Tachymètre Solex

v2.1

1 Objet :

Pour mesurer le régime d'un Solex ou d'un moteur, on peut :

- capter le courant bougie (moteurs à essence uniquement), le résultat dépend du cycle moteur : une étincelle par tour pour un deux-temps, ou une étincelle tous les deux tours pour un quatre-temps (ce n'est pas toujours vrai, certains moteurs ont une étincelle perdue en fin d'échappement)
- capter le courant de la génératrice (alternateur, volant magnétique, maléfique ou mirifique selon qu'on est à l'apéro ou au digestif) L'alternateur, peut aussi servir avec réserves, car certains anciens diesels, sans bougie d'allumage ni calculateur d'injection ne disposaient que d'un alternateur mû par une courroie, et donc avec une erreur permanente de glissement, et de surcroît variable, dépendant de la charge électrique.

Ce signal est interprété par une petite électronique, puis affiché en digital ou analogique.

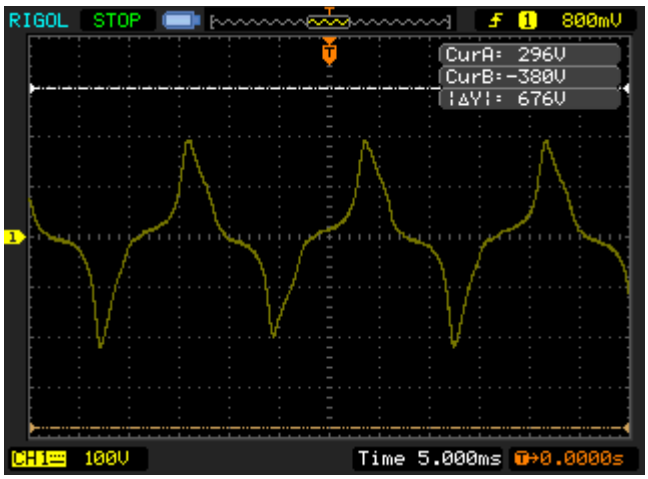
2 Capture

2.1 Bougie

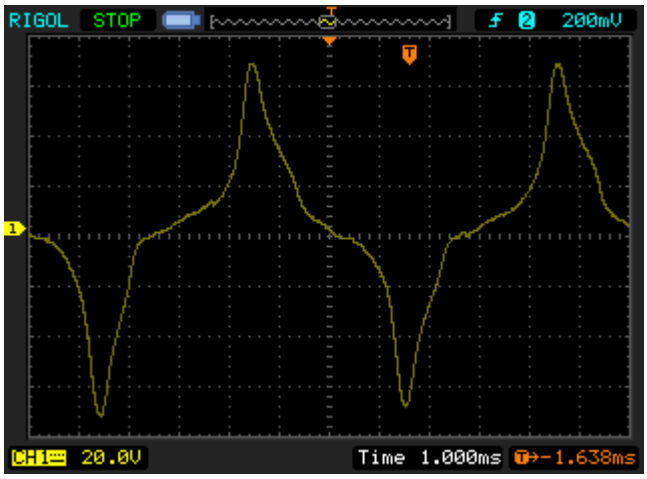
La capture du signal par un fil de bougie (un fil enroulé autour du fil de bougie) donne une impulsion très raide qui peut être utilisée par un circuit à très haute impédance d'entrée (circuit CMOS comme certains 555 et des monostables) Il est indispensable de protéger efficacement l'entrée des circuits. On ne fait plus aujourd'hui soi-même ses monostables ! La capture du signal de bougie, si envoyée à un µC doit durer plus de 500ns pour être bien prise en compte par une interruption.

2.2 Volant

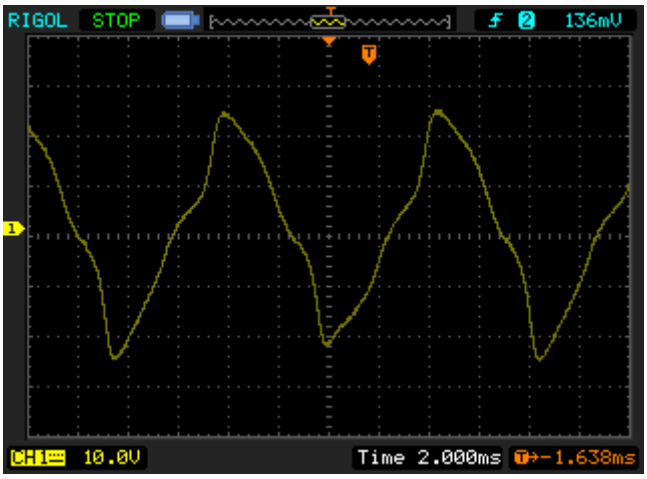
La capture du signal du volant demande un écrêtage du signal de la bobine d'éclairage, sa tension variant de 3 V à 50 V voire nettement plus selon le régime et l'allumage ou non des lampes. On pourrait théoriquement utiliser directement la tension de sortie de la bobine qui est linéairement proportionnelle au régime (merci Lenz et Faraday), mais on s'interdit alors tout éclairage (la tension varierait avec le degré d'usure des lampes). Ça s'est pourtant fait il y a des décennies avec les « dynamos tachymétriques ».
Le volant est aussi capable de fournir l'alimentation du montage mais demande une sorte de régulation, minimale par des lampes par exemple, ou plus élaborée, par de l'électronique.
La tension à vide est loin d'être sinusoïdale



à haut régime, ça monte fort :



en charge ce n'est guère mieux :

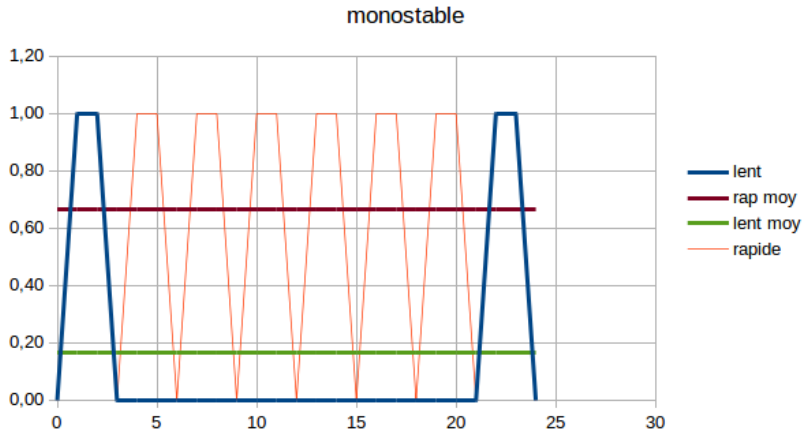


Le signal peut entrer sur le µC (atmel mega328p déjà câblé) avec une résistance un simple clamping sur les tensions d'alimentation

3 le compte-tours à monostable

3.1 fonctionnement

Le monostable est un grand classique de l'électronique, cette fonction fournit un signal de sortie de durée constante dès qu'il est déclenché par une impulsion.



La tension de sortie est intégrée (moyennée) par un galvanomètre à aiguille, ou dans un circuit à résistance-condensateur suivi d'un voltmètre digital. Lorsque les impulsions sont éloignées (régime lent) la tension moyenne est faible, lorsque les impulsions sont rapprochées (régime rapide), la tension moyenne est plus élevée. Cette technique souffre d'un défaut intrinsèque de linéarité :

C'est une fonction actuellement assurée par un circuit intégré spécifique ou reconstruit à partir d'un circuit logique (4011 par exemple)

La charge d'un condensateur est une exponentielle bien connue, une impulsion démarre le monostable et le condensateur se charge, puis, dès que la tension atteint un seuil (donc après un certain temps), le système bascule et revient au départ, prêt à recevoir une nouvelle impulsion. Le seuil de basculement est un point critique de ce système : dans un circuit spécifique comme le 555 et ses descendants, ce seuil est déterminé par un pont diviseur et les déclenchement sont assurés par des comparateurs analogiques. Les résistances forment un pont diviseur dont le rapport est parfaitement constant et indépendant de la tension d'alimentation. Comme la charge du condensateur est habituellement assurée par une résistance reliée à la tension d'alimentation, on a un ensemble dont la stabilité est assurée même pour des variations de la tension d'alimentation.

Un monostable comme le HC123 voit son condensateur chargé par une résistance (comme le 555), mais le seuil de déclenchement est assuré par une porte logique, avec donc les incertitudes inhérentes à l'utilisation analogique d'un circuit logique : grande variabilité de la tension selon l'exemplaire considéré, selon la tension d'alimentation[1] et fortement avec la température, normal, un circuit logique est conçu pour ne voir que des 0 ou des 1 (alim ou masse) ! Le HC123 est un barbarisme technologique, ce qui n'empêche pas son utilisation courante, si on n'oublie pas son

1 Un CMOS de base est constitué de deux transistors complémentaires un PMOS et un NMOS. Le NMOS a sa tension de basculement référencée à la masse, le PMOS, référencé à l'alim, la zone d'incertitude entre les deux varie avec la tension d'alimentation.

imprécision. Il faut en plus ajouter les dérives en température des condensateurs et résistances. Un monostable à portes genre HC4011 est affublé des mêmes défauts intrinsèques.

La précision et la stabilité du voltmètre d'affichage entrent bien entendu en ligne de compte pour la qualité de l'ensemble.

3.2 alimentation

Son alimentation entre en ligne de compte (monostable à circuit logique) pour la stabilité et la précision, et doit donc être soignée. Elle doit pouvoir assurer l'alimentation du monostable (quelques milliwatts) mais aussi celle de l'afficheur : quelques milliwatts pour un galva ou un afficheur à LCD, et quelques centaines de mW pour un afficheur digital à LED.

4 Le compte-tours à processeur

Ce compte-tours utilise une micro-carte à processeur déjà monté avec ses annexes. Il en existe plusieurs variantes sur le marché chinois ou français. On peut le trouver avec l'étalon de temps par un résonateur céramique ou par un quartz (il suffit de regarder la photo sur I.B.). Même le résonateur est d'une précision et d'une stabilité de plusieurs dizaines ou centaines de milliers de fois meilleures qu'un simple circuit RC avec circuit logique. Son prix est inférieur à 3€. Je ne veux pas parler d'arduino, puisque je n'utilise rien de chez arduino (environnement, philosophie, langage pourri), pour moi ce n'est qu'une carte bon marché et de grande diffusion, que je programme directement en c, via le port universel ICSP.

4.1 Capture du signal

j'ai choisi de prendre le signal de la bobine d'éclairage, plus facile à capter que le signal à la bougie dont les composants de capture doivent permettre une durée suffisante pour déclencher à coup sûr une interruption du processeur (une phase d'horloge, soit ici 500ns). Ce signal voit sa fréquence directement proportionnelle au régime moteur et sa tension dépendant du régime et de la charge électrique connectée.

Pour un solex, à quatre demi-aimants, la fréquence est double du régime de rotation

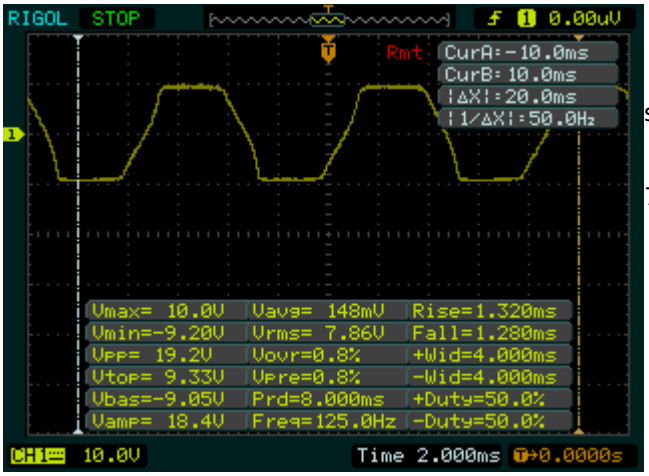
600 rpm = 20 Hz (période 50 ms)

6000 rpm = 200 Hz (période 5 ms)

l'écrêtage est assuré par une résistance série 30kΩ 1/2W vers la bobine, et des diodes Schottky vers la masse et le + de la capa de filtrage (qui dépend de l'utilisation ou non d'un afficheur à LED).

4.2 Alimentation de l'électronique

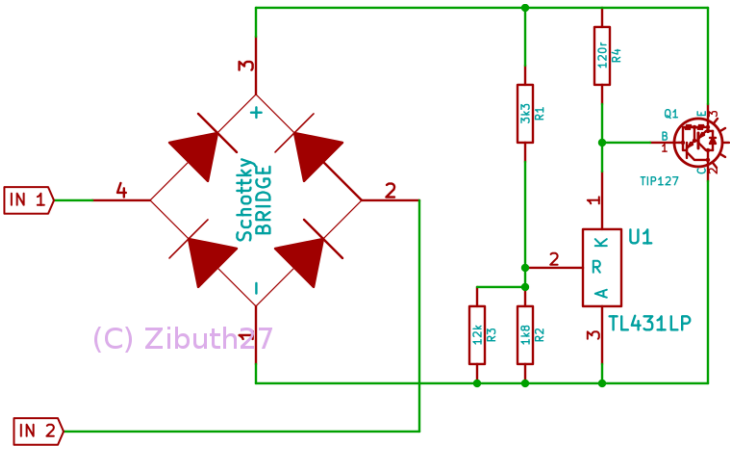
La consommation est de l'ordre de 30 mA (sortie sur galva seulement) à 60 mA (voltmètre digital) soit au mini 200mW (200mW) à 400 mW(digital)
La tension a été régulé sur les Mobs, dans les seventies, avec des Zener doubles, qui n'existent plus. Je propose donc une zener électronique pour obtenir ça :

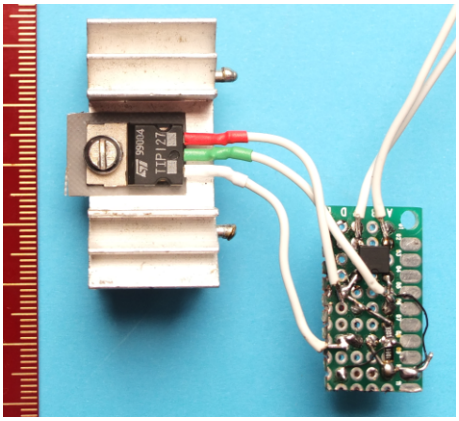


tension du volant avec le régulateur shunt

tension efficace vraie équivalente = 7,8V, supportable par les lampes
tension crête +- 10V

Vous pouvez bien entendu utiliser le montage de régulation qui vous va bien, et si vous avez une ancienne zener double Tobec, tant mieux pour vous, sinon vous pouvez utiliser ce régulateur shunt (aucun réglage si on utilise les valeurs indiquées)



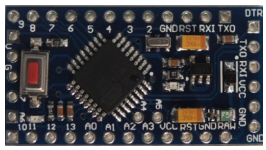


Ce montage n'atténue pas le courant utile pour les lampes, il se contente d'éliminer l'excès.
Il faut envoyer au compte-tours une tension max de 26V crête (tenue du LM2941 – 5 utilisé)
L'utilisation de cette super-zener demande juste un condensateur de 470 à 1000 µF / 16V aux bornes du transistor. Le compte-tours y est aussi relié.

Pour une utilisation avec un galvanomètre seulement, on pourrait envisager de n'utiliser qu'une zener, j'ai pas utilisé, et vous laisse donc le soin d'estimer les valeurs.

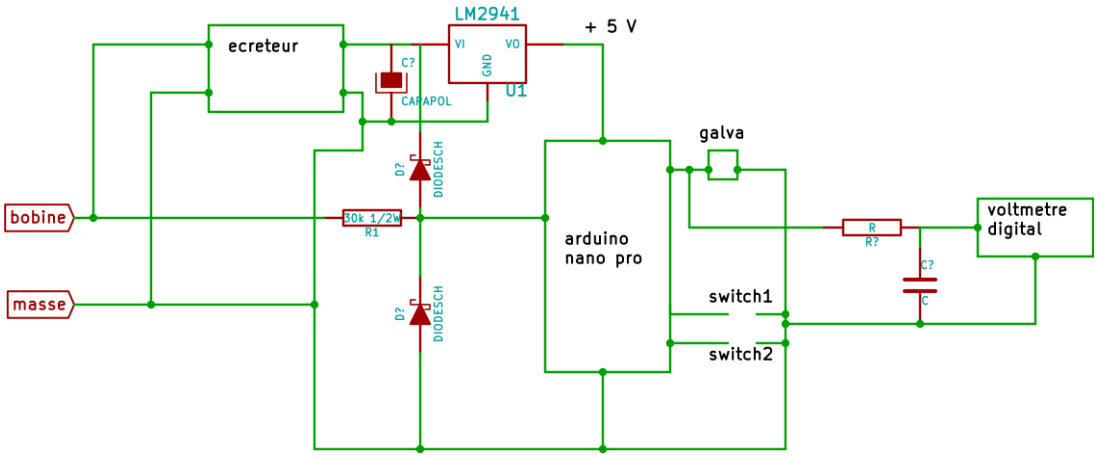
5 Le processeur

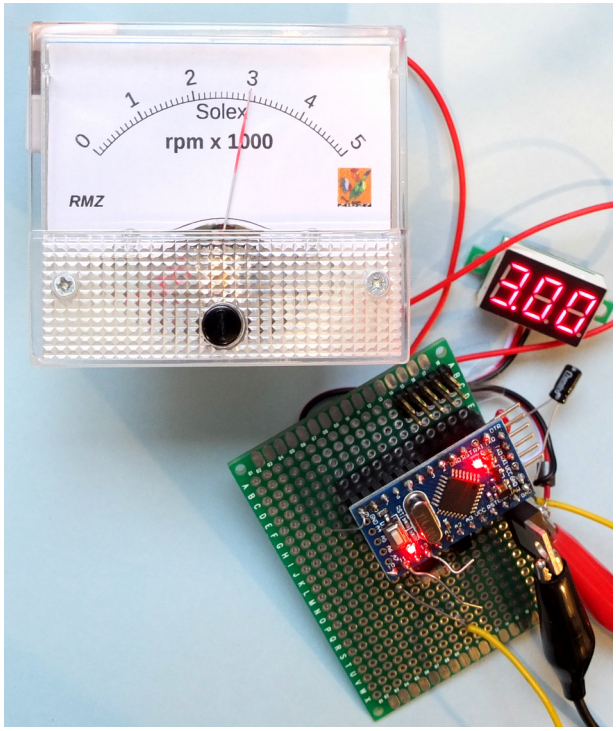
C'est un microcontrôleur Atmel atmega328p monté avec son circuit minimal sur une carte format timbre-poste

taille réelle 

Ses bornes sont identifiées à la sauce arduino et prédéfinies (sorties « digitales » avec sérigraphie de 2 à 13 et « analogiques » de A0 à A7). L'utilisation hors monde arduino permet d'utiliser le processeur sans restrictions et dans toute sa puissance.
Le signal d'entrée (tension bobine d'éclairage) déclenche une interruption processeur à chaque front montant. L'état du compteur 16bits TCNT1 (qui s'incrénente toutes les 3,2µs) est alors relevé. Un calcul donne alors le régime et l'affiche par une sortie PWM 8 bits (PWM = modulation à largeur d'impulsion). Le galvanomètre peut lire directement la valeur de 0 è 5 V, et il est intéressant d'avoir un galvanomètre 0-5V (pas besoin d'étalonnage supplémentaire). Pour cela, il est nécessaire que le régulateur d'alimentation soit le plus précis possible. Le régulateur intégré n'est pas toujours suffisamment précis (le mien faisait 5,2V, j'en ai eu de meilleurs). Il faut donc utiliser un régulateur externe comme le LM2941 à faible chute interne (permet donc de fonctionner dès une tension plus faible que le classique LM7805)
L'afficheur digital est un vrai générateur de parasites, je lui ai donc collé des perles ferrite sur chacun de ses trois fils, ça le calme (ou plutôt ne propage pas trop ses saletés)

Le premier programme assure la fonction compte-tours, but de la présente démo, et je n'ai pas encore implanté la fonction vitesse de l'engin, ni les réglages de diamètre de râpeur et de glissement.




0
1
2
3
4
5
Solex
rpm x 1000
RMZ

6 le programme

```
/*
 *            solex tachy
 *
 * atmega 328p arduino nano pro
 * crystal clock 16 MHz
 * compiler avr-gcc 4.3.5
 *
 * PB0  D8                    DIO
 * PB1  D9      OC1A          CLK           LED
 * PB2  D10     OC1B
 * PB3  D11     MOSI    OC2A  ICSP    1     OC2A    galva or DVM
 * PB4  D12     MISO          ICSP    9     LED 4 d
 * PB5  D13     SCK           ICSP    7     LED arduino
 *
 * PC0  A0                                  PMOS pulse cmd prim
 * PC1  A1                                  ADC1    coil primary
 * PC2  A2                                  ADC2    coil secondary
 * PC3  A3                                  encoder phase B
 * PC4  A4                    SDA           LED 5   e
 * PC5  A5                    SCL           LED 6   f
 * PC6  -       RESET         ICSP    5
 *
 * ADC6 A6                                  PMOS pulse command sec
 * ADC7 A7                                  LED expanded
 *
 * PD0  RXI
 * PD1  TXO
 * PD2  D2      INT0          OC2B          pulse in
 * PD3  D3      INT1                        encoder mode sw
 * PD4  D4                                  LED 7   g
 * PD5  D5      OC0B                        LED 8 dp
 * PD6  D6      OC0A          AIN0          LED
 * PD7  D7                    AIN1          jumper serial out
 *
 * RMZ # 230
 *
 * lfuse FF    hfuse DA efuse 00 arduino nano pro (resonator delivery fuses)
 * lfuse FF hfuse DA efuse 05 (nano pro with quartz)
 * lfuse 7F clock divided / 8
 *
 * status:
 * keywords:
 *
 * Zibuth27 2016/09/25
 */

#include <avr/io.h>
#include <util/delay.h>
#include <util/delay_basic.h>
#include <avr/interrupt.h>
// #define F_SCL 100000L                        // i2c clock to be 100 kHz

#define        AIN0    PD6

volatile uint16_t duree;
volatile uint32_t rpm;

int main (void)       {

       // settings

// ports
       DDRB = 0xff;                             // PORTB as output
       DDRC = 0xfd;
       DDRD = 0x32;
```

```
        PORTD |= (1<<PD7);                      // jumper pull-up

// timer0
        TCCR0A = 0x80;                                  // used for galvanometer
        TCCR0B = 0x05;                          // CTC
        OCR0A = 253;                            // start counter
        //TIMSK0 = 0x02;                        // 16.08 ms

// timer1
        TCCR1A = 0x80;                                  // for period measure
        OCR1A = 10000;                                  //
        TCCR1B = 0x04;
        TIMSK1 |= (1<<TOIE1);

// timer2                                               // used for PWM to analog galvanometer
        TCCR2A = 0x83;
        TCCR2B = 0x06;                          // 0x07 for 17 ms loop, 0x01 for ESR
        TCCR2B = 0x07;                          // start counter
        OCR2A = 0x00;                           // galvannometer control

// ADC
        ADMUX = 0x41;                           //
        ADCSRA = 0x06;                          //  ADC speed 125 kHz

//USART
        UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00);      // 8 bit
        UBRR0H = 0;
        UBRR0L = 103;
        PORTD |= (1<<PD1);
        UCSR0B |= (1<<TXEN0);

// comparator

// INT1
        EICRA |= (1 << ISC00)|(1<<ISC01);       // INT1 rising edge
        EIMSK |= (1 << INT0);

// IRQs
        sei();                                  // interrupts enable

while (1) {
// MAIN LOOP
        ;                                               // nop
        }                                               // end of while
  return 0;
}

ISR (TIMER1_OVF_vect) {                                 // rpm too low, stop LED OC2A hi-Z
        TCNT1 = 0;
//      PORTB &= ~(1<<PB5);
        DDRB &= ~(1<<PB3);
}

ISR (INT0_vect) {
        duree = TCNT1;
        DDRB |= (1<<PB3);
        TCNT1 = 0;
        PORTB ^= (1<<PB5);                      // toggle at freq/2
        rpm = 97300/duree;                              // 782000
        if (rpm<10) rpm = 0;
        if (rpm >255) rpm=255;

        OCR2A = rpm ;
}
```

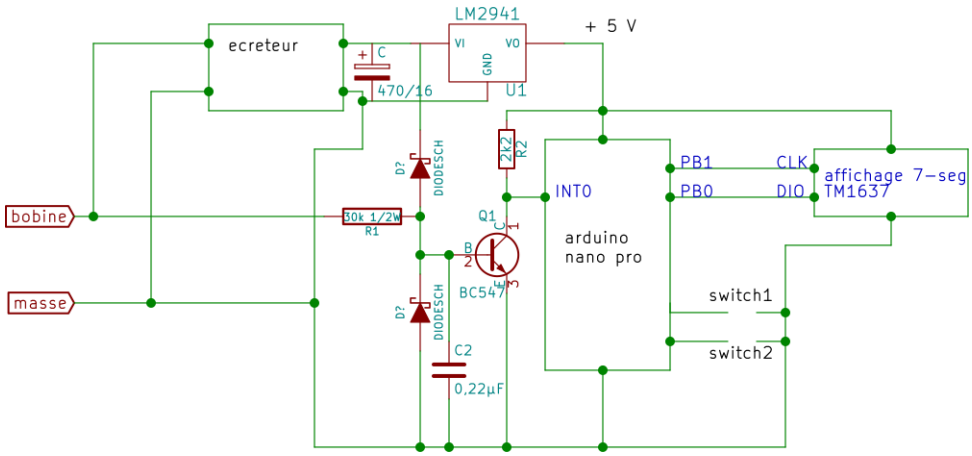
7 Version digitale

La version décrite ci-dessus reste en partie analogique, et est sensible à la précision des régulateurs 5V et à celle des afficheurs.

Une version purement digitale utilise un afficheur à 4 digits dit « digital tube » supposé utiliser un protocole I2C, c'est pipeau et baratin marketing ! Enfin , c'est pas cher et une fois maîtrisé, ça marche bien, pour 1,99$.

http://www.ebay.com/itm/2PCS-4Bits-Digital-Tube-LED-Display-TM1637-Module-With-Clock-Display-for-Arduino-/401051342850?hash=item5d6085d802:g:poUAAOSwkl5XdgPv

Du coup le hard se simplifie drastiquement, devient insensible à toute dérive et ne nécessite aucun réglage.



Les switches sont prévus pour les autres sortes de galets : 4-temps, 3 ou 4 aimants,...

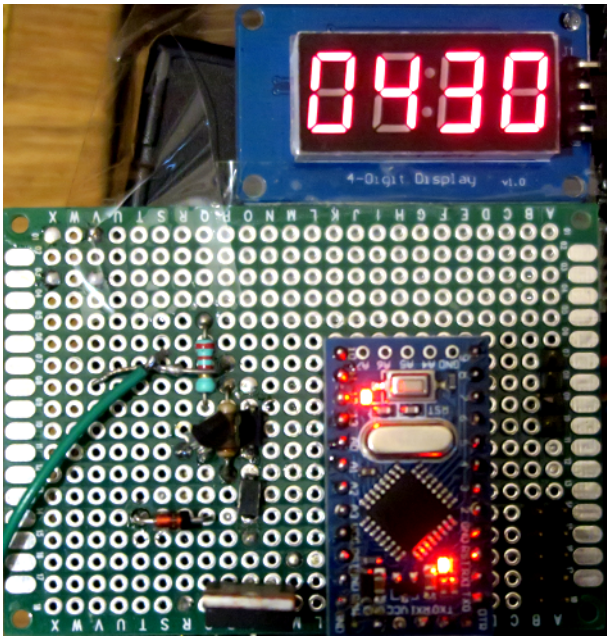
l'étage d'entrée (BC547) permet de déclencher l'interruption de mesure à une tension « fixe » de 0,7V. Il est conseillé de le mettre aussi sur le montage précédent, lequel montage précédent est sensé marcher avec un capteur entourant le fil de bougie

Un inconvénient apparent (car il existe aussi dans la version analogique, mais invisible à l'œil), est le pas d'affichage qui progresse par sauts, mais cela devrait rester dans l'instabilité naturelle de fonctionnement d'un moteur.

Résolution :

avec diviseur TCCR1B à 0x02 (div/8) elle est de 16 bits, dans la gamme de 500µs (1ms = 60000rpm) à 32ms (306Hz =9200rpm)

TCCR1B à 0x04 (div/256) gamme 78125Hz = 4 687 500 rpm à 71 rpm



le proto

Video disponible sur mon site :
http://www.hackerschicken.eu/www/electronics/variateur_banc.MOV

Programme

```
/*
 *            solex tachy
 *
 * atmega 328p arduino nano pro
 * crystal clock 16 MHz
 * compiler avr-gcc 4.3.5
 *
 * PB0  D8                    DIO
 * PB1  D9      OC1A          CLK           LED
 * PB2  D10     OC1B
 * PB3  D11     MOSI    OC2A  ICSP   1      OC2A   galva or DVM
 * PB4  D12     MISO          ICSP   9
 * PB5  D13     SCK           ICSP   7      LED arduino
 *
 * PC0  A0                                  jumper
 * PC1  A1                                  jumper
 * PC2  A2
 * PC3  A3
 * PC4  A4                    SDA
 * PC5  A5                    SCL
 * PC6  -       RESET         ICSP   5
 *
 * ADC6 A6
 * ADC7 A7
 *
 * PD0  RXI
 * PD1  TXO
 * PD2  D2      INT0          OC2B
 * PD3  D3      INT1
 * PD4  D4
 * PD5  D5      OC0B
 * PD6  D6      OC0A          AIN0
 * PD7  D7                    AIN1
 *
 * RMZ # 230
 *
 * lfuse FF    hfuse DA efuse 00 arduino nano pro (resonator delivery fuses)
 * lfuse FF hfuse DA efuse 05 (nano pro with quartz)
 * lfuse 7F clock divided / 8
 *
 * status:
 * keywords: TM1637, 4 digit LED display,
 *
 * Zibuth27 2016/10/06
 */


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

#define        DIO            PB0
#define        CLK            PB1
#define        LED            PB5
#define        CLK1           PORTB |= (1 << CLK)    // CLK 1
#define        CLK0           PORTB &= ~(1 << CLK)   // CLK 0
#define        DIO1           PORTB |= (1 << DIO)    // DIO 1
#define        DIO0           PORTB &= ~(1 << DIO)   // DIO 0
#define        HALFBIT _delay_loop_1 (4)             // 2µs  if 10, clock is 80kHz = OK
#define        ACKBIT  _delay_loop_1 (10)    // 5µs  50
#define        timer1_sp      0x02
#define        rpm_cte        60400000

volatile uint8_t ack, m, c, d, u;
volatile uint16_t duree, duree1, duree2, duree3, duree4, duree5;
```

```
volatile uint32_t dureetot, rpm, rpm_const = rpm_cte;
volatile uint8_t msg[4] = {0x06,0xdb,0x4f,0x66};
volatile uint8_t segments[16] = {0x3f,0x06,0x5b,0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f,0 ,0 ,0 ,0 ,0 ,0 };
       // 7 segments

int main ()    {
// variables
       //uint8_t switch_read;


// ports
       DDRB = 0xff;                                    // PORTB as output
       DDRC = 0xff;
       DDRD = 0x32;                                    //

// timer0

// timer1

       TCCR1A = 0x00;
       OCR1A = 10000;                                          //
       TCCR1B = timer1_sp;
       TIMSK1 |= (1<<TOIE1);

// timer2

//USART
       //UCSR0B |= (1<<TXEN0);
       //UCSR0C |= (1<<UCSZ01)|(1<<UCSZ00); // 8 bit
       //UBRR0H = 0;
       //UBRR0L = 103;

// IRQs
       //TIMSK1 |= (1<<TOIE1);                                 // enable timer1 IRQ
       //EICRA = 02;                                     // INT0 rising edge
       //EIMSK = 01;                                     // INT0 enable, OCR1A match
       //sei();

// INT1
       EICRA |= (1 << ISC01);//|(1<<ISC01);        // INT1 rising edge
       EIMSK |= (1 << INT0);

       sei();

       _delay_us(100);

       while (1){
// main loop
/*
       PORTB |= (1<<PB5);                              // scope sync, on board LED
       _delay_us(50);                                  // this can be
       PORTB &= ~(1<<PB5);                             // suppressed
*/
       start_i2c ();                                   // write command mode = auto increment address
       wr_byte (0x40);
       ask_i2c ();
       stop_i2c ();

       start_i2c ();                                   // start address = digit 1
       wr_byte (0xc0);
       ask_i2c ();

       wr_byte (msg[0]);                                       // digit 1
       ask_i2c ();

       wr_byte (msg[1]);                                       // digit 2
       ask_i2c ();

       wr_byte (msg[2]);                                       // digit 3
       ask_i2c ();
```

```
        wr_byte (msg[3]);                                   // digit 4
        ask_i2c ();

        stop_i2c();

        start_i2c ();                                       // brightness control 8f = max
        wr_byte (0x8f);
        ask_i2c ();
        stop_i2c();

        //_delay_us(10);


        moyennage ();
        _delay_ms(1);
        DDRB |= (1<<PB3);

//      PORTB ^= (1<<PB5);                                  // toggle at freq/2
        rpm = rpm_const/duree;                              // 782000

        m = rpm/1000;
        c = (rpm-(m*1000))/100;
        d = (rpm-(m*1000) -(c*100))/10;
        u = (rpm-(m*1000)-(c*100)-(d*10))/1;
        u=0;                                                // 3 digits are enough

        msg[0] = segments [m];
        msg[1] = segments [c];
        msg[2] = segments [d];
        msg[3] = segments [u];
//      PORTB ^= (1<<PB5);

// the writing to the TM1637 chip has to be repeated constantly, as the chip has no buffer

        }
  return 0;
}

void start_i2c () {
        CLK1;                           // DIO 1
//      HALFBIT;
        DIO1;                           // CLK 1
        HALFBIT;
        DIO0;                           // DIO 0
//      CLK0;
        HALFBIT;
}

void stop_i2c () {
        CLK0;                           // CLK 0
        HALFBIT;
        DIO0;                           // DIO 0
        HALFBIT;
        CLK1;                           // CLK 1
        HALFBIT;
        DIO1;                           // DIO 1
        HALFBIT;
        HALFBIT;
//      PORTB &= ~(1<<PB5);
}

void ask_i2c () {
        CLK0;
        DDRB = 0xfe;   //&= ~(1<<DIO);              // Master releases DIO for reading ACK state
        _delay_us (4);
        //CLK1;
        //ACKBIT;
        while (PINB & (1<<DIO)) ;
//      PORTB |= (1<<PB5);    // NACK
```

```
        DDRB = 0xff;   //|= (1<<DIO);               // Master takes over the bus again
        CLK1;
        HALFBIT;
        CLK0;
}

//Titan micro uses "i2c" but LSB first (i2c = strictly MSB first)

void wr_byte (uint8_t octet) {
        uint8_t i;
        for (i=0;i<8;i++) {           // was i=0;i<8;i++ for LSB first
                CLK0;                         //
                if(octet & 0x01) {
                        DIO1;
                }
                else {
                        DIO0;
                }
                HALFBIT;       //_delay_loop_1 (3);  //30
                octet = octet >>1;    //
                CLK1;
                HALFBIT;       //_delay_loop_1 (3);
                HALFBIT;
        }
}
/*
void frame_error () {
        ;
}
*/
void moyennage () {

        dureetot = (duree + duree1 + duree2 + duree3 + duree4 + duree5)/6;
        duree5 = duree4;
        duree4 = duree3;
        duree3 = duree2;
        duree2 = duree1;
        duree1 = duree;
}

ISR (TIMER1_OVF_vect) {                                 // rpm too low, stop LED OC2A hi-Z
        //TCNT1 = 0;                                    // used for checking the longest period allowable
        //PORTB ^= (1<<PB5);
        //DDRB &= ~(1<<PB3);
}

ISR (INT0_vect) {
        duree = TCNT1;
        TCCR1B = 0;

        PORTB ^= (1<<PB5);

        TCNT1 = 0;
        TCCR1B = timer1_sp;

}
```