane applications. Many small park fliers do not need this feature. Most of my planes are designed to maximize steady flight time at a constant motor setting, so I don't use it much either.

The microprocessor will detect when the throttle stick is in the lowest 15% of its range and set the motor brake, pin 3, high. This can be used to turn on a braking transistor across the motor. When this transistor is on, the motor is shorted to itself through it so the wind energy imparted to the propeller is turned into heat energy in the transistor, slowing and eventually stopping the propeller. The transistor does not have to be a MOSFET, I have used a PNP bipolar with success, but the base current needed is troublesome. Darlington transistors do not work.

### **Low Voltage Detect**

An input of the microprocessor, pin 7, is dedicated to monitoring the voltage level of the battery. When it is low, further operation of the ESC is not allowed unless the stick is brought into the off or brake region. Then the motor may be restarted as long as the battery has regained sufficient voltage. This pin is wired to the +5 supply in the circuit shown, defeating this feature.

Another version of the microprocessor has an onboard A/D. This eliminates the need for a hardware low voltage detect circuit.

### **Arming Switch**

The code also supports an arming switch, pin 6, which must be grounded for the motor to be energized. An adjacent pin provides a handy ground, so wiring the two pins together at the PWB effectively defeats this feature. If you want to use an arming switch, simply connect a wire from these two pins through a switch.

## Battery Eliminator Circuit

The BEC is deceptively simple. Put in any voltage up to 45 Volts and get 5 volts out - perfect for supplying your RC receiver and servos. But, ah! the details...

The drop out voltage (V<sub>do</sub>) of a linear regulator is the minimum voltage between the input and output required to maintain the output voltage. The old standby 7805 has a 2 volt dropout voltage, meaning that at least 7 volts in is required to maintain the 5 volt output.

A Low Drop Out (LDO) linear regulator is optimized to reduce the dropout voltage to a minimum, typically less than .5 Volt. The National Semiconductor LM2941 provides up to 1 amp of current to the servos and receiver at 5.0 Volts. The downside is that the LDO is only rated to 1 amp and 150 degrees C. When it reaches either limit, it will shut off. Micro and mini servos are little current hogs, frequently drawing 2 or 3 times the current of a standard servo. It is easy to move several servos simultaneously and exceed the maximum current rating of the LDO. When this happens, the LDO will turn off and everything is given a break to cool down – except you, the pilot, who is moving the sticks madly wondering why nothing is happening. The LDO requires at least 22 uF capacitor on the output for stabilization.

All of the semiconductor heat calculations apply here too. If your servos draw 1 Amp and your battery voltage is 15 Volts, the heat generated in the BEC will be the current through it times the voltage across it or 10 Watts (1 A x 10 Volts = 10 W). The thermal resistance of this circuit is also about 40 deg/W, so the temperature just shot up to 400 degrees! However, the servos only draw high currents when they are moving, so the average current through the BEC is limited to about 300 mA (120 degree max -> 3 W max, 3 W / 10 Volts = .3 A).

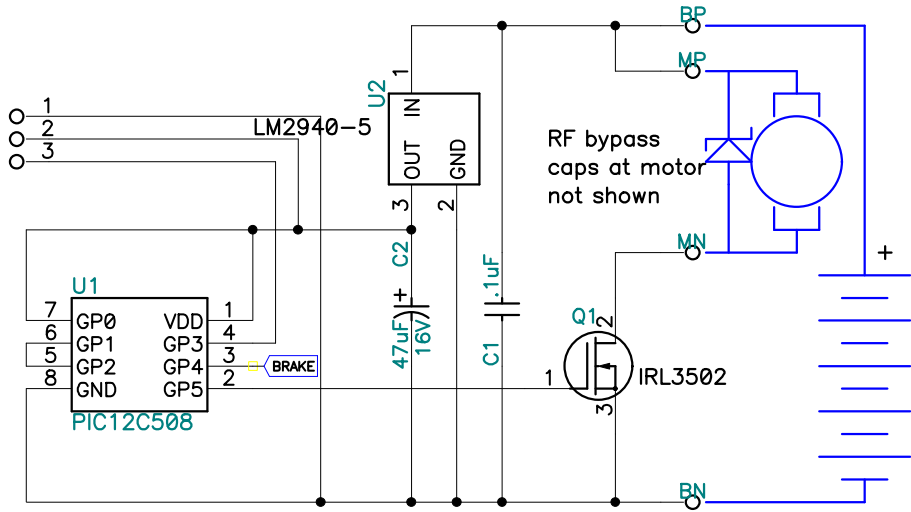
### Why?

So, what do I think about the project? I like being able to gin up a motor control out of parts from my junk box. I like the fact that after you have developed the code, you can scale the application to fit as large or small a motor as needed. Was it cost effective? No way! It would have been way cheaper to buy a commercial ESC. But, where's the fun in that?

If you have any questions, you can contact me at [ssarns@flying-eye.com](mailto:ssarns@flying-eye.com) the latest revision of downloadable source code is available at [www.flying-eye.com/support/downloads](http://www.flying-eye.com/support/downloads).

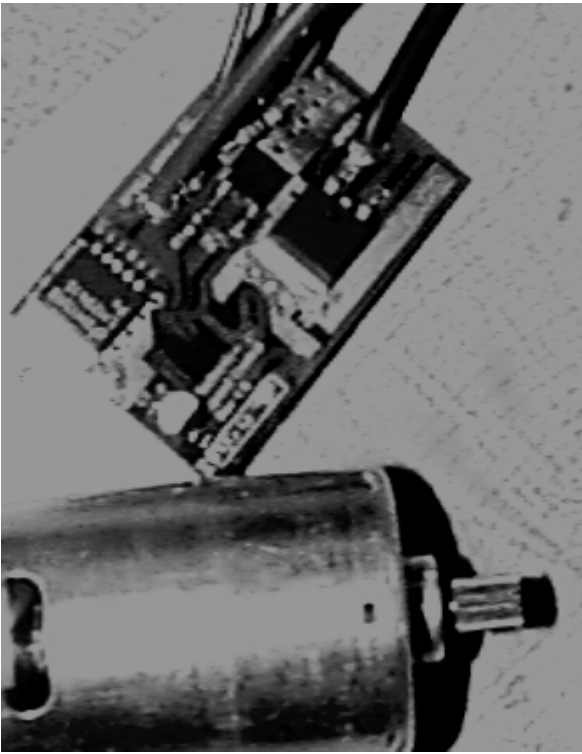
Good luck on your project!

# Schematic – 17 Amp ESC



Bad photo showing 40 A ESC

(better photos coming)



```

; RCMC accepts "R/C Servo" style input pulses and outputs PWM to MOSFET
; Servo input pulse is normally 1.25 to 1.75 ms, usually 21 ms frame rate

; file registers 7 to 1F are general pupose
TMR0      equ    01      ; address of timer 0
STATUS    equ    03      ; STATUS register F3
OSCAL     equ    05      ; oscillator calibration register
GPIO      equ    06      ; I/O file register
PWM_UP_COUNT equ    08      ; pwm amount from signal input
PWM_DN_COUNT equ    09      ; pwm amount from signal input
TEMP      equ    0a      ; Holds pwm while up/dn counting
FLAGS     equ    0b      ; collection of status flags
COUNTER   equ    0c      ; Loop counter for 100 pwm pulses
PWM       equ    0d      ; used as a working register

; destination declarations
W         equ    0        ; w register designation
FILE      equ    1        ; file register destination

; flag bit declarations
C_FLAG    equ    0        ; Carry bit in status register
Z_FLAG    equ    2        ; zero bit in status register
OFF_MODE  equ    1        ; off mode
MAX_MODE  equ    2        ; 100% on mode
BRK_MODE  equ    3        ; brake mode
SAF_MODE  equ    4        ; safe mode
LOBAT_FLAG equ    5        ; low battery

; pin declarations (bits)
PWROK_PIN equ    0        ; power monitor input pin, input
ARM_PIN   equ    1        ; arming sw input
LED_PIN   equ    2        ; LED/diagnostic output
RX_PIN    equ    3        ; input pin from receiver, input
BRK_PIN   equ    4        ; output pin for motor brake, output
PWM_PIN   equ    5        ; output pin to speed control, output

; value declarations
BRK_STK   equ    028      ; anything less is brake engaged
OFF_STK   equ    040      ; anything less is PWM off
MAX_STK   equ    0C0      ; anything greater is 100% output
FREQ      equ    3        ; equal approx freq in kHz
IO_CONF   equ    b'00001011'; 1=input, 0=output, 6 bits
OP_CONF   equ    b'10000001'; WU disable, PU enable, T0 prescaler=1:4

    org    0

    movwf  OSCAL          ; set the oscillator calibration value
    bcf   GPIO, PWM_PIN   ; set outputs to value before enable
    bcf   GPIO, BRK_PIN   ; brake is off
    movlw IO_CONF
    tris  GPIO            ; enable outputs
    movlw OP_CONF
    option           ; Timer 0 increments each 4 uS, roll in 1.024 mS
    clrf  FLAGS          ; set all flags = 0
    btfss GPIO, PWROK_PIN ; test battery
    bsf   FLAGS,LOBAT_FLAG; battery is low

; monitor transmitter for one second before starting
    clrf  TEMP          ; 50 x .02 sec/loop = 5.12 sec
w1     btfsc GPIO, RX_PIN ; get signal
        goto w1          ; wait for signal to go low
w2     btfss GPIO, RX_PIN ; get signal
        goto w2          ; wait for input signal to go high
    decfsz TEMP, F      ; 50 loops
        goto w1          ; not done yet

; ***** this is the main routine *****
main1  bcf   FLAGS, SAF_MODE ; clear safe mode flag
        btfsc GPIO, ARM_PIN ; test arming pin
        bsf   FLAGS, SAF_MODE ; set off mode if arm pin high

```

```

; measure the input signal
; set brake, off and max mode flags

s0    movwf    TMR0          ; set the zero flag
      btfsz   STATUS, Z_FLAG ; test if timer 0 has reached zero
      incfsz  COUNTER, F    ; COUNTER was zero upon entry
      goto   off_mode      ; 256 mS of input signal low
      btfsz   GPIO, RX_PIN  ; wait for low
      goto   s0            ; 7 uS/loop, asynch wrt timer 0 (4 uS)
s1    btfsz   GPIO, RX_PIN  ; wait for rising edge
      goto   s1            ; input is low, wait

      movlw   6             ; want 1.000 mS timer
      movwf   TMR0         ; preload with 6 counts x 4 uS/count = 24 uS
s2    movwf   TMR0
      btfsz   STATUS, Z_FLAG
      goto   s2            ; wait 1.000 mS

      clrf    TMR0         ; clear timer0
s3    btfsz   GPIO, RX_PIN  ; wait for input to fall, loop is 5 uS resolution
      goto   s3            ; signal is still high
      movfw   TMR0         ; get timer 0 = input pulse high time x 4 uS/bit
      movwf   PWM          ; store timer 0 result in PWM

      movlw   BRK_STK      ; less than 28H is brake mode
      subwf   PWM, W       ; test if PWM value is less than threshold
      btfsz   STATUS, C_FLAG ; carry flag clear if result is negative
      goto   s5            ; not in brake mode
      bsf    FLAGS, BRK_MODE ; is in brake mode
      bcf    FLAGS, OFF_MODE ; brake flag is set, off flag cleared
      bcf    FLAGS, MAX_MODE ; max flag is clear
      goto   main2         ; value rx is less than BRK_STICK-> exit

s5    movlw   OFF_STK      ; less than 40H is off mode
      subwf   PWM, W       ; test if PWM value is less than threshold
      btfsz   STATUS, C_FLAG ; carry flag clear if result is negative
      goto   s6            ; not in off mode
      bcf    FLAGS, BRK_MODE ; is in off mode
      bsf    FLAGS, OFF_MODE ; off flag is set, brake flag clear
      bcf    FLAGS, MAX_MODE ; clear max flagclear
      goto   main2         ; value rx is less than OFF_STICK-> exit

s6    movlw   MAX_STK      ; greater than C0 is max mode
      subwf   PWM, W       ; test if PWM value is greater than threshold
      btfsz   STATUS, C_FLAG ; carry=1 if >=
      goto   s7            ; not in max mode, result is 40 to BF
      bcf    FLAGS, BRK_MODE ; is in max mode
      bcf    FLAGS, OFF_MODE ; max flag is set, brake and off flags are cleared
      bsf    FLAGS, MAX_MODE ; set max flag
      goto   main2         ; value rx is more than MAX_STICK-> exit

s7    movlw   OFF_STK      ; PWM contains 40-BF
      subwf   PWM, F       ; PWM contains 0-7F
      rlf    PWM, F        ; PWM contains 0-FF

      bcf    FLAGS, BRK_MODE ; in PWM mode, brake flag cleared
      bcf    FLAGS, OFF_MODE ; off flag cleared
      bcf    FLAGS, MAX_MODE ; max flag cleared

main2 btfsz   FLAGS, SAF_MODE
      goto   off_mode
      btfsz   FLAGS, BRK_MODE ; vector to appropriate routine
      goto   brake_mode     ; ESC is in brake mode
      btfsz   FLAGS, OFF_MODE ;
      goto   off_mode       ; ESC is off
      btfsz   FLAGS, MAX_MODE ;
      goto   max_mode       ; ESC is 100% on

PWM_mode
      btfsz   FLAGS, LOBAT_FLAG ; ESC is in PWM mode
      goto   main1         ; batt too low

```

```

pwm2    clrf    COUNTER      ; loop counter
        bsf    GPIO, PWM_PIN ; set output pin high

        movfw  PWM          ; get PWM value
        movwf  TEMP         ; will decr TEMP until<0
        movlw  FREQ         ; decr by constant amount
pwm3    subwf  TEMP, F      ; decrement until <0
        btfs  STATUS, C_FLAG ; carry=0 if negative
        goto  pwm3         ; 4 uS/loop, 1Khz if FREQ=1

        btfss  GPIO, PWROK_PIN ; test battery with motor on
        bsf    FLAGS, LOBAT_FLAG
        bcf    GPIO, PWM_PIN ; set output low

        movfw  PWM          ; get PWM
        movwf  TEMP         ; will incr TEMP until>FF
        movlw  FREQ         ; incr by const amount
pwm4    addwf  TEMP, F      ; incr until >FF
        btfs  STATUS, C_FLAG ; carry=1 when >FF
        goto  pwm4         ; 4 uS/loop, 1Khz if FREQ=1

        incfsz COUNTER, F   ; test loop counter
        goto  pwm2         ; generate another pulse
        btfs  PWM, 7        ; test PWM MSE
        goto  main1        ; leave PWM output low, <50%
        bsf    GPIO, PWM_PIN ; PWM>50%, set PWM pin high
        goto  main1        ; 256 pulses have been generated

brake_mode
        bcf    GPIO, PWM_PIN ; turn off PWM
        bsf    GPIO, BRK_PIN ; turn on brake
        bcf    FLAGS, LOBAT_FLAG ; clear only in brake and off mode
        btfs  GPIO, PWROK_PIN ; test battery
        bsf    FLAGS, LOBAT_FLAG ; power is low
        goto  main1

off_mode
        bcf    GPIO, PWM_PIN ; turn off both
        bcf    GPIO, BRK_PIN
        bcf    FLAGS, LOBAT_FLAG ; clear only in brake and off mode
        btfs  GPIO, PWROK_PIN ; test battery
        bsf    FLAGS, LOBAT_FLAG ; power is low
        goto  main1

max_mode
        btfs  FLAGS, LOBAT_FLAG
        goto  main1        ; batt too low
        bcf    GPIO, BRK_PIN ; turn off brake
        bsf    GPIO, PWM_PIN ; set PWM on
        btfs  GPIO, PWROK_PIN ; test battery w/ mtr on
        bsf    FLAGS, LOBAT_FLAG ; power is low
        goto  main1

END

```