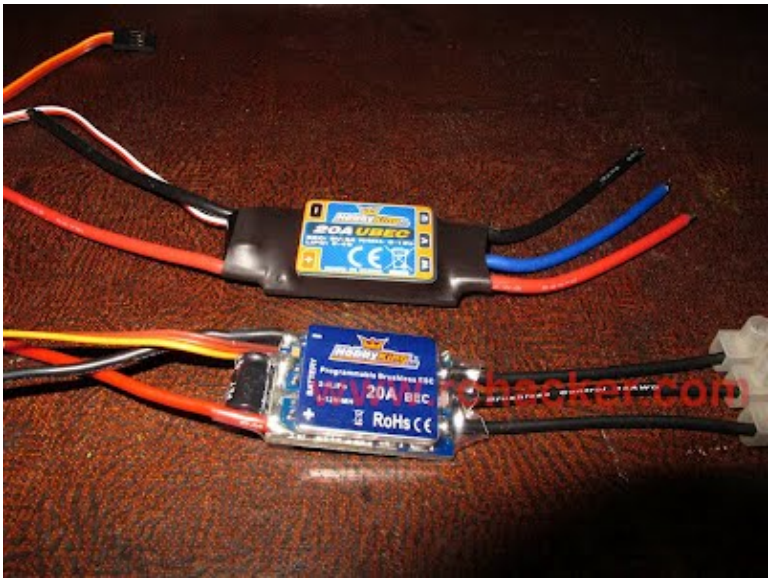
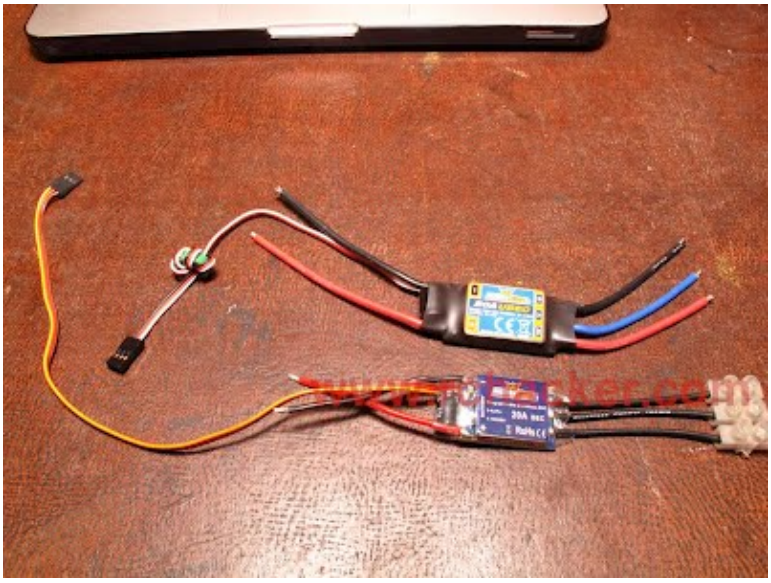
DIY >

# SimonK ESC firmware flashing HK Blue Series 20A and HK F20A

For those seriously into quads you would have heard of SimonK and his firmware for Atmega based ESCs. I am going to focus on flashing a HK Blue Series 20A ESC. These are popular so if you cannot get hold of one the Mystery Blue 20A is pretty much the same unit. Another very good ESC for flashing is the HobbyKing F-20A. You have to build your own flashing adapter, but once you have that they are much easier. I will cover that here also.

Here they are side by side:



The Blue series is smaller and lighter, but I like the F20-A better as it cheaper and has longer leads. Also if you mount it right is much easier to re-flash once it is fitted on your quad.

Difficulty: medium.

Hardware required.

- USBasp programmer.
- HK Blue Series 20A
- or Mystery Blue 20A
- or HobbyKing F-20A
- 6 very small screws.
- Soldering iron.

- 2 servo extension leads.

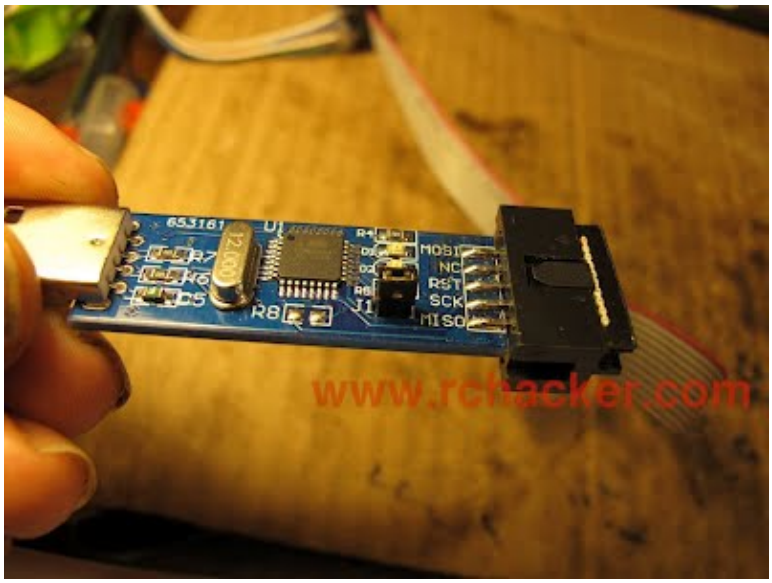
**Step 1: Install your USBasp drivers.**

No drivers are needed for Linux or OSX.

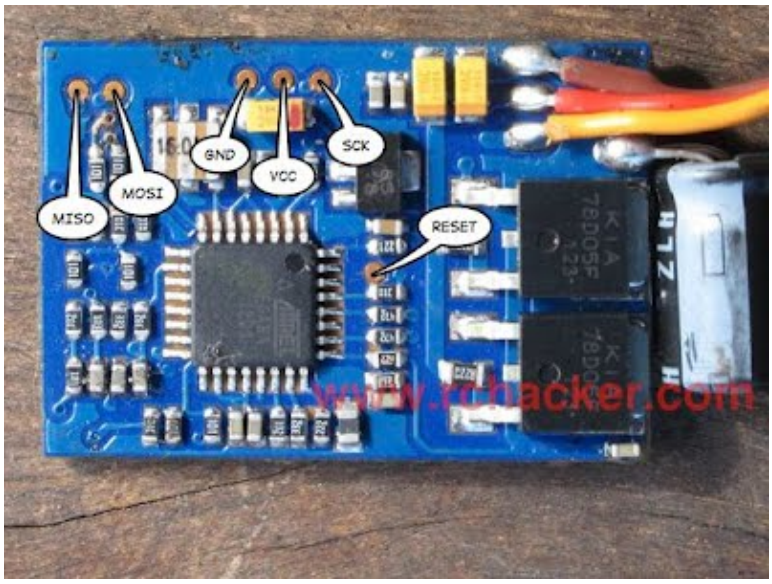
Windows get your drivers from here you have to manually install the drivers. Josh on Flite Test shows how to install the drivers on Windows 7, and also how to use KKMultiTool on a HK controller board. Worth a look.

**Step 2: Programmer to ESC connections.**

The HK USBasp programmer. has nice labels with handy points for testing with your multimeter.

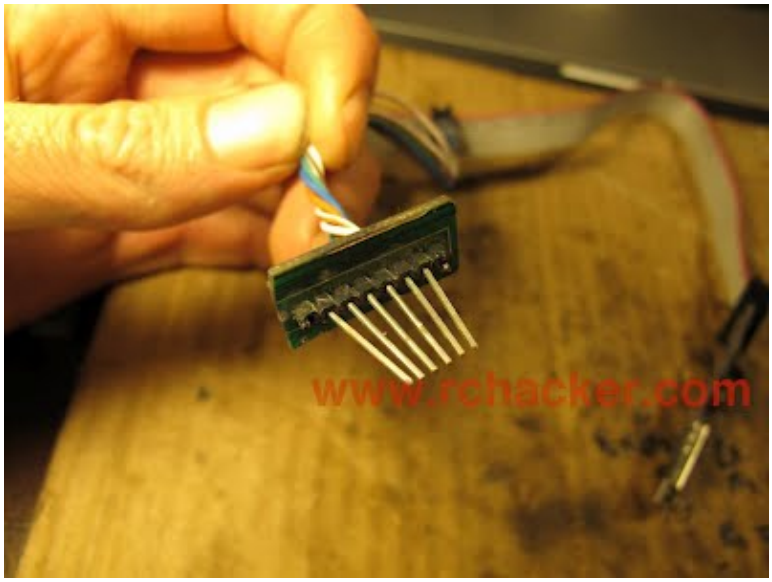



Here is the business side of a blue series with the programming pads labeled. Do not cut your cover off like I have here, as you will see there is no need.




And here is the F20-A with the pads exposed.


Just a small strip of the covering cut off at the corner of the pcb near the red motor wire.
With a sharp hobby knife make a small incision first and then lift from underneath to cut it. This way you wont cut any tracks.




I made this for the F20-A with headers that were cut out of an old circuit board. The pins were bent inwards to fit and then ground flat with the face of a Dremel cutter. I simply hold it in place while programming. You could also attach a peg to this setup but I am already in too much trouble with the missus.


I made it into a permanent fixture on my programmer by attaching it to the seldom used 10pin header. The solid cores of stripped network wires are pushed in and held with hot glue.

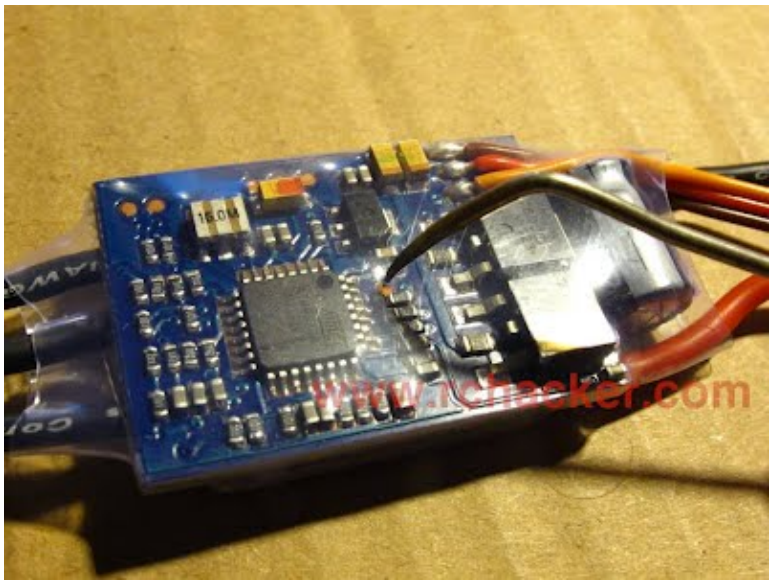
That is the F20-A setup ready. Back to the blue series:

Prepare our leads. I used the female ends of two servo extension wires. You will have to cut off the outer plastic.


I find it easier to slide the pins out before I solder them on.


Next you will need four very small and relatively long screws. Most people should have a few dead servos lying about. These were from my Esky Belt heli.

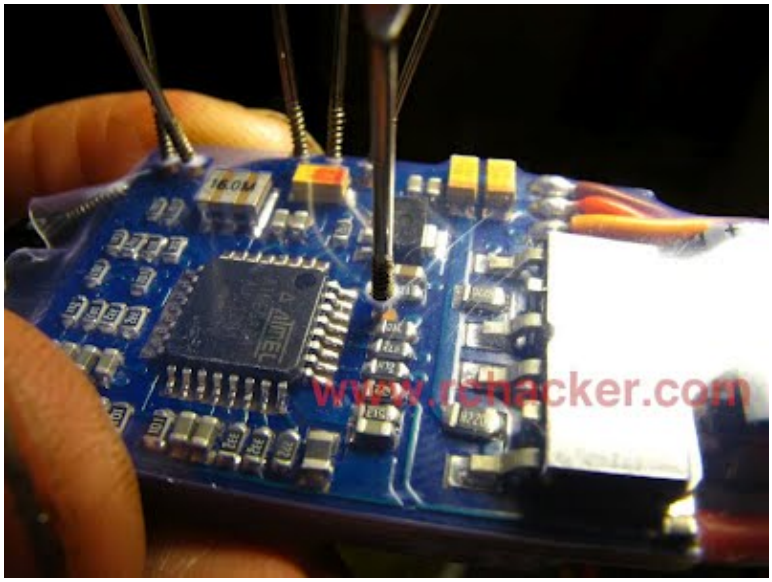

What are looking for is a pointed and threaded end. If they are not pointed it will just make it a bit more difficult but not impossible.



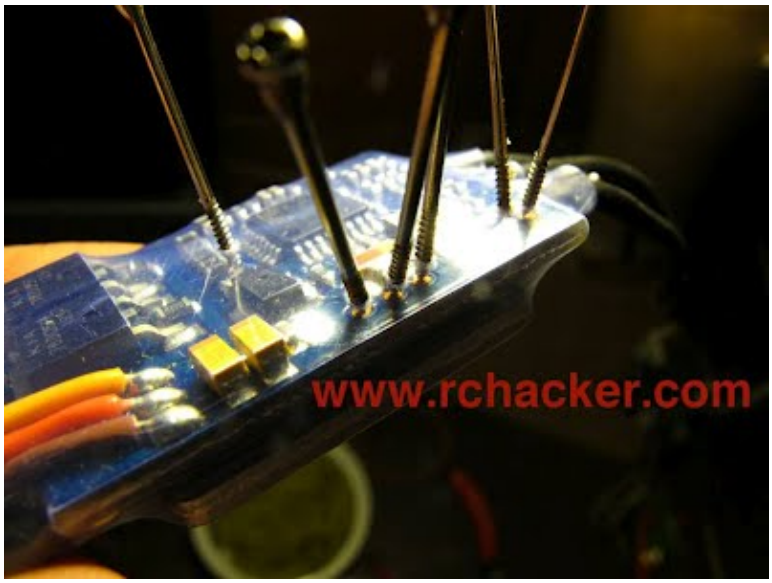
Here I have my favorite pointy tool which is a stainless steel dental tool. Any sharp pin held with pliers will do the same job.

Obviously we want the hole right over the pad.

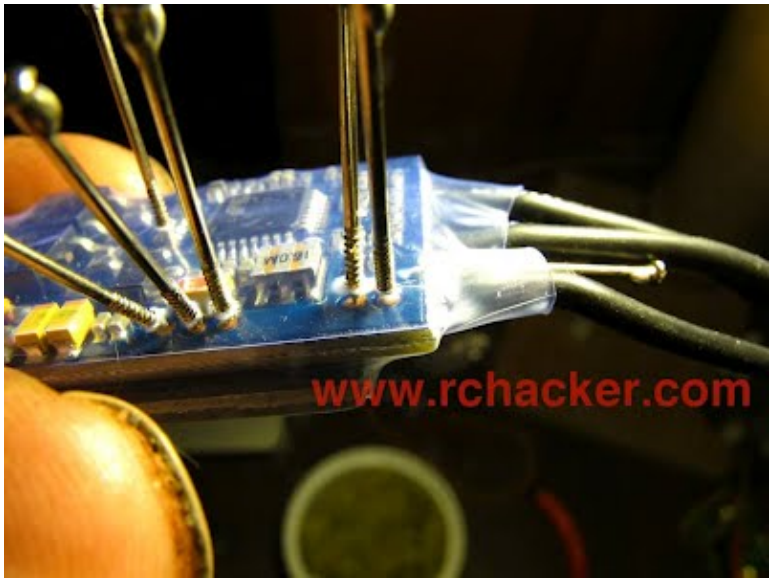

Next get some solder on your screws, you do not want the screws getting too hot while they are stuck in the heat shrink.



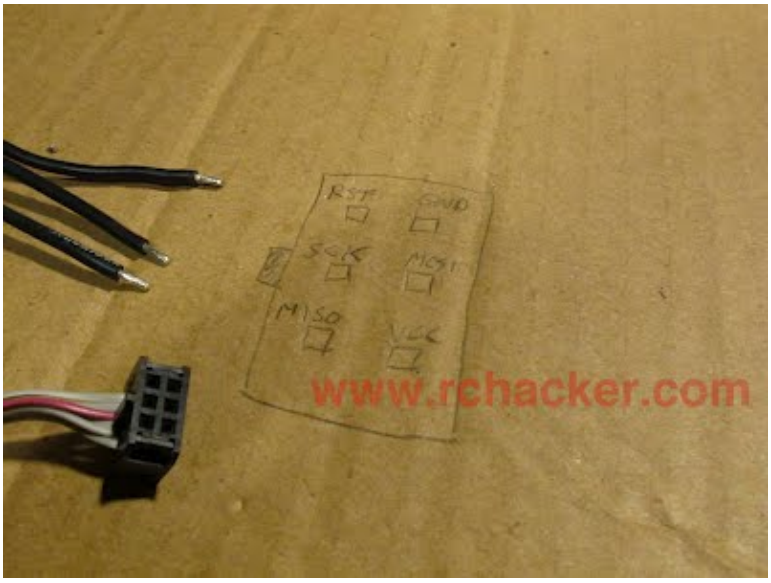

With a small screwdriver or even you fingers and a small amount of pressure twist the screw into the hole until it bites.



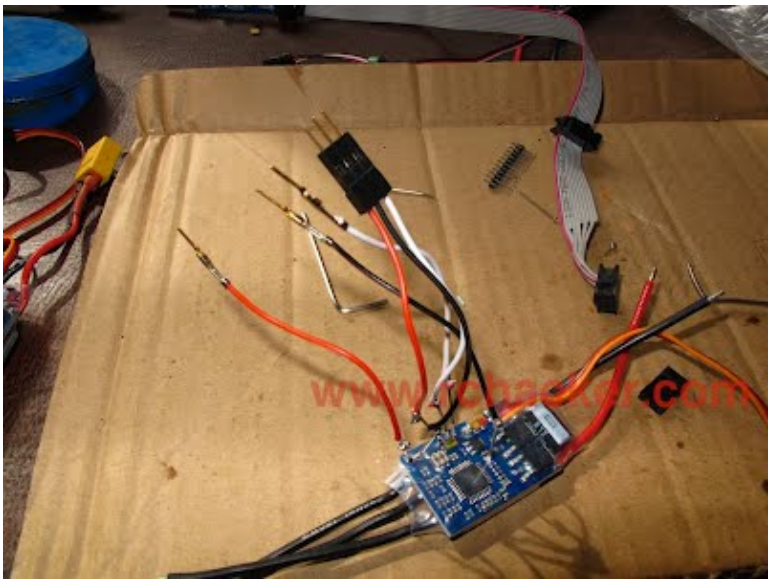

These three you can offset the holes a little to avoid shorts.



Apologies for the dirty nails. These two are tricky because the heat shrink is much closer to the board, lifting it with a screwdriver helps.
Once they are all in, we can solder our wires on then match them up with the right connections on the programmer.




This diagram should help.



Part way there...


Double check all your connections with a multimeter and that each screw is touching its pad and not other screws Put it aside for now we now have to sort the software out.

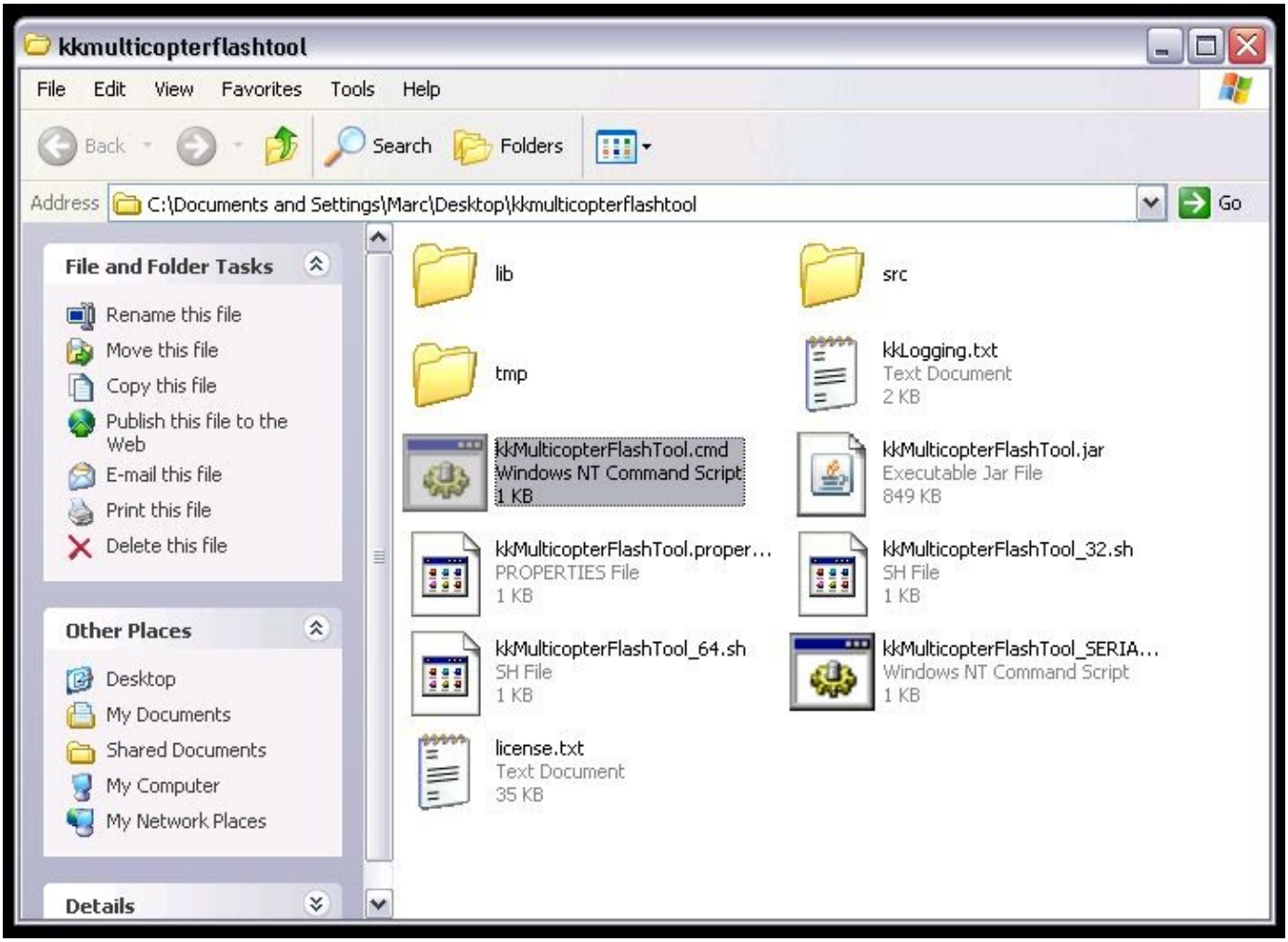
## Step 3: Flashing.

Two options here KKMultiTool or the good old command line.

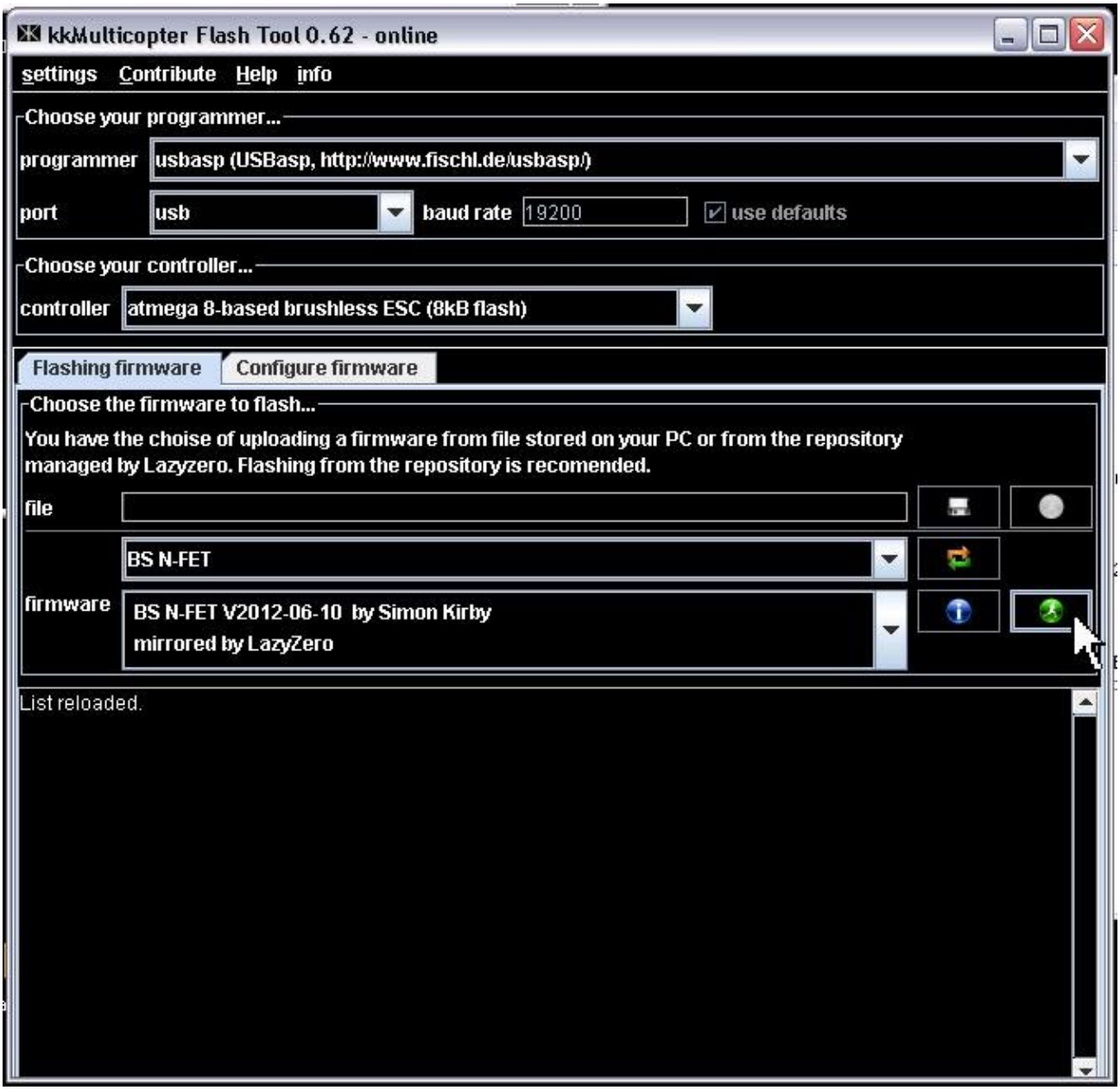
## Flashing with KKMultiTool.

Go to the kkMultiTool website and download the latest stable version.

Unzip it open the folder and double click on **kkMultiCopterFlashTool.cmd**




You may not see the .cmd extension.




Baud rate to 9600!
You need to change the controller to **atmega 8-based brushless ESC (8kB flash)**
And select **BS N-FET** for the firmware.
WARNING this next step will completely erase the HK firmware if you have a programming card it will no longer work and you cannot go back.
Plug in your programmer, check the connections to the pads and click on the green running man everything should happen.

If you see AVRDude successful you are done.

## Configuring, compiling and flashing from the source.

Here I want to show you how to compile (assemble) from source. The process is not that hard and it will allow you to change some compile time options like the motor direction, brake and car style forward/reverse.

*SimonK has also recently implemented a boot loader which means once flashed the first time you can subsequently re-flash the firmware via just the servo plug. No need to muck about connecting to pads. He does this via the Turnigy USB Linker. Lazyzero has a brief tutorial on using it with KKMultiTool here.*

**Install avrasm2**

First you are going to need an AVR assembly compiler. This will take the .asm and .inc files and make them into a .hex file. ie turn assembly code into machine code.

Simon uses avra on linux. Avra is open source and they say is almost 100% compatible with avrasm2 (Atmels compiler).

Gurus can go ahead and compile avra from source (it might already be on your linux box). I am going to use avrasm2 which is a little easier to install. In fact it is just one standalone executable, no installer, no dlls.

Unfortunately you cannot simply download avrasm2 from Atmel. In their wisdom it only comes with the monstrous (1 gig installed) AVR Studio and at that buried way down in some obscure directory. Being such a great guy, I have made a nice zip file with avrasm2.exe in it. I can even do this legally as long as I include the license file with it.

*Usually avrasm2 also comes with a bunch of include files for all the processors, so if you are going to get into assembler you will need to track them down. SimonK has the relevant include file included with his source so all you need here is avrasm2.exe.*

Copy avrasm2.exe to a directory somewhere and then execute it from the command line. You should see something like this:

```
AVRASM: AVR macro assembler 2.1.51 (build 39 May  4 2012 15:31:44)
Copyright (C) 1995-2012 ATMEL Corporation
```

Along with a couple of lines about usage and accessing help.

I run avrasm2 under wine on osx. I installed wine using macports. I also found a nice little shell script so I can run avrasm2 from the command line just like you would in windows. For those interested here is the shell script.

```
#!/bin/bash
wine ~/avrassembler/avrasm2.exe -I ~/avrassembler/include -fI $1 $2 $3 $4
$5
```

Send me an email and I will add detailed instructions for the mac users.

**Install AVRDude**

~~Download and install winavr. It is a package of AVR tools that includes avrdude.~~

*Ok I've been bashing my head installing this stuff on windows and my poor XP virtual machine ran out of space and chucked a wobbly. I can see know why the avrdude developers do not make a nice windows exe easily available. They are thinking: please Atmel ditch this windows crap.*

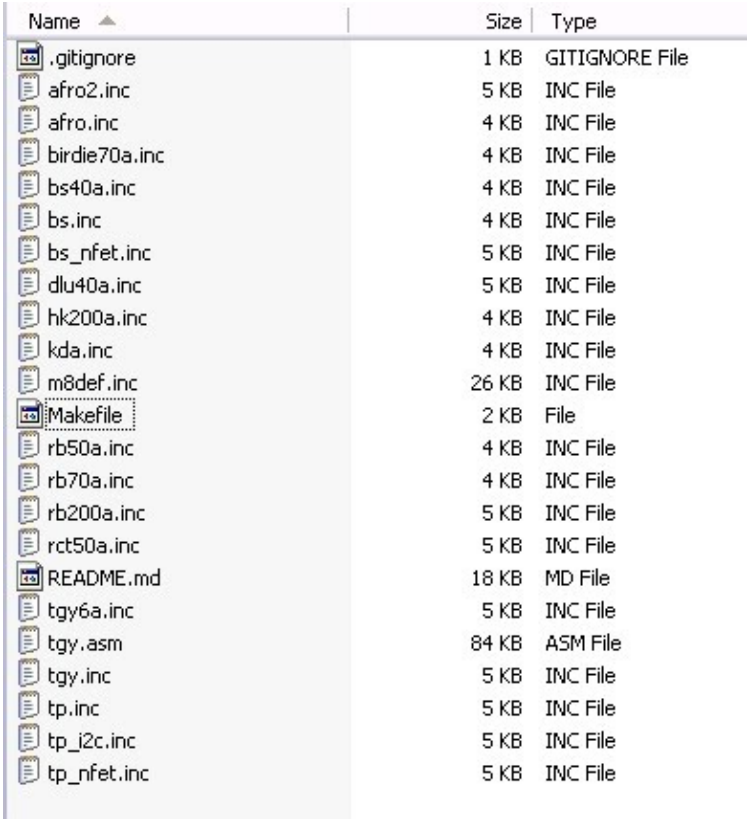
On your mac with macports just type `sudo port install avrdude` suck on that Bill.

Sorry I lost it there a bit...

I have taken avrdude.exe and avrdude.conf (configuration file) out of the winavr package and supplied it below. Put it with avrasm2 and our super light weight AVR development environment is ready to go.

Next get the source code. Simon has named the project **tgy**. You can get the latest from github or use the exact same version that I used when making the Bang Buck Quad.

Unzip it and have a look at the contents. You can use any text editor to look at the files.



| Name | Size | Type |
|---|---|---|
| .gitignore | 1 KB | GITIGNORE File |
| afro2.inc | 5 KB | INC File |
| afro.inc | 4 KB | INC File |
| birdie70a.inc | 4 KB | INC File |
| bs40a.inc | 4 KB | INC File |
| bs.inc | 4 KB | INC File |
| bs_nfet.inc | 5 KB | INC File |
| dlu40a.inc | 5 KB | INC File |
| hk200a.inc | 4 KB | INC File |
| kda.inc | 4 KB | INC File |
| m8def.inc | 26 KB | INC File |
| Makefile | 2 KB | File |
| rb50a.inc | 4 KB | INC File |
| rb70a.inc | 4 KB | INC File |
| rb200a.inc | 5 KB | INC File |
| rct50a.inc | 5 KB | INC File |
| README.md | 18 KB | MD File |
| tgy6a.inc | 5 KB | INC File |
| tgy.asm | 84 KB | ASM File |
| tgy.inc | 5 KB | INC File |
| tp.inc | 5 KB | INC File |
| tp_i2c.inc | 5 KB | INC File |
| tp_nfet.inc | 5 KB | INC File |

**.gitignore** is used by github where the code is stored.
**Makefile** is a script that compiles all the binaries at once. We are only interested in one type of ESC at a time so I am not going to cover it here.
**README.md** read it! Its got lots of information and instructions on calibrating your ESC once flashed.
The inc files are for each of the different types of ESCs. We will be using **bs_nfet.inc**
**tgy.asm**. This is the main source file and here we will make some changes. Open it up with any text editor.

tgy.asm

tgy.asm:83 <No selected symbol>

```
;
; Don't set WDTON if using the boot loader. We will enable it on start.
;
;-- Board ---------------------------------------------------------------
;
; The following only works with avra or avrasm2.
; For avrasm32, just comment out all but the include you need.
.if defined(afro_esc)
.include "afro.inc"      ; AfroESC (ICP PWM, I2C, UART)
.elif defined(afro2_esc)
.include "afro2.inc"         ; AfroESC 2 (ICP PWM, I2C, UART)
.elif defined(birdie70a_esc)
.include "birdie70a.inc"     ; Birdie 70A with all nFETs (INT0 PWM)
.elif defined(bs_esc)
.include "bs.inc"        ; HobbyKing BlueSeries / Mystery (INT0 PWM)
.elif defined(bs_nfet_esc)
.include "bs_nfet.inc"       ; HobbyKing BlueSeries / Mystery with all nFETs (INT0 PWM)
.elif defined(bs40a_esc)
.include "bs40a.inc"         ; HobbyKing BlueSeries / Mystery 40A (INT0 PWM)
.elif defined(dlu40a_esc)
.include "dlu40a.inc"        ; Pulso Advance Plus 40A DLU40A inverted-PWM-opto (INT0 PWM)
.elif defined(hk200a_esc)
.include "hk200a.inc"        ; HobbyKing SS Series 190-200A with all nFETs (INT0 PWM)
.elif defined(kda_esc)
.include "kda.inc"       ; Keda Model 12A - 30A (INT0 PWM)
.elif defined(rb50a_esc)
.include "rb50a.inc"         ; Red Brick 50A with all nFETs (INT0 PWM)
.elif defined(rb70a_esc)
.include "rb70a.inc"         ; Red Brick 70A with all nFETs (INT0 PWM)
.elif defined(rb200a_esc)
.include "rb200a.inc"        ; Red Brick 200A (black) with all nFETs (INT0 PWM)
.elif defined(rct50a_esc)
.include "rct50a.inc"        ; RCTimer 50A with all nFETs (INT0 PWM)
.elif defined(tp_esc)
.include "tp.inc"        ; TowerPro 25A/HobbyKing 18A "type 1" (INT0 PWM)
.elif defined(tp_i2c_esc)
.include "tp_i2c.inc"        ; TowerPro 25A/HobbyKing 18A "type 1" (I2C)
.elif defined(tp_nfet_esc)
.include "tp_nfet.inc"       ; TowerPro 25A with all nFETs "type 3" (INT0 PWM)
.elif defined(tgy6a_esc)
.include "tgy6a.inc"         ; Turnigy Plush 6A (INT0 PWM)
.else
.include "tgy.inc"       ; TowerPro/Turnigy Basic/Plush "type 2" (INT0 PWM)
.endif

.equ    BOOT_LOADER = 1 ; Enable or disable boot loader

.equ    I2C_ADDR    = 0x50  ; MK-style I2C address
.equ    MOTOR_ID    = 1 ; MK-style I2C motor ID, or UART motor number

.equ    TIME_ACCUMULATE = 0 ; Accumulate 4 commutations timing method
.equ    TIME_HALFADD    = 0 ; Update half timing method
```

Scroll down to the list of include files. I have pretty colors because I am using xcode. Since we are not using the makefile we need to comment out everything but the include file we need. Comments in assembler are any line starting with a semicolon.

tgy.asm

tgy.asm:119 <No selected symbol>

```
;
; Don't set WDTON if using the boot loader. We will enable it on start.
;
;-- Board ---------------------------------------------------------------
;
; The following only works with avra or avrasm2.
; For avrasm32, just comment out all but the include you need.
;.if defined(afro_esc)
;.include "afro.inc"         ; AfroESC (ICP PWM, I2C, UART)
;.elif defined(afro2_esc)
;.include "afro2.inc"        ; AfroESC 2 (ICP PWM, I2C, UART)
;.elif defined(birdie70a_esc)
;.include "birdie70a.inc"    ; Birdie 70A with all nFETs (INT0 PWM)
;.elif defined(bs_esc)
;.include "bs.inc"       ; HobbyKing BlueSeries / Mystery (INT0 PWM)
;.elif defined(bs_nfet_esc)
.include "bs_nfet.inc"      ; HobbyKing BlueSeries / Mystery with all nFETs (INT0 PWM)
;.elif defined(bs40a_esc)
;.include "bs40a.inc"        ; HobbyKing BlueSeries / Mystery 40A (INT0 PWM)
;.elif defined(dlu40a_esc)
;.include "dlu40a.inc"       ; Pulso Advance Plus 40A DLU40A inverted-PWM-opto (INT0 PWM)
;.elif defined(hk200a_esc)
;.include "hk200a.inc"       ; HobbyKing SS Series 190-200A with all nFETs (INT0 PWM)
;.elif defined(kda_esc)
;.include "kda.inc"      ; Keda Model 12A - 30A (INT0 PWM)
;.elif defined(rb50a_esc)
;.include "rb50a.inc"        ; Red Brick 50A with all nFETs (INT0 PWM)
;.elif defined(rb70a_esc)
;.include "rb70a.inc"        ; Red Brick 70A with all nFETs (INT0 PWM)
;.elif defined(rb200a_esc)
;.include "rb200a.inc"       ; Red Brick 200A (black) with all nFETs (INT0 PWM)
;.elif defined(rct50a_esc)
;.include "rct50a.inc"       ; RCTimer 50A with all nFETs (INT0 PWM)
;.elif defined(tp_esc)
;.include "tp.inc"       ; TowerPro 25A/HobbyKing 18A "type 1" (INT0 PWM)
;.elif defined(tp_i2c_esc)
;.include "tp_i2c.inc"       ; TowerPro 25A/HobbyKing 18A "type 1" (I2C)
;.elif defined(tp_nfet_esc)
;.include "tp_nfet.inc"      ; TowerPro 25A with all nFETs "type 3" (INT0 PWM)
;.elif defined(tgy6a_esc)
;.include "tgy6a.inc"        ; Turnigy Plush 6A (INT0 PWM)
;.else
;.include "tgy.inc"      ; TowerPro/Turnigy Basic/Plush "type 2" (INT0 PWM)
;.endif

.equ    BOOT_LOADER = 1 ; Enable or disable boot loader

.equ    I2C_ADDR    = 0x50  ; MK-style I2C address
.equ    MOTOR_ID    = 1 ; MK-style I2C motor ID, or UART motor number

.equ    TIME_ACCUMULATE = 0 ; Accumulate 4 commutations timing method
.equ    TIME_HALFADD    = 0 ; Update half timing method
```

Done. Scroll down a little to see the compile time options that we may be interested in.

```
.equ    MOTOR_BRAKE = 0 ; Enable brake
.equ    MOTOR_REVERSE   = 0 ; Reverse normal commutation direction
.equ    RC_PULS_REVERSE = 0 ; Enable RC-car style forward/reverse throttle
.equ    SLOW_THROTTLE   = 0 ; Limit maximum throttle jump to try to prevent overcurrent
```

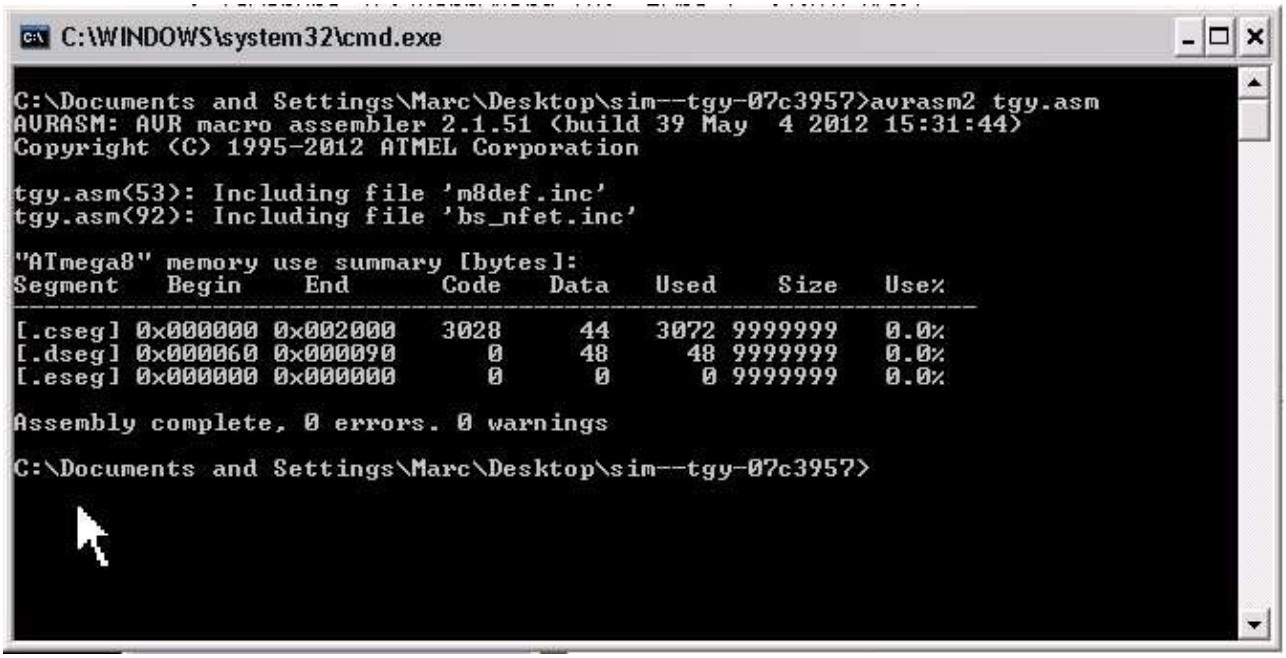
Nothing complicated `1` is enabled and `0` is disabled. For a multicopter setup the only one you might be interested in is `MOTOR_REVERSE`, and only then if you have hardwired your ESC.

Save your changes.

Lets compile it.

Copy avrasm2.exe, avrdude.exe and avrdude.conf into the same folder with the source files.

On the command line type:
**avrasm2 -fI tgy.asm**




Done, you should now have `tgy.hex` in the same folder.

Now connect up your programmer to your ESC and computer. The programmer supplies power to the ESC during programming so there is no need to provide power or have it plugged into a reciever.

WARNING this next step will completely erase the HK firmware if you have a programming card it will no longer work and you cannot go back.

```
avrdude -c usbasp -p m8 -U flash:w:tgy.hex
```

If all goes well avrdude should do its thing. As long as you get the message AVRDude Successful you are done. Don't worry about any SCK errors if you see them.

Screen shots coming when I re-flash one. :)

If you are doing any variation on what I have described above, ie a different ESC, it might be a safe bet to do your first tests with a current limited supply or a 9V battery.

More information and help can be found on this gigantic rcgroups thread.

Happy flashing.