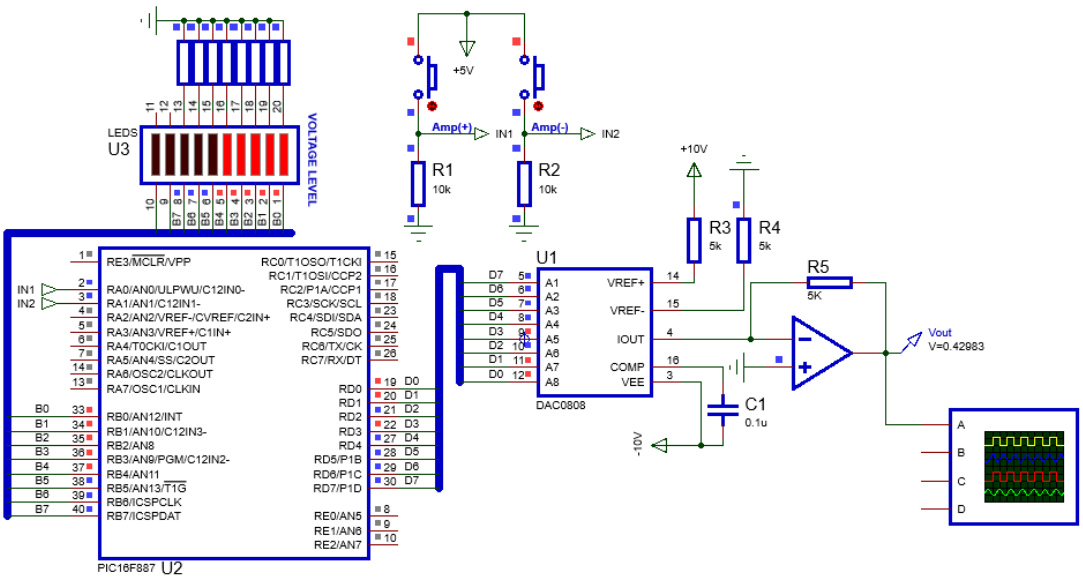


Objectifs

- Savoir utiliser la fonction rand() et son initialisation
- Savoir créer un décodeur (code 1 => code 2)
- Savoir créer une nouvelle fonction MikroC
- Se familiariser à l'utilisation du convertisseur N/A (DAC)
- Comprendre le principe de fonction du générateur du bruit
- Etc.

Fonctionnement

Le montage est un générateur d'une séquence pseudo-aléatoire numérique d'amplitudes variables. Il est constitué des éléments suivants :

- Boutons poussoirs Amp(+) & Amp(-) : Ils servent à augmenter ou réduire l'amplitude du signal. On dispose de neuf niveaux N en fonction de l'intensité du signal : 0, 1, 2, 3, …, 7, 8. Lorsqu'on appuie sur le bouton Amp(+) on passe de niveau N à N+1, et N à N-1 dans le cas d'appuie sur Amp(-).

- LEDs : Permettent de visualiser l'intensité du signal à la sortie du DAC. Les LEDs sont connectés avec le port B du microcontrôleur. Un mot de 8 bits est envoyé uniquement au moment de l'appuie sur l'un des boutons poussoirs. Le mot initial est « 0x00 » qui correspond à N=0.
- Le convertisseur A/N: Un mot de 8 bits est envoyé au convertisseur A/N en permanent. Le convertisseur permet de convertir la valeur numérique en un signal analogique. Voir le projet source de tension pour plus de détails.

Décodeur

Fonctionnement

Un décodeur est un composant sous forme logiciel ou matériel qui permet de convertir un code 1 en un autre code 2. Ici on a besoin de convertir les valeurs « 0 » à « 8 » à un autre format exploitable par les LEDs d'une part, et par le convertisseur N/A d'autre part. Ci-dessous la correspondance entre les deux codes :

Code 1 (entrée) ==> Code 2 (sortie):

- 0 ==> 0x00
- 1 ==> 0x01
- 2 ==> 0x03
- 3 ==> 0x07
- 4 ==> 0x0F
- 5 ==> 0x1F
- 6 ==> 0x3F
- 7 ==> 0x7F
- 8 ==> 0xFF

Note: En résumé, pour passer du code 1 au code 2 il suffit de

positionner à « 1 » le bit du poids fort suivant et maintenir les autres dans l'état « 1 ». Ce type de codage nous permettra de multiplier par deux (ou diviser/2) l'amplitude à une unité prêt : Valeur(N+1)=2*Valeur(N)+1.

La fonction Code1_2_Code2()

Elle convertie le code 1 en code 2. Elle prend en entrée le code 1, puis elle renvoie le code 2. Ci-dessous la syntaxe et la déclaration de la nouvelle fonction.

```
Syntaxe : unsigned short Code1_2_Code2( unsigned short num_bitt)
```

```
// Déclaration
unsigned short Code1_2_Code2( unsigned short num_bitt)
{
  unsigned short val_DAC=0;
  switch (num_bitt)
 {
    case  0: val_DAC = 0x00; break;
    case  1: val_DAC = 0x01; break;
    case  2: val_DAC = 0x03; break;
    case  3: val_DAC = 0x07; break;
    case  4: val_DAC = 0x0F; break;
    case  5: val_DAC = 0x1F; break;
    case  6: val_DAC = 0x3F; break;
    case  7: val_DAC = 0x7F; break;
    case  8: val_DAC = 0xFF; break;
 }
 return   val_DAC;
```

```
}
```

Fonctions du générateur du bruit

Rand()

Elle constituée le cœur du générateur. Elle renvoie une valeur pseudo-aléatoire en format entier (16 bits) comprise entre 0 et 32767.

```
Syntaxe : int rand();
```

Srand()

La fonction rand() renvoie toujours la même séquence des nombres aléatoires, d'où l'appellation « pseudo-aléatoire. En effet, on peut réinitialiser la valeur du départ de la fonction afin d'obtenir une séquence variable. La fonction srand() force une valeur de départ de la séquence. Ci-dessous la syntaxe.

```
Sytanxe: void srand(unsigned x);
```

> Note: Dans notre cas, on va réinitialiser la valeur de départ de la séquence lorsqu'on appuie sur l'un des deux boutons poussoirs Amp(+/-) (voir le programme principe).

Comment adapter le format 16 bits de rand() au format 8 bits du DAC ?

Comme vous l'avez constaté la fonction rand renvoie une valeur en format entier positif. En revanche, le DAC ne peut avoir que 8 bits à l'entrée ! En conséquent, l'adaptation des deux formats est nécessaire. Ci-dessous quelques propositions non exhaustives :

1. Retenir uniquement le poids faible sur 8 bits de la valeur aléatoire (perte du poids fort) :

```
rand_val=rand();
```

```
rand_val= (rand_val & 0x00FF);
```

2. Retenir le poids fort (perte du poids faible) :

```
rand_val=rand();
```

```
rand_val=(rand_val>>8 & 0x00FF);
```

3. Mélanger les deux poids: Permets de maintenir l’ensemble de l’information

```
rand_val=rand();
```

```
rand_val= (rand_val & 0x00FF) | (rand_val>>8 & 0x00FF);
```

Note: Le format 3 sera utilisé

Programme MikroC

```
#include <built_in.h>

unsigned short  DAC_val=0;
unsigned short  DAC_step=1;
float DAC_fval=0.0;
int rand_val;
unsigned short num_bit=0;

// Paramètres de la Tension de sortie (DAC)
const float Res_5=5000;
const float Courant_LSB=7.8125E-6;
float Tension=0.0;

// Décodeur pour affichage (LEDs)
unsigned short Code1_2_Code2( unsigned short num_bitt)
{
  unsigned short val_DAC=0;
  switch (num_bitt)
 {
   case  0: val_DAC = 0x00; break;
   case  1: val_DAC = 0x01; break;
   case  2: val_DAC = 0x03; break;
   case  3: val_DAC = 0x07; break;
   case  4: val_DAC = 0x0F; break;
   case  5: val_DAC = 0x1F; break;
   case  6: val_DAC = 0x3F; break;
   case  7: val_DAC = 0x7F; break;
   case  8: val_DAC = 0xFF; break;
 }
 return   val_DAC;
}
```

```
void main()
{
     // Initialisation du générateur
     srand(128);
     // Configure AN pins as digital
     ANSEL  = 0;
     ANSELH = 0;

     // Port A en entrée
     TRISA=0xFF;

     // Configuration en sortie des ports
     TRISB=0;
     TRISD=0;

     // Init des ports
     PORTB=0x00;
     PORTD=0x00;
     // Lecture et affichage de la donnée ADC
     while(1)
     {
      // Lecture des entrées & Mise à jour DAC
      if (Button(&PORTA, 0, 1, 1))
      {
         // Mise à jour  de nombres des bits & du générateur
         num_bit=num_bit+1;
         if (num_bit>8)
         {
            num_bit=0;
            srand(rand_val); // Réinitialiser le génerateur
         }
         // Affichage de la valeur (LEDS)
```

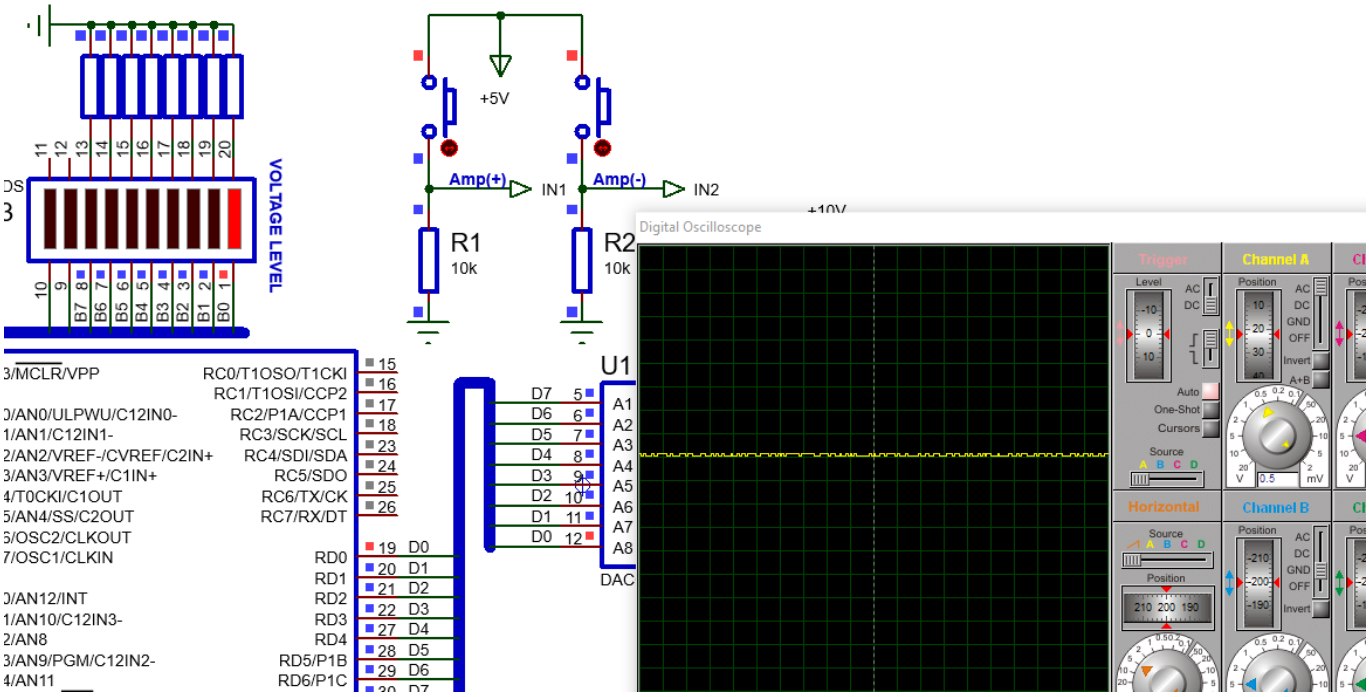
```
        PORTB=Code1_2_Code2(num_bit);
        // Tempo
        Delay_ms(200);
    }
    if (Button(&PORTA, 1, 1, 1))
    {
        num_bit=num_bit-1;
        if (num_bit>250)
        {
            num_bit=8;
            srand(rand_val);
        }
        PORTB=Code1_2_Code2(num_bit);
        Delay_ms(200);
    }

    // Lecture du générateur
    rand_val=rand();
    // Mise en forme de la donnée: 16 bits => 8 bits
    rand_val= (rand_val & 0x00FF) |  (rand_val>>8 & 0x00FF);
    // Mise à jour de l'amplitude
    rand_val =rand_val>>(8-num_bit);
    // Envoi de la valeur au DAC
    PORTD=rand_val;
}
```

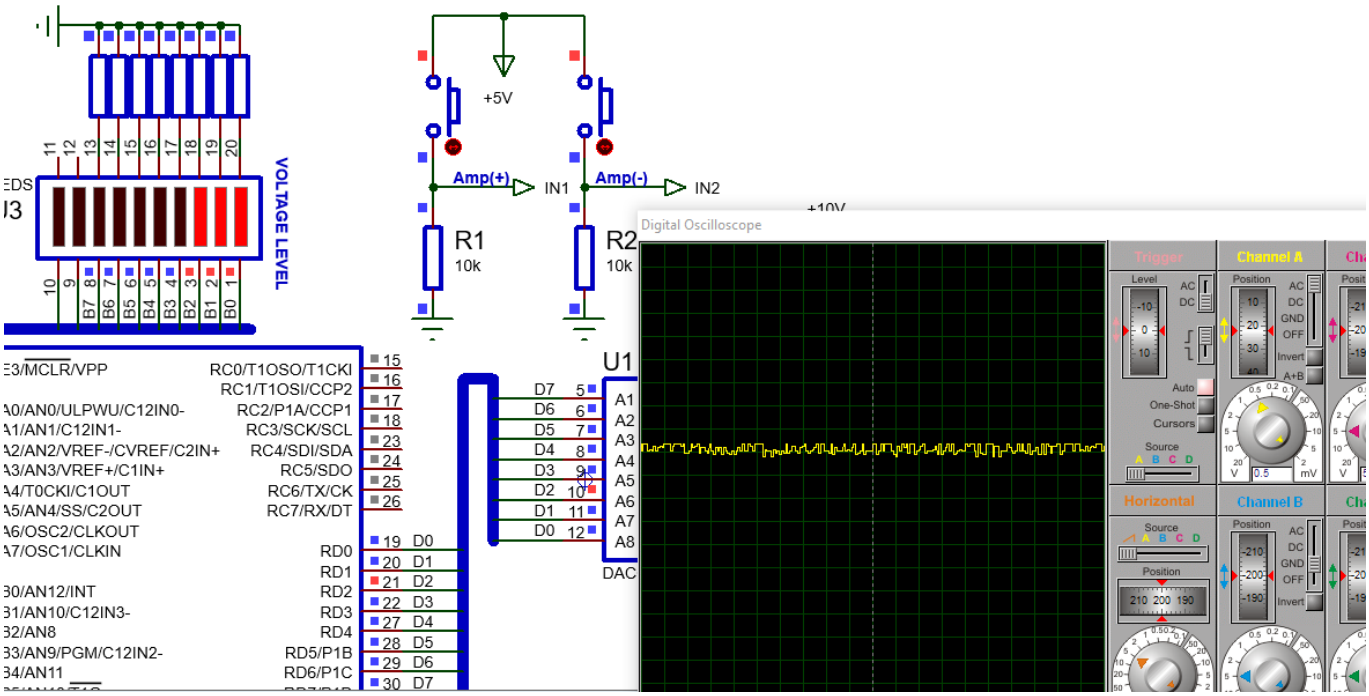
```
}
```

Simulations

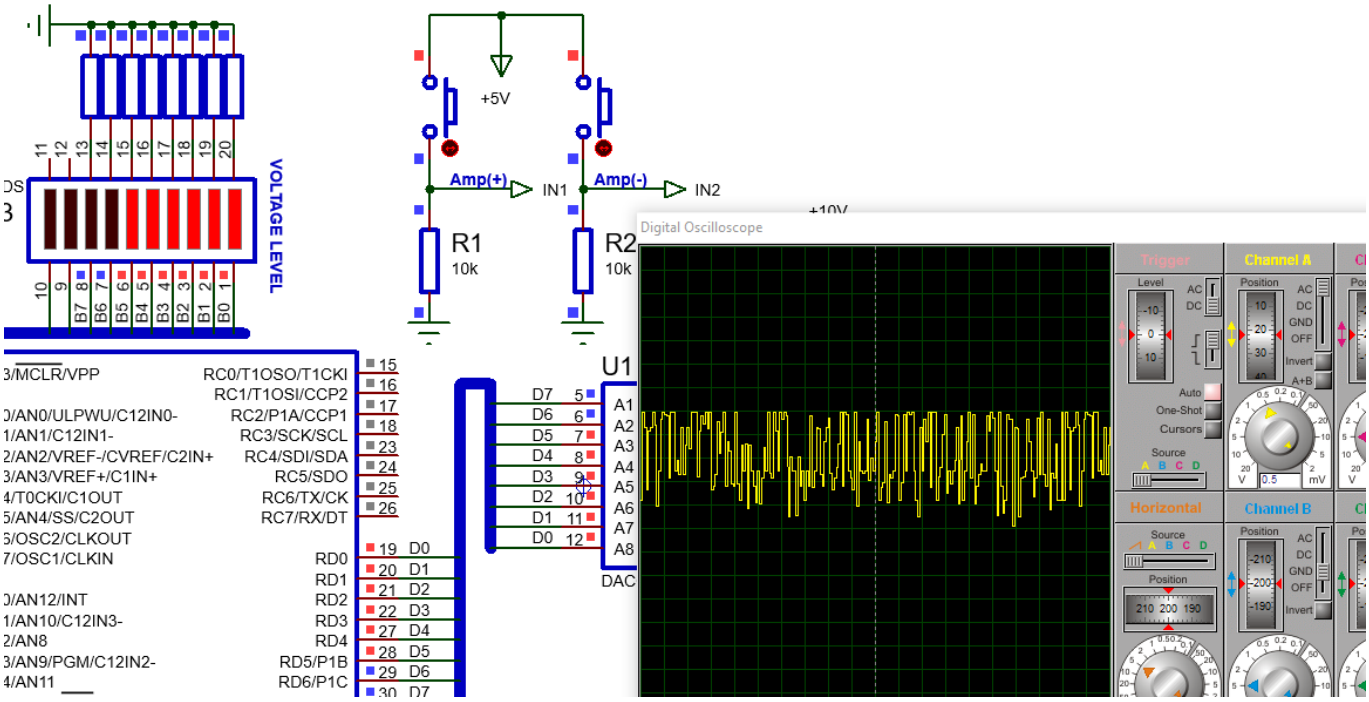
- **Amplitude “0x01”**



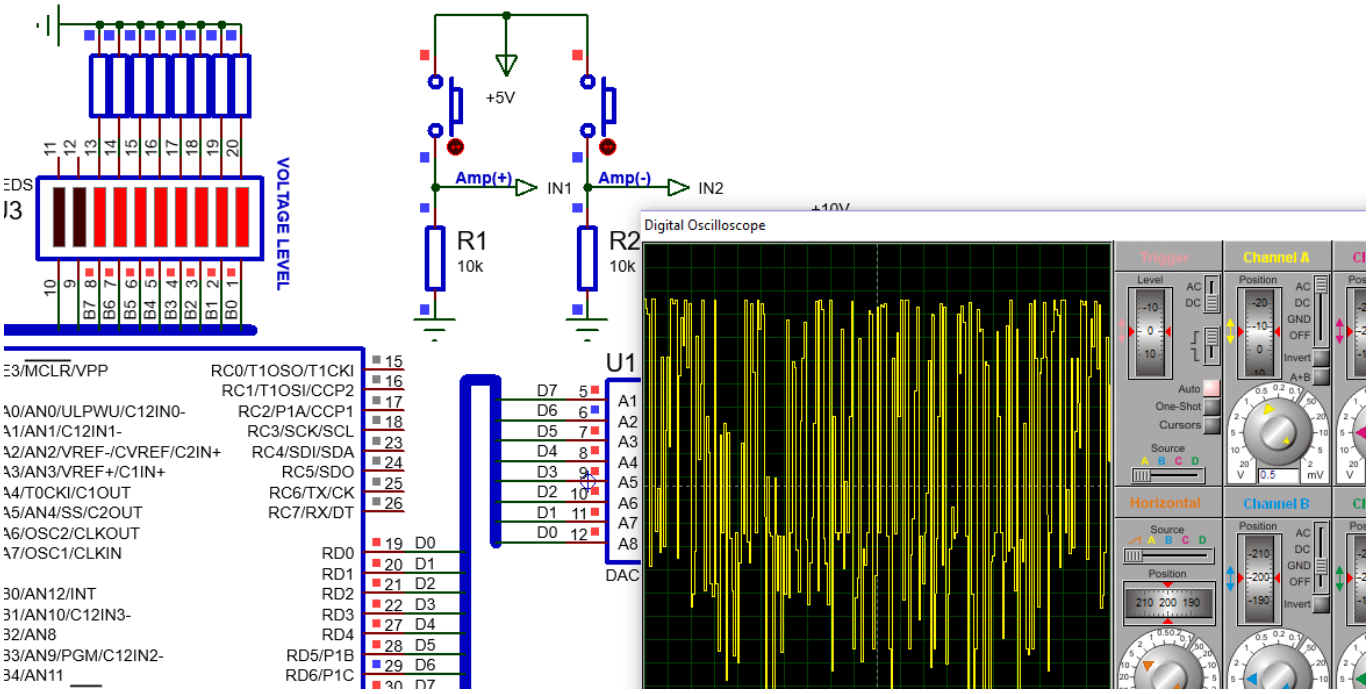
- **Amplitude “0x03”**



- **Amplitude “0x3F”**



- **Amplitude “0xFF”**



Téléchargement

- MikroC: Générateur du bruit
- ISIS: Générateur du bruit

Retour à l’accueil MikroC