LA GARE DE VERBROUCK BÉNÉFICIERAIT LARGEMENT D'UNE SIGNALISATION LUMINEUSE FONCTIONNELLE SELON LES PRINCIPES DÉCRITS ICI PAR CHRISTIAN BÉZANGER.

Y. BAUDE

Dynamisez

LA SIGNALISATION LUMINEUSE DE VOTRE RÉSEAU

On voit parfois de très beaux réseaux, hélas dotés d'une signalisation lumineuse statique. Il est par exemple bien dommage de voir un signal au rouge, franchi par un train qui ne s'arrête pas. Pourtant, avoir une signalisation lumineuse dynamique n'est pas très compliqué avec Arduino.

Texte et illustrations : Christian Bézanger (christian.bezanger@gmail.com>) sauf mention contraire

Cet article va donc vous expliquer comment gérer la signalisation lumineuse de votre réseau et vous fournira un plan de montage et un programme de démonstration capable de gérer 16 LED. Si vous comprenez bien le principe, vous pourrez étendre le montage à 24 LED voire plus, et ainsi gérer toute la signalisation de votre réseau avec une simple carte Arduino Uno. Nous avons même implanté ce programme dans un ATtiny45 et cela fonctionne.

Économiser des sorties

Lorsqu'on pense signalisation lumineuse et Arduino, on imagine un montage du genre de la **figure 1** où chaque LED du signal est reliée à une sortie d'Arduino. Or, un signal SNCF peut aller de deux lampes (carré violet) à neuf lampes (cible de type H) sans compter l'œilleton lorsqu'il existe. Suivant le nombre de signaux et leurs types installés sur votre réseau, une carte Uno risque de ne pas suffire.

Plutôt qu'utiliser une carte Mega, nous allons faire appel au circuit intégré 74HC595 **(voir Fiche Pratique III.72 dans LR 901 d'août 2022)** qui permet d'augmenter le nombre de sorties d'une carte Uno (ou d'un ATtiny). Ce circuit intégré est présenté dans la fiche pratique incluse dans ce numéro, avec un montage et un programme permettant d'avoir 16 LED commandées par seulement 3 sorties d'Arduino. La **figure 2** montre comment effectuer le montage avec deux circuits 74HC595 (représentés en gros plan).

Attention à l'orientation des deux circuits intégrés, le point blanc repère la broche 1 du CI.

Chaque LED du montage représente une LED d'un signal lumineux à cathodes communes ; la LED est commandée par son anode (signaux à cathodes communes) et elle est allumée si celle-ci est à un niveau HIGH (5 V). Ce serait un niveau LOW (0 V) qui allumerait la LED si elle était commandée par sa cathode ; pour plus d'informations, lisez l'article https://locoduino.org/spip.php?article209.

Nous allons continuer en supposant que les signaux sont à cathodes communes mais nous verrons en fin d'article comment utiliser des signaux à anodes communes. Le programme qui fait fonctionner le montage de la **figure 2** envoie deux octets, soit 16 bits qui se retrouvent sur les LED (si le bit est à 1 la LED est allumée, s'il est à 0 elle est éteinte). Pour gérer vos signaux, il suffit donc d'envoyer les bons bits dans le bon ordre (voir fiche pratique déjà citée).

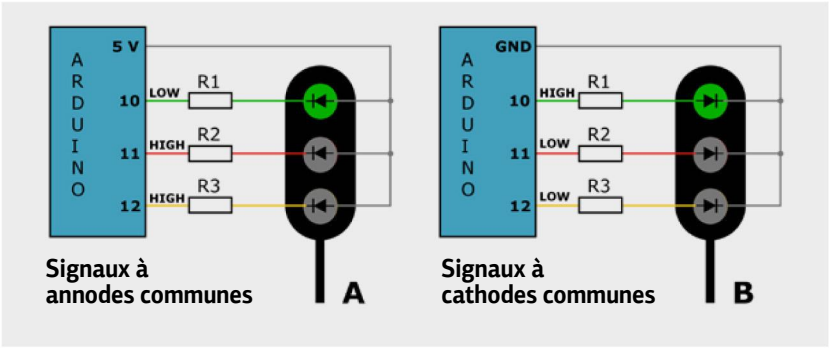


FIGURE 1.

Organisation des LED en signaux lumineux SNCF

Les 16 LED du montage représentent donc les 16 bits des deux octets envoyés par le programme et le bit le moins significatif est sur la LED de droite. Nous allons décomposer ces 16 LED en signaux SNCF et ce découpage dépend bien sûr des signaux qui se trouvent sur votre réseau. La **figure 3 (page suivante)** montre un exemple de découpage avec de gauche à droite un carré violet (2 LED), deux signaux de B.A.L (Block Automatique Lumineux, deux fois trois LED), deux carrés (deux fois quatre LED). Pour chaque signal, le feu le plus haut est à gauche. Pour les carrés, l'œilleton n'a pas été représenté mais il me semble qu'il est allumé en même temps que le rouge du sémaphore et peut donc être commandé par la même sortie ; dans ce cas, il faut baisser la valeur de la résistance pour que les deux LED soient commandées ensemble.

Le découpage doit être fait de façon astucieuse et non dans l'ordre des signaux du réseau afin de ne pas laisser de place vide. Par exemple, un carré avec ralentissement ou rappel de ralentissement occupe 6 LED, il reste deux places pour un carré violet. Un signal peut aussi occuper plus de 8 bits (cible H à 9 lampes) et être à cheval sur les deux octets envoyés. Lorsque deux LED sont allumées ensemble (ralentissement ou rappel de ralentissement par exemple) on peut n'utiliser qu'une seule sortie. Bref, une seule chose à retenir : avec 2 CI 74HC595, on dispose de 16 bits pour commander 16 LED.

Repérage des signaux dans le programme

Pour commander les différentes lampes d'un signal, il va falloir commander les 16 bits d'une donnée de type entier (**int** en langage d'Arduino). Par exemple, si je veux allumer un « carré fermé » sur le signal le plus à droite du montage, et laisser tout le reste éteint, la valeur à transmettre en binaire est :

0000000000001010 ce qui s'écrit 0b0000000000001010

On visualise bien ce qui est allumé : cela correspond aux bits à 1.

Mais on ne veut pas allumer qu'un seul signal. Par exemple, si on veut initialiser les feux de la façon suivante : feu blanc du carré violet allumé, les deux signaux B.A.L au vert (Voie libre) et les deux carrés fermés (deux feux rouges allumés), alors la valeur binaire est :

int data = 0b1010010010101010;

On pourrait calculer la valeur binaire pour chaque façon d'allumer les signaux mais plus il y a de signaux sur le réseau et plus cela multiplie les possibilités. On va donc procéder autrement. Tout d'abord, on définit une numérotation pour chaque signal afin de le repérer plus facilement. CV désignera un carré violet, BA un signal de B.A.L et CA un signal de type carré et ceci doit être suivi d'un numéro car il se peut qu'il y ait plusieurs signaux de même nature : c'est d'ailleurs le cas dans notre exemple et on trouve CV1, BA2, BA1, CA2 et CA1. Les signaux sont numérotés de la droite vers la gauche mais ceci est arbitraire, vous pouvez faire différemment.

Il faut ensuite définir pour chaque signal les différents visuels possibles, correspondant aux façons d'allumer le signal. Par exemple, CV1W désigne le carré violet feu blanc allumé (W pour white qui est blanc en anglais) et CV1P désigne le carré violet feu violet allumé (P pour purple qui est violet en anglais). J'ai choisi l'anglais ici ▶▶

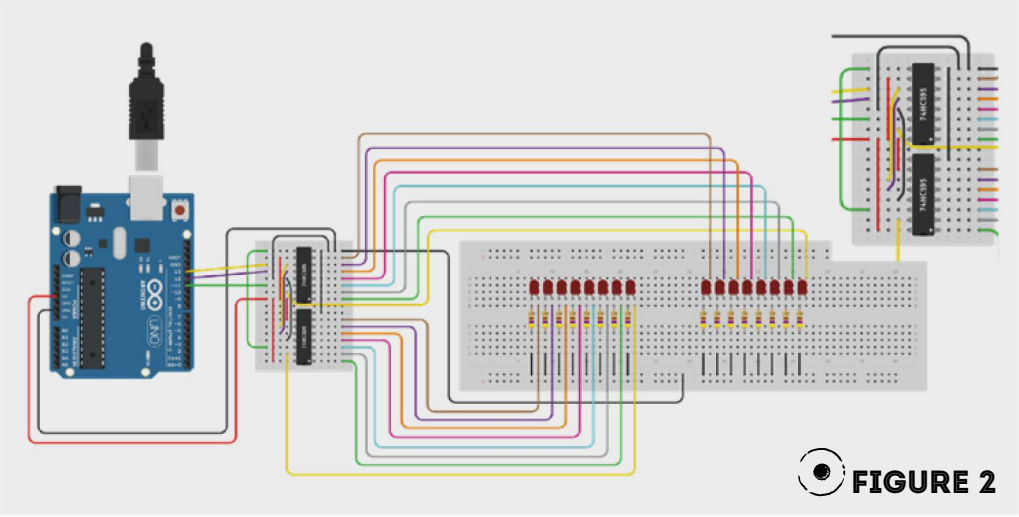
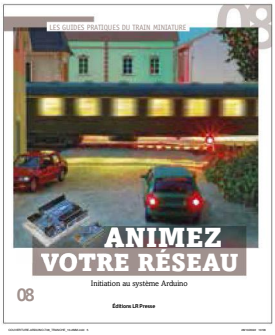


FIGURE 2



Retrouvez votre guide pratique n°8 "Initiation au système Arduino", à paraître début 2023

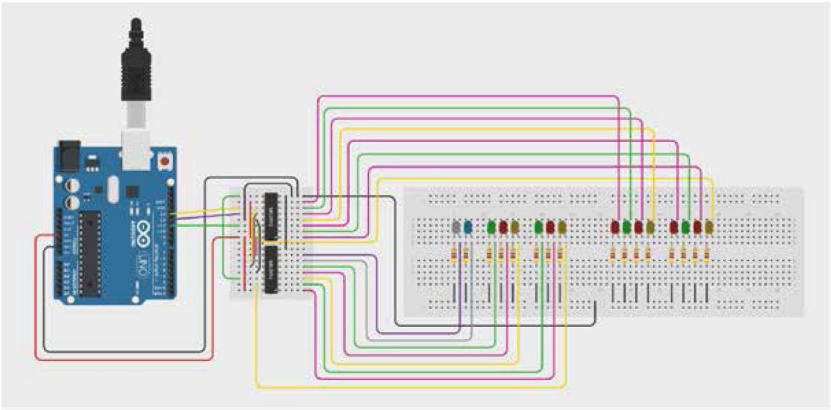


FIGURE 3

pour ne pas confondre avec le V de CV, mais pour les autres signaux, j'utilise l'initiale de la couleur en bon français et C pour désigner le carré (deux feux rouges). Par exemple, pour le premier signal de type BAL, on peut avoir BA1V, BA1R ou BA1J pour les feux vert, rouge et jaune respectivement.
Toujours dans notre système, la valeur binaire à attribuer aux différents visuels correspond à : CV1W = 0b10 et CV1P = 0b01
BA1V = 0b100, BA1R = 0b010 et BA1J = 0b001
CA1C = 0b1010, CA1V = 0b0100, CA1R = 0b0010 et CA1J = 0b0001.
Encore une fois, sachant qu'un 1 correspond à la LED allumée, ce n'est pas très compliqué à comprendre. Ce qui va suivre se corse un peu (mais pas trop).

Changement du visuel d'un signal

Pour modifier le visuel d'un signal, il faut d'abord éteindre ce signal complètement puis afficher la valeur du nouveau visuel, tout cela sans toucher aux signaux qui ne changent pas. Les opérations logiques sur les bits vont nous permettre de résoudre ce problème. Par exemple, pour éteindre un signal, il suffit de mettre tous ses bits à 0 sans modifier les bits des autres signaux. Un ET bit à bit entre la donnée (appelée data plus haut dans le texte) et un code d'effacement (composé de 1 là où il ne faut rien toucher et de 0 là où on veut éteindre) va permettre de réaliser l'opération. Voyons cela sur un exemple pour éteindre le signal B.A.L numéro 1 (le plus à droite des deux signaux B.A.L). La **figure 4** montre la valeur de data : les bits à ne pas toucher sont appelés x (peu importe leur valeur) et les bits à mettre à zéro sont appelés y (peu importe leur valeur). Le code d'effacement correspondant au signal BA1 est 1111100011111111. Le ET bit à bit correspond en quelque sorte à une multiplication, si on multiplie par 1 on retrouve la valeur, si on multiplie par 0 on obtient 0. La figure 4 montre le résultat de cette opération. La donnée n'a pas changé sauf à l'endroit correspondant au signal BA1 où les bits valent 0 (signal éteint).
Dans le programme, l'opération se note :
data = data & effacement ;

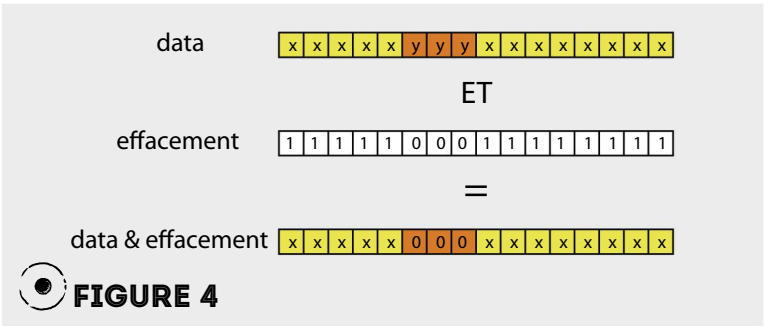


FIGURE 4

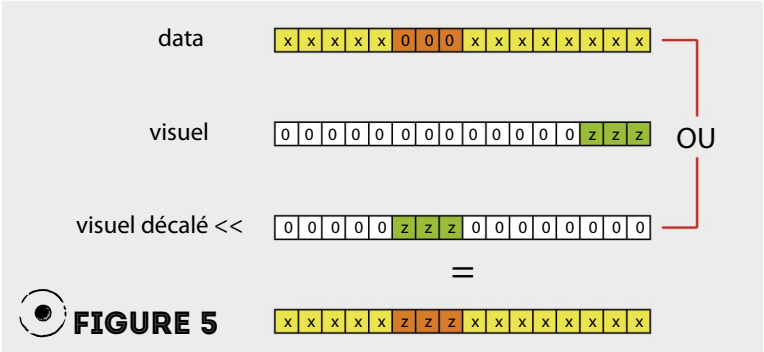


FIGURE 5

Ne pas confondre & (le ET bit à bit) et && (le ET logique). Il faut définir un code d'effacement pour chaque signal possible.
Maintenant que le signal est éteint, il faut introduire le nouveau visuel et le faire au bon endroit pour ne pas modifier les autres signaux. Cette fois, c'est un OU bit à bit qui va réaliser l'opération mais avant cela, il va falloir décaler le visuel pour le mettre au bon endroit. La **figure 5** montre le détail de ce qui va se passer. La donnée data est composée de bits x (peu importe leur valeur) et de 0 à l'endroit du signal à modifier, ceci en raison de ce que nous venons de faire précédemment. Le nouveau visuel est une valeur binaire composée de bits z (peu importe la valeur) précédés de 0 : appelons-le « motif ». On commence par décaler le nouveau visuel vers la gauche d'une certaine quantité pour arriver là où on doit faire la modification ; chaque décalage ajoute un 0 à droite des bits z. Une fois l'opération terminée, on réalise le OU bit à bit ; le OU opère en quelque sorte comme une addition dont le résultat serait limité à 1 (x OU 0 donne x, 0 OU z donne z, 1 OU 1 donne 1). Le nouveau visuel se retrouve donc au bon endroit et la donnée data peut alors être transmise aux CI 74HC595. Dans le programme, l'opération se note :
data = data | (motif<<decalage) ;
Ne pas confondre | (le OU bit à bit) et || (le OU logique). Finalement, pour chaque visuel possible, il faut définir la valeur du décalage à faire pour être au bon endroit. Ce décalage dépend de la position du signal dans notre ensemble de 16 bits ; le programme analyse d'ailleurs la valeur du décalage pour en déduire quel signal est concerné et quel code d'effacement doit être utilisé.

Exemple

Notre signal B.A.L N°2 est au vert et doit passer au rouge car un train vient de la franchir. Le nouveau visuel est BA2R = 0b010 et son décalage doit être de 11. Avant l'opération, la

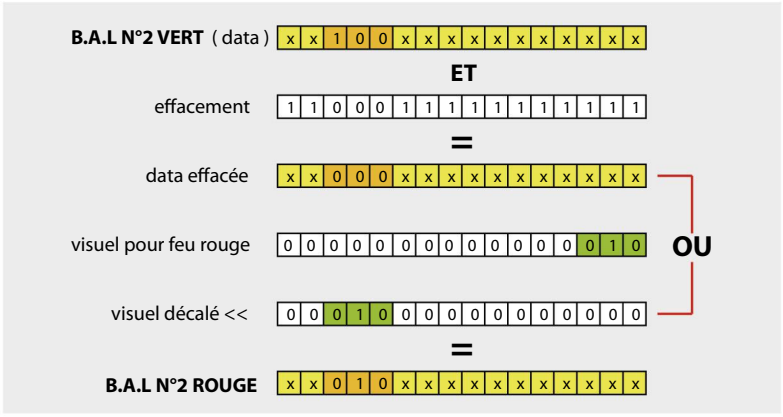


FIGURE 6

donnée data est constitué de bits x qui doivent être conservés après l'opération. La **figure 6** montre ce qui est réalisé.

Voilà, le plus difficile est passé. Ce qu'il faut retenir sur le plan pratique est qu'il faut définir les visuels de chacun de vos signaux (un nombre binaire comportant autant de bits qu'il y a de LED dans le signal) et un décalage (une valeur entière de décalage vers la gauche pour arriver au bon endroit). Il faut aussi définir le code d'effacement de chaque signal ; celui-ci est constitué de 1 sauf aux endroits (bits) où les LED du signal sont connectées (les bits sont alors égaux à 0).

Une fois que vous avez défini tout cela en fonction de votre réseau, vous pouvez utiliser la fonction de mise à jour du signal. Cette fonction est appelée **« updateLight »** et utilise deux arguments : d'une part le motif ou visuel correspondant à ce que vous voulez afficher et d'autre part le décalage qu'il faut appliquer à ce visuel pour qu'il se retrouve au bon endroit. Cette fonction commence par faire un test de type switch...case sur la valeur de décalage afin de déterminer quel signal est concerné et la valeur pour effectuer l'effacement (afin d'éteindre le signal). Par exemple, pour faire passer le carré N°2 en voie libre, il faut écrire : updateLight(CA2V, decCA2V) ;

(On rappelle que CA2V = 0b0100 et decCA2V = 4. On rappelle aussi que 0b0100 est la même chose que 0b0000000000000100 et que si on la décale 4 fois à gauche, on obtient 0b0000000001000000 : 4 zéros ont été perdus à gauche et 4 zéros ont été rajoutés à droite).

Application pratique

Commencez par reproduire le montage de la figure 3 soit en réel, soit en virtuel avec Tinkercad **(voir *Loco-Revue* 891 d'octobre 2021)**. Téléchargez le programme sur le forum Arduino de *Loco-Revue* et utilisez-le pour faire fonctionner votre montage. Ce programme propose une simulation d'un train qui quitte la gare lorsque son carré est en voie libre, puis se déplace de droite à gauche en modifiant la signalisation B.A.L en fonction du canton occupé. Un délai de 5 secondes simule l'avancement du train sur chaque canton et suivant le canton sur lequel il évolue, cela modifie les signaux en amont pour les trains qui suivent.

Cas des signaux à anodes communes

La **figure 7** montre comment relier ces signaux au montage ; cette fois, les LED sont commandées par leur cathode et les anodes en commun sont à relier au 5 V. Et ce qui allume la LED est un bit égal à 0. Pour pouvoir utiliser le programme précédent qui a été prévu pour des signaux à cathodes communes, il faut le modifier légèrement.

Tout d'abord, on part d'un motif d'initialisation où ce sont des 0 qui allument la LED ; ce motif s'appelle « data » dans le programme. Chaque fois qu'on veut le modifier, on l'inverse bit à bit par l'opérateur « bitwise NOT » décrit à la page https://www.arduino.cc/reference/en/language/structure/bitwise-operators/bitwisenot/ (et réalisé par le symbole tilde ~) : le 0 devient 1 et réciproquement. On se retrouve alors dans le cas du programme décrit dans cet article. On effectue le traitement de data, comme expliqué plus haut, puis on inverse à nouveau data pour l'envoyer aux 74HC595 qui réalisent l'affichage. On a donc ajouté deux lignes de programme dans la fonction « updateLight » comme le montre la **figure 8.** Ce n'est pas plus compliqué que cela, et la modification du programme nous a permis d'économiser deux circuits intégrés inverseurs comme les ULN2803. Ceci démontre la force de l'électronique programmable par rapport à l'électronique classique.

Vous pouvez vous inspirer de ce programme pour modéliser votre propre réseau ; ce n'est pas difficile du tout. Bien évidemment, l'appel de la fonction « updateLight » doit se faire en fonction de ce qui circule sur votre réseau, mais avec ce montage, il vous reste suffisamment d'entrées sur votre carte Uno pour y connecter des capteurs (I.L.S ou bien détecteur d'occupation par consommation de courant ou simple contact sur un moteur d'aiguillage). Ainsi, la signalisation lumineuse de votre réseau dépendra vraiment de la position des trains et des aiguillages et cela impressionnera le public lors d'exposition. ■

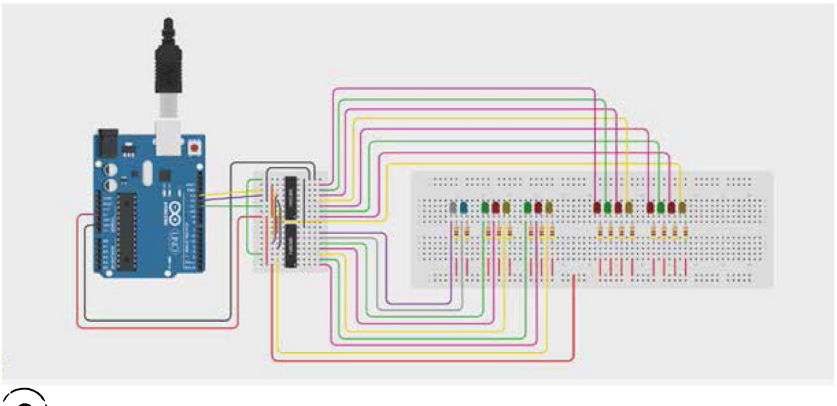


FIGURE 7

```
75    data = ~data;  // Inversion des bits car anodes communes
76    data = data & effacement;  // Efface signal concerné
77    data = data | (motif<<decalage);  // Décale le motif
78    data = ~data;  // Inversion des bits car anodes communes
```

FIGURE 8