Montage 6

Extension de port

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation de plusieurs broches comme sorties (`OUTPUT`) ;
- le registre à décalage de type 74HC595 avec 8 sorties ;
- la commande du registre à décalage par trois lignes de la carte Arduino ;
- la définition d'une fonction particulière ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- les autres sketches ;
- l'instruction `shiftOut` ;
- un exercice complémentaire.

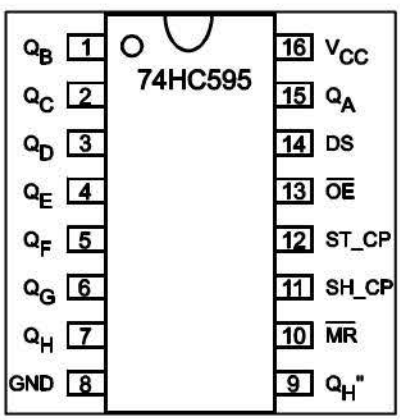
Le registre à décalage

Nous avons vu dans le montage précédent comment programmer la commande des multiples LED d'un séquenceur de lumière. Votre carte Arduino ne disposant que d'un nombre limité de sorties numériques, ces précieuses ressources pourraient finir par vous manquer pour ajouter d'autres LED à votre séquenceur de lumière. Par ailleurs, vous voulez peut-être aussi connecter quelques capteurs sur des entrées numériques.

Vous aurez donc encore moins de broches numériques à disposition. Comment résoudre ce problème ? Il existe plusieurs solutions, en voici une. Je dois utiliser pour cela un registre à décalage. Vous vous demandez certainement ce que c'est et comment il opère. Dans cette expérimentation, un circuit intégré (IC pour *Integrated Circuit*) sera relié pour la première fois à votre carte Arduino. Un registre à décalage est un circuit géré par un signal d'horloge et doté de plusieurs sorties disposées l'une derrière l'autre. À chaque période d'horloge, le niveau présent à l'entrée du registre est transmis à la sortie suivante. Cette information passe ainsi par toutes les sorties existantes.

Le circuit intégré 74HC595, que nous utilisons ici, dispose d'une entrée série par laquelle les données sont entrées et de huit sorties équipées de registres mémoire internes pour conserver les états. Seules trois broches numériques sont nécessaires à l'alimentation, lesquelles fournissent des données au module qui, de son côté, commande ses huit sorties. C'est en soi une économie majeure car le circuit 74HC595 peut être cascadé, permettant ainsi une expansion quasi illimitée des sorties numériques. De quoi s'agit-il exactement ? Voyons de plus près les différentes entrées et sorties de ce circuit. La figure 6-1 illustre le brochage du circuit, vu de dessus.

Figure 6-1 ▶
Brochage du registre à décalage 74HC595



Le tableau 6-1 récapitule les différentes broches et leur signification.

Broche	Signification
V_{cc}	Tension d'alimentation
GND	Masse 0 V
Q_A-Q_H	Sorties parallèles 1 à 8
Q_H"	Sortie série (entrée pour un deuxième registre à décalage)
MR	Master Reset (actif LOW)
SH_CP	Registre à décalage, entrée d'horloge (Shiftregister clock input)
ST_CP	Registre mémoire, entrée d'horloge (Storageregister clock input)
OE	Activation de la sortie (Output enable/actif LOW)
DS	Entrée série (Serial data input)

Tableau 6-1
Signification des broches du registre à décalage 74HC595

Le mode de fonctionnement du registre à décalage peut se résumer ainsi : quand le niveau à l'entrée d'horloge SH_CP passe de LOW à HIGH, le niveau à l'entrée série DS est lu, transmis à l'un des registres internes et enregistré temporairement. Mais cela ne signifie pas pour autant transmis aux sorties Q_A à Q_H. C'est seulement une impulsion d'horloge à l'entrée ST_CP de LOW à HIGH qui fait transmettre toutes les informations des registres internes aux sorties. C'est utile car ce n'est que quand toutes les informations ont été lues à l'entrée série qu'elles sont censées être détectées aux sorties. Le changement du niveau logique de LOW à HIGH est appelé contrôle par front montant d'horloge, car une action n'est entreprise que quand un changement de niveau se produit de la manière décrite.

Voyons maintenant un peu ce qui se passe dans le registre à décalage…

Voici justement SH_CP au travail. Quand il tourne la pancarte de LOW à HIGH, le candidat potentiel, qui se trouve dans la zone DS-area, passe dans le registre suivant et attend la suite de son voyage vers la sortie.

Figure 6-2
SH_CP préparant les données série

La figure 6-3 montre ST_CP en train de faire partir les données des registres internes vers les sorties.

Figure 6-3 ▶
ST_CP autorisant le départ des données des registres vers les sorties

Quand il tourne la pancarte de `LOW` à `HIGH`, les portes des registres internes s'ouvrent et alors seulement les données peuvent trouver le chemin de la sortie. Le procédé croqué ici sera reproduit dans plusieurs sketches pour que vous puissiez voir en direct comment marche le registre à décalage. Nous allons tout faire de A à Z, mais vous verrez à la fin qu'il existe une instruction bien commode pour toutes les actions à entreprendre l'une derrière l'autre, qui vous épargnera bien du travail et vous facilitera les choses.

Composants nécessaires

	1 registre à décalage 74HC595
	8 LED rouges
	8 résistances de 330 Ω
	1 résistance de 10 kΩ

1 bouton-poussoir

Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Voici le code du sketch pour commander le registre à décalage 74HC595 au moyen de trois lignes de sorties numériques. Les broches suivantes sont nécessaires sur le registre :

- SH_CP (registre à décalage, entrée d'horloge) ;
- ST_CP (registre mémoire, entrée d'horloge) ;
- DS (entrée série pour les données).

Des variables sont affectées aux trois lignes de données, variables auxquelles j'ai donné les noms suivants :

- SH_CP est `shiftPin` ;
- ST_CP est `storagePin` ;
- DS est `dataPin`.

Ce sketch met l'entrée série DS sur `HIGH`, niveau qui est ensuite transféré dans le registre interne quand l'entrée d'horloge SH_CP du registre à décalage passe de `LOW` à `HIGH`. Les sorties sont alors programmées et enregistrées via les registres internes au moyen de l'entrée d'horloge ST_CP du registre mémoire.

```
int shiftPin = 8;         //SH_CP
int storagePin = 9;       //ST_CP
int dataPin = 10;         //DS
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  resetPins();            //Mise de toutes les broches sur LOW
  //Mise de DS sur HIGH pour reprise ultérieure par SH_CP
  digitalWrite(dataPin, HIGH); //DS
  delay(20); //Brève pause avant traitement
  //Transmission du niveau à DS dans registres mémoire internes
  digitalWrite(shiftPin, HIGH); //SH_CP
  delay(20); //Brève pause avant traitement
  //Transmission des registres mémoire internes aux sorties
```

```
  digitalWrite(storagePin, HIGH); //ST_CP
  delay(20);
}

void loop(){/* vide */}

//Réinitialisation de toutes les broches → niveau LOW
void resetPins(){
  digitalWrite(shiftPin, LOW);
  digitalWrite(storagePin, LOW);
  digitalWrite(dataPin, LOW);
}
```

Revue de code

Les variables suivantes sont techniquement nécessaires à notre programmation.

Tableau 6-2 ▶
Variables nécessaires et leur objet

Variable	Objet
shiftPin	N° broche SH_CP
storagePin	N° broche ST_CP
dataPin	N° broche DS

Les variables sont d'abord pourvues des informations de broche nécessaires puis toutes les broches sont programmées en tant que sorties au début de la fonction `setup`. Vous rencontrez pour la première fois dans ce montage une fonction écrite par vous-même. Une fonction n'a en soi rien de nouveau pour vous puisque `setup` et `loop` font déjà partie de cette catégorie de structures logicielles. Je souhaite cependant revenir sur le sujet pour en préciser le sens et l'objet. Une fonction peut être considérée comme une sorte de sous-programme qui peut toujours être appelé dans le déroulement normal d'un sketch. Elle est invoquée par son nom et peut tout aussi bien retourner une valeur à l'appelant qu'enregistrer plusieurs valeurs transférées nécessaires au calcul ou au traitement. La structure formelle d'une fonction est la suivante :

Figure 6-4 ▶
Structure de base d'une fonction

```
Type de donnée de retour Nom (paramètre)
{
   //Une ou plusieurs instructions
}
```

La partie entourée est appelée signature de la fonction et représente l'interface formelle avec la fonction. Cette dernière est comparable à une

black box, que vous connaissez déjà. En fait, vous n'avez pas besoin de savoir comment elle fonctionne. Il vous suffit de connaître la structure de l'interface et de savoir sous quelle forme une valeur est retournée le cas échéant. Vous programmez ici bien entendu la fonction elle-même et devez pour le moins connaître la logique qu'elle renferme. Certaines fonctions peuvent également être obtenues par exemple sur Internet, dans la mesure où leur usage n'est pas limité techniquement par une licence, et utilisées dans votre montage. Peu importe de savoir comment elles fonctionnent du moment qu'elles ont été programmées et testées avec succès par d'autres. Le principal est qu'elles fonctionnent ! Mais revenons à notre définition de la fonction. Si elle renvoie une valeur en retour à l'appelant, comme le fait par exemple `digitalRead`, vous devez indiquer le type de donnée en question dans votre fonction.

Supposons que vous vouliez retourner des valeurs qui sont toutes des nombres entiers, le type de donnée est alors `Integer` défini par le mot-clé `int`. Si toutefois aucun retour n'est requis, vous devez le faire savoir par le mot-clé `void` (traduction : *vide*) qui précède déjà les deux fonctions principales `setup` et `loop`.

Excusez-moi mais j'ai une question. Vous avez dit que les fonctions sont toujours invoquées par leur nom. Mais qu'en est-il des deux fonctions `setup` et `loop` ? Pas besoin de stipuler quelque part dans le code qu'elles doivent être appelées et pourtant ça marche. Comment est-ce possible ?

Cette question est pertinente, pourtant ce comportement n'étonne personne la plupart du temps. `setup` et `loop` sont des fonctions systémiques qui sont appelées implicitement. Comme vous l'avez remarqué, vous n'avez pas besoin de vous en occuper.

Pour aller plus loin

Si cela vous intéresse, vous trouverez dans le répertoire d'installation sous `arduino-1.x.y\hardware\arduino\cores\arduino` le fichier `main.cpp` que vous pourrez ouvrir avec un éditeur de texte. Vous verrez alors ce qui suit :

```
#define ARDUINO_MAIN
#include <Arduino.h>

int main(void)
{
    init();

#if defined(USBCON)
    USB.attach();
#endif

    setup();

    for (;;) {
        loop();
        if (serialEventRun) serialEventRun();
    }

    return 0;
}
```

La fonction directement appelée en début de programme avec C++ est nommée `main`, comme c'est le cas ici. Elle sert quasiment de point d'accès, pour que le programme sache par quoi il doit commencer. `main` contient plusieurs appels de fonction qui sont traités l'un après l'autre. On y trouve entre autres la fonction `setup` et l'appel de la fonction `loop` dans une boucle sans fin définie par `for(;;)`. Vous reconnaissez certainement les déroulements ou plutôt les relations qui se créent en coulisses au début d'un sketch quand il s'agit d'appeler `setup` ou `loop`.

Si une ou plusieurs variables sont à fournir à votre fonction, celles-ci sont indiquées entre parenthèses derrière son nom, séparées par des virgules, avec leur type de donnée correspondant. Les parenthèses sont nécessaires même s'il n'y a rien entre elles faute de variables. La signature est suivie du corps de fonction, formé par la paire d'accolades. Toutes les instructions, qui se trouvent entre ces deux accolades, font partie de la fonction et sont traitées séquentiellement de haut en bas lors de l'appel. Mais revenons au code. En quoi est-ce utile d'écrire une fonction particulière ? Très simple ! Ça l'est toujours quand les mêmes instructions sont à exécuter plusieurs fois dans le code et c'est ici le cas. Je dois exécuter la suite d'instructions qui suit à divers endroits pour réinitialiser – autrement dit, pour remettre sur `LOW` – les niveaux sur les différentes broches numériques. Sans fonction, le sketch compterait un grand nombre de lignes de code en plus et manquerait donc de clarté.

```
digitalWrite(shiftPin, LOW);
digitalWrite(storagePin, LOW);
digitalWrite(dataPin, LOW);
```

Pour aller plus loin

Le code source, qui revient plusieurs fois avec la même séquence d'instructions dans le sketch, est appelé code redondant ou redondance de code. Le mieux est de le stocker dans une fonction à laquelle vous donnez un nom suffisamment évocateur pour en saisir le sens. Si vous devez procéder à une modification, vous intervenez de manière centrale au sein de la fonction et non pas un peu partout dans le code, ce qui est générateur de bien des erreurs et très chronophage.

Au début du sketch, l'appel de fonction :

```
resetPin() ;  //Mettre toutes les broches sur LOW
```

permet de mettre les broches 8, 9 et 10 au niveau `LOW`.

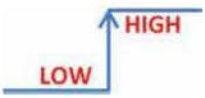
Le premier signal de niveau `HIGH` est ensuite appliqué à DS par la ligne :

```
digitalWrite(dataPin, HIGH); //DS
```

Puis une attente de 20 ms s'écoule avant que la ligne suivante ne transmette le niveau `HIGH` de DS au registre mémoire interne :

```
digitalWrite(shiftPin, HIGH); //SH_CP
```

Il faut ici tenir compte du fait que ce n'est possible qu'au moyen d'un contrôle par front montant de `LOW` vers `HIGH`.



Il n'y a pas encore de transfert en direction du port de sortie. Une nouvelle attente de 20 ms s'écoule, et enfin la ligne suivante déclenche la transmission des registres mémoire internes aux sorties, ce qui revient à commander les LED dans le cas présent :

```
digitalWrite(storagePin, HIGH); //ST_CP
```

Un changement de niveau de `LOW` à `HIGH` est ici aussi nécessaire, d'où le recours à la fonction `resetPin` qui permettra plus tard de changer encore le niveau de `LOW` à `HIGH`.

Schéma

Le schéma montre les différentes LED avec leurs résistances série de 330 ohms, qui sont commandées par le registre à décalage 74HC595. L'entrée *master reset* de la puce est connectée, à travers la résistance *pull-up*, à la tension d'alimentation +5 V, si bien que le reset ne se déclenche pas tant que le bouton-poussoir n'est pas enfoncé puisque l'entrée MR est active au niveau `LOW`. On notera la présence d'un trait horizontal au-dessus de MR, qui correspond à une négation. L'entrée

output enabled est également active au niveau `LOW` et reliée par un fil à la masse car les sorties doivent être toujours actives. Le registre à décalage est commandé par les broches Arduino 8, 9 et 10 avec les fonctions décrites plus haut.

Si vous lancez le sketch, la première LED s'allume immédiatement sur la sortie Q_A car vous avez entré une seule fois `1` dans le registre à décalage. Vous devez actionner non seulement le bouton-poussoir du circuit mais aussi le bouton de reset de la carte Arduino pour effectuer un reset (réinitialisation).

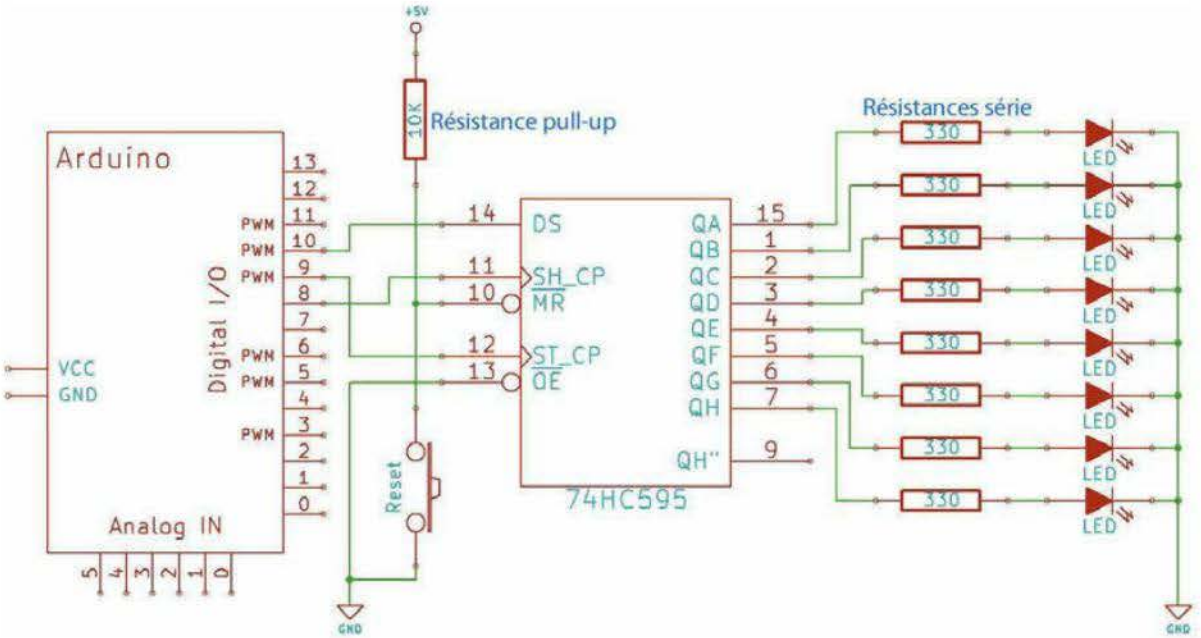


Figure 6-5 ▲
Carte Arduino commandant le registre à décalage 74HC595 par trois lignes de signaux

Réalisation du circuit

Votre plaque d'essais se remplit et les choses deviennent intéressantes, n'est-ce pas ?

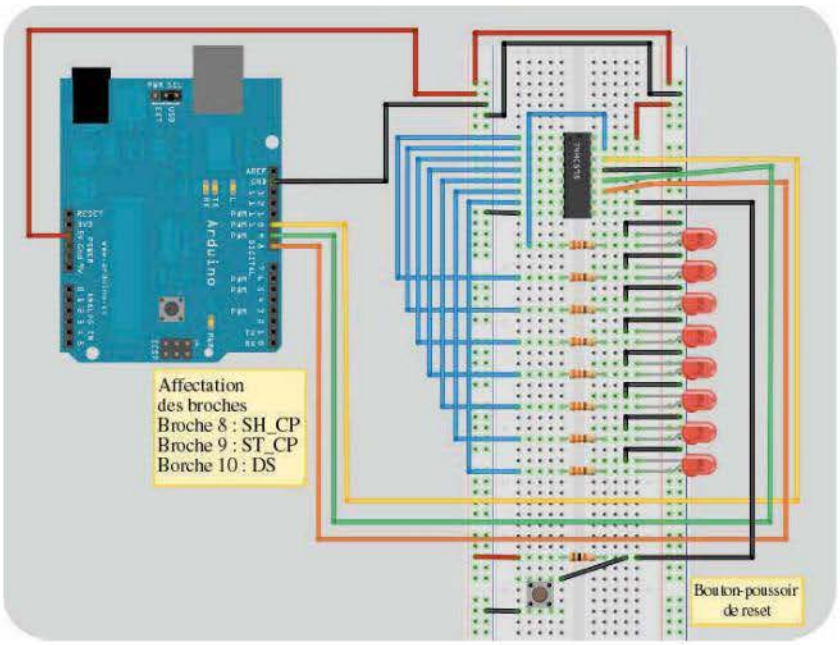


Figure 6-6
Réalisation du circuit avec Fritzing

Extension du sketch : première partie

Complétons maintenant un peu le sketch de telle sorte que vous puissiez rentrer plusieurs valeurs dans l'entrée série. Ce n'est encore qu'un degré intermédiaire et non pas la solution finale que je souhaite vous présenter. Ce code doit transmettre au registre à décalage une séquence stockée dans un tableau de données. La construction du circuit reste la même.

```
int shiftPin = 8;          //SH_CP
int storagePin = 9;        //ST_CP
int dataPin = 10;          //DS
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  resetPins();            //Mettre toutes les broches à LOW
  putPins(dataArray);     //Régler les broches sur le tableau
                          //de données
  //Transmission des registres mémoire internes aux sorties
  digitalWrite(storagePin, HIGH);   //ST_CP
}
```

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

void resetPins(){
  digitalWrite(shiftPin, LOW);
  digitalWrite(storagePin, LOW);
  digitalWrite(dataPin, LOW);
}

void putPins(int data[]){
  for(int i = 0; i  <  8; i++){
    resetPins();
    digitalWrite(dataPin, data[i]); delay (20);
    digitalWrite(shiftPin, HIGH); delay (20);
  }
}
```

Voyons maintenant comment le code accomplit son travail. Tout tourne autour du tableau de données qui contient le modèle de commande des différentes LED. Il s'agit donc de la ligne de déclaration et initialisation suivante :

```
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
```

Le code lit les éléments du tableau de gauche à droite et rentre les valeurs dans le registre à décalage. Un `1` signifie une LED allumée et un `0` une LED éteinte.

Un moment s'il vous plaît. Vous avez utilisé les valeurs `1` et `0` pour commander les LED. Ne vaut-il pas mieux travailler avec les noms de constante `HIGH` et `LOW` ?

J'ai préféré utiliser les valeurs `1` et `0` parce que ce sont elles précisément qui se cachent derrière les constantes `HIGH` et `LOW`. Normalement, je ne suis pas pour les *magic numbers* (voir page 224) mais il m'a semblé que dans ce cas, je pouvais faire une exception. `1` et `0` étant aussi les valeurs logiques, vous ne devriez pas avoir trop de mal à comprendre ! Vous pouvez bien entendu écrire à la place de :

```
int dataArray[] = {1, 0, 1, 0, 1, 1, 0, 1};
```

la ligne suivante :

```
int dataArray[] = {HIGH, LOW, HIGH, LOW, HIGH, HIGH, LOW, HIGH};
```

Mais revenons au code et à sa manière d'exploiter le tableau. La chose n'est pas si simple car j'ai ajouté une fonction nommée `putPins`, qui a pour tâche de remplir le registre à décalage. Elle présente un

paramètre de transmission capable d'enregistrer non pas une variable normale mais tout un tableau de données. Il suffit simplement de transmettre le tableau de données comme argument dans la fonction :

```
putPins(dataArray);
```

La fonction est définie comme suit :

```
void putPins(int data[]){
  for(int i = 0; i  < 8; i++){
    resetPins(); //Remise à zéro des broches et préparation
                 //à la commande par front montant
    digitalWrite(dataPin, data[i]); delay(20);
    digitalWrite(shiftPin, HIGH); delay(20);
  }
}
```

On peut voir qu'un tableau du type de donnée `int` a été déclaré avec deux crochets dans la signature de la fonction. Au moment où la fonction est appelée, le tableau initial `dataArray` est copié dans `data`, qui est ensuite exploité dans la fonction.

Puis chaque élément du tableau est envoyé dans l'entrée série, au moyen de la boucle `for` (que vous connaissez déjà), via :

```
digitalWrite(dataPin, data[i]);
```

et placé lors de l'étape suivante dans le premier registre interne via :

```
digitalWrite(shiftPin, HIGH);
```

Le tout s'effectue huit fois (de `0` à `7`), chaque registre interne transmettant ses valeurs au suivant. Les figures suivantes montrent les choses encore plus clairement.

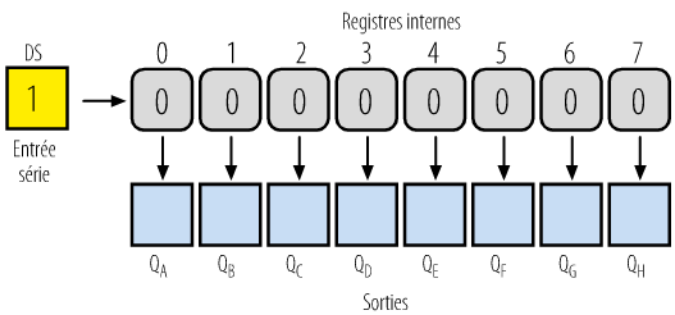
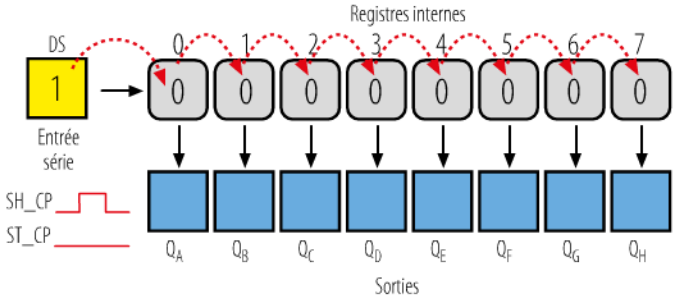


Figure 6-7
Registre à décalage

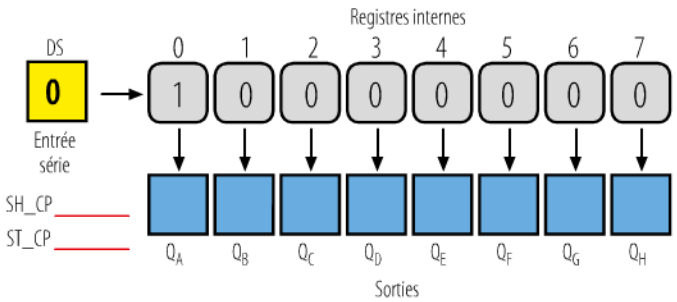
Au début, les registres sont encore tous vides. Un `1` attend toutefois déjà à l'entrée d'être transféré dans le premier registre interne.

Figure 6-8 ▶
Registre à décalage pendant le premier temps SH_CP



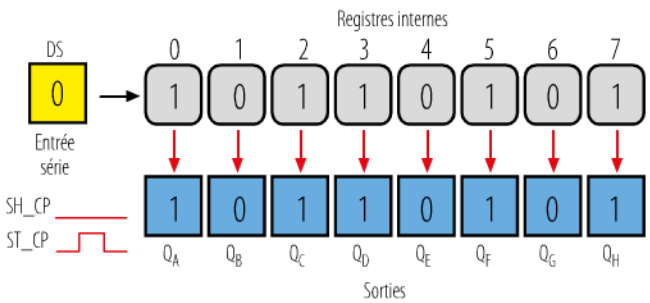
Le 1 qui se trouve à l'entrée série est entré, pendant le front montant de SH_CP, dans le premier registre interne. Les contenus de tous les registres sont décalés d'une position vers la droite. Cette action donne les états illustrés à la figure 6-9.

Figure 6-9 ▶
États du registre à décalage après la première impulsion de SH_CP



À l'entrée se trouve maintenant un 0 qui sera lui aussi entré à la prochaine impulsion de SH_CP dans le premier registre interne. Mais avant, l'état du septième registre interne sera passé au huitième, le sixième au septième, etc., et enfin le 0 est entré dans le premier registre. Passons directement au moment où toutes les valeurs du tableau ont été entrées, conformément au schéma ci-dessus, dans les registres internes et où l'impulsion ST_CP a recopié les registres vers les sorties.

Figure 6-10 ▶
États du registre à décalage après lecture des valeurs du tableau et après l'impulsion ST_CP



Les valeurs du tableau lu sont appliquées aux sorties seulement maintenant, la première valeur entrée se trouvant complètement à droite et la dernière complètement à gauche.

> Comment fait-on pour inverser le comportement ? Je voudrais maintenant que la première valeur du tableau se trouve complètement à gauche et que la dernière complètement à droite se trouve à la sortie, de telle sorte que l'ordre soit quasiment inversé.

Pas de problème car où les broches ont-elles été déterminées ? Exact, dans la fonction `putPins` ! C'est la boucle `for` qui fixe l'ordre des différentes broches. Celui-ci sera inversé si vous appelez la dernière valeur à la place de la première et la transmettez au registre à décalage. Voici le code modifié de la boucle `for` :

```
for(int i = 7; i  >= 0; i--){
   // ...
}
```

Extension du sketch : deuxième partie

Maintenant que vous en savez assez sur le registre à décalage 74HC595, je voudrais vous présenter une instruction spéciale qui vous épargnera du travail. `shiftOut` est vraiment facile à utiliser. Mais je dois auparavant vous livrer quelques informations sur la sauvegarde des données dans l'ordinateur, informations qui sont vraiment importantes pour comprendre le fonctionnement d'un microcontrôleur. Je me sers pour mes réalisations du type de donnée `byte`, dont la taille est de 8 bits et qui peut stocker des valeurs comprises entre `0` et `255`. La figure 6-11 montre la valeur décimale `157` sous sa forme binaire `10011101`.

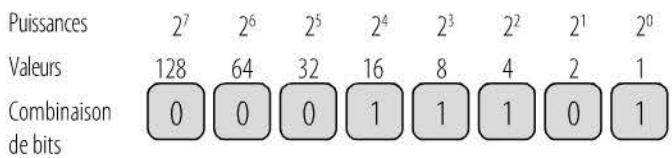


Figure 6-11
Combinaison binaire pour le nombre entier `157`

Si on regarde les puissances de plus près, on voit que la base est le chiffre 2. Nous autres humains comptons en base 10 en raison des dix doigts de nos mains. Les valeurs des différents chiffres d'un nombre sont donc 10^0, 10^1, 10^2, etc. Pour le nombre 157, cela donne $7 \times 10^0 + 5 \times 10^1 + 1 \times 10^2$ qui font bien sûr 157. Le microcontrôleur ne

pouvant cependant stocker que deux états (`HIGH` et `LOW`), le système binaire (du latin *binarius*, deux chacun) est fondé sur la base 2. La valeur décimale de ladite combinaison binaire se calcule par conséquent comme suit, en commençant en principe par la valeur – ou bit – la plus petite :

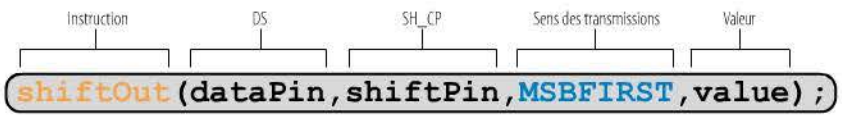
$$1\times2^0+0\times2^1+1\times2^2+1\times2^3+1\times2^4+0\times2^5+0\times2^6+1\times2^6=157_{10}$$

Pour aller plus loin

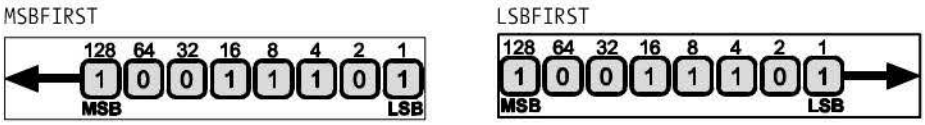
La base figure derrière le nombre pour plus de clarté quand différents systèmes de numération sont utilisés.

Avec un nombre de 8 bits (ou 1 octet), vous pouvez représenter 256 valeurs distinctes (de 0 à 255). Ceci dit, revenons à l'instruction `shiftOut`, qui possède différents paramètres dont nous allons faire le tour.

Figure 6-12 Instruction `shiftOut` avec ses nombreux arguments



Les arguments `dataPin`, `shiftPin` ou encore la valeur à transmettre sont censés être clairs. Mais que signifie la constante `MSBFIRST` ? Cet argument permet de définir le sens de transmission des bits. Dans le cas d'un octet, le bit de poids fort est nommé *Most Significant Bit* (MSB) et le bit de poids faible *Least Significant Bit* (LSB). Vous pouvez aussi définir quel bit doit être transféré en premier dans le registre à décalage.



Voici le code complet avec l'instruction `shiftOut`. Le circuit n'a pas non plus besoin ici d'être modifié.

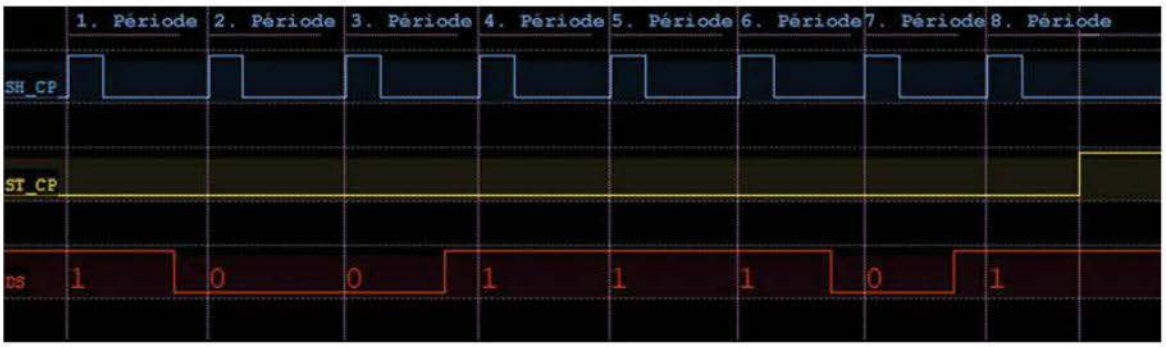
```
int shiftPin = 8;     //SH_CP
int storagePin = 9;   //ST_CP
int dataPin = 10;     //DS
byte value = 157;     //Valeur à transférer
void setup(){
  pinMode(shiftPin, OUTPUT);
  pinMode(storagePin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop(){
```

```
  digitalWrite(storagePin, LOW);
  shiftOut(dataPin, shiftPin, MSBFIRST, value);
  digitalWrite(storagePin, HIGH);
  delay(20);
}
```

Pour aller plus loin

Vous pouvez saisir directement la combinaison binaire au lieu du nombre décimal 157 lors de l'initialisation des variables et éviter ainsi la conversion. Tapez seulement `B10011101`. Le préfixe `B` indique qu'il s'agit d'une combinaison binaire avec laquelle la variable doit être initialisée.

Ce chronogramme vous montre les niveaux des trois lignes de données, les unes par rapport aux autres, pour commander le registre à décalage pendant le déroulement chronologique.



▲ **Figure 6-13**
Chronogramme pour le nombre transféré `157` (`B10011101`)

On voit tout en haut le signal d'horloge SH_CP pour la prise en charge des données à l'entrée série DS. À l'issue de la 8e période, le niveau de ST_CP passe de `LOW` à `HIGH` et les données des registres internes sont transmises aux sorties. Essayez avec différentes valeurs et différents sens de transmission pour bien comprendre.

Pour aller plus loin

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

- 75HC595 ;
- 75HC595 fiche technique ;
- 74HC595 datasheet.

Problèmes courants

Si les LED ne s'allument pas l'une après l'autre, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la plaque d'essais correspondent-elles vraiment au circuit ?
- Pas de court-circuit éventuel ?
- Les différentes LED sont-elles correctement branchées ? La polarité est-elle correcte ?
- Les résistances ont-elles bien les bonnes valeurs ?
- Le registre à décalage est-il correctement câblé ? Contrôlez encore une fois tous les raccordements qui sont nombreux.
- Le code du sketch est-il correct ?

Qu'avez-vous appris ?

- Vous savez ce qu'est un registre à décalage de type 74HC595 avec une entrée série et huit sorties.
- Le premier sketch permet de commander les trois lignes de données SH_CP, ST_CP et DS, les signaux d'horloge étant contrôlés par un front d'horloge montant, autrement dit ne réagissant que quand le niveau passe de `LOW` à `HIGH`.
- L'instruction `shiftOut` permet d'envoyer au registre à décalage des combinaisons de bits au moyen de nombres non seulement décimaux mais aussi binaires.
- Vous pouvez initialiser une variable du type de donnée `byte` avec un nombre entier, par exemple `157`, ou à l'aide de la combinaison de bits correspondante qui doit être précédée du préfixe `B`, soit par exemple `B10011101`.

Exercice complémentaire

Dans cet exercice complémentaire, je vous propose d'abord de faire s'allumer les LED au gré de toutes les combinaisons de bits possibles entre `00000000` et `11111111`.

Puis je vous invite à concevoir différents motifs ou séquences, selon lesquels les LED doivent clignoter. Pour cela, je vous donne l'exemple de la figure 6-14.

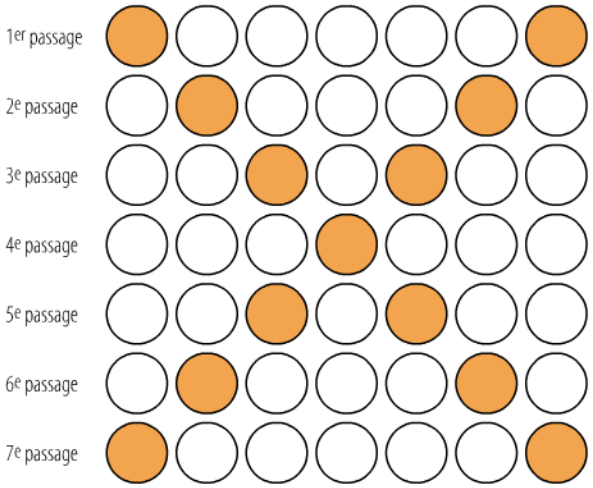


Figure 6-14
Séquence de LED commandée par un registre à décalage 74HC595

Le motif du 7e passage est le même que celui du 1er et la séquence revient au début. Il s'agit de deux LED allumées qui se rapprochent l'une de l'autre pour s'éloigner à nouveau. Vous pouvez répéter ce déroulement trois fois. Toutes les LED doivent clignoter à la fin pendant ∫ seconde l'une après l'autre, puis le cycle doit reprendre au début.