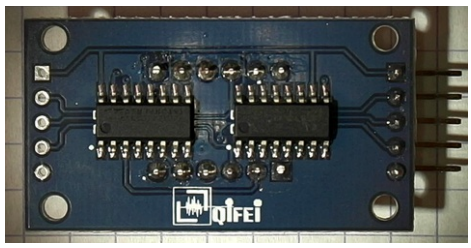
Ce compte-tours, destiné à un banc d'essais de dynamos, utilise un afficheur LED, plus lisible qu'un afficheur LCD. L'afficheur est un modèle à 4 chiffres trouvé sur I.B., vendu 2$ ou moins : *http://www.ebay.com/itm/4-Bits-LED-Digital-Tube-Display-module-four-serial-module-595-drives-M57-/291244188611?pt=LH_DefaultDomain_0&hash=item43cf81a3c3.*

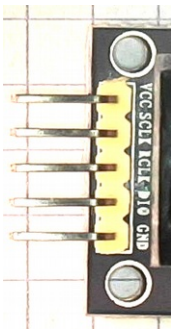
Il est fait d'un bloc de quatre afficheurs 7 segments et de deux registres à décalage série/parallèle 74HC595.

Aucune documentation disponible sur le Net ou chez le vendeur, il faut donc reconstituer le schéma et le fonctionnement qui en découle.

Pas de résistances de segments visibles, elles pourraient être incluses dans le bloc à LEDs, car la consommation globale d'une centaine de mA ainsi que l'intensité lumineuse ne font pas penser à une liaison sans résistances entre registres et LED (module 5V).



Les modules d'affichage sont sérialisables par simples câbles parallèles pin à pin, on obtient des afficheurs à 4, 8, 12 LED ou plus, par groupe de 4.



entrée et sortie du module

visiblement pas de composants (résistances) sous l'afficheur

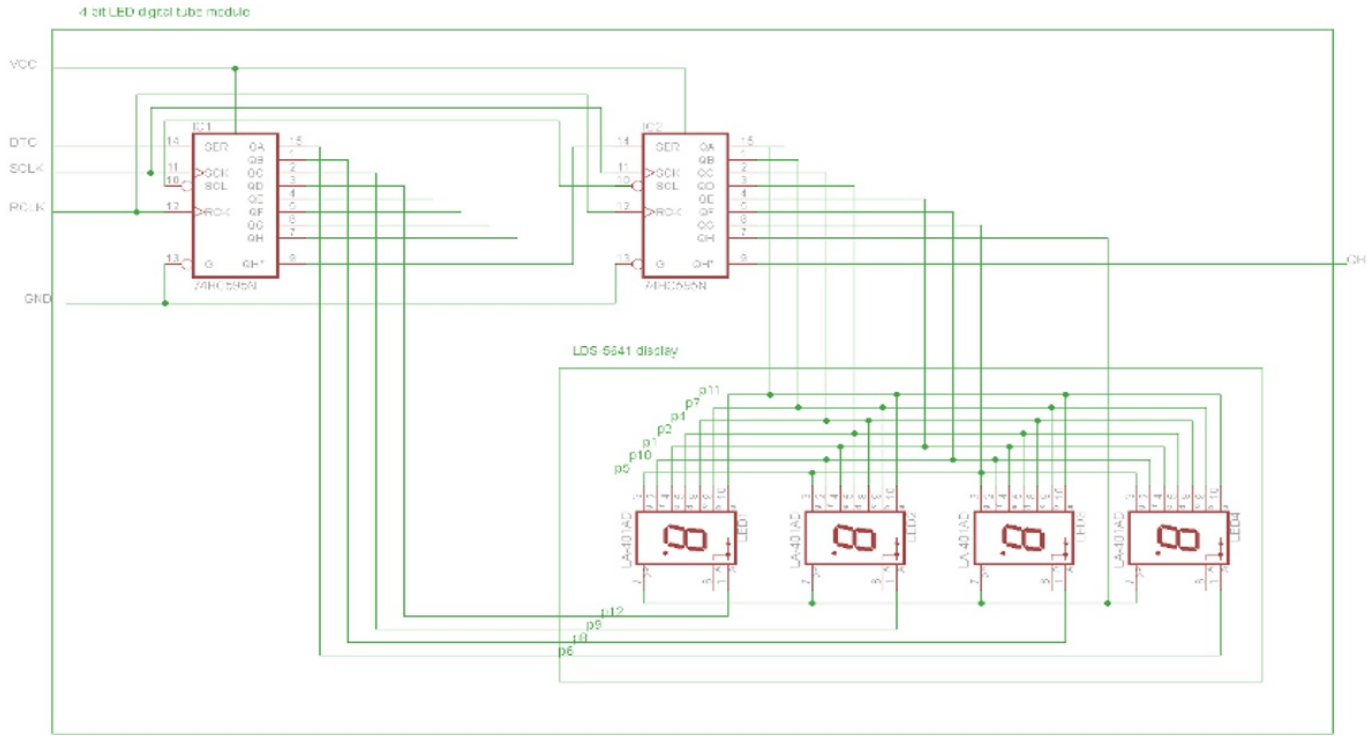
Reconstitution du schéma et de l'utilisation du module d'affichage

Le bloc de LEDs est référencé ZS3641BS, je n'ai trouvé aucune datasheet de ce composant. Une mesure des LEDs donne une correspondance fonctionnelle avec le LDS5641 à anode commune.

Chaque digit est accessible par un mot de 16 bits :

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
content	Dig 4	Dig 3	Dig 2	Dig 1	NC	NC	NC	NC	Seg a	Seg b	Seg c	Seg d	Seg e	Seg f	Seg g	dp

  zibuth27@gmx.fr



En raison de la sérialisation choisie par le fabricant, le codage est dans l'ordre : gfedcba, les bits sont envoyés dans l'ordre Little Indian (LSB en premier).

Digit affiché :

segment	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Code (hex)	0xFC	0x60	0xDA	0xF2	0x66	0xB6	0xBE	0xE0	0xFE	0xF6	0xEE	0x3E	0x9C	0x7A	0x9E	0x8E

L'adresse du digit est codée dans le second octet, c'est donc un affichage multiplexé, affichage séquentiel digit après digit :

digit	1	2	3	4
Adresse (hex)	10	20	40	80

Exemples :
pour afficher « A » sur le premier digit, le mot à envoyer au module est 0xEE10
pour afficher « 6 » sur le premier digit, le mot à envoyer au module est 0xBE40
pour ajouter le point décimal, simplement ajouter 1 mot

L'algorithme d'affichage est le suivant

1. placer le mot dans une variable
2. tester le LSB
3. placer le bit correspondant sur DIO
4. générer le strobe SCLK
5. décaler à droite d'un pas
6. répéter les étapes 2 à 5, quinze fois
7. générer le strobe RCLK

Le microcontrôleur

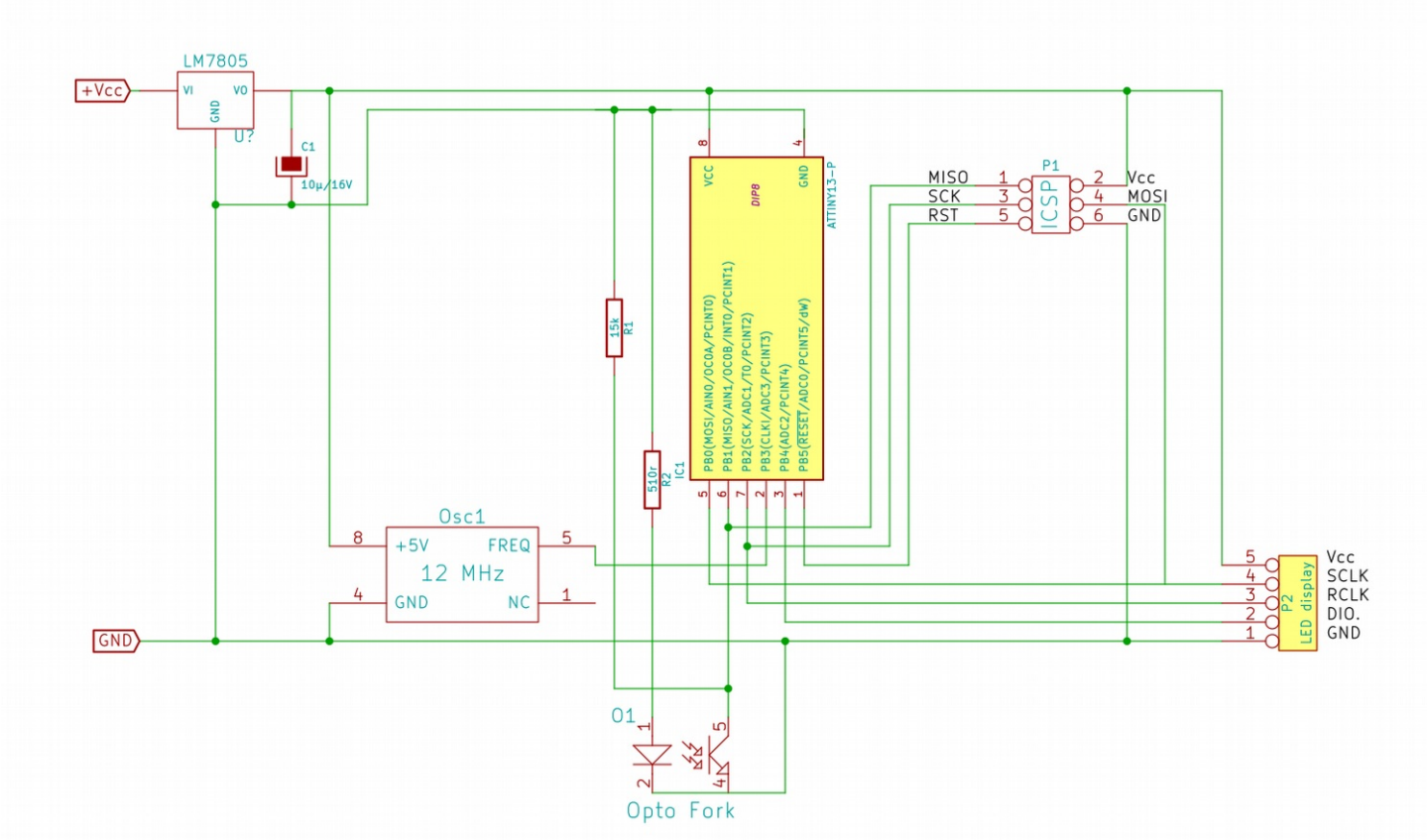
Un atmel tiny13 possède suffisamment de pins et de vitesse pour faire le job, pour un demi-euro.

Le montage utilise toutes les pins disponibles (sauf Reset qui ne sert que lors de la programmation). Un connecteur ICSP 6 pins est installé, et comme tous les signaux sont à impédance de plus de 2kΩ, la programmation par le port ICSP peut se faire en live, sans rien déconnecter (il faut toutefois empêcher le phototransistor de conduire, en introduisant un objet opaque dans la fourche, ou en tournant l'arbre de mesure).

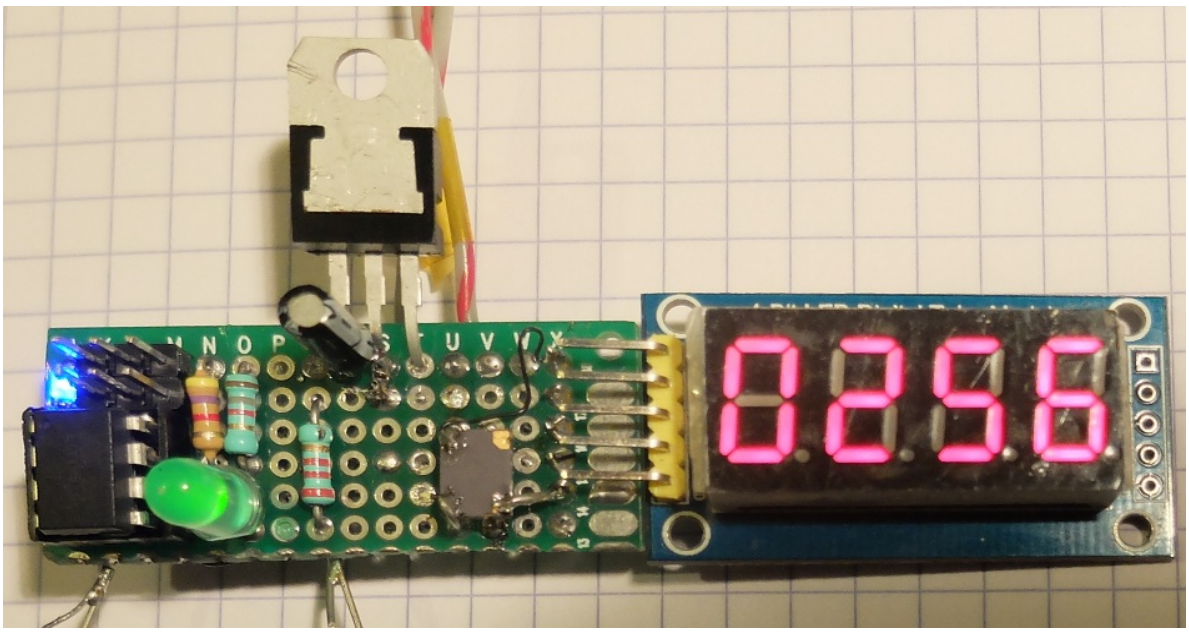
Pour ne pas être tributaire de la mauvaise stabilité de l'oscillateur interne (comme tous les µC), un oscillateur à quartz externe de 12 MHz est utilisé. Il faut donc programmer le lfuse pour cela. Par exemple par l'envoi de l'instruction avrdude suivante :
sudo avrdude -p t13 -c usbasp -U lfuse:w:0x78:m (« -c usbasp » = pour carte de prorgammation à 3$: USBASP)
attention : après cette programmation du fusible, on ne peut plus désormais accéder au µC, pour test ou reprogrammation, qu'en fournissant une horloge à la pin 2, par exemple le montage de compte-tours lui-même.
Pour réutiliser le µC dans une autre application, ne pas oublier de remettre le lfuse à 0x7A !

Schéma du compte-tours

super-simple (KISS : keep it simple[,] stupid)
Un régulateur de tension basique assure la fourniture du 5V. Un oscillateur à quartz est relié à la pin 2. Le capteur à fourche optique est alimenté par une résistance de 510Ω, sa sortie est reliée à la pin 6, configurée en entrée haute impédance et générant l'interruption INT0.



L'affichage, sans filtre rouge



Le montage électronique nu, et en boîtier

Il mesure des vitesses de rotation de 20 à 9999 t/min

Programme :

En tâche cachée, le timer 0 est activé, sans prédiviseur, sans sortie vers l'extérieur (OC0), mais en générant une interruption de débordement chaque fois qu'il cycle 256 pas. Cette interruptions arrive à 46875kHz / 21.33µs, sa seule tâche est d'incrémenter le compteur de boucles.

Une autre interruption est déclenchée par le front montant du capteur optique. Elle lit le nombre de boucles puis remet à zéro le compteur de boucles.

La tâche de fond (main) s'occupe en permanence de l'affichage, après les initialisations d'usage (déclaration

des variables, direction des ports, configuration du timer et des interruptions). Cela commence par la décomposition de la valeur du nombre de boucles en quatre chiffres, leur codage en valeurs lisibles par l'afficheur, envoi des bits petit poids en tête (Little Indian), et envoi des strobes validant les données. Il a fallu utiliser des variables en 32 bits pour pouvoir assurer correctement le calcul de la vitesse. La place occupée par le programme est de 916 octets sur les 1024 disponibles

```
/* t13 rpm meter
 *
 * attiny13
 * crystal clock 12 MHz
 * compiler avr-gcc 4-8-2.1
 * lfuse = 0x78 (external clock), full speed
 *
 * Zibuth27 06/10/2015
 * 916 bytes
 * status:
 * keywords: 4-bit LED digital tube module, crystal oscillator
 */

/* connections
*  pin2 PB3 crystal oscillator
*  pin3 PB4 DIO
*  pin5 PB0 SCLK
*  pin6 PB1 INT0 input
*  pin7 PB2 RCLK
*/

/* data
 * 0 = FC
 * 1 = 60
 * 2 = DA
 * 3 = F2
 * 4 = 66
 * 5 = B6
 * 6 = BE
 * 7 = E0
 * 8 = FE
 * 9 = F6
 * A = EE
 * B = 3E
 * C = 9C
 * D = 7A
 * E = 9E
 * F = 8E
 *
 * addresses
 * dig1 = 80
 * dig2 = 40
 * dig3 = 20
 * dig3 = 10
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#define F_CPU 12000000UL // 12 MHz
#include <util/delay.h>



volatile uint32_t value;
volatile uint16_t counting, loops;

// generates the serial data clock pulse (SCLK)
void serial_ck()
{
           PORTB |= (1<<PB0);
           PORTB &=~(1<<PB0);
           PORTB |= (1<<PB0);
}

// transfers the data to output register (RCLK)
void parallel_reg()
{
```

```
            PORTB |= (1<<PB2);
            PORTB &=~(1<<PB2);
            PORTB |= (1<<PB2);
}

// send data part of the word (DIO)
void send_data(uint8_t word)
{
    for(uint8_t i=0;i<8;i++)
            {
                    if(word & 1) PORTB &=~ (1<<PB4);     // data inversion (common anode)
                           else PORTB |= (1<<PB4);
                    serial_ck();
                    word = word >> 1;
            }
}

// generates the address part of the word (DIO)
void send_addr(uint8_t digit)
{
            for(uint8_t i=0;i<8;i++)
            {
                    if(digit & 1)PORTB |= (1<<PB4);             // data non inverted (anode address)
                           else PORTB &=~ (1<<PB4);
                    serial_ck();
                    digit = digit >> 1;
            }
      parallel_reg();
}

// conversion of digits (0 . . 9) to seven segments display
uint8_t decode(uint8_t input)
{
    uint8_t sevenseg;

    switch (input)
    {
            case 0 : sevenseg = 0xFC; break;
            case 1 : sevenseg = 0x60; break;
            case 2 : sevenseg = 0xDA; break;
            case 3 : sevenseg = 0xF2; break;
            case 4 : sevenseg = 0x66; break;
            case 5 : sevenseg = 0xB6; break;
            case 6 : sevenseg = 0xBE; break;
            case 7 : sevenseg = 0xE0; break;
            case 8 : sevenseg = 0xFE; break;
            case 9 : sevenseg = 0xf6; break;
            default : sevenseg = 0x00; break;    // blank
    }
    return sevenseg;
}

//  MAIN
void main(void)
{
    uint8_t data1, data2, data3, data4, addr;

// port settings
    DDRB = 0x1d;
    PORTB = 0x1d;

// timer setting
    TCCR0A = 0x00;                      // OC0A OC0B disconnected
    TCCR0B = 0x01;//|= (1<<CS00);       // no prescaling timer mode 0
    TIMSK0 |= (1<<TOIE0);               // timer overflow interrupt

// external ISR setting
    MCUCR =0x03;                        // interrupt at rising edge
    GIMSK |= (1<<INT0);                 // pin change interrupt
    PCMSK |= (1<<PCINT0);               // PB0 as interrupt source

    sei();                              // enable interrupts

// endless loop
  for (;;)
```

```
  {
            value = 2812500/counting;     // timer0 OVF (46875 x 60)


// decomposition of the number in 4 digits
            data1=value/1000;
            data1=decode(data1);
            value=value%1000;
            data2=value/100;
            data2=decode(data2);
            value=value%100;
            data3=value/10;
            data3=decode(data3);
            value=(value%10-1);
            data4=value;
            data4 = decode(data4);



// send digits to display module
            send_data(data1);
            addr=0x10;
            send_addr(addr);
            _delay_ms(5);         // multiplexing adjustment for LED brightness

            send_data(data2);
            addr=0x20;
            send_addr(addr);
            _delay_ms(5);         // multiplexing adjustment for LED brightness

            send_data(data3);
            addr=0x40;
            send_addr(addr);
            _delay_ms(5);         // multiplexing adjustment for LED brightness

            send_data(data4);
            addr=0x80;
            send_addr(addr);
            _delay_ms(4);         // multiplexing adjustment for LED brightness


            _delay_ms(1);

  } // end of infinite loop
  return 0;
}   // end of main

ISR (TIM0_OVF_vect)
{
    ++loops;                              // 46875kHz / 21.33µs
}

ISR (INT0_vect)
{
            counting = loops;
            loops = 0;
}
```

  zibuth27@gmx.fr