Ignition Pulse Generator

But

Génerer les impulsions de commande d'allumage moteur
L'environnement est particulièrement pourri !
l'éclateur (réglable, sous 1bar, remplace la bougie) est à 30 cm du montage
le montage est dans une boîte métal avec traversées capacitives (by-pass)
le transistor de sortie est à l'extérieur de la boîte

Environnement

la commande d'allumage est le RMZ décrit ailleurs sur le site (http://www.hackerschicken.eu/www/electric/commande_allumage.pdf), c'est un montage particulièrement increvable. Les transistors sont des bipolaires soigneusement choisis, les transistors « spéciaux allumage » MOS ou IGBT me semblent peu fiables et semblent ne pas être réellement utilisés par les constructeurs auto puisque les bobines sont commandées en courant (et nécessitent maintenant une diode interne), les MOS sont particulièrement inadaptés au fonctionnement en régime linéaire. L'allumeur RMZ commande les bobines en tension et possède aussi une limitation en courant (à 3,9A ici), il peut commander n'importe quel type de bobine et se montre particulièrement costaud et insensible à ces environnements. Il est ici placé à 10cm de la sortie bobine et à 15cm de l'éclateur. L'éclateur est à l'air libre, c'est un cas particulièrement difficile, la bougie qu'il représente se trouve dans la culasse et donc dans un blindage électromagnétique important. Les premiers essais, avant la boîte finale, ont détruit 3 cartes arduino. Mon PC se plante toujours lors des essais, même placé à 1 m !
C'est clairement la composante parasites conduits qui pollue et tue mes arduinos, puisque le montage fonctionne maintenant boîte ouverte. La composante parasites rayonnés est nettement moins gênante.

La carte arduino

C'est une carte avec connecteur mini-USB (ça m'intéresse pas) mais avec un connecteur ICSP pour la programmation, directement connectable et compatible de ma carte de programmation usbasp (avec adaptateur 6 pins). La carte de programmation permet aussi d'alimenter la carte pendant la programmation et les essais de mise au point.
Toutes les liaisons passent par des inductances, des perles ferrite et des condensateurs de traversée (bypass).
Elle est alimentée en fonctionnement normal sur batterie 12V 3Ah par son régulateur interne (entrée RAW) avec découplages 470µF/25V et céramique 0,22/630V. La même batterie sert à alimenter la bobine et son circuit d'allumage.
Le transistor bipolaire de sortie est dans un milieu pollué de parasites, seule sa base est reliée à l'arduino.
Fidèle à mes habitudes je n'utilise pas le « language » arduino mais du c universel.

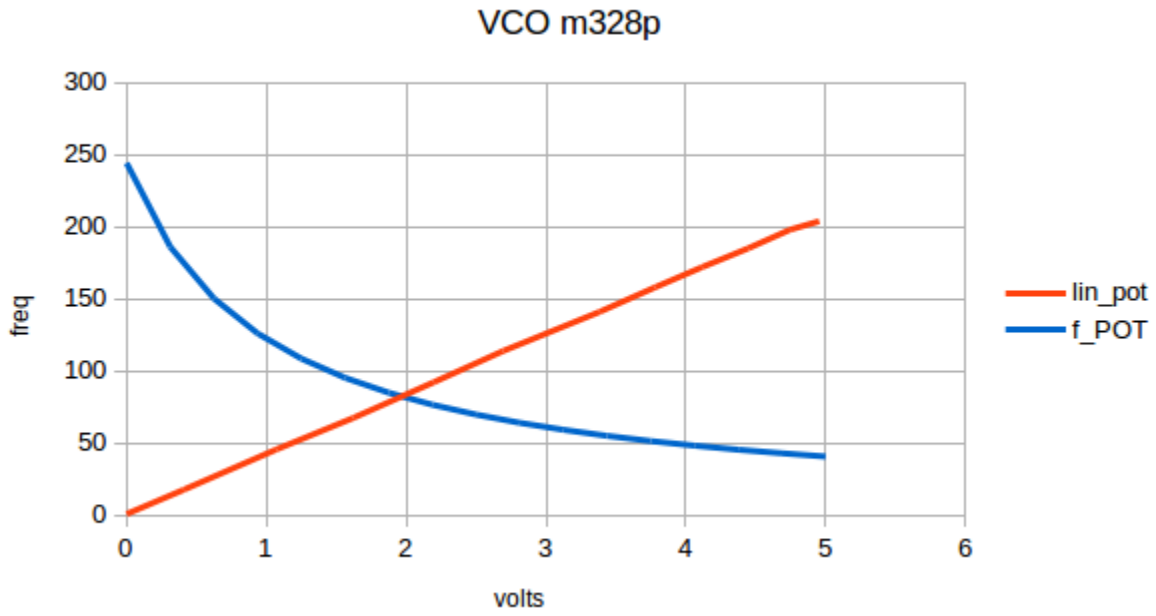
Le programme

Le µC utilise son timer 16 bits en mode fast PWM (mode 14 = RAZ compteur à ICR1) tandis que ses sorties waveform basculent au moment prévu et donnent un temps de conduction de l'allumage de 4ms, ce qui énergise à donf toute bobine standard et permet d'atteindre les 200Hz (6000rpm en caisse, 12000rpm en Soufflex). Pas de dwell, qui n'est que la pâle traduction mécanique du besoin de 4ms d'énergisation de la bobine. Les registres OCR1A et OCR1B sont dédiés à ce temps.
L'utilisation d'une carte arduino me permet d'ajouter facilement des fonctions comme un affichage LED de la fréquence.
Le montage est donc un VCO (voltage controlled oscillator) où la fréquence est fonction de la tension d'entrée (le potentiomètre).

La linéarité de commande

Les montages équivalents à monostable (555 ou HC123) ont une commande linéaire. Leur commande se fait par un potentiomètre dont la valeur fait varier la constante de temps k x RC. Avec un potentiomètre linéaire le déplacement du bouton permet un affichage

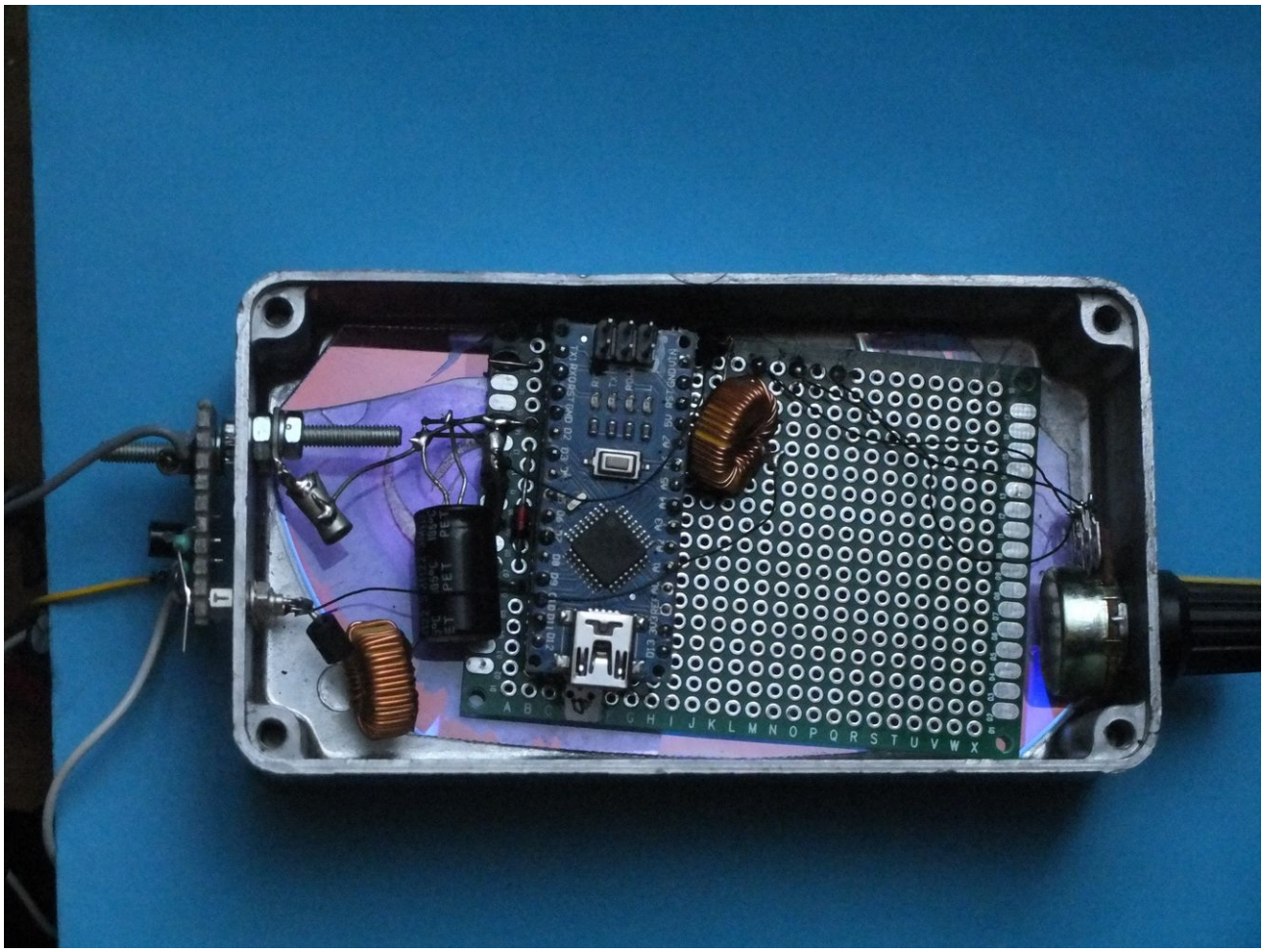
« linéaire », en fait en arcs de cercles réguliers, l'arc entre 50 et 100Hz est le même qu'entre 100 et 150Hz. Avec l'arduino on commande en période (1/f), désagréable, on a l'impression que rien ne bouge pendant un demi-tour et que tout se passe dans le dernier 1/4 de tour !



Une petite astuce d'inversion soft et la commande devient linéaire, et bien plus agréable. Elle agit ici à l'envers, fréquence max pour 5V, alors que le potentiomètre d'origine donne la fréquence max pour 0V. Pas vraiment un problème, on inverse les fils masse et alim 5V sur le potentiomètre, et le tour est joué.

Video
http://www.hackerschicken.eu/www/electric/DSCF5814.MOV

Ignition Pulse Generator



Le programme :

```
/* at mega328p
 * arduino nano pro
 * potentiometer controlled ignition tester
 *
 * PC0 arduino A0     potentiometer wiper
 * potentiometer GND on hi-speed side
 * PB5 arduino 13     LED & output
 * output needs an external transistor (out of the metal box used to protect µC;
 * (I blew up several µC during development, I suggest strongly a good shielding)
 *
 * dwell fixed by program (can be changed with OCR1A & OCR1B values)
 *
 * status:
 * keywords: main loop in interrupts, timer 16b, ADC 10 bits
 * fuses l=e2   h=d9   e=07
 *
 * RMZ#239
 * 15/05/13 04/08/17
 * (CC) zibuth27 by sa nc 2017
 */

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/sfr_defs.h>
```

Ignition Pulse Generator

```
int main()
{
        // variables

        // pins
        DDRB = (1<<PB5)|(1<<PB0)|(1<<PB1)|(1<<PB2);         // LED output enable
        DDRD = (1<<PD4)|(1<<PD6)|(1<<PD1);
        PORTC|=(1<<PC1)|(1<<PC2)|(1<<PC3);                  // inputs pull-up

        // set timer1
        TCCR1A|=(1<<COM1A0)|(1<<COM1A1)|(1<<WGM11);         //  mode 14, fast PWM
        TCCR1B = 0x1d;                                      // CTC mode, prescaler
        TIMSK1|=(1<<OCIE1B)|(1<<OCIE1A)|(1<<TOIE1);         // ISR mask
        OCR1A = 30;
                                                            // dwell = OCR1A-OCR1B = 4ms
        OCR1B = 8;
        ICR1  = 1000;

        // set ADC
        ADMUX=0x40 ;                                        // mux0 PC0
        ADCSRA= 0x7;                                        // ADEN, ADPS = prescaler 128 = 125kHz
        ADCSRA |= (1<<ADEN);                                // enable ADC

        // enable ISRs
        sei();

        // initialization of values


  while (1)                                                 // endless loop
  {

//              _delay_ms(1);
  }     // end of while
}       // end of main

ISR (TIMER1_OVF_vect)
{
               ADCSRA |= (1<<ADSC);
               while (ADCSRA & (1<<ADSC));                  // wait for conversion complete
               ADMUX |= (1<<ADLAR);
               uint16_t pot = ADCH;                         // read potentiometer value

               pot = 9500/(1*(1+pot));                      // inversion for linearization V/f
               if (pot >= 2000) pot = 2000;                 // practical limits
               if(pot <= 34) pot = 34;                      // practical limits
               ICR1 = pot;                                  // timer reloaded

}



ISR(TIMER1_COMPB_vect)                      // ignition start : energizing coil
{
               PORTB |= (1<<PB5);

}

ISR(TIMER1_COMPA_vect)                      // cycle rate, ignition coil trigger
{
               PORTB &= ~(1<<PB5);
}
```