Montage 18

Numérique appelle analogique

Au sommaire :

- la fabrication d'un shield générateur de signaux analogiques ;
- comprendre ce qu'est un convertisseur numérique-analogique ;
- savoir ce qu'est un réseau de résistances R2R ;
- savoir ce qu'est un registre de port ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

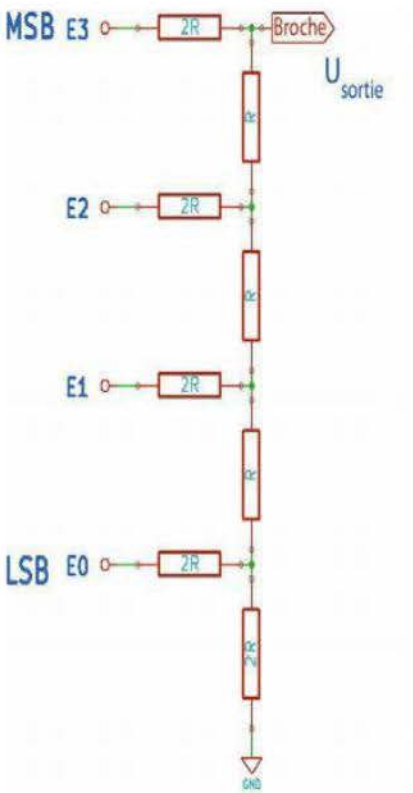
Comment convertir des signaux numériques en signaux analogiques ?

L'exploitation de signaux analogiques est relativement simple par les entrées analogiques avec votre carte Arduino. Le sens inverse – donc produire et distribuer une tension analogique à l'aide du microcontrôleur – n'est faisable qu'en utilisant les sorties numériques capables de MLI. Quand vous aurez vu la forme de la courbe des signaux MLI, vous saurez combien elle diffère de celle d'un signal analogique. La plupart des microcontrôleurs courants ne convertissent pas un signal numérique en signal analogique. Il leur faudrait pour ce faire intégrer un convertisseur N/A.

Notre montage consiste à fabriquer un tel convertisseur, appelé aussi CNA (*Digital-Analog-Converter* en anglais ou DAC), avec des moyens simples. Tout tourne ici autour du réseau R2R. Cette appella-

tion est due au fait que le convertisseur est composé de plusieurs résistances, disposées en cascade et qui doivent se trouver entre elles dans un rapport déterminé. La disposition des éléments fait penser à une échelle, c'est pourquoi ce type de circuit est également nommé réseau en échelle de résistances dans la littérature spécialisée. Retenons seulement que le réseau de résistances sert à répartir une tension de référence qui, dans notre cas, est de +5 V. La figure 18-1 montre un réseau de résistances R2R avec une entrée de 6 bits.

Figure 18-1 ▶
Réseau de résistance R2R avec une entrée de 6 bits



Vous vous demandez peut-être d'où vient ce nom R2R. Si vous regardez le schéma de plus près, vous verrez que les résistances n'ont pas une valeur fixée, et que seuls les rapports de résistance sont indiqués. Les valeurs des résistances (horizontales), qui sont reliées aux connexions E_0 à E_5 des sorties numériques, sont le double de celles des résistances (verticales), qui relient les résistances précédentes et mènent au point de sortie U_{sortie}. La résistance du bas, qui est reliée à la masse, a la même résistance 2R que les résistances horizontales. La formule suivante peut être utilisée pour déterminer la tension de sortie :

$$U_{sortie} = \frac{U_{e5}}{2} + \frac{U_{e4}}{4} + \frac{U_{e3}}{8} + \frac{U_{e2}}{16} + \frac{U_{e1}}{32} + \frac{U_{e0}}{64}$$

Pour cet exemple avec cinq entrées, la résolution suivante peut être obtenue :

$$U_{résolution} = \frac{U_{ref}}{64}$$

U_{ref} est ici la tension avec laquelle les différentes entrées sont commandées. Pour une tension U_{ref} de 5 V, le résultat serait donc le suivant :

$$U_{résolution} = \frac{U_{ref}}{64} = \frac{5\ \text{V}}{64} = 78{,}13\ \text{mV}$$

Cette valeur représente le plus petit pas de progression obtenu chaque fois que la valeur binaire de l'entrée à 6 bits est incrémentée de 1. Le tableau 18-1 fournit les quatre premières valeurs ainsi que la dernière.

Tableau 18-1
Combinaisons binaires et tensions de sortie arrondies

Valeur binaire	Tension de sortie
000000	0 V
000001	78,13 mV
000010	156,26 mV
000011	234,39 mV
...	...
111111	5 V

Nous avons donc, pour le shield de conversion N/A prévu, une définition de 6 bits ($2^6 = 64$).

Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- réseau R2R ;
- réseau de résistances en échelle.

Vous avez peut-être remarqué que je n'ai donné jusqu'ici aucune valeur de résistance. Ce n'est pas utile tant que le rapport des résistances est exactement de 2:1. En outre, la tolérance des différentes résistances doit être aussi faible que possible pour obtenir des résultats relativement précis. Nous n'en tiendrons cependant pas compte dans ce montage.

Composants nécessaires

17 résistances de 47 kΩ

1 jeu de connecteurs femelles empilables (2 × 8 + 2 × 6)

1 carte de shield

Fils, si possible de différentes couleurs

Réflexions préliminaires

Le réseau R2R avec rapports de résistance 2:1 peut s'avérer difficile à réaliser car vous devez trouver des valeurs de résistance qui sont dans ce rapport entre elles. La solution n'est certes pas simple. J'ai choisi une résistance de 47 kΩ pour que les courants en circulation ne soient pas trop élevés.

Vous vous demandez peut-être si une résistance de 23,5 kΩ existe. Non seulement je ne crois pas, mais cette valeur est en plus très facile à obtenir. Quand on branche deux résistances de même valeur en parallèle, le résultat obtenu est l'exacte moitié de la résistance en question. Donc si $R_1 = R_2$, on obtient l'équation suivante.

$$\frac{1}{R_{totale}} = \frac{1}{R_1} + \frac{1}{R_2} = \frac{1}{R} + \frac{1}{R} = \frac{2}{R}$$

$$\text{donc } R_{totale} = \frac{R}{2}$$

C'est élémentaire, n'est-ce pas ?

Code du sketch

```
int pinArray[] = {8, 9, 10, 11, 12, 13};
byte R2RPattern;
void setup(){
  for(int i = 0; i < 6; i++)
    pinMode(pinArray[i], OUTPUT);
  R2RPattern = B000001; //Configuration binaire pour commander
                        //les sorties numériques
}

void loop(){
  for(int i = 0; i < 6; i++){
    digitalWrite(pinArray[i], bitRead(R2RPattern,i) == 1?HIGH:LOW);
   }
}
```

Ce sketch, vraiment très court, commande les sorties numériques sur lesquelles se trouve le réseau R2R. Elles sont commandées via la variable `R2RPattern`, qui délivre une tension correspondante à la sortie du réseau.

Revue de code

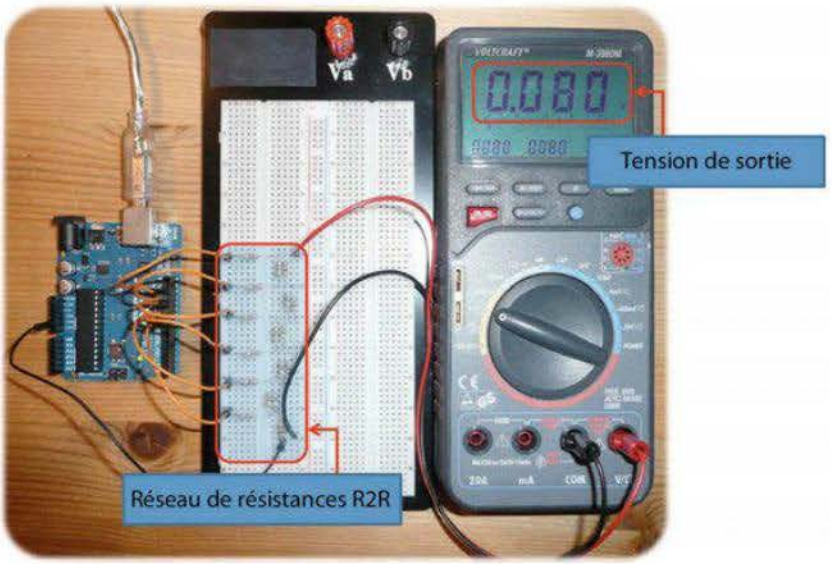
Du point de vue logiciel, les variables indiquées dans le tableau suivant sont nécessaires à notre montage.

Tableau 18-2
Variables nécessaires et leur objet

Variable	Objet
`pinArray`	Tableau unidimensionnel pour stocker les broches connectées à l'afficheur
`R2Rpattern`	Contient la combinaison de bits utilisée pour commander le réseau R2R

La figure 18-2 illustre le circuit que j'ai créé sur une plaque d'essais avant de le reporter sur le shield R2R.

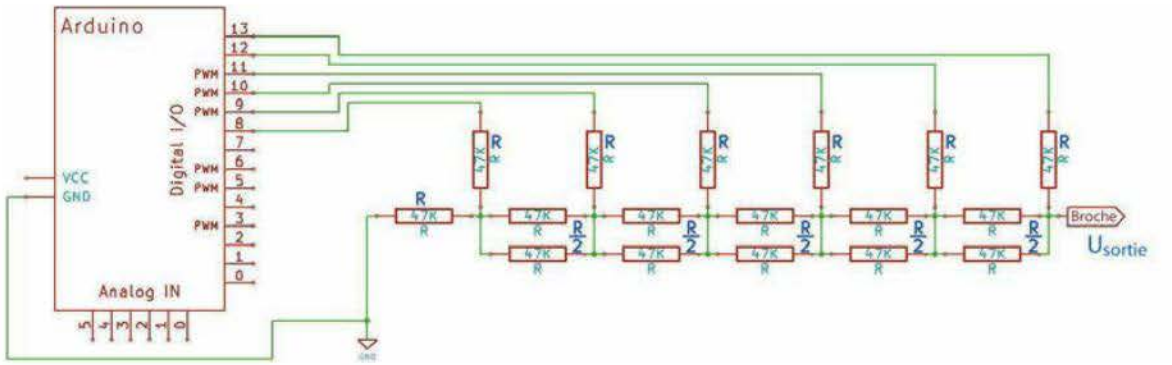
Figure 18-2 ▶
Réseau R2R sur une plaque d'essais (tension de sortie pour une combinaison binaire de 000001)



Le réseau est commandé avec la combinaison de bits 000001 tirée du sketch et le multimètre affiche une tension de 0,080 V, soit 80 mV. Dans le tableau 18-1, la valeur est de 78,13 mV pour la combinaison de bits. La valeur de sortie de 80 mV ne correspond donc pas tout à fait à la valeur calculée du tableau, mais c'est tout de même correct quand on sait que le résultat se trouve par exemple légèrement faussé par les tolérances matérielles des résistances utilisées ou par des erreurs d'affichage du multimètre. Il m'est arrivé de construire un réseau R2R où les valeurs coïncidaient presque toutes jusqu'à deux chiffres après la virgule, mais ce n'était que pur hasard.

Schéma

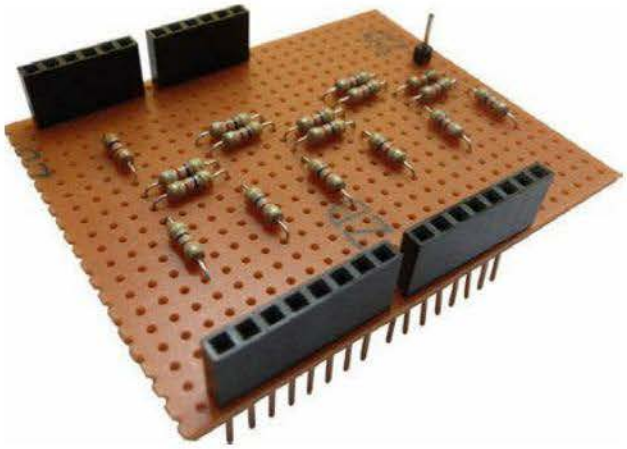
Comme vous pouvez le voir dans la figure suivante, le circuit est uniquement composé de résistances reliées d'une manière particulière pour constituer un réseau R2R.



Les résistances R ont naturellement une valeur de 47 kΩ. Les paires de résistances R/2 ont comme valeur résultante 23,5 kΩ.

▲ **Figure 18-3**
Commande du réseau R2R par 6 sorties numériques

Réalisation du shield



◀ **Figure 18-4**
Réalisation du réseau R2R sur un shield dédié

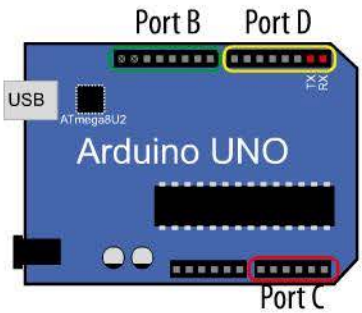
La figure montre bien le réseau de résistances, la broche en haut du shield étant la sortie sur laquelle vous pouvez brancher votre multimètre pour mesurer la tension de sortie.

Commande du registre de port

Je ne vous apprendrai rien en vous disant que la carte Arduino ne communique que par les entrées et les sorties. Ceci vaut également pour commander des LED, des moteurs, des servomoteurs et pour lire, entre autres, les valeurs d'un capteur de température ou d'une résistance réglable ou photosensible.

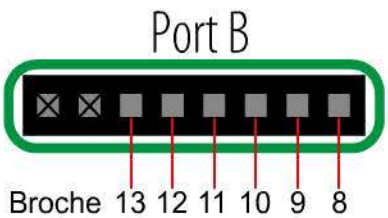
Votre microcontrôleur ATmega328 travaille en interne avec ce qu'on appelle des registres, raccordés aux entrées et sorties (broches). Dans le domaine informatique, ce sont des zones de mémoire à l'intérieur d'un processeur, qui sont reliées directement à l'unité centrale de calcul. Ainsi l'accès à ces zones est très rapide puisque le détour par des circuits mémoire externes est évité. Les différentes broches de votre carte Arduino sont reliées en interne à des registres de port, encartouchés en couleurs (vert, rouge ou jaune) et nommés port B, C ou D dans la figure 18-5.

Figure 18-5 ▶
Registres de port de la carte Arduino



Regardons par exemple le port B de plus près (figure 18-6).

Figure 18-6 ▶
Registre de port B



On reconnaît immédiatement les entrées et sorties numériques (broches 9 à 13). Les deux broches de gauche sont pour nous sans intérêt, car elles sont reliées à *Aref* (entrée pour la tension de référence du convertisseur analogique-numérique) et à la masse et ne peuvent être manipulées. Six bits en tout sont donc disponibles dans le registre de port B. Ils vont nous servir à faire des choses diverses. Comme par hasard, notre réseau de résistances est lui aussi commandé avec 6 bits. Je ne vous en dis pas plus pour l'instant. Chacun des trois ports est sollicité dans un sketch au moyen des identifiants suivants :

- `PORTB` ;
- `PORTC` ;
- `PORTD`.

Nous savons déjà comment les différents ports sont sollicités mais nous nous en tiendrons, comme je l'ai dit plus haut, au port B dans notre exemple.

J'ai d'emblée une question à poser. Quand on programme des broches numériques, on doit définir dans la fonction `setup` si elles vont servir d'entrée ou de sortie. Dans le cas d'un registre de port, comment dois-je lui dire de servir d'entrée ou de sortie ?

Et bien Ardus, vous m'offrez là une transition rêvée pour passer au point suivant. Mais je dois vous dire quelque chose avant : vous pouvez bien sûr attribuer à chaque bit du registre de port un sens de circulation des données particulier. Le registre complet ne fonctionne pas de telle sorte que toutes les broches servent d'entrées ou de sorties. Chaque broche peut être configurée séparément. D'autres registres sont en effet spécialement prévus pour influer sur le sens de circulation des données de chaque broche. Ils ont pour nom DDRx, x indiquant le port à solliciter. Le registre DDRB est par conséquent celui de notre `PORTB` – DDR, pour *Data Direction Register*, signifie à peu de choses près registre de direction de données. Voyons comment tout cela fonctionne dans le détail. Avant d'utiliser un port, je dois donc définir le sens de circulation des données par le DDR correspondant. Dans la figure 18-7, les flèches indiquent les sens de circulation des données que nous voulons obtenir avec notre programmation.

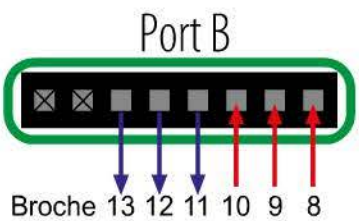


Figure 18-7
Registre de port B avec divers sens de circulation des données selon les broches

La configuration est donc la suivante :

- entrées : broches 8, 9 et 10 ;
- sorties : broches 11, 12 et 13.

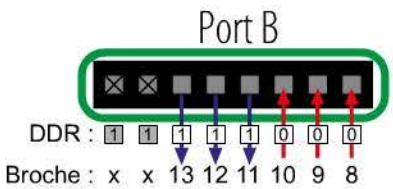
Pour affecter un sens de circulation des données à une broche, la valeur suivante doit être entrée dans le DDR.

Tableau 18-3 ▶ Valeurs pour le DDR

Valeur	Mode de fonctionnement
0	Broche servant d'entrée, comparable à `pinMode(pin, INPUT);`
1	Broche servant de sortie, comparable à `pinMode(pin, OUTPUT);`

Cela nous donne pour le DDR la programmation de la figure 18-8.

Figure 18-8 ▶ Initialisation du DDR pour les différents sens de circulation des données



Nous pouvons maintenant mettre par exemple les sorties numériques des broches 11, 12 et 13 au niveau `HIGH` par l'instruction `PORTB`. Voici un extrait correspondant tiré d'un sketch :

```
void setup(){
  DDRB = 0b11111000;      //Broches 8, 9, 10 = INPUT. Broches 11,
                          //12, 13 = OUTPUT
  PORTB = 0b00111000;     //Mise des broches 11, 12, 13 au niveau HIGH
}

void loop(){/ * vide * /}
```

Les deux bits les plus significatifs pour les broches non utilisées ont simplement été pourvus d'un 1 dans le DDR. Cela n'a aucune importance dans notre cas. Si vous regardez la mise des sorties au niveau `HIGH`, que constatez-vous de différent par rapport à la manipulation des broches habituelle ? Je vous donne les deux variantes pour comparaison :

```
digitalWrite(11, HIGH);          PORTB = 0b00111000;
digitalWrite(12, HIGH);
digitalWrite(13, HIGH);
```

Aucune idée ? Bon. La manière traditionnelle de gauche met les différentes broches l'une après l'autre au niveau `HIGH`. Celle de droite met par contre toutes les broches *en même temps* au niveau `HIGH` avec une seule instruction, la configuration binaire étant alors appliquée simultanément à toutes les broches.

Mieux vaut donc choisir la nouvelle variante de manipulation des ports pour aller plus vite. Le sketch suivant génère une tension en dents de scie à la sortie du réseau de résistances :

```
void setup(){
  DDRB = 0b11111111;      //Toutes les broches programmées
                          //en tant que sorties
}

void loop(){
  for(int i = 0; i <= 63; i++)  //63 = B00111111
    PORTB = i;                  //Commande du registre de port B
}
```

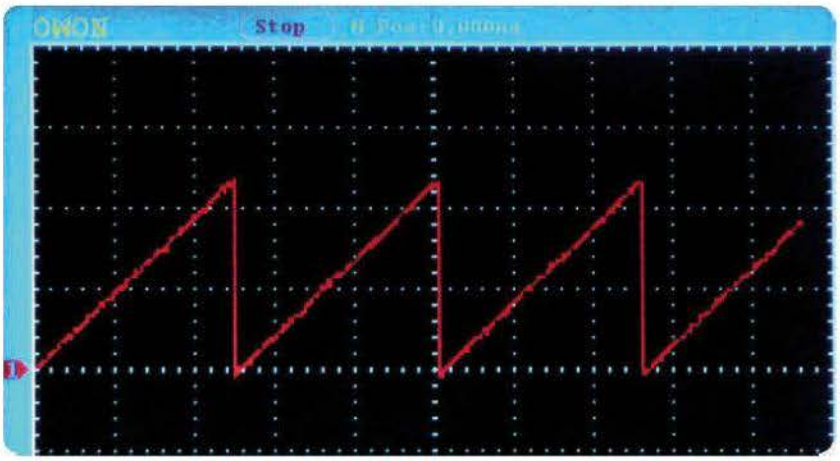


Figure 18-9
Oscillogramme avec une courbe en dents de scie

D'après vous, comment adapter le sketch pour obtenir la courbe suivante ?

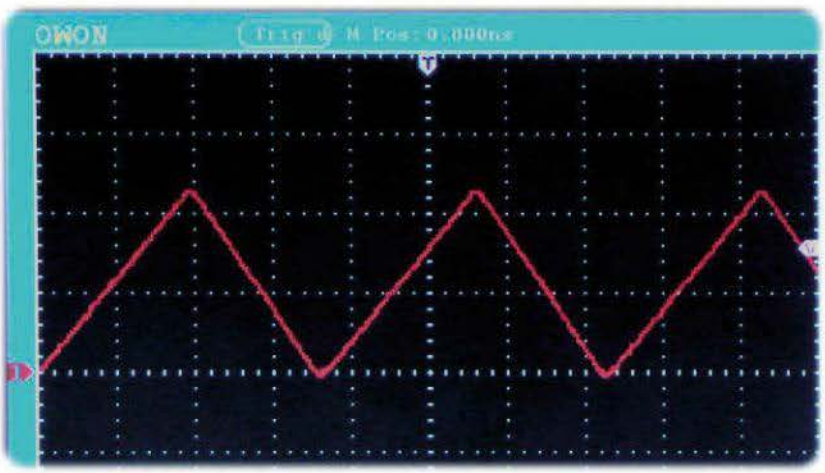


Figure 18-10
Oscillogramme avec une courbe en triangles

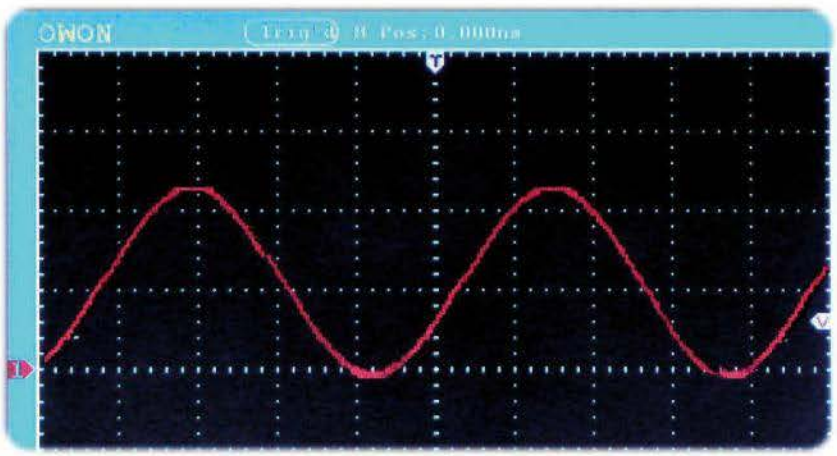
Bien joué si vous avez trouvé la solution.

```
void loop(){
  for(int i = 0; i <= 63; i++)
```

```
    PORTB = i;          //Commande du registre de port B
                        //(front montant)
  for(int i = 63; i >= 0; i--)
    PORTB = i;          //Commande du registre de port B
                        //(front descendant)
}
```

Quelles sont les autres courbes ? Qu'en est-il d'une courbe sinusoïdale ? La fonction sinus ayant besoin d'un certain temps pour calculer les valeurs, on a eu l'idée de créer des tables de correspondance ou *Lookup-Tables* (LUT). Les résultats d'un calcul y sont déjà enregistrés. On peut ainsi reproduire la courbe d'une fonction sinus en s'aidant des points qui se trouvent sur la courbe, par exemple.

Figure 18-11 ▶
Oscillogramme avec une courbe sinusoïdale



Le sketch pour générer la courbe sinusoïdale est vraiment laborieux à taper du fait que la LUT est très longue.

```
byte LUT[] =
{31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 38, 38, 39, 39, 40, 40,
 41, 41, 42, 42, 43, 43, 44, 44, 45, 45, 46, 46, 47, 47, 48, 48, 49, 49,
 50, 50, 50, 51, 51, 52, 52, 52, 53, 53, 54, 54, 54, 55, 55, 55, 56, 56,
 56, 57, 57, 57, 58, 58, 58, 59, 59, 59, 59, 60, 60, 60, 60, 60, 61, 61,
 61, 61, 61, 61, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 61, 61, 61,
 61, 61, 61, 60, 60, 60, 60, 60, 59, 59, 59, 59, 58, 58, 58, 57, 57, 57,
 56, 56, 56, 55, 55, 55, 54, 54, 54, 53, 53, 52, 52, 52, 51, 51, 50, 50,
 50, 49, 49, 48, 48, 47, 47, 46, 46, 45, 45, 44, 44, 43, 43, 42, 42, 41,
 41, 40, 40, 39, 39, 38, 38, 37, 36, 36, 35, 35, 34, 34, 33, 33, 32, 32,
 31, 30, 30, 29, 29, 28, 28, 27, 27, 26, 26, 25, 24, 24, 23, 23, 22, 22,
 21, 21, 20, 20, 19, 19, 18, 18, 17, 17, 16, 16, 15, 15, 14, 14, 13, 13,
 12, 12, 12, 11, 11, 10, 10, 10, 9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 6, 5, 5,
 5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
```

```
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
 7, 7, 7, 8, 8, 8, 9, 9, 10, 10, 10, 11, 11, 12, 12, 12, 13, 13, 14, 14,
 15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23,
 24, 24, 25, 26, 26, 27, 27, 28, 28, 29, 29, 30, 30, 31};

void setup(){
  DDBR = 0b11111111;  //Toutes les broches programmées
                      //en tant que sorties
}

void loop(){
  for(int i = 0; i <=360; i++)
    PORTB = LUT[i];   //Commande du registre de port B
}
```

Attention !

Vous courez le risque non négligeable de programmer votre microcontrôleur de telle sorte qu'il ne réagisse plus par la suite. Si vous regardez le port D, vous verrez que les signaux de contrôle RX et TX se trouvent respectivement sur les broches 0 et 1. RX sert à recevoir et TX à envoyer les données. Le sens de circulation des données est donc le suivant : RX = `INPUT` et TX = `OUTPUT`. Si vous modifiez par inadvertance la programmation de ces valeurs via DDRD, vous ne pourrez à coup sûr plus transmettre aucun sketch sur votre carte Arduino. Vous devez par conséquent être sûr de ce que vous faites. Mieux vaut vérifier trois fois votre sketch avant de l'envoyer au microcontrôleur.

Vous trouverez des informations plus précises sur :

http://www.arduino.cc/en/Reference/PortManipulation

Problèmes courants

Si la tension de sortie du réseau R2R ne correspond pas aux valeurs souhaitées pour la combinaison binaire, vérifiez les points suivants.

- Toutes les résistances utilisées pour le réseau R2R ont-elles la même valeur ?
- Aucune connexion au réseau n'a été oubliée ? (Je sais de quoi je parle car j'avais oublié un point de jonction, et j'ai passé près de dix minutes à trouver l'erreur !)

Qu'avez-vous appris ?

- Ce montage vous a présenté un réseau de résistances R2R.
- Avec ce réseau, vous avez pu réaliser un convertisseur numérique/analogique simple.
- Vous avez découvert les registres de port de votre carte Arduino et manipulé les sorties numériques au moyen du port B.

Exercice complémentaire

Essayez, en ajustant le tableau LUT, de créer des courbes de formes différentes. Vérifiez dans tous les cas que seuls 6 bits sont à votre disposition pour représenter une courbe. Le domaine de valeurs va de 0 à 63. Si vous êtes au-dessus, ni le circuit ni votre microcontrôleur n'en souffrira, mais la courbe ne ressemblera à coup sûr pas à ce que vous auriez souhaité.