Dr.Edgar Huckert
Mail: edgar.huckert@huckert.com
V2.1 09-2019

# Controlling brushless motors: two solutions

## 1. Controlling a brushless motor via an ESC with Arduino

I am rather a beginner in the area of modelling. I had however some parts to start with. Controlling a normal DC motor is rather simple: you need an analog output (or PWM with simple additional components) and a current switch like a MosFet or a normal transistor or a special chip.

Things are a little bit more complicated for brushless motors. The have a minimum of three connection pins. The input current scheme is rather comparable to a 3-phase current connection. The ESCs (ESC = electronic speed control) on the market do the job for you while being controlled **via simple PWM** as in classical modelling.

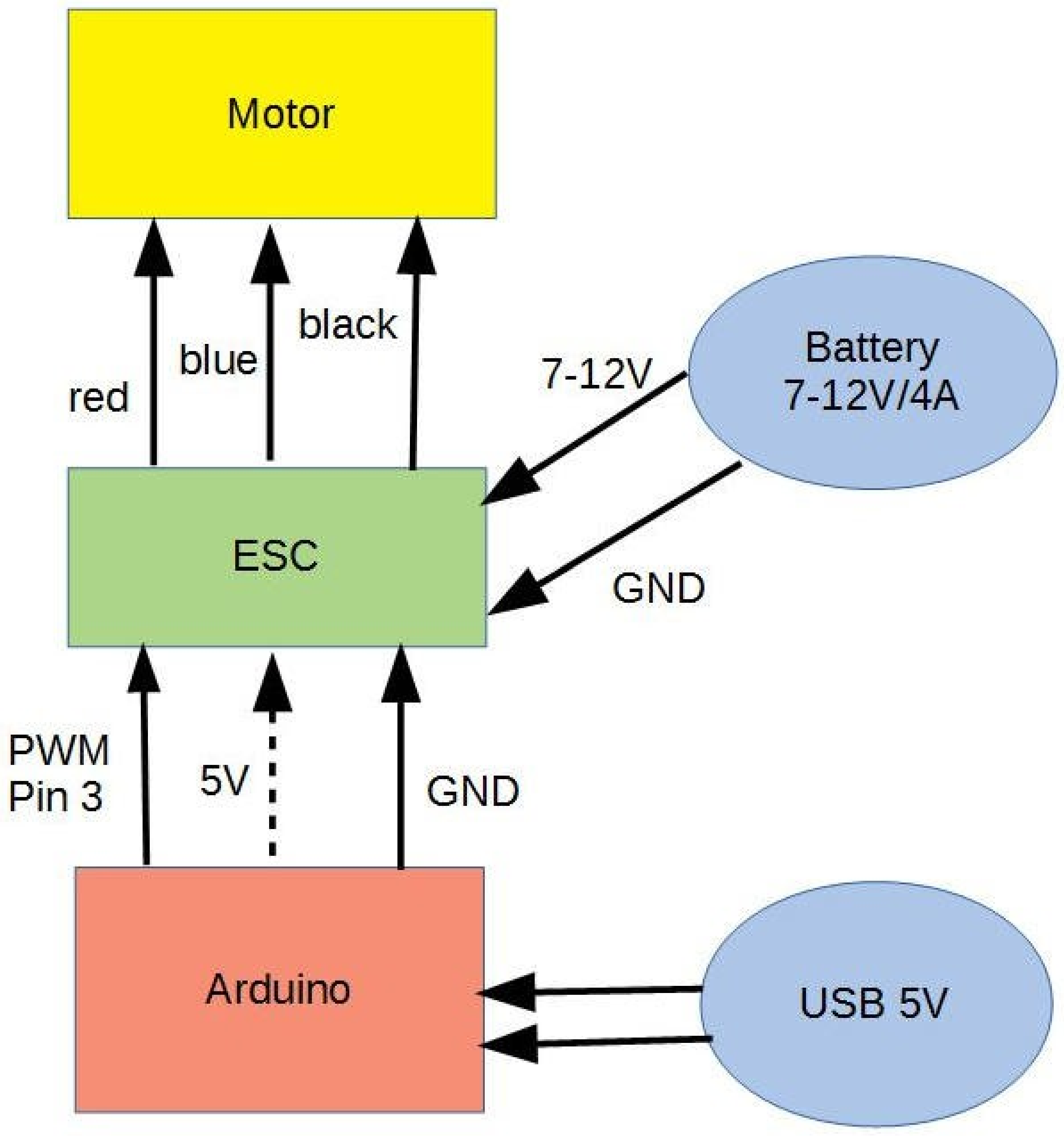
I just wanted to get the motor running and to test **speed variation**.  The Arduino replaces in this configuration the remote control device. It is however not a suitable replacement for the usual remot control devices that also communicate with an ESC via PWM signals.

My **configuration**:

- **brushless motor**: Simprop Magic Torque 10-11
- **ESC** (electronic speed control): Robbe ROXXY BL-Control 810
- **Arduino**: PWM on Pin3, all classic Arduinos should work

While there a many proposals on the Internet how to connect a brushless motor directly to an Arduino using MosFets  I found practically not description how an ESC works. Even the input wires were not marked in my data sheet for the ESC.This ESC accepts a PWM signal on the input and works as a motor driver on the output - no additional MosFets or normal transitors are needed.

**Diagram** for the configuration:



Note: the 5V connection from Arduino to ESC is not needed (dotted line). If you use other PWM pins on the Arduino be sure that they are speed-compatible with you ESC (490 or 980 Khz). The colours of the cables may be different if other components are used.

My motor started to work from a PWM value of ca. 140 onward. With a vPWM value of 200 it consumed nearly 4A under 10V (40 Watt). When I skipped the initialisation routine then the results were unpredictible. I cannot explain this as I found no clear documentation for this ESC.

If you need multiple motors or if you have to control more parameters than speed you may need multiple ESCs. A single Arduino has normally more than one PWM output pin - so the Arduino should not be the problem.

**Conclusions**:

- the initialisation (for the ESC) called in *setup()* seems to be necessary. If other ESCs are used the initilisation may be different.
- the motor consumes up to 4A with 10V. Take care: this is a rather small motor - the Ampère value is bigger if a bigger motor is used! Your ESC must support this.

PWM is a wide spread feature on nearly all current microprocessors (Rasp-berry, BeagleBone etc). The concept should thus work with nearly all of them if the sketch (the software) is adapted.

Here is the **Arduino sketch**:

```
// PWM output on Pin 3
// EH 07-2019
// The initialisation seems to be necessary
// Comsumes up to 4A with 10V
//
const int PwmPin = 3; // 3,5,6,10,11
int       outputValue = 255;
int       loopCount   = 0;
const int MAXIDX      = 12;
int       idx         = 0;
//
// table of values for initialisation
int       table1[MAXIDX] =
{ 0, 20, 40, 60, 80, 100, 120, 120, 120, 120, 120, 0 };
// table of values for test
int       table2[MAXIDX] =
{ 125, 145, 145, 150, 155, 160, 170, 180,
  190, 195, 140, 130 };

// ---------------------------------------------------
// ESC initialisation: uses table1
void initMotor()
{
  Serial.println("Initializing...");
  for (int n=0; n < MAXIDX; n++)
  {
    outputValue = table1[n];
    analogWrite(PwmPin, outputValue);
    delay(300);
  }
  Serial.println("End Initialisation");
}   // end initMotor()

// --------------------------------------------------
void setup()
{
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);
  idx         = 0;
  // initialise motor/ESC
  initMotor();
}   // end setup()
```

```
// --------------------------------------------------
// production" loop: uses table table2
void loop()
{
  outputValue = table2[idx++];
  if (idx >= MAXIDX) idx = 0;
  //
  Serial.print("value=");
  Serial.println(outputValue);
  Serial.print("count,value=");
  Serial.print(loopCount);
  Serial.print("  ");
  Serial.println(outputValue);
  analogWrite(PwmPin, outputValue);
  delay(800);
  loopCount++;
}   // end loop()
```

## 2. Controlling a brushless motor with LPC1768 and HC-12

This time I used an **NXP LPC1768** microcontroller to generate the PWM signal for the ESC controlling the brushless motor. After having played around with different Arduino versions (Duemillanove, Nano, DUE) and after having experienced failures due to different PWM basic speeds I found that using an LPC1768 was the simplest solution. The MBED library offers an uncomplicated set of functions for PWM. It also offers the concept of **callback routines** for simple reception of serial bytes (this is a simplified version of the "interrupt service routines" known found in real time systems). BTW: I use here a global variable *rxVal* set in the callback routine and read in the *main()* routine. In more complicated programs a lock mechanism (mutex etc.) may be necessary to avoid simultaneous access to the shared variable.

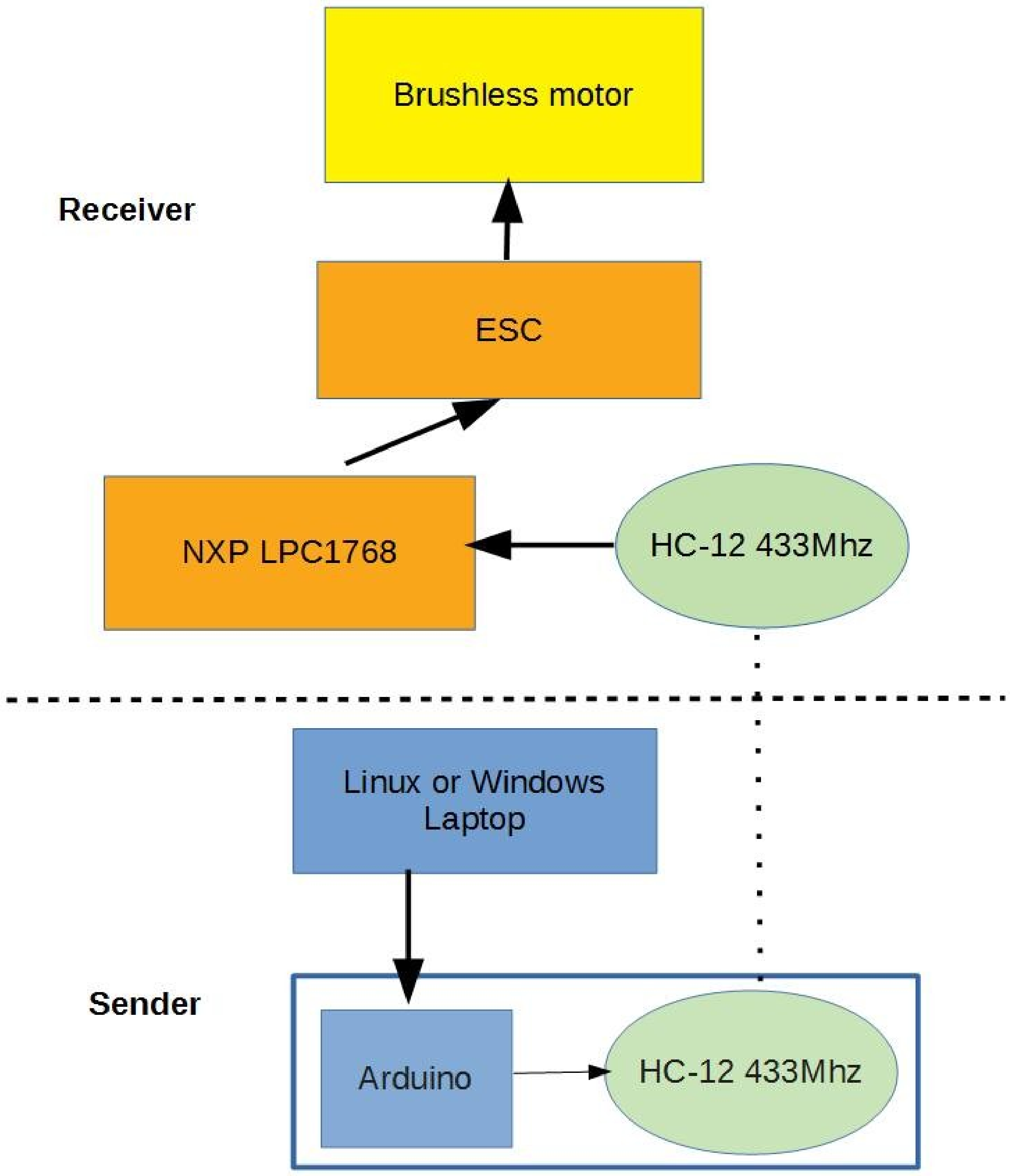
The receiver is mainly the **LPC1768** microcontroller and a **HC-12** 433 Mhz module acting as receiver. No arduino is needed here: the LPC1768 does the complete job including the production of the PWM signal.

As **sender** I use here an **Arduino nano** connected to a **second HC-12** module. This module receives simple control bytes in the range '0' - '9'. These control bytes are translated to PWM duty cycle values (range 0.0 to 1.0) via a simple table. I enter the control bytes via **TeraTerm** (under Windows) or **minicom** (under Linux). It is clear that this can be replaced by a potentiometer (variable resistance) connected to an Arduino that transmits these (translated) values to a HC-12 module. The Arduino is used here as a simple *USB to 5V-serial converter.*

The parts and components I used are the following:

- receiver side: NXP **LPC1768**
- receiver side: **HC-12** module for 433Mhz reception
- receiver side: **ESC** Robbe ROXXY BL-Control 810
- sender side: **HC-12** module for 433Mhz transmission
- sender side: TeraTerm (windows) or minicom (Linux)

Here is a **diagram** showing this configuration:



Suggestion for the use of **minicom** under Linux:

- start minicom: `sudo minicom -device=/dev/ttyUSB0`
- in minicom: set the appropriate baudrate (normally 9600 Bd)

Note that the device */dev/ttyUSB0* connecting the Arduino with your computer may be named differently. Take a look at the devices by typing `ls -l /dev/tty*`.

In **TeraTerm** under Windows things are also simple:

- under menu *Setup/Serial port* choose the appropriate serial port (for me: *com3*, normally higher than *com2*)
- under the same menu choose the baudrate (normally 9600Bd)

Note that under Windows also the port can be named differently.

Typing numbers under *TeraTerm* or *minicom* from 1-5 should result in a reaction of the motor. In my setup the motor started from 2 on. 1 should stop it.

Here is the **C code** for the NXP LPC1768. The code must be compiled and loaded with the MBED compiler:

```
// module PWM433Mhz
// Dr.Edgar Huckert 08-2019
// V2.1 - is OK
//
// Uses a HC-2 433Mhz module to receive bytes (9600 Baud)
// Uses PWM(500 Hz) to control an ESC for a brushless motor
// The ESC is: Robbe
// The brushless motor is: Simprop Magic Torque
//
// on the other side (remote control sender):
// HC-12 module used in TX direction (RX pin connected to Arduino TX)
// Arduino Nano used as USB-Serial (5V) converter
// On PC/Laptop (Windows): TeraTerm with pseudo serial(USB) port
//
#include "mbed.h"
//
DigitalOut myled(LED1);
PwmOut PWM1(p23);
Serial ser(p28, p27); // RX from HC-12 on pin 27
float rxVal = 0.5;
float rxValTable[10] =
{
    0.0, 0.50, 0.55, 0.60, 0.65,
    0.70, 0.75, 0.80, 0.85, 0.90
};

// ------------------------------------------------
// the callback routine: receives bytes from the HC-12 module
// Received bytes in range '0' - '9'
// Sets the global variable rxVal
void gotBytes()
{
    char ch = ser.getc();
    // normalize the received char (byte) to the range '0 - '9'
    ch = ch & 0x7f;
    printf("ori %c", ch);
    if (ch <= ' ') ch = '0';
    if (ch < '0')  ch = '0';
    if (ch > '9')  ch = '9';
    rxVal = rxValTable[ch - '0'];
    printf(" %c", ch);
    printf(" rxVal=%f\r\n", rxVal);
    return;
}   // end gotBytes()

// ---------------------------------------
void initMotor()
{
    // configure the serial interface connected to a HC-12 module (433MHz)
    printf("configure ser.interf. for HC-12\r\n");
    // register the call back routine
    ser.attach(& gotBytes);
    ser.baud(9600);
    ser.format(8, SerialBase::None, 1);
    //
    // start PWM on pin 23
    printf("initMotor()\r\n");
    PWM1.period(0.002); // set PWM period to 2 ms=500 Hz
    PWM1 = 0.5;         // duty cycle 50%
    for (int n=0; n < 5; n++)
    {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
    printf("initMotor() OK\r\n");
}   // end initMotor()

// ---------------------------------------
// S1ets a new PWM duty value
// parameter val: the new duty cycle
void driveMotor(float val)
{
    printf("duty cycle=%f\r\n", val);
    // set the duty cycle value
    PWM1 = val;         // set duty cycle
}   // end driveMotor()

// ---------------------------------------
int main()
{
    float oldRxVal = rxVal;
    unsigned count = 0;
    //
    printf("PWM433Mhz starting...\r\n");
    //
    initMotor();
    //
    // Note: the control bytes varying the PWM ducty cycle
    //       are received in callback routine gotBytes()
    while (1)
    {
      if (rxVal != oldRxVal)
        driveMotor(rxVal);
      oldRxVal = rxVal;
      wait(0.333);
      count++;
      if ((count % 100) == 0)
        printf("alive\r\n");
    }
}   // end main()
```

The **Arduino sketch code** for the simple 433Mhz sender is here. Note that the two standard serial pins (RX, TX) on the Arduino are used for different things (see the comments):

```
const int ledPin = 13;
unsigned  ledVal = 0;
unsigned  count  = 0;
char      ch     = 0;

// -----------------------------------------
void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, ledVal);
}   // end setup()
```

```
// --------------------------------------------
void loop()
{
  if (Serial.available() > 0)
  {
    // read from the Arduino RX pin connected to
    // PC/USB (TX pin)
    ch = Serial.read();
    // Send  the received char via TX pin
    // This TX pin on the Arduino is connected
    // to RX on the HC-12
    Serial.print(ch);
    count++;
    // blink the LED
    ledVal =  ! ledVal;
    digitalWrite(ledPin, ledVal);
  }
  else
  {
    delay(1);  // wait 1 ms
  }
}   // end loop()
```