Albis motor controller IC

Setup manual

version 2.50

Adliswil, Switzerland
bmp72@hotmail.com

Some general remarks:

Before connecting the battery make sure the output stage polarity is set correctly. Use a fuse in the battery line. There should be no diode in the battery line. Make sure the current sensors are connected conform the schematic and that the default calibration values are written. An error here can cause severe damage to the output stage (again, make sure to have a fuse in the battery line).

The menu system was designed using 'gtkterm' under Ubuntu. The baudrate is 115200 baud, 8 data bits, 1 stop bit, no parity, no handshaking.

Under windows the freely available program 'termite' can be used, I tested version 2.8. The main difference between the two is that gtkterm transmits characters as they are typed while termite waits for an return before sending the characters. This means that whenever the chip waits for 'press any key', in termite the chip will only respond correctly when the return key is pressed. In the serial port settings of termite select 'append CR' and disable 'local echo'

The chip can be placed in the setup mode by closing the setup switch connecting pin 19 to ground while resetting the chip. Upon entering setup mode a keypress on the PC is necessary before the chip displays the main setup menu.

Except for the output stage polarity all data during setup is recorded in RAM. It is only stored in EEPROM for motor use when the correct menu option for this has been selected. Do not turn off the power before writing the new setup to EEPROM !

Some menu options have a yes/no or high/low setting. Selecting this type of menu option toggles the setting, no further input is required.

Every time the chip displays a menu all internal 16 bit variables are translated into decimal numbers. When new numbers are entered the controller IC immediately translates these into a 16 bit number (which it uses for running the motor). The consequence of this is that in between entering a new number and the displaying of the updated menu 2 rounding operations have occurred, first after translating to 16 bits and then after translating back to decimal. This gives an insignificant discrepancy between the entered data and displayed data.

The chip does not perform any checks to see whether data is valid, this is up to the users common sense. Entering negative numbers or letters where positive numbers are required will result in the refreshed menu displaying data not correlated to the entered characters.

Some displayed values are calculated using more than one 16 bit variable, changing one variable can change the displayed decimal value of more than one menu option. Notable examples of far-reaching variables: sensor transimpedance and f_sample

Pressing the setup key while in motor mode will write the current sensor gain and offset calibration values to EEPROM. This is indicated by all four drive LEDs lighting up.

```
########################################
#   (c)opyright 2015, B.M. Putter      #
#   Adliswil, Switzerland              #
#   bmp72@hotmail.com                  #
#                                      #
#  version 2.50                        #
#  experimental, use at your own risk  #
########################################


0) mode: HF tone
a) PWM parameters
b) current settings
c) throttle setup
d) erpm limits
e) battery
f) current sensor calibration
g) control loop coefficients
h) filter bandwidths
i) FOC motor impedance
j) CAN setup
k) recovery only
l) hall sensored only
m) temperature sensors
n) miscellaneous
z) store parameters in ROM for motor use
```

This is the main setup menu as displayed when the chip is in the setup mode. From this menu all the sub menus can be selected.

Option 0 toggles between the three motor start options of this version: HF tone, Sensorless or Hall sensored .

```
a) PWM frequency: 21kHz
b) deadtime: 499ns
c) dutycycle testsignal: 50%
d) toggle high side polarity, now active HIGH
e) toggle low side polarity, now active HIGH
f) test PWM signals

g) autocomplete

h) loop sample frequency: 41.03 kHz

z) return to main menu

------>
```

This is the setup menu for the output stage. Option a sets the PWM frequency that is used for operating the output stage. The deadtime used between the switching of the high/low side transistors is set using option b. With option c the dutycycle for the testsignal (option f) is set, this option has no effect when the controller is not in setup mode.

Options d & e MUST BE SET FIRST AND BEFORE THE BATTERY IS CONNECTED !!! These options determine whether a high (active HIGH) or low (active LOW) signal is used to turn on the high/low side FET. Unlike all other options (which are only saved to EEPROM when the controller IC is instructed to do so) these options are directly written to EEPROM.

With option f a test-signal having the properties of options a-c is generated. After selecting this option the controller will ask you to press any key before the test signal is turned on. After again pressing a key the test signal is turned off and the controller returns to the menu.

Option g is the autocomplete. In every menu where there is an autocomplete option, all the options below it are autocompleted. In this case option h, which is the frequency at which the control loops operate (the frequency at which the chip makes measurement, calculates the new output signals and writes new values to the PWM output stages). The value for h is based on the PWM frequency, times 2 minus 1 kHz (but below 45 kHz). Option h can of course be manually entered by selecting it.

```
                                                        b) current settings

    a) current sensor transimpedance: 100.00 mV/A
    b) maximum motor phase current: 13.9 A
    c) maximum battery current, motor use: 13.9 A
    d) maximum battery current, regen: 0.0 A

    e) autocomplete

    f) HF current, base level (HF only): 0.6 A
    g) HF current, proportional factor (HF only): 1.0000
    h) maximum phase current in drive 2 (HF only): 6.6 A
    i) phase current for forcing motor position: 2.7 A
    j) maximum shutdown error current, fixed: 3.4 A
    k) maximum shutdown error current, proportional: 1.7 A
    l) applied braking current (phase) on direction change: 0.0 A
    m) offset filtering (phase) current limit: 0.0 A
    n) maximum field weakening current: 0.0 A

    z) return to main menu

    ------>
```

This is the setup menu for anything current related. Options a-d are the minimum settings that must be entered by the user, the rest can be autocompleted. Options a sets the current sensor transconductance, option b the maximum motor phase current amplitude, option c the maximum battery current in motor mode and option d the maximum battery (charge) current during regen.

When the option is set (in a later menu) to run position sensing at standstill by means of a High Frequency tone, option f sets the amplitude of the tone for throttle closed. As the throttle is opened the amplitude of the HF tone increases proportionally to the phase current as indicated by option g. Since part of the current sensor range is used for the HF tone, not all is available for the actual powering phase current. The max powering phase current remaining is given by option h. At startup and during measurements later on the motor must be brought in a pre determined position, this is done with a current as specified in option I.

```
                                                        b) current settings

    a) current sensor transimpedance: 100.00 mV/A
    b) maximum motor phase current: 13.9 A
    c) maximum battery current, motor use: 13.9 A
    d) maximum battery current, regen: 0.0 A

    e) autocomplete

    f) HF current, base level (HF only): 0.6 A
    g) HF current, proportional factor (HF only): 1.0000
    h) maximum phase current in drive 2 (HF only): 6.6 A
    i) phase current for forcing motor position: 2.7 A
    j) maximum shutdown error current, fixed: 3.4 A
    k) maximum shutdown error current, proportional: 1.7 A
    l) applied braking current (phase) on direction change: 0.0 A
    m) offset filtering (phase) current limit: 0.0 A
    n) maximum field weakening current: 0.0 A

    z) return to main menu

    ------>
```

Options j and k set the maximum allowed error current at 0 signals to the motor (option j) and at max signal amplitude to the motor (sum of j+k). The error current serves as a fault detection mechanism, when the level is violated the controller will default to drive_0.

When reverse is used the motor cannot instantly change direction. The mechanical energy of the spinning motor / moving vehicle must be reduced by either letting it spool down freely (option l = 0 A) or by specifying a regen braking current to actively slow the motor down (option l > 0 A). Note that when a braking current is specified, option d must be set to allow for regen current.

In drive_3 the controller can perform auto-offset calibration of the current sensors. This is not always a good idea, especially when currents are high and the sensors are non-linear. With option m the chip only calibrates the current sensors when the phase current is below the set value. By specifying 0 the auto offset calibration is effectively turned off. If auto calibration is used dependent on the motor and current sensors the motor can start bucking after about 30 seconds in drive_3, this is an indication that option m must be reduced.

```
                                                        b) current settings

    a) current sensor transimpedance: 100.00 mV/A
    b) maximum motor phase current: 13.9 A
    c) maximum battery current, motor use: 13.9 A
    d) maximum battery current, regen: 0.0 A

    e) autocomplete

    f) HF current, base level (HF only): 0.6 A
    g) HF current, proportional factor (HF only): 1.0000
    h) maximum phase current in drive 2 (HF only): 6.6 A
    i) phase current for forcing motor position: 2.7 A
    j) maximum shutdown error current, fixed: 3.4 A
    k) maximum shutdown error current, proportional: 1.7 A
    l) applied braking current (phase) on direction change: 0.0 A
    m) offset filtering (phase) current limit: 0.0 A
    n) maximum field weakening current: 0.0 A

    z) return to main menu

    ------>
```

Option n sets the maximum field weakening current. Field weakening is applied automatically upto the set current level (keep n at 0.0 A for no field weakening). Field weakening will increase dissipation, but gives a (motor and speed dependent) 'virtual' battery voltage increase of

4 * 3.14 * (e_rpm / 60) * L * I_fieldweak Volt.

with L the inductance indicated by the FOC motor impedance menu.

```
                                                        c) throttle setup

    a) calibrate throttle 1
    b) calibrate throttle 2
    c) polynomial coefficients throttle 1 (x, x^2, x^3): 1.0000, 0.0000, 0.0000
    d) polynomial coefficients throttle 2 (x, x^2, x^3): -0.0002, -0.0002, -0.0002
    e) use analog throttle 1: YES
    f) use analog throttle 2: NO
       receive throttle over CAN: NO
    g) TX throttle over CAN: NO
    h) test throttle

    z) return to main menu

    ------>
```

Upto 2 analog throttles can be connected to the controller IC. Options a and b are for calibration. The IC will measure the throttle voltage for throttle closed and throttle open. The throttle closed voltage must be lower than the throttle open voltage.

Either one of the throttle channels can be used for variable strength regen. This can be achieved by using negative polynomal coefficients (see the next slide)

Toggle options e and f indicate to the IC which analog throttle to use. When both are 'NO' the IC will automatically set the 'receive throttle over CAN' option. When the analog throttles are used there is an option to transmit the throttle information over CAN bus to other motor controller IC's.

When analog throttles are used also the 'reverse' switch can be used to select reverse. The state of this switch will also be transmitted over CAN. An IC receiving throttle information over CAN will also receive the 'reverse' information.

Based on the calibration information the throttle voltage will be transformed into variable x (x_1 for throttle 1, x_2 for throttle 2) in the range of 0 to 1. Out of range voltages are rounded to 0 or 1. Variables x_1, x_2 and the state of the reverse switch are transmitted or received over CAN bus.

Variables $x_{1,2}$ are transformed into variables $y_{1,2}$ by means of a polynomal function (the purpose of which is to be able to make different types of throttle response curves):

$$y_1 = a_1 x_1 + b_1 x_1^2 + c_1 x_1^3$$
$$y_2 = a_2 x_2 + b_2 x_2^2 + c_2 x_2^3$$

Options c and d are used to input the coefficients $a_{1,2}$, $b_{1,2}$ and $c_{1,2}$. Valid values are between -7.999 and +7.999.

Based on the throttle information the motor's phase current is given by:

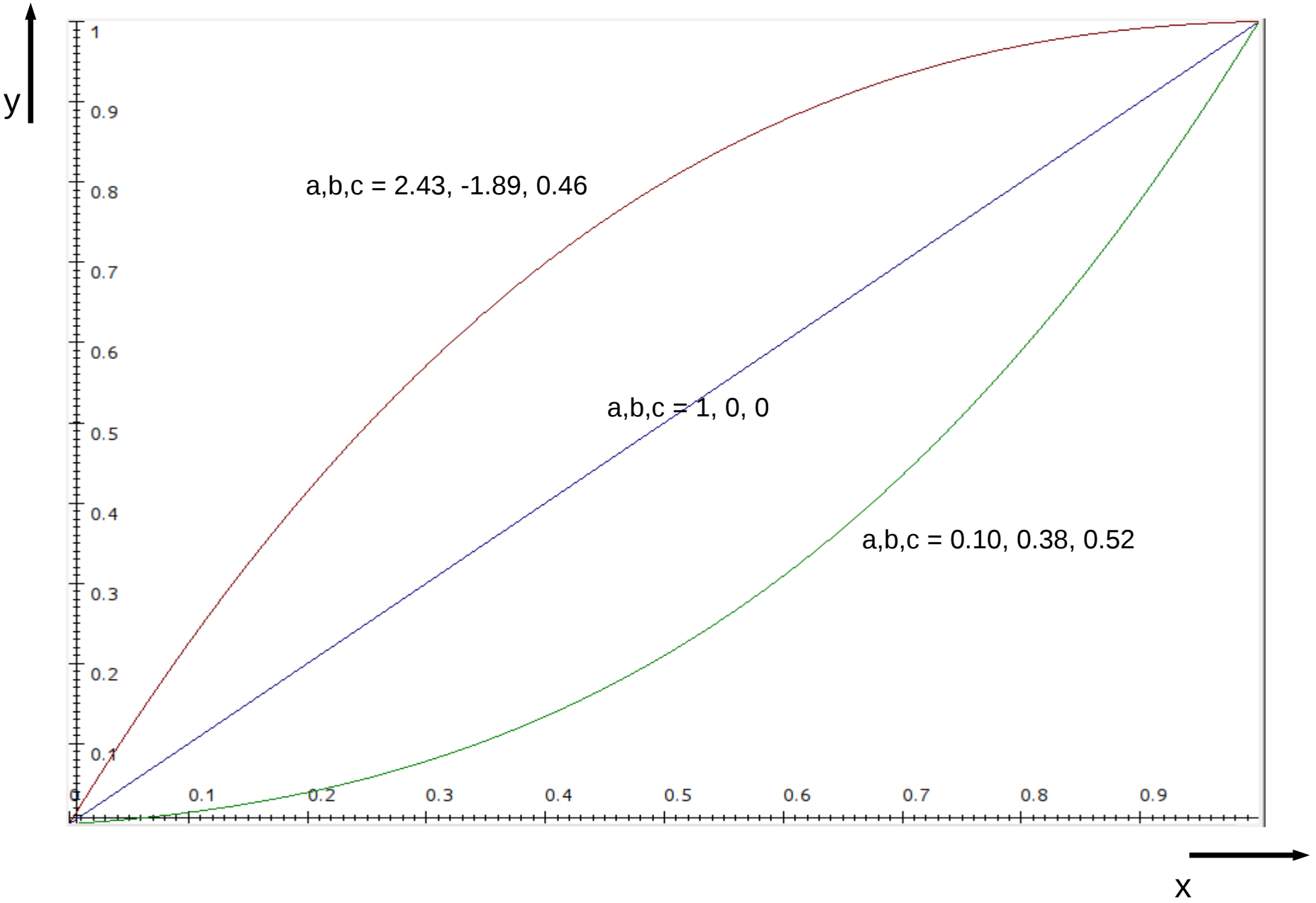
phase current = maximum phase current * (y_1 + y_2)

The maximum phase current is entered under menu d, option c.

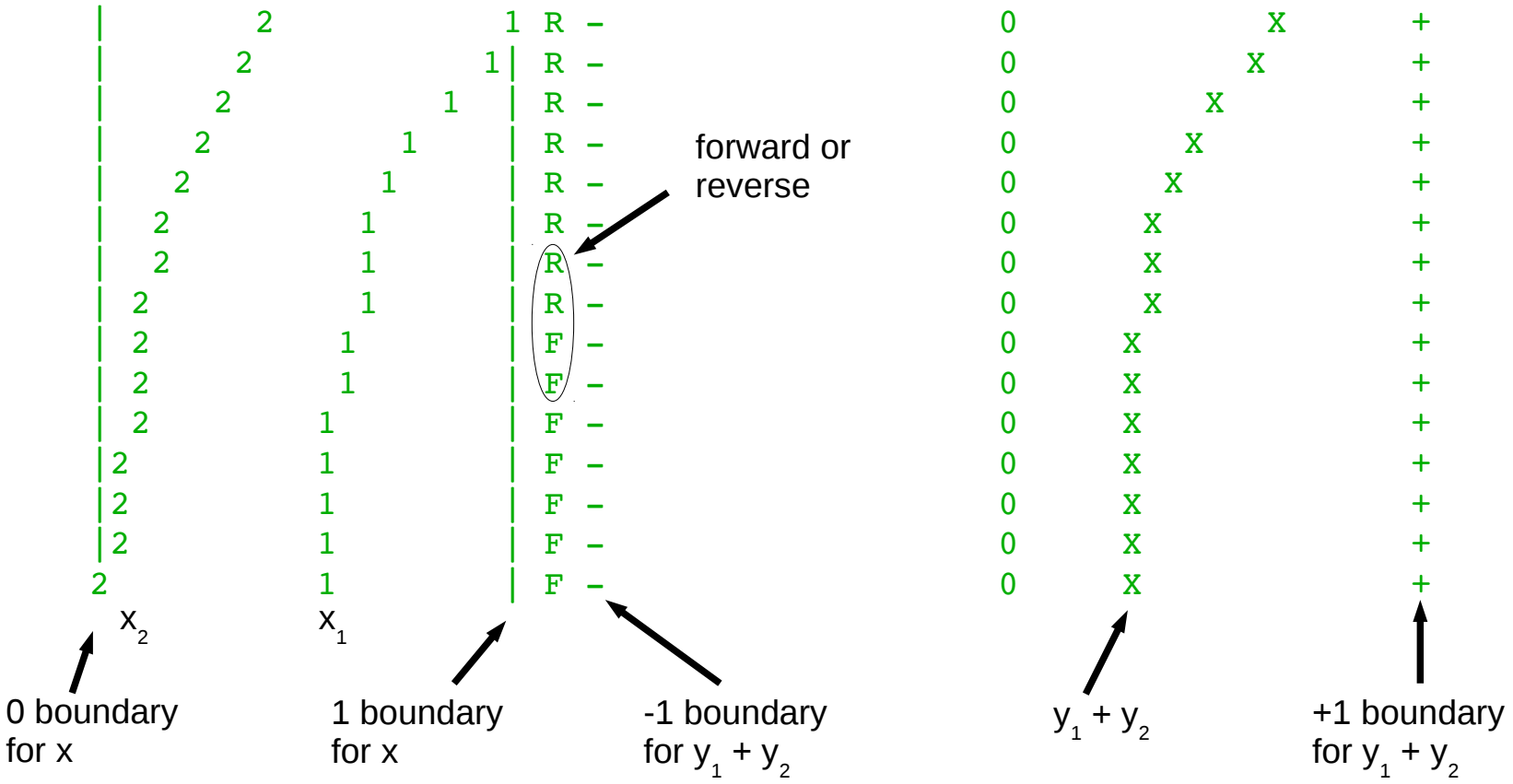
Variables x_1, x_2 are shared over CAN bus. Every motor controller IC however has its own set of a, b and c coefficients and its own maximum phase current setting. This allows the combining of motors with different ratings to operate of a shared throttle.

A negative phase current means current will flow to the battery, this is how variable strength regen can be obtained. When the conditions are such that the throttle requested phase current means that the maximum battery current or maximum battery regen current will be violated the phase current is automatically reduced.

some example throttle curves :



Because of the complexity of the throttle setup and it's importance for safety a test function has been implemented (option h).



The throttle information as shown above is updated around 30 times a second. The receive / transmit over CAN functionality is active so it's possible to test a multi-controller setup.

```
                                                        d) erpm limits

    a) erpm limit, forward: 49.96 k-erpm
    b) erpm limit, reverse: 49.96 k-erpm
    c) accept direction change below: 75 erpm
    d) transition erpm drive 2 -> 3: 789 erpm
    e) transition erpm drive 3 -> 2: 187 erpm

    z) return to main menu

    ------>
```

The chip has 4 different running mode, called drive_0 to drive_3. When not using recovery, drive_0 is the startup mode where the chip ends up at reset or after an error current event. It transitions to drive_1 when the throttle is closed and the motor is at standstill. Drive_1 is only used in combination with the HF tone, in this mode the motor is brought to a preset position (the force position current (option I of the current menu) must be high enough to do this !) Finally drive_2 is the mode in which the motor is started (either with or without HF tone) and drive_3 is the main FOC running mode in which 99% of the driving around is done. The drive mode 1 is different when recovery is selected, see later in this manual.

In the erpms menu, options a and b set the maximum motor speed (electrical rpm !) for forward and and reverse direction. The reverse pin is treated as a request for reverse. When the request comes the motor is spooled down with the braking current as set in the currents menu, and the request is granted once the motor speed is below the speed set in option c.

The transition erpms between starting the motor and the FOC running mode are set with options d and e. Note that some hysteresis should be build in by specifying option d larger than option e.

```
                                                                e) battery

    a) battery voltage: 64.1 V

    z) return to main menu

    ------>
```

The battery menu at the moment only allows you to enter the battery voltage. This value is (at the moment only) used to calculate the correct motor inductance value in a later menu.

```
                                                    f) current sensor calibration

   a) restore calibration, autocomplete
   b) perform offset measurement
      sensor a: 0.0 mV
      sensor b: 0.0 mV
      sensor c: 0.0 mV
   c) perform gain measurement
      channel a: 99.99 %
      channel b: 99.99 %
      channel c: 99.99 %
   d) online gain calibration update rate: 0.302 %

   z) return to main menu

   ------>
```

The current sensors can be calibrated for gain and offset. Option a resets all calibration values to default. In practise I found gain and offset calibration is only necessary for the HF mode.

Option b performs the offset measurement. Good offset calibration is necessary for smooth (very very) low sensorless operation. When you select option b the motor will make a noise as the controller makes the measurement.

Option c performs the gain calibration. The position detection at standstill with the HF is based on the inductances in the motor having different values, it is based on different 'gains' from the controller output voltages via the motor to the current sensors. Any gain error in the current sensors is added / taken into account in this detection method. As we only want gain differences from the motor inductances and not from differences in the current sensors, the gain differences between the current sensors must be calibrated for.

When you perform the gain measurement the motor will make a noise and move over an e-rotation. The motor must be able to spin unhindered during this measurement. If the motor does not move smoothly, the positioning current (option I in the current menu) must be increased. Make sure that after the measurement the gain % add up to close to 300%, if not restore calibration and try again.The gain calibration also runs when the motor is spinning in drive 2 (with the HF tone), the update speed is set with option d (every e-rotation the gains are updated with this magnitude).

```
a) autocomplete                                          g) control loop coefficients

  phase control loop, drive 3
b) 1st order: 480
c) 2nd order: 48.0000
d) 3rd order: 0.6000
  phase control loop, drive 2
e) 1st order: 480
f) 2nd order: 48.0000
g) 3rd order: 0.0299
  amplitude control loop
h) 1st order: 200
i) 2nd order: 3.0000
j) 3rd order: 0.0000
  field weakening control loop
k) 1st order: 50
l) 2nd order: 5
m) maximum amplitude: 100 %

z) return to main menu
```

Lets assume a f_sample of 40kHz (PWM menu, option g)

Every cycle of f_sample (so 40000 times a second) the controller decides by how much to upgrade the phase and amplitude of the 3 signals driving the motor. The coefficients b to d and e to g are the maximum updates for the phase loop, the coefficients h to j the max updates for the amplitude loop. To make the algorithm fast to calculate, an update is either to add the coefficients or subtract the coefficients from the internal variables.

The internal phase is called phi. phi has a range from 0 to 65535 (32 bit, with 16 before the comma and 16 after) representing 0 to 360 degrees.

The internal amplitude is called ampli (inventive, ey ?) and has a range from -32767 to +32767 (again 32 bit with 16 after the comma), with 32767 being the maximum voltage to the motor (with Vbat lets say 60V at max ampli (so 32767) the sines to the motor will have 1.15*60 = 69 V peak to peak).

The controller uses moving-midpoint, meaning it can output sine waves with a peak-peak 15% larger than the supply, this is the reason for the 1.15. The output voltage are like the tips of a 3-prong propellor blade, the moving midpoint means the center of the propellor moves up and down. All very much like the rotor in a Wankel engine.

Every cycle the controller determines whether to increase or decrease phi and ampli (so phase and amplitude of the 3 motor signals).

b, e: every cycle this coefficient is added or subtracted from the phase phi, but it's contribution will be forgotten by the next cycle. It is kind of a momentary jump in phase, one which will be forgotten by the next cycle. It is a coefficient necessary for loop stability, it's minimum value is 10 times coefficient f. The value 480 represents (480/65536)*360 = 2.64 degrees

c, f: every cycle this coefficient is added to or subtracted from the phase phi but its contribution is not forgotten, it represents the phase advance (or retreat) every cycle. When a motor is running you need a constant phase advance, as the back emf sine waves keep advancing and the controller needs to keep up with this. The value of 48 represents (48/65536)*360 = 0.26 degrees.

d, g: every cycle the phase phi is also increased by a constant phase increase which I call phi_int. Phi_int is updated every cycle by the value of coefficient g. So if phi_int for instance is 100, it means every cycle the phase phi is automatically advanced by 100 (/65536 * 360 = 0.55 degrees). This automatic advance comes on top of the advance of coefficient c,f. For stability d,g must be less than 1/40th of c,f.

When the motor is running at a constant speed, all contributions from coefficients b,e and c,f will average out to 0. The phase advance of the internal variable phi purely comes from its automatic updating with phi_int. When the motor slows down or speeds up, the controller will immediately respond with phase updates from coefficient b,e and c,f, but coefficient d,g means also the automatic phase update phi_int will increase (speed up) or decrease (slow down). Phi_int is a measure of motor speed. With phi_int = 100, phi will be increase 40000 times a second with 100, so a total phase advance of (100 * 40000) / 65536 times 360 = 61 times 360 degrees, so 61 e-rotations per second (times 60 for per minute = 3660 erpm)

the amplitude coefficients work the same.

h: is added to the amplitude every cycle but will be forgotten by the next. With 60V battery, 200 represents a peak-peak increase/decrease of (200/32767) * 1.15 * 60 = 0.421 V
i: is the update for the amplitude variable that will not be forgotten from cycle to cycle, 3 represents 6.3 mV
j: and this coefficient accumulates in the automatic amplitude update variable which will be added to the amplitude every cycle...

The phase loop coefficients can be chosen differently for drive_2 (startup) and drive_3 (normal running). Drive_2 benefits from a lower 3rd order coefficient as too high a value puts a lot of noise on the motor speed variable phi_int which can make the controller jump too early to drive_3. Higher values for the 1st and 2nd order coefficient makes the controller faster to respond (bigger phase steps) which is good when the motor moves and you want the controller to immediately catch on.

```
                                                          h) filter bandwidths

    a) autocomplete

    b) throttle current filter -3dB freq: 100 Hz
    c) error current 50% step response time: 5.005 msec
    d) induction position filter 45 degree delay speed: 5.93 k-erpm
    e) drive 2 speed filter 50% step response time: 319.8 msec

    z) return to main menu

    ------>
```

This menu sets the filter bandwidths for various filter in the controller IC.

Option b specifies the frequency for the throttle current filtering. High is bad because it makes the FOC algorithm jumpy, low is bad because it makes it unstable. Inbetween is good.

The error currents are filtered so that the controller doesn't conk out to drive_0 for spikes which are higher than the error current settings. The response time for this filter can be set using option c.

The HF algorithm at a certain point arrives at two variables X and Y which follow a circle as the motor turns. The angle from the circle center is a measurefor the angle at which the motor is positioned. X and Y are noisy and are filtered with the filter specified under option d. Not enough filtering (high value) and the position detection is noisy. Too much filtering means the delay through the filter cause the XY position to seriously lag behind the actual motor position (when the motor is running at speed). The autocomplete calculates the correct filter frequency based on the drive_2 to 3 transition rpm.

The internal motor speed variable is too noisy to reliably make the changes between drive modes 2 and 3. Therefore a very slow filter is added (option e) and the drive mode transitions are based on the output of this filter. This prevents incorrect transitions coming from spikes on the internal variable phi_int.

```
a) autocomplete                                          i) FOC motor impedance

b) FOC measurement current: 6.9 A
c) FOC measurement erpm: 11.98 k-erpm
d) inductance scaling factor: 100 %
e) perform impedance measurement

 measured inductance: 98.5 uH
 measured resistance: 365.5 mOhm

z) return to main menu

------>
```

For the Field Oriented Control the motor impedance must be measured.

Option a autocompletes the measurement currents and erpms. Option e must be manually selected and will perform the actual measurement (the motor will make a noise). In order to display the correct impedance value, the battery voltage must be set correctly (main menu, option e).

Option d must be set to
- 100% plus
- an additional 100% for every other controller driving the same motor phases
- an additional 87% for every controller driving other motor phases having a 30 degree phase difference

So, 100% except for when you're called John and live in Costa Rica :-)

```
                                                        j) CAN setup
   a) CAN 'address': 16383
   b) CAN CFG1 as per Microchip 30F manual: 65535
   c) CAN CFG2 as per Microchip 30F manual: 65535
      RS232 output rate: 3636 Hz
   z) return to main menu

   ------>
```

This menus sets the properties of the CAN bus. Option a sets the 'address' (acceptance filter in CAN speak), the addresses of the transmitting and receiving controller must match for communication to occur. Multiple master/slaves with different addresses can use a single CAN bus. Valid values are in the 0 to 2046 range.

Options b and c configure the CAN bus data rate, see the 30F manual from Microchip. For a typical robust 100 kHz bitrate setup, use '14' for option b and '664' for option c. Throttle and reverse information (60 to 70 bits) is sent at a rate of 100 Hz.

During motor use the RS232 is dormant. Normally no information is transmitted but when the controller receives a single lower-case letter according to the table it will start transmitting 16 bit data at a rate indicated here under 'RS232 output rate'. Data is outputted as 2's complement with the high byte first. Transmission will stop once a character not in the table is received.

a	phi (slide 16)
b	phi_int (slide 16)
c	phase current, filtered
d	phase current, requested from throttle
e	amplitude (slide 18)
f	throttle 1 (x1)
g	throttle 2 (x2)
h	combined throttle after polynomals (y1+y2)

i	calibration value current sensor A
j	calibration value current sensor B
k	calibration value current sensor C

```
                                                                  j) CAN setup
a) CAN 'address': 16383
b) CAN CFG1 as per Microchip 30F manual: 65535
c) CAN CFG2 as per Microchip 30F manual: 65535
   RS232 output rate: 3636 Hz
z) return to main menu

------>
```

l	temperature (*2) reading from sensor 0
m	sensor 1
n	sensor 2
o	sensor 3
p	sensor 4
q	sensor 5
r	sensor 6
s	sensor 7
t	Max phase current based on temperature sensors
u	Field weakening current

```
                                                          k) recovery only
a) autocomplete

  phase control loop, recovery
b) 1st order: 0
c) 2nd order: 120.0000
d) 3rd order: 3.0000
  amplitude control loop, recovery
e) 1st order: 240
f) 2nd order: 12.0000
g) 3rd order: 0.0000
h) pulse when current drops below: 0.9 A
i) pulse width: 19 usec
j) pulse % for exit: 50
k) pulse % filter 50% step response time: 51.5 msec
l) speed filter 50% step response time: 7.0 msec
m) try restart for: 499 msec
n) check for spinning motor, drive_0: disabled
o) check for throttle closed, drive_0: disabled
  exit from startup to recovery at current
p) current to check: total current
q) fixed part: 3.1 A
r) proportional to throttle current, factor: 100 %
s) current filter 50% step response time: 3.0 msec

z) return to main menu

------>
```

Recovery is done by pulsing the motor (shorting it out) and then observing the currents and trying to get phase and amplitude information from this. As phase and amplitude information come closer and closer to the correct value, a pulse generates less and less current. To maintain the current level the pulse rate will increase. When the pulse rate has reached a certain level the motor is 'recovered' and normal running will commence.

When recovery is used it is indicated by the drive_1 LED (with drive_0 controller startup, drive_2 motor start with wiggle and drive_3 normal FOC running).

A pulse has the length of time as given by option i. During a pulse the motors backemf voltage will induce a current in the windings. Because of the inductance this current will keep flowing after the pulse has ended. When the current drops below the level of option h a new pulse is given. As the controller gets closer and closer to being in sync, the current induced from every pulse will be less and less. So after every pulse the current level of option h is reached sooner, to maintain this current level the pulse rate will go up as controller gets more and more in sync. The pulse rate is passed through a filter (option k) and when it reaches the level of option j the motor is declared 'recovered' and normal running will commence.

Options b to g are the phase and amplitude control loop coefficients used during recovery. They are set rather high to increase speed of recovery. Too high however and the phase/amplitude information will jump around a lot, making it difficult to accurately capture the motor. In practise this means the controller will not reach a high enough pulse rate to exit drive_1.

To start normal running the controller needs to obtain speed information during recovery, this information is filtered using option l. The number here must be less than the pulse filter option k, as the speed filter must be settled before pulse filter settles.

Whenever an error or glitch occurs the controller will exit to drive_1 and try to recover the motor. In case of a serious problem like a blown FET the controller will not be able to recover. Recovery is tried for the amount of time specified under option m, if unsuccesfull the controller will go to drive_0. Here the controller will yes/no wait for motor stop (option n) and/or throttle closed (option o). Having option n set to enabled will also detect a shorted out FET. Option o is there to prevent the motor from starting with high torque when the controller is powered on for the first time. Both options can be disabled (this enables for on the fly controller reset or turn-on).

In Toneless with recovery drive_0 is the startup mode where the controller can wait for throttle closed and motor standstill. Drive_1 is the recovery mode, drive_2 is sensorless motor start with wiggle and drive_3 is sensorless FOC. An error or glitch in drive_2 or drive_3 will make the controller jump to drive_1, the recovery mode. If recovery is unsuccesfull the controller goes back to drive_0.

With a well-setup controller it should never have to use the recovery mode during normal use.

Drive_3 is exited to recovery when the error current level is tripped or complete sync is lost.

Drive_2 is exited when the total or error current (option p) , after filtering with option s, exceeds a level calculated as the sum of option q plus the throttle phase current times option r.

The options mentioned above should be set such that the controller jumps to recovery only under abnormal circumstances. For instance: lets assume the controller is on a bike ridden off-road. At speed the rider locks the wheel to make a slide and then releases the brake to continue riding. What should happen to the controller is the following: before the slide the motor is running in drive_3, the normal running mode. Locking the wheel with the brake is an instantaneous large speed change, the error current detection should trip and the controller goes to recovery mode: drive_1. As the controller is very fast (a 1000 yards/sec rifle bullet only travels for an inch in one controller cycle) it will almost immediately declare the motor recovered as it is not rotating during the slide. Because of the low detected speed the controller will go to drive_2, motor start. As the brake is still on, dependent on the throttle a certain amount of phase current flows but the motor does not rotate. Then, at the end of the slide when the brake is released the motor will instanteneous speed up due to the bike still moving. Based on the settings and the acceleration of the motor either the controller will keep up or not. When it keeps up, it will go from drive_2 to drive_3 and all is well. When it does not keep up settings p to s in the recovery menu must trip and the controller must go to recovery again (drive_1). Now with the motor at speed the controller will recover and exit recovery mode to drive_3. If this doesn't happen but instead the motor stays locked up in drive_2, settings p to s must be corrected....

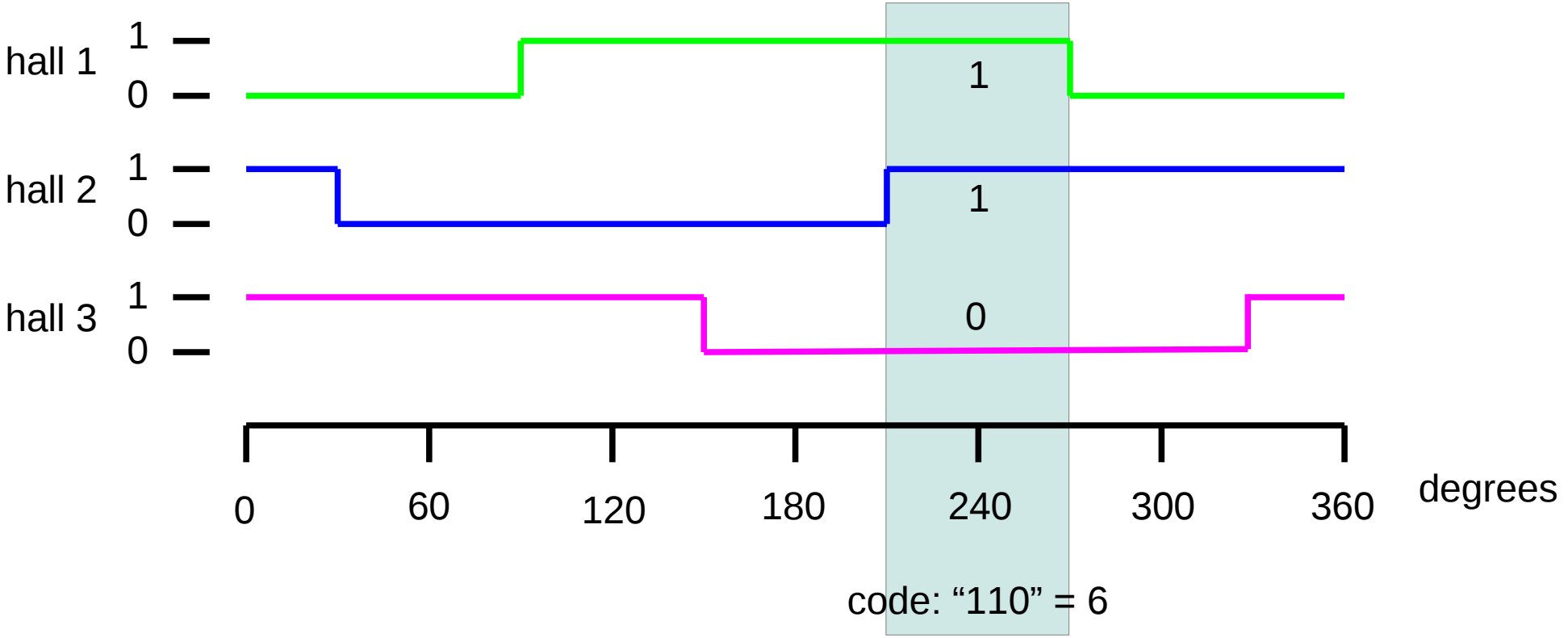
```
                                                                l) hall sensored only
 code: 0, angle: 111 deg, confidence: 0, used: no
 code: 1, angle: 59 deg, confidence: 7, used: yes
 code: 2, angle: 298 deg, confidence: 7, used: yes
 code: 3, angle: 357 deg, confidence: 7, used: yes
 code: 4, angle: 177 deg, confidence: 7, used: yes
 code: 5, angle: 118 deg, confidence: 7, used: yes
 code: 6, angle: 239 deg, confidence: 7, used: yes
 code: 7, angle: 358 deg, confidence: 0, used: no

a) toggle hall usage
b) calibrate hall positions: no

z) return to main menu

------>
```

The hall sensor menu gives information about the current hall calibration, allows you to use or disregard different 'codes' and to enable the automatic hall calibration.

The hall calibration works as follows. The chip must be set in hall mode (option 0 of the main menu), the calibrate hall positions in the above menu must be turned on and the 'online parameter save' in the store in ROM menu must be enabled. The chip must be setup completely and the parameters must be saved.

Turn off the controller, remove the setup signal and power on the controller. The controller will enter motor mode, and because the calibrate hall is selected it will not go into hall mode but into sensorless mode. Make sure the motor is unloaded and give a bit of throttle. The motor will start spinning as it would in sensorless mode. Increase the motor speed until the controller is in drive 3 and the motor runs at a few 1000 **erpm**. After a few seconds in drive 3, press the setup button to save all the setting. The hall settings will now be newly calibrated and saved for use. The calibrate hall (option b above) will automatically be turned off, so the controller is ready for running sensored straight away.

Even through it is not necessary, after calibration you can go into the hall menu and have a look at the calibration results, and turn on or off certain 'codes'



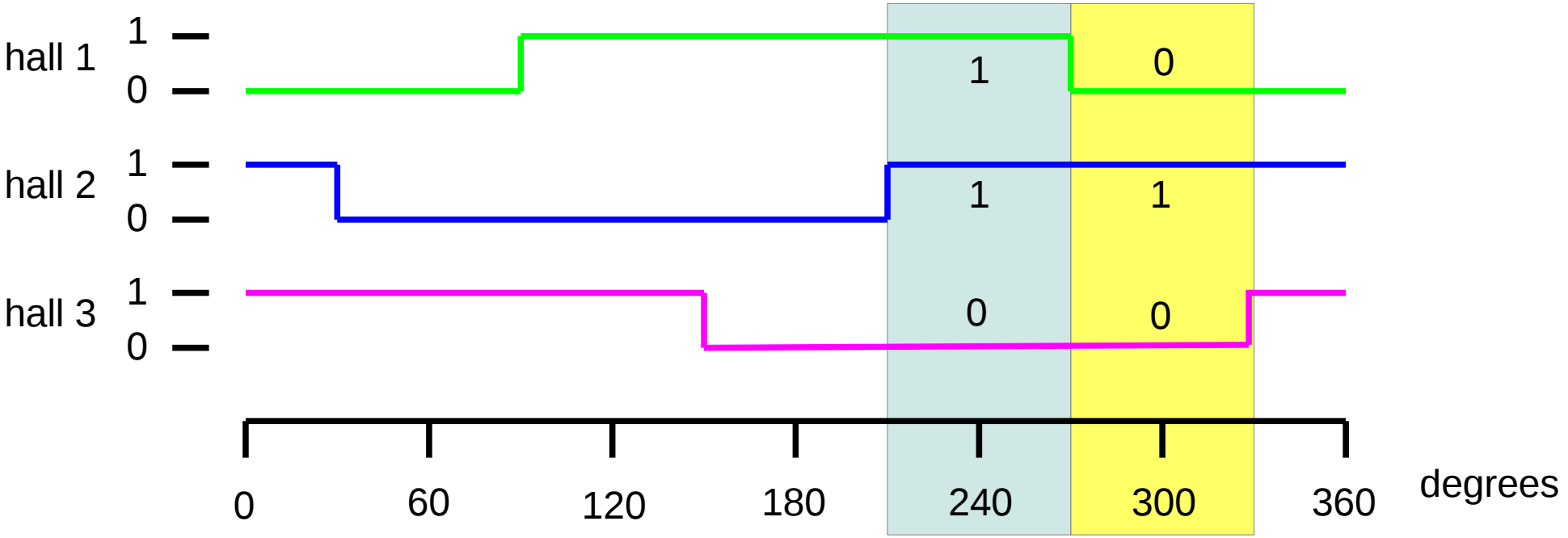
The figure above shows how the three hall signals toggle as the motor moves from 0 to 360 e-degrees.

The controller looks at the three hall signals as if they are part of a binary number. In the blue box for instance, hall 1 is digital '1', hall 2 is digital '1' and hall 3 is '0'. Together this makes the binary number “110”, which translates to the decimal number 6. This is called the 'code'.

When running the motor, the controller looks at the hall signals, transforms this into the code and then looks in an array to find the correct e-degrees at which to apply the signals to the motor.

```
code: 0, angle: 111 deg, confidence: 0, used: no
code: 1, angle: 59 deg, confidence: 7, used: yes
code: 2, angle: 298 deg, confidence: 7, used: yes
code: 3, angle: 357 deg, confidence: 7, used: yes
code: 4, angle: 177 deg, confidence: 7, used: yes
code: 5, angle: 118 deg, confidence: 7, used: yes
code: 6, angle: 239 deg, confidence: 7, used: yes
code: 7, angle: 358 deg, confidence: 0, used: no
```

```
l) hall sensored only
```

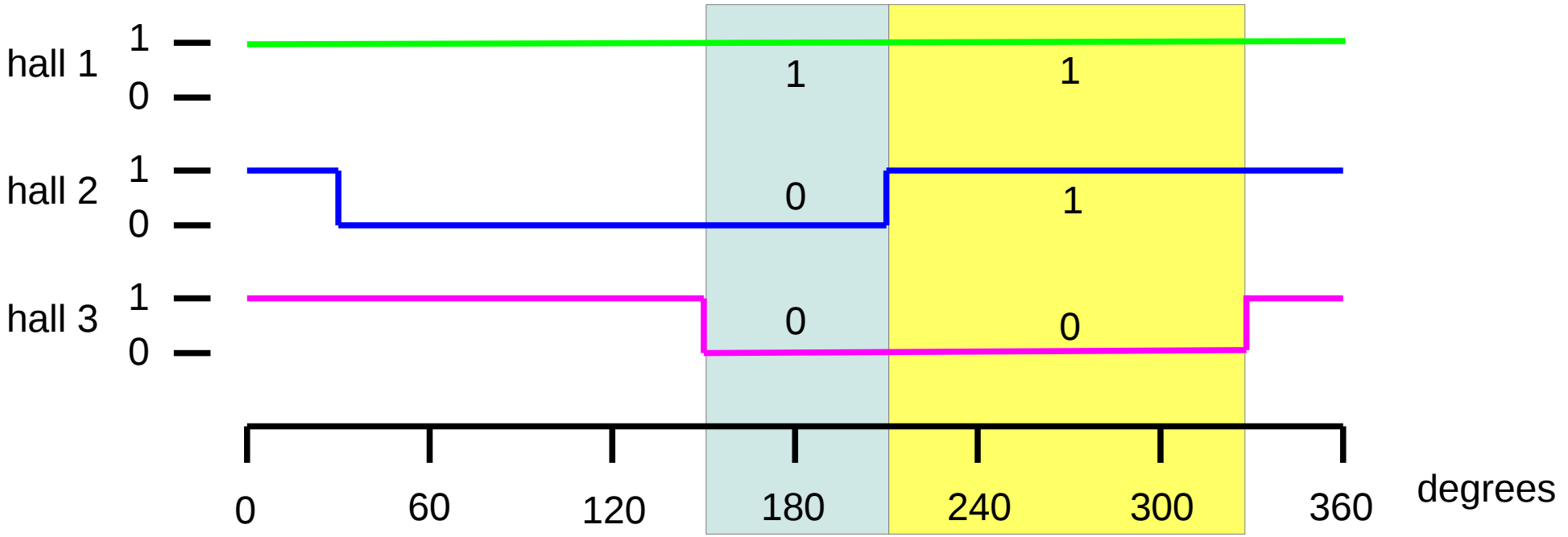


The table in the hall menu shows the calibration result. If you look at code 6, you can see the chip measured an average of 239 degrees, as indicated by the blue box in the graph. In a similar way, code 2 is indicated by the yellow box.

The confidence in the measurement is indicated by a number between 0 and 7. The confidence is computed based on how often a code occurred during the calibration run and how wide the range of degrees is for the code. Above for instance the confidence in codes 0 and 7 is 0, these codes were not seen by the chip during the calibration run. A low confidence (below 2.3) means the chip will not use the code when running the motor. You can override whether a code will be used of not with option a. If the chip encounter an unused code when running the motor, it will use the last valid code it saw.

```
code: 0, angle: 358 deg, confidence: 0, used: no
code: 1, angle: 358 deg, confidence: 0, used: no
code: 2, angle: 358 deg, confidence: 0, used: no
code: 3, angle: 358 deg, confidence: 0, used: no
code: 4, angle: 177 deg, confidence: 6, used: yes
code: 5, angle: 85 deg, confidence: 6, used: yes
code: 6, angle: 267 deg, confidence: 6, used: yes
code: 7, angle: 357 deg, confidence: 6, used: yes
```

l) hall sensored only

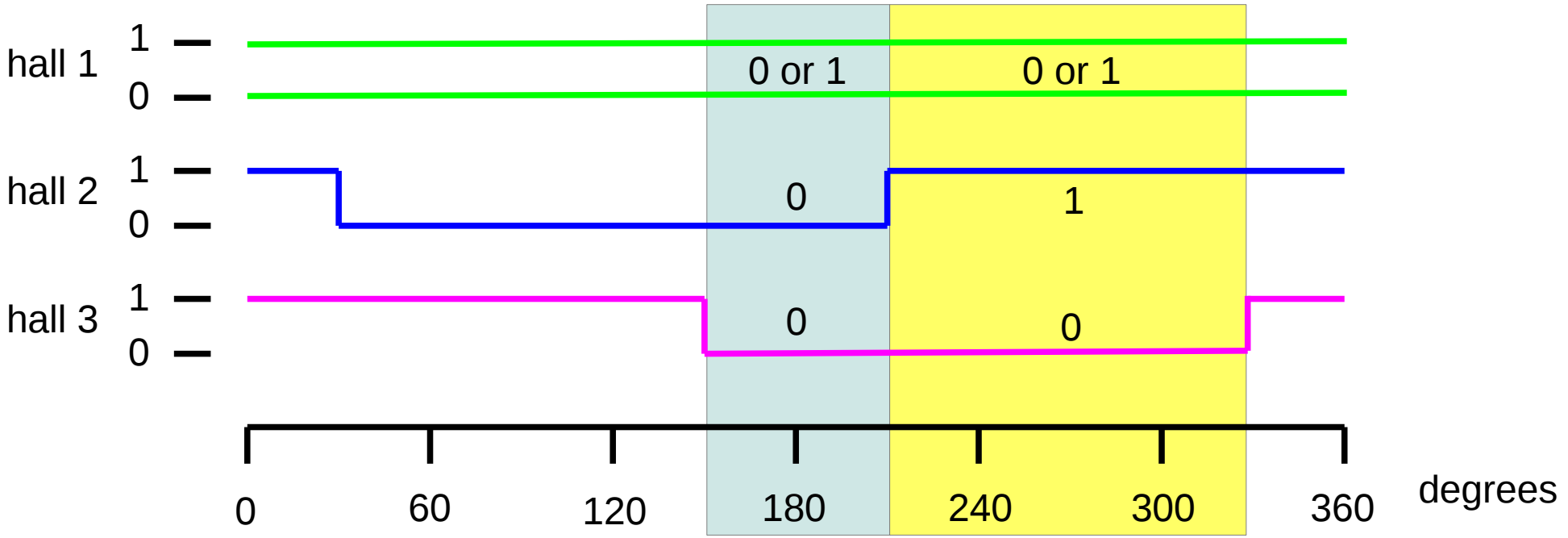


The system also works when a hall sensor is broken, above for instance hall 1 is for some reason always showing a high signal. The blue square shows code 4 which is indeed centered around 177 degrees. The yellow box shows code 6, which now is centered around 267 degrees. Confidence has dropped from 7 to 6, either because the codes don't occur so often (the narrow blue box) or because the degree range is wide (wide yellow box). Codes 0 to 3 don't occur, confidence is 0 and they are not used.

The controller will still be able to run the motor. Especially in the wide (yellow) ranges the motor control is not so accurate as the specific code covers a range of 120 e-degrees. But since most of the time the controller runs completely sensorless and the halls are only used for motor start, performance is not really affected.

```
code: 0, angle: 177 deg, confidence: 6, used: yes
code: 1, angle: 85 deg, confidence: 6, used: yes
code: 2, angle: 265 deg, confidence: 6, used: yes
code: 3, angle: 355 deg, confidence: 6, used: yes
code: 4, angle: 175 deg, confidence: 6, used: yes
code: 5, angle: 85 deg, confidence: 6, used: yes
code: 6, angle: 267 deg, confidence: 6, used: yes
code: 7, angle: 357 deg, confidence: 6, used: yes
```

```
l) hall sensored only
```



Above shows the case when hall 1 is broken and outputs random 1's and 0's. Now both codes 0 and 4 correspond to the same blue box, codes 2 and 6 map to the yellow box.

Since all codes occur at random and therefore both quite often, confidence is high at 6. All codes are used.

```
                                                          m) temperature sensors

    a) use temp sensors: yes
    b) identify temp sensors
    c) temperature readings
    z) return to main menu

     0: 5E00 0802 BBD6 1010
     1: 8F00 0802 BBF8 6E10
     2: 5400 0802 A8C4 E110

    0) reduce max phase current above 50.0 degC by 0.9 A/degC
    1) reduce max phase current above 60.0 degC by 1.9 A/degC
    2) reduce max phase current above 74.0 degC by 3.4 A/degC

    ------>
```

Option a turns the use of temperature sensors on or off.

From the factory each produced temperature sensor gets its own unique 64 bit code. With this code the sensors can be addressed independently, even though they are all connected in parallel. Option b searches for all connected temperature sensors and retreives their 64 bit codes (upto 8 sensors max). These are then displayed after option z.

For each temperature sensor you can set above which temperature the motor phase current must be reduced and by how many Amperes per degree Celsius.

Option c will show multiple columns, one for each sensor diplaying the temperature, with the final column showing the temperature allowed max phase current (based on the user entered data under 0), 1) etc)

```
                                                          n) miscellaneous
    a) autocomplete

    b) motor standstill voltage threshold: 0.48 V
    c) low side pulsing in drive 0: enabled
    d) low side pulsing rate: 20 Hz
    e) low side pulsing width: 20 usec
    f) wiggle range: 19 deg
    g) wiggle rate: 9 Hz
    h) minimum # of cycles going from drive 2 to 3: 1000
    i) number of cycles going from drive 3 to 2, HF only: 200
    j) too fast acceleration detection: enabled

    z) return to main menu
```

Options b-e are discussed on the following pages.

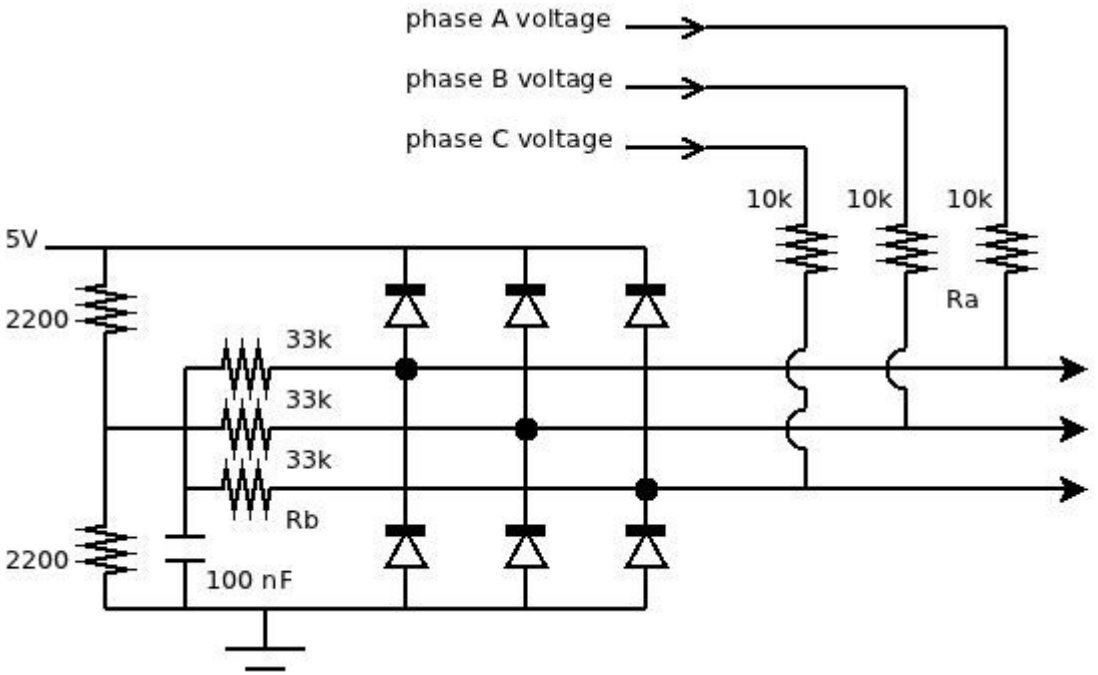
When toneless start is selected, a wiggle on the phase (drive_2 only) is used to intise a response from the motor that the sensorless algorithm can work with. Option f sets the amount of e-phase the the motor should be wiggled over, option g sets the rate. The wiggle is especially usefull for losening up RC motors which have a relatively large amount of cogging. The wiggle can be disabled by making option f 0.

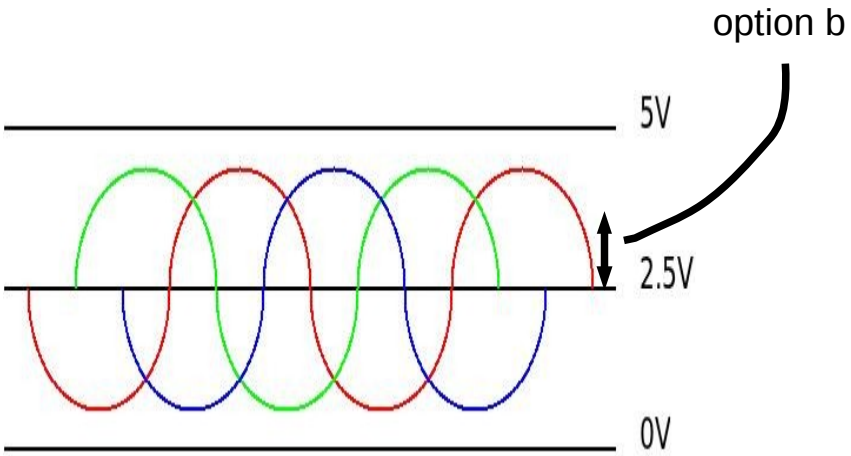
The actual backemf based sensorless will see the wiggle and try to follow it. When the chip transides to drive_3, the wiggle can be disruptive and cause an error current event. Therefore there is an inbetween mode (drive_2to3) during which error current detection is disabled. Option h sets the minimum amount of cycles to be spend in this transitional mode. After this amount of cycles, the chip checks for low error current before transiding to drive_3.

Option i is only active in HF mode. The HF tone is off in drive_3, and must be re-started before entering drive_2. This must happen very fast, as the motor is slowing down meaning backemf sensorless information is rapidly disappearing. Option i specifies how much time is spend to restart the HF tone. Note that the HF tone must pass through through some filters (option d of the filters menu) and that this takes time, option i cannot simply be made 0).

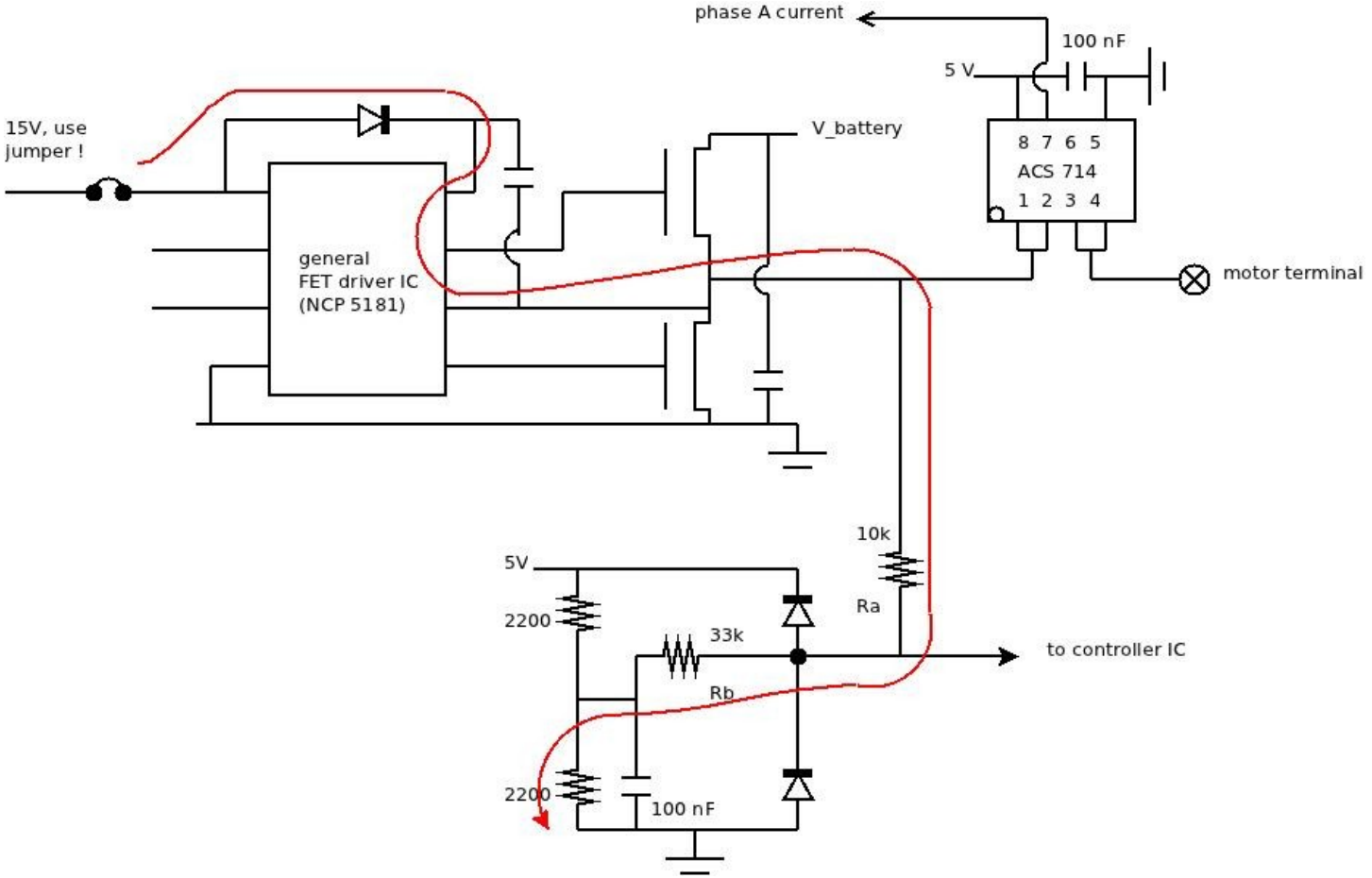
When option j is set the controller goes to drive_1 when it detects an impossibly fast accelleration (an error which can happen during sensorless start in drive_2)

Since the controller powers all motor terminals all of the time a mechanism has to be in place to tell the controller when it's safe to take control of the motor. This is determined in drive_0, the only mode in which the motor terminals are 'released'. To transide out of this mode the throttle has to be closed (or on regen) and the motor speed has to be low. The motor voltage amplitudes as seen by the controller have to be below the voltage of option b.

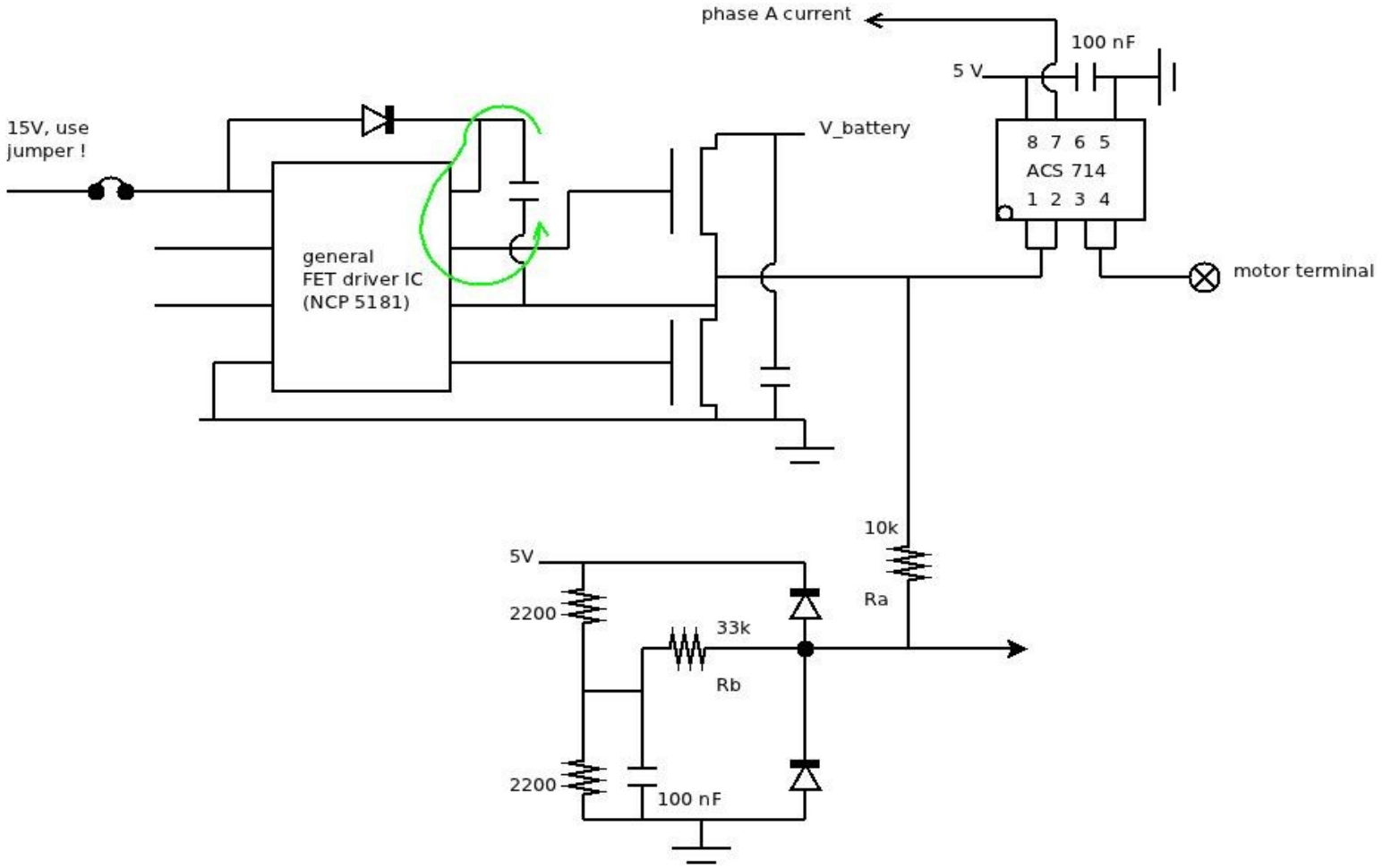




Dependent on the type of output driver a bias current can pass through resistors Rb in which case the detected voltages will never fall below 2.5V+option b.



The figure shows a typical output stage driver built using a NCP 5181. The high side driver is supplied from a capacitor which is charged using a diode. When the capacitor is not fully charged the diode will conduct, causing a current to flow according to the red path. The current passes through Rb, raising the voltage to the controller IC. When the voltage stays above 2.5V+option b the IC will stay in drive_0.
To prevent this from happening the high side driver's supply capacitor must be fully charged. The diode will then no longer conduct and no parasitic current will flow though Rb.



The figure shows the high side driver's supply current path when its supply capactor is properly charged.

To charge the supply capacitor the controller offers the option to pulse-wise turn on the low side FET. When the low side FET is on the supply capacitor will be charged, enabling the correct detection of 2.5V+option b.

The pulsing of the low side FET during drive 0 is turned on or off with option c. Options d and e set the pulse frequency and duration. The options should be set low enough not to drastically brake the motor and high enough to keep the high side driver supply capacitor charged to a sufficient level.

```
                                                    z) store parameters in ROM for motor use

    a) save data to ROM for motor use
    b) print data in HEX format
    c) enter data in HEX format
    d) online parameter save: disabled

    z) return to main menu

    ------>
```

This menu allows for the saving of data to the chips internal ROM memory, so that all the settings can be used for running the motor. For this option a must be used.

Option b print out all the variables in HEX format, which can then be saved in a text file on the computer.

```
------> b

 save the following HEX lines in a text file, including the '*' termination character

0x0085    0x0085    0x00D6    0x0015    0x0005    0x0002    0x7FBC    0x0623
0x0000    0x0F5B    0x0000    0x07AD    0x03D6    0x0E92    0x0400    0x0400
0x0400    0xAAAA    0xAAAA    0xAAAA    0x02CA    0x0248    0x000C    0x02DB
0x000B    0xF44C    0x0403    0x1000    0x0000    0x0000    0xFFFF    0xFFFF
0xFFFF    0x00D1    0x01D7    0xFFFF    0xFFFF    0x1EB6    0x0189    0x03D6
0x0000    0x0800    0x0258    0x0064    0xFFFF    0xFFFF    0xFFFF    0xFFFF
0x0000    0x4CCD    0x000C    0x0000    0x00F0    0xFFFF    0xB333    0xFFF4
0x0000    0xFF10    0x0000    0x07AE    0x0018    0x0000    0x01E0    0xFFFF
0xF852    0xFFE8    0x0000    0xFE20    0x0003    0x0000    0x0078    0x0000
0x0000    0xFFFD    0x0000    0xFF88    0x0000    0x0000    0x003C    0x0003
0x0000    0x0000    0x0000    0xFFC4    0xFFFD    0x0000    0x0000    0x0000
0x00F0    0x000C    0x0000    0x0000    0x0000    0xFF10    0xFFF4    0x0000
0x0000    0x0000    0x038A    0x6400    0x05DC    0x05DC    0x0F5B    0x0035
0x00E4    0x02C6    0x013F    0x0007    0x017A    0x0042    0x0010    0x0E10
0x0000    0x03E8    0x00C8    0x5027    0x03B6    0x6000    0x0623    0x013F
0x4700    0x2A3F    0xD441    0xFD3F    0x7D3F    0x5441    0xAA3F    0xFF00
0xFFFF    0xC519    0x764B    0x5482    0x41B3    0x35C3    0x2D7A    0x276B
0x22C9    0x1F1E    0x1C28    0x19B5    0x17A6    0x15E6    0x1463    0x1312
0x11EB    0x10E4    0x0FFB    0x0F28    0x0E6B    0x0DC0    0x0D23    0x0C94
0x0C10    0x0B97    0x0B27    0x0ABF    0x0A5F    0x0A05    0x09B1    0x0962
*
```

The data from the text file can later be read into the controller IC by using option c. Most terminal programs have the option to send a raw file over RS232, this can be used to send a previously saved HEX file to the chip.

After using option c, press enter to restore the menu and then use option a to save to ROM.

Finally, when option d is turned on, gain and offsetcalibration data gathered while running the motor can be stored to ROM. While in motor mode, activate setup to store the data. All drive mode indicating LEDs will light up and the chip will return to drive_0.