Montage

11

L'afficheur sept segments

Au sommaire :

- la commande d'un afficheur à sept segments ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Qu'est-ce qu'un afficheur sept segments ?

Pour visualiser des états logiques (vrai ou faux) ou des données (`14`, `2.5`, `"Hello User"`) sous une forme quelconque, il nous faut commander les LED dans un premier temps et revenir au Serial Monitor dans un deuxième temps. Il existe en électronique d'autres éléments d'affichage que les LED, l'afficheur sept segments étant l'un deux. Comme son nom l'indique, cet afficheur se compose de sept segments qui, disposés d'une certaine manière, peuvent représenter des chiffres et, dans une moindre mesure, des signes. La figure 11-1 présente un tel afficheur de manière schématisée.

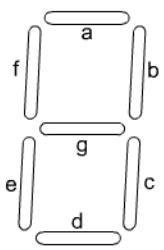


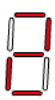
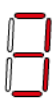
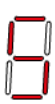
Figure 11-1
Afficheur sept segments

On voit que chaque segment est pourvu d'une petite lettre. L'ordre n'est pas primordial mais la forme montrée ici s'est imposée et a été adoptée pratiquement partout. Aussi l'utiliserons-nous également toujours sous cette forme. Si maintenant nous commandons les différents segments avec habileté, nous pouvons afficher des chiffres allant de 0 à 9. On peut aussi afficher des lettres, nous y reviendrons plus tard. Votre quotidien est sûrement rempli de ces afficheurs sept segments sans que vous n'y ayez jamais prêté attention. Faites un tour en ville et vous verrez à quel point ils sont courants. Voici d'ailleurs une petite liste des possibilités d'utilisation :

- l'affichage des prix sur les stations-service (toujours en hausse hélas !) ;
- l'affichage de l'heure sur certains bâtiments ;
- l'affichage de la température ;
- les montres numériques ;
- les tensiomètres médicaux ;
- les thermomètres numériques.

Le tableau 11-1 indique une fois pour toutes, en vue de la programmation, quels sont les segments à allumer pour chacun des chiffres.

Tableau 11-1 ▶
Commande des sept segments

Afficheur	a	b	c	d	e	f	g
	1	1	1	1	1	1	0
	0	1	1	0	0	0	0
	1	1	0	1	1	0	1
	1	1	1	1	0	0	1
	0	1	1	0	0	1	1
	1	0	1	1	0	1	1

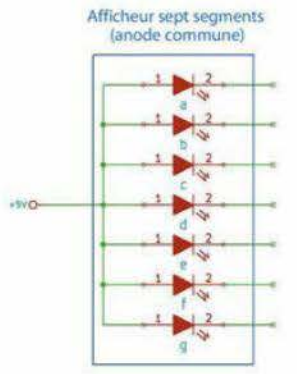
Afficheur	a	b	c	d	e	f	g
	1	0	1	1	1	1	1
	1	1	1	0	0	0	0
	1	1	1	1	1	1	1
	1	1	1	1	0	1	1

◀ Tableau 11-1 (suite)
Commande des sept segments

Le chiffre `1` dans ce tableau ne signifie pas forcément niveau `HIGH`, mais c'est la commande de l'allumage du segment concerné. Celle-ci peut se faire soit avec le niveau `HIGH` que nous connaissons (+5 V résistance série incluse), soit avec un niveau `LOW` (0 V). Vous voulez peut-être savoir maintenant en fonction de quoi on choisit une commande. Cela dépend en fait du type de l'afficheur sept segments. Deux approches sont possibles :

- la cathode commune ;
- l'anode commune.

En cas de cathode commune, toutes les cathodes des diverses LED de l'afficheur sept segments sont réunies en interne et reliées à la masse à l'extérieur. Les différents segments sont commandés par des résistances série dûment raccordées au niveau `HIGH`. Notre exemple porte cependant sur un afficheur sept segments avec anode commune. Ici, c'est exactement le contraire : toutes les anodes des diverses LED sont reliées entre elles en interne et raccordées au niveau `LOW` à l'extérieur. Les segments sont commandés par des résistances série correctement dimensionnées, en passant par les différentes cathodes des LED qui sont accessibles à l'extérieur.



Dans le circuit pour afficheur sept segments avec anode commune de gauche, toutes les anodes des diverses LED en service sont reliées à la tension d'alimentation +5 V. Les cathodes sont reliées par la suite au sorties numériques de votre carte Arduino et pourvues des différents niveaux de tension conformes au tableau de commande. Nous utilisons pour notre essai un afficheur sept segments avec anode commune de type SA 39-11 GE. La figure suivante illustre le brochage de cet afficheur.

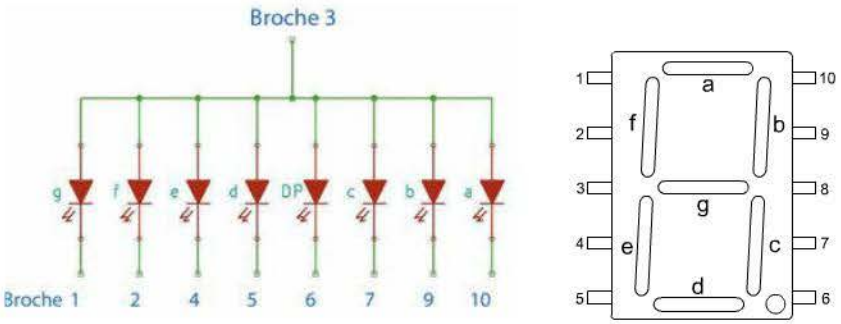
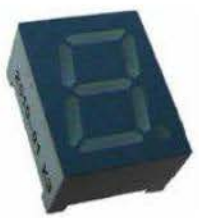


Figure 11-2 ▶
Commande de l'afficheur sept segments de type SA 39-11 GE

Le graphique de gauche montre les broches utilisées de l'afficheur sept segments, et le graphique de droite le brochage du type utilisé. DP est la forme abrégée de point décimal.

Composants nécessaires

	1 afficheur sept segments (par exemple de type SA 39-11 GE avec anode commune)
	7 résistances de 330 Ω
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

```
int segments[10][7] = {{1, 1, 1, 1, 1, 1, 0}, //0
                       {0, 1, 1, 0, 0, 0, 0}, //1
                       {1, 1, 0, 1, 1, 0, 1}, //2
                       {1, 1, 1, 1, 0, 0, 1}, //3
```

```
                        {0, 1, 1, 0, 0, 1, 1}, //4
                        {1, 0, 1, 1, 0, 1, 1}, //5
                        {1, 0, 1, 1, 1, 1, 1}, //6
                        {1, 1, 1, 0, 0, 0, 0}, //7
                        {1, 1, 1, 1, 1, 1, 1}, //8
                        {1, 1, 1, 1, 0, 1, 1}} ; //9
int pinArray[] = {2, 3, 4, 5, 6, 7, 8} ;

void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pinArray[i], OUTPUT);
}
void loop(){
  for(int i = 0; i < 10; i++){
    for(int j = 0; j < 7; j++)
      digitalWrite(pinArray[j], (segments[i][j]==1)?LOW:HIGH);
    delay(1000);  //Pause de 1 seconde
    }
}
```

Revue de code

Du point de vue logiciel, les variables suivantes sont nécessaires à notre programmation expérimentale.

Tableau 11-2
Variables nécessaires et leur objet

Variable	Objet
`segments`	Tableau bidimensionnel pour stocker l'information des segments pour chaque chiffre
`pinArray`	Tableau unidimensionnel pour stocker les broches connectées à l'afficheur

Un tableau bidimensionnel s'impose d'emblée pour stocker les informations sur les segments à allumer pour chaque chiffre de `0` à `9`. Ces valeurs sont définies dans la variable globale segmente en début de sketch :

```
int segments[10][7] = {{…},
                       …
                       {…}};
```

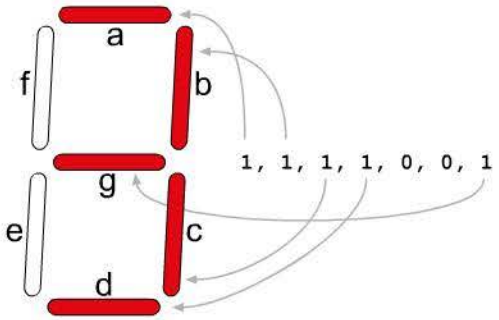
Le tableau comprend 10×7 cases mémoire, le contenu de chacune d'elles pouvant être obtenu par les coordonnées :

```
segments[x][y]
```

La coordonnée `x` sert pour tous les chiffres de `0` à `9` (soit 10 cases mémoire), et la coordonnée `y` pour tous les segments de `a` à `g` (soit 7 cases mémoire). On détermine par exemple les segments à allumer du chiffre `3` en écrivant la ligne :

`segments[3][y]`

les résultats pour la variable *y* allant de `0` à `6` étant obtenus par une boucle `for`. Les données des segments sont alors celle de la figure suivante.



Minute, s'il vous plaît ! Vous avez dit que ce type d'afficheur sept segments disposait d'une anode commune. Pourtant, il y a un 1 là où il devrait y avoir une mise à la masse dans le tableau des segments. Ce n'est donc pas le cas alors !

Je confirme la première partie de ce que vous venez de dire. Mais pour la deuxième, vous n'avez sûrement pas été totalement attentif. J'ai dit qu'un `1` ne voulait pas forcément dire niveau `HIGH`, mais simplement que le segment en question devait être allumé. Dans le cas d'un afficheur sept segments à cathode commune, on commande l'allumage du segment souhaité avec le niveau `HIGH`, tandis que dans le cas d'un afficheur sept segments à anode commune, on le commande avec le niveau `LOW`. On écrit ainsi la ligne suivante :

```
digitalWrite(pinArray[j], (segments[i][j]==1) ?LOW:HIGH);
```

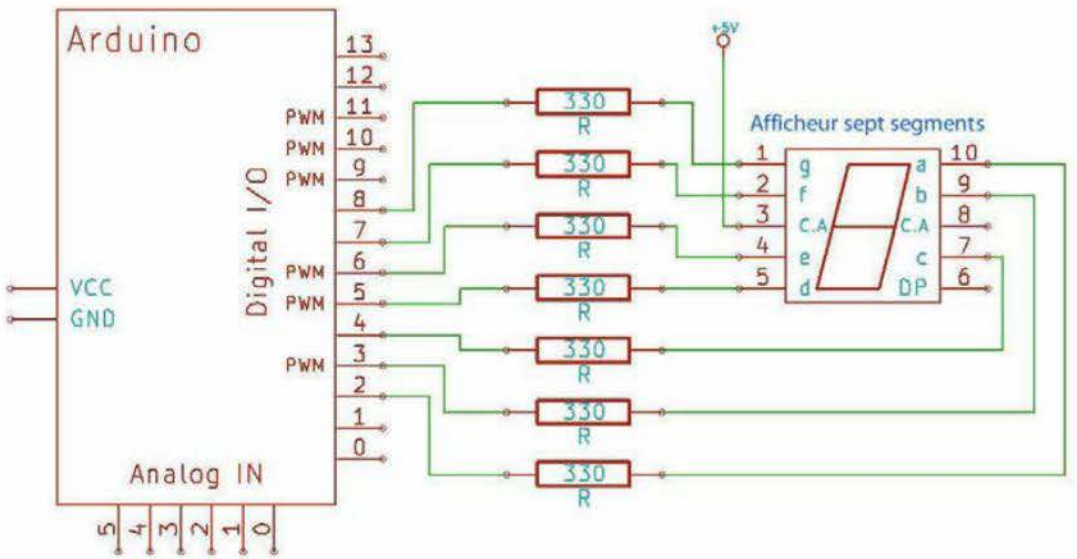
Si l'information est un `1`, `LOW` est alors transmis comme argument à la fonction `digitalWrite`. Sinon, c'est `HIGH`. Le segment correspondant s'allume si c'est `LOW`, et se voit géré de manière à rester éteint si c'est `HIGH`. Notre sketch affiche tous les chiffres de `0` à `9` au rythme d'une seconde. Le code suivant est utilisé pour ce faire :

```
for(int i = 0; i < 10; i++){
  for(int j = 0; j < 7; j++)
    digitalWrite(pinArray[j], (segments[i][j]==1)?LOW:HIGH);
  delay(1000); //Pause de 1 seconde
}
```

La boucle extérieure avec la variable de contrôle `i` sélectionne dans le tableau le chiffre à afficher tandis que la boucle intérieure avec la variable `j` sélectionne les segments à allumer.

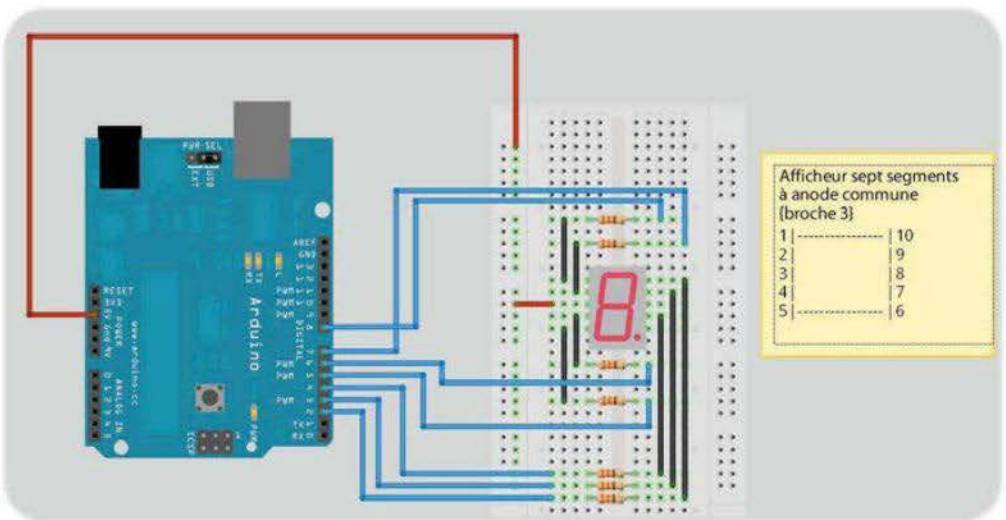
Schéma

Le circuit ressemble à celui du séquenceur de lumière. Mais pas si vite, il va se compliquer.



▼ Figure 11-3
Commande de l'afficheur sept segments

Réalisation du circuit



▼ Figure 11-4
Réalisation du circuit de l'afficheur sept segments avec Fritzing

Sketch amélioré

Les divers segments d'un chiffre étaient commandés jusqu'ici au moyen d'un tableau bidimensionnel, la première dimension servant à sélectionner le chiffre désiré, et la deuxième les différents segments. Le sketch suivant va nous permettre de tout faire avec un tableau unidimensionnel. Comment ? C'est simple puisque bits et octets n'ont déjà plus de secret pour vous. L'information de segment doit maintenant tenir dans une seule valeur. Quel type de donnée s'impose ici ? Nous avons affaire à un afficheur sept segments, et à un point décimal que nous laisserons de côté pour l'instant. Cela fait donc 7 bits, qui tiennent idéalement dans un seul octet de 8 bits. Chaque bit est simplement affecté à un segment et tous les segments nécessaires peuvent être commandés avec un seul octet. J'en profite pour vous montrer comment initialiser directement une variable par le biais d'une combinaison de bits :

```
void setup(){
  Serial.begin(9600);
  byte a = B10001011;    //Déclarer + initialiser la variable
  Serial.println(a, BIN)  //Imprimer en tant que valeur binaire
  Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale
  Serial.println(a, DEC); //Imprimer en tant que valeur décimale
}

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

La ligne décisive est bien sûr la suivante :

```
byte a = B10001011;
```

Ce qui est remarquable pour ne pas dire génial là-dedans, c'est le fait que le préfixe `B` permet de représenter une combinaison de bits qui sera affectée à la variable située à gauche du signe `=`. Cela simplifie les choses quand par exemple vous connaissez une combinaison de bits et souhaitez la sauvegarder. Il vous faudrait sinon convertir la valeur binaire en valeur décimale avant de sauvegarder. Cette étape intermédiaire n'est ici plus nécessaire.

Je ne comprends pas bien. Le type de donnée **byte** est bien – du moins, il me semble – un nombre entier. Type de donnée et nombres entiers sont bien composés de chiffres allant de *0* à *9*. Pourquoi maintenant peut-on commencer par la lettre **B** et la faire suivre d'une combinaison de bits ? Ou s'agit-il d'une chaîne de caractères ?

Le type de donnée `byte` est un type de nombre entier. Vous avez raison sur ce point. Là où vous avez tort, c'est sur le fait qu'il pourrait s'agir d'une chaîne de caractères. Celle-ci serait alors entre guillemets. Il s'agit en fait de tout autre chose. Aucune idée ? Je ne dirai qu'un mot : `#define`. Ça vous dit quelque chose ? Voyez plutôt. Il existe dans les tréfonds d'Arduino un fichier nommé `binary.h` qui se trouve dans le répertoire : `arduino-1.x.y\hardware\arduino\cores\arduino`.

Voici un court extrait de ce fichier, dont les nombreuses lignes n'ont pas toutes besoin d'être montrées.

```
#ifndef Binary_h
#define Binary_h

#define B0 0
#define B00 0
#define B000 0
#define B0000 0
#define B00000 0
#define B000000 0
#define B0000000 0
#define B00000000 0
#define B1 1
#define B01 1
#define B001 1
#define B0001 1
#define B00001 1
#define B000001 1
#define B0000001 1
#define B00000001 1
```

Ce fichier contient toutes les combinaisons de bits possibles pour les valeurs de 0 à 255, qui y sont définies en tant que constantes symboliques. Je me suis permis de retirer la ligne pour la valeur 139 (déconseillé, à moins de restaurer ensuite l'état initial !) pour voir comment le compilateur réagit. Voyez plutôt :

```
void setup(){
  Serial.begin(9600);
  byte a = B10001011; //Déclarer + initialiser la variable
  Serial.println(a, BIN); //Imprimer en tant que valeur binaire
  Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale
  Serial.println(a, DEC); //Imprimer en tant que valeur décimale
}
void loop ( ){/*Vide*/}
```

```
'B10001011' was not declared in this scope

DiceGroupwithLibrary.cpp: In function 'void setup()':
DiceGroupwithLibrary.pde:-1: error: 'B10001011' was not
declared in this scope
```

Le message d'erreur indique que le nom `B10001011` n'a pas été trouvé. Il me faut encore vous expliquer les lignes suivantes avant d'en revenir au projet :

```
Serial.println(a, BIN); //Imprimer en tant que valeur binaire
Serial.println(a, HEX); //Imprimer en tant que valeur hexadécimale
Serial.println(a, DEC); //Imprimer en tant que valeur décimale
```

La fonction `println` peut accueillir, en plus de la valeur à imprimer, un autre argument qui peut être indiqué séparé par une virgule. Je vous ai mis ici les trois plus importants. Vous en trouverez d'autres sur la page de référence des instructions Arduino sur Internet. Des explications parlantes figurent sous forme de commentaires derrière les lignes d'instructions. L'impression dans le Serial Monitor est alors la suivante :

```
10001011
8B
139
```

Passons maintenant à la commande de l'afficheur sept segments au moyen du tableau bidimensionnel. Voici auparavant le sketch complet que nous allons analyser :

```
byte segments[10] = {B01111110,   //0
                     B00110000, //1
                     B01101101,   //2
                     B01111001,   //3
                     B00110011,   //4
                     B01011011,   //5
                     B01011111,   //6
                     B01110000, //7
                     B01111111,   //8
                     B01111011};  //9
int pinArray[] = {2, 3, 4, 5, 6, 7, 8};

void setup(){
  for(int i = 0; i < 7; i++)
    pinMode(pinArray[i], OUTPUT);
}
void loop(){
  for(int i = 0; i < 10; i++){   //Commande du chiffre
    for(int j = 6; j >= 0; j--){ //Interrogation des bits pour
                                 //les segments
      digitalWrite(pinArray[6-j],bitRead(segments[i],j)==1?LOW:HIGH);
    }
    delay (500); //Attendre une demi-seconde
  }
}
```

Dans la figure 11-5, on voit très bien quel bit est en charge de quel segment au sein de l'octet.

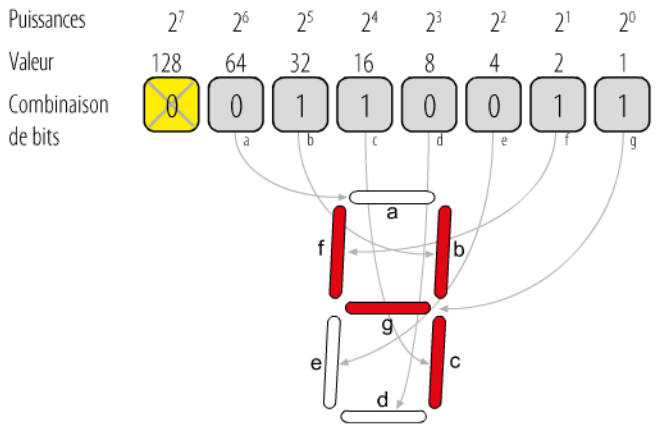


Figure 11-5
Un octet gère les segments de l'afficheur (ici par exemple pour le chiffre 4).

Ayant seulement sept segments à commander et ne tenant pas compte du point décimal, j'ai constamment donné au MSB (rappelez-vous : MSB = bit le plus significatif) la valeur `0` pour tous les éléments du tableau.

Tout se joue bien entendu encore – et comment en serait-il autrement – à l'intérieur de la fonction `loop`. Jetons-y un coup d'œil :

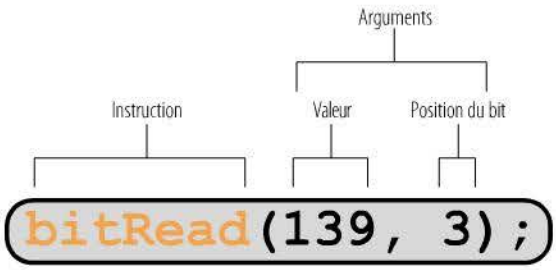
```
void loop(){
  for(int i = 0; i < 10; i++){     //Commande du chiffre
    for(int j = 6; j >= 0; j--){   //Interrogation des bits pour
                                   //les segments
      digitalWrite(pinArray[6-j],bitRead(segments[i],j)==1?LOW:HIGH);
    }
    delay(500);  //Attendre une demi-seconde
  }
}
```

La boucle extérieure `for` avec la variable de contrôle `i` commande encore les divers chiffres de `0` à `9`. C'était déjà le cas dans la première solution. Le code est ensuite différent. La boucle intérieure `for` avec la variable de contrôle `j` est chargée de choisir le bit dans le chiffre sélectionné. Je commence du côté gauche par la position `6`, qui est en charge du segment `a`. Le tableau des broches gérant cependant la broche `8` pour le segment `g` à la position `6` de l'index, la commande doit se faire en sens inverse. On y parvient en soustrayant le nombre `6` puisque j'ai gardé tel quel le tableau des broches du premier exemple :

```
pinArray[6 - j]
```

Voici maintenant à une fonction intéressante, permettant de lire un bit déterminé dans un octet. Elle porte le nom `bitRead`.

Figure 11-6 ▶
Instruction `bitRead`



Cet exemple donne le bit de la position `3` pour la valeur décimale `139` (binaire : `10001011`). Le comptage commence pour l'index `0` au LSB (bit le moins significatif) du côté droit. La valeur renvoyée serait par conséquent un `1`. La ligne :

```
digitalWrite(pinArray[6-j],bitRead(segments[i],j) == 1?LOW:HIGH);
```

permet de vérifier que la lecture du bit sélectionné renvoie bien un `1`. Si c'est le cas, la broche sélectionnée est commandée avec le niveau `LOW`, autrement dit le segment s'allume. N'oubliez pas : anode commune ! Sauriez-vous expliquer la différence entre les deux solutions ?

Laissez-moi réfléchir. Bon ! Dans la première version avec le tableau bidimensionnel, le chiffre à afficher est sélectionné par la première dimension tandis que les segments à commander le sont par la deuxième. Cette information se trouve dans les différents éléments du tableau. Dans la deuxième version, le chiffre à afficher est également sectionné par la première dimension. S'agissant d'un tableau unidimensionnel, elle est cependant la seule dimension. Seulement, l'information pour commander les segments est contenue dans les diverses valeurs de l'octet. Ce qui était fait auparavant par la deuxième dimension est maintenant fait par les bits d'un octet.

Très bien, Ardus ! La technique est comparable.

Problèmes courants

Si l'affichage ne correspond pas aux chiffres 1 à 9 ou si des combinaisons incohérentes s'affichent, vérifiez les choses suivantes.

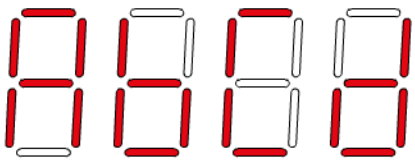
- Vos fiches de raccordement sur la maquette correspondent-elles bien au circuit ?
- Vérifiez qu'il n'y a pas de court-circuit entre elles.
- Le code du sketch est-il correct ?
- Si des caractères incohérents s'affichent, il se peut que vous ayez interverti des lignes de commande. Vérifiez le câblage avec le schéma ou la fiche technique de l'afficheur sept segments.
- Le tableau des segments est-il initialisé avec les bonnes valeurs ?

Qu'avez-vous appris ?

- Dans ce montage, les principes de la commande d'un afficheur sept segments vous sont expliqués.
- L'initialisation d'un tableau vous permet de définir les différents segments de l'affichage pour pouvoir les commander à votre aise par la suite.
- Le fichier d'en-tête `binary.h` contient un grand nombre de constantes symboliques que vous pouvez utiliser dans votre sketch.
- Vous savez comment convertir un nombre à imprimer dans une autre base numérique en ajoutant un deuxième argument (`BIN`, `HEX` ou `DEC`) à la méthode `println`.
- La fonction `bitRead` vous permet de lire l'état de certains bits d'un nombre.

Exercice complémentaire

Élargissez la programmation du sketch de telle sorte que certaines lettres puissent s'afficher à côté des chiffres 0 à 9. Ce n'est certes pas possible pour tout l'alphabet, donc à vous de trouver lesquelles pourraient convenir. La figure suivante vous fournit quelques exemples pour commencer.



Pour aller plus loin

Il existe un nombre infini de déclinaisons d'afficheurs sept segments. L'affichage peut être de différentes couleurs, telles que :

- jaune ;
- rouge ;
- vert ;
- rouge très clair.

Il faut bien entendu s'assurer du type de connexion avant d'acheter :

- l'anode commune ;
- la cathode commune.

Ils ont des tailles différentes. En voici deux proposées par le fournisseur Kingbright :

- type *SA-39* : hauteur des chiffres = *0,39" = 9,9 mm* ;
- type *SA-56* : hauteur des chiffres = *0,56" = 14,2 mm*.