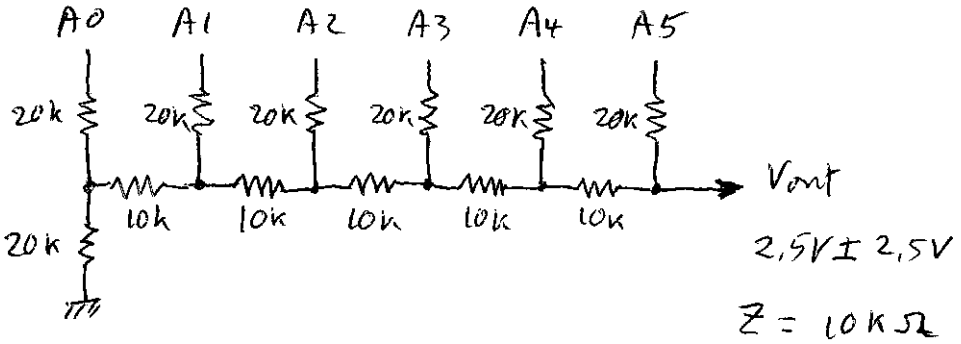
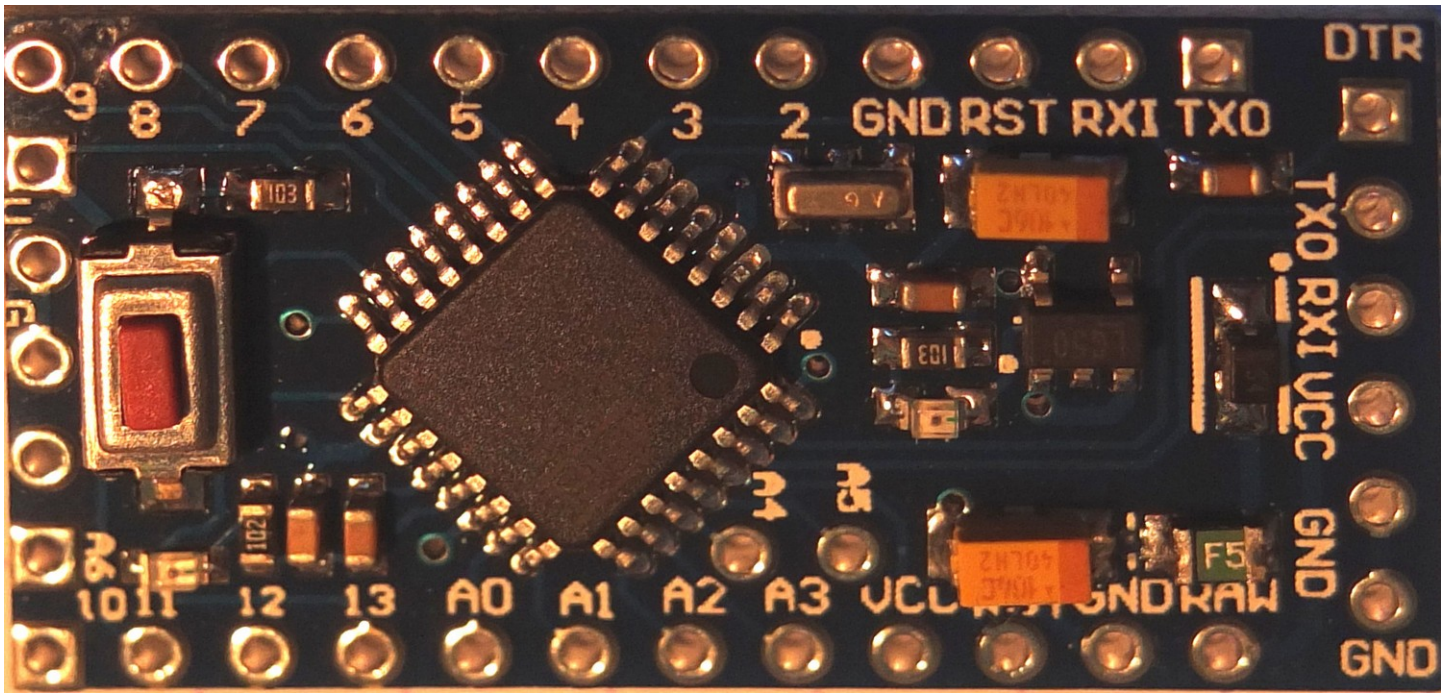
Générateur DDS R2R

V0.0

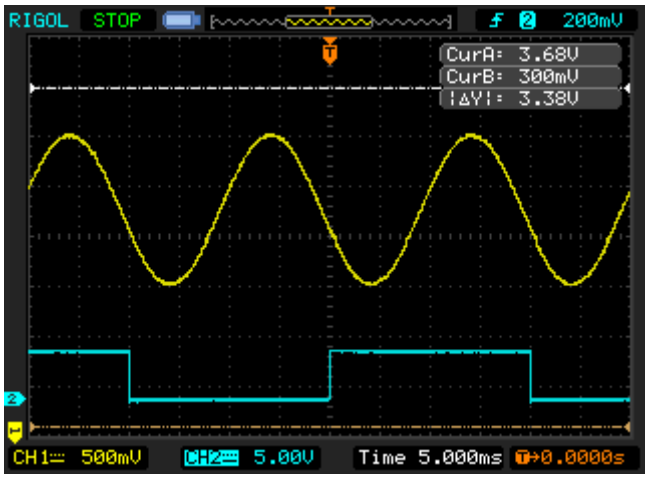
Générateur très simple
6 bits = 1,5 % distorsion due au système (la tolérance des résistances augmente l'imprécision)
fréquence adaptable

Hardware simple :
les résistances sont reliées aux ports A0 – A5 de l'arduino





Résultat , avec filtrage 1µF



Programme simplissime en c :

le déroulement du programme fait juste deux lignes dans le while(1)
le timer 1 est utilisé en fréquence variable qui sert d'entrée au timer 0 qui est le descripteur de signal
la forme du signal est stockée en table (LUT) qui peut être sinus, triangle, carré, ou toute forme au choix, décrite en 64 états.
L'augmentation de la résolution demande une table plus grande, fonction du nombre de bits, et l'utilisation d'autres ports, compliquant un peu l'écriture

pour un sinus à 50Hz, il faut que les bits du DAC soient activés à 12800Hz
A cause de l'entrelacement des interruptions, le montage ne monte guère au-delà de 2500Hz

timer1 :
pas de prescaler
ICR1 = 5000
CTC = mode 14 (WGM = 1110x)
TIMSK = TOI1E

timer0 :
mode 2 WGM = 010
OCR0A = 64
TIMSK0 = TOIE0

```
/* 50Hz sinus generator
 * at mega328p
 * arduino nano pro
 *
 * DAC R2R
 *
 * PB5 LED output
 * PC0 DAC bit A0
 * PC1 DAC bit A1
 * PC2 DAC bit A2
 * PC3 DAC bit A3
 * PC4 DAC bit A4
 * PC5 DAC bit A5
 * PD4 TO, input of timer0
 *
 * crystal 16MHz
```

```
 *
 * status: OK
 * keywords: timer 1 variable frequency (ICR1), DAC R2R 6 bits
 *
 * (CC) nc, sa, by, zibuth27 2015/02/10,
 */

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


       // global varianbles
volatile uint8_t outval;
volatile uint8_t sinus64[] =
       {
              0x00, 0x00, 0x00, 0x01,
              0x02, 0x04, 0x05, 0x07,
              0x09, 0x0C, 0x0E, 0x11,
              0x14, 0x17, 0x1A, 0x1D,
              0x20, 0x23, 0x26, 0x29,
              0x2C, 0x2F, 0x31, 0x34,
              0x36, 0x38, 0x3A, 0x3C,
              0x3D, 0x3E, 0x3F, 0x3F,
              0x3F, 0x3F, 0x3F, 0x3E,
              0x3D, 0x3C, 0x3A, 0x38,
              0x36, 0x34, 0x31, 0x2E,
              0x2B, 0x28, 0x25, 0x22,
              0x1F, 0x1C, 0x19, 0x16,
              0x13, 0x10, 0x0D, 0x0B,
              0x09, 0x06, 0x05, 0x03,
              0x02, 0x01, 0x00, 0x00
       };

void main()
{
       // local variables

       // output pins
       DDRB |= (1<<PB5);                          // LED output enable
       DDRC  = 0x3f;
       DDRD |= (1<<PD4)|(1<<PD6);                 // PD4 enable (T0=clock timer0)

       // set timer0
       TCCR0A |= (1<<WGM01);                      // CTC mode 2, recycle on OCR0A
       TCCR0B |= (1<<CS02)|(1<<CS01)|(1<<CS00);   // clock source on T0 pin (PD4)
       OCR0A   = 63;
       TIMSK0 |= (1<<OCIE0A);                     // ISR mask OVF0

       // set timer1
       TCCR1A |=(1<<WGM11);                       // CTC mode 14
       TCCR1B |=(1<<WGM13)|(1<<WGM12)|(1<<CS10);  // CTC mode, prescaler
       TIMSK1 |=(1<<TOIE1);                       // ISR mask OVF1
       ICR1    = 250;                             // frequency adjust :
                                                  // ICR1 = 16MHz / (outfreq*64)
                                                  // = 5000 for 50Hz
                                                  // = 50 for 4500Hz
                                                  // = 568 for 440Hz
                                                  // = 250 for 1000Hz
                                                  // ICR1 >= 50

       // set ADC

       // set USART

       // interrupts enable
       sei();

  while (1)                                       // endless loop
  {
              outval = sinus64[TCNT0];
              PORTC = outval;
  }
```

```
}

ISR(TIMER1_OVF_vect)                                    // 3200Hz
{
                PORTD ^= (1<<PD4);
                PORTD ^= (1<<PD4);
}

ISR(TIMER0_COMPA_vect)                                  // 50Hz
{
                PORTB ^= (1<<PB5);                      // LED flag
}
```