Chapitre 30 - Le microcontrôleur ATtiny45

Ce microcontrôleur de marque Atmel, se présente sous la forme d'un boitier DIP à 8 broches (voir figure 30.1). C'est peu mais les broches sont multiplexées, ce qui signifie qu'une même broche peut réaliser plusieurs fonctions. Il dispose de 6 broches d'E/S mais nous n'en utiliserons que 5 par commodité car la sixième sert aussi au reset du µC, ce qui reste très utile tout de même. Son prix est inférieur à 1,50 € (en 2016). Il est parfait pour réaliser par exemple de petites animations lumineuses ou d'autres petits périphériques. Nous allons donc commencer par l'étudier puis nous verrons la marche à suivre pour le programmer aussi simplement que nous programmions nos modules Uno.

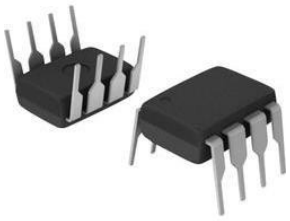


Figure 30. 1

Sa tension d'alimentation varie de 2,7 à 5,5 V (de 1,8 à 5,5 V pour la version ATtiny45V). Une simple pile de 4,5 V suffit donc pour l'alimenter. Le plus de l'alimentation est connecté à la broche 8 et le moins à la quatre, soit deux broches diagonalement opposées comme c'est souvent le cas pour un CI. Sa fréquence d'horloge utilisable dépend de la tension d'alimentation ; s'il est alimenté en 5 V, la fréquence d'horloge peut aller de 0 à 20 MHz.

La mémoire de programme est de 4 kilo-octets, la mémoire RAM est de 256 octets tout comme la mémoire EEPROM. Cela peut paraitre peu mais c'est largement suffisant pour de petites applications. Ces chiffres sont à diviser par 2 si vous optez pour le petit frère ATtiny25 et à multiplier par 2 si vous choisissez le grand frère ATtiny85.

Ce microcontrôleur est capable de produire de la PWM, dispose de timers, d'un convertisseur analogique-numérique sur 10 bits, de sources d'interruption externes et internes, un oscillateur interne calibré et un port SPI pour communiquer, ce qui permet de le programmer. À défaut d'être le mouton à 5 pattes, voici une petite merveille de technologie à 8 pattes et il serait dommage de s'en passer.

La figure 30.2 montre les fonctions des broches du microncontrôleur ATtiny45.

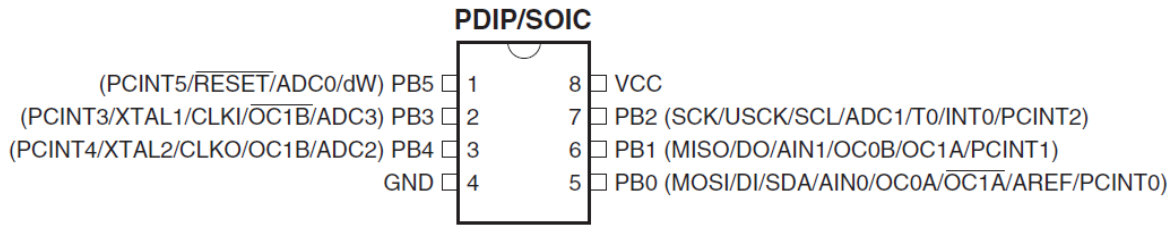


Figure 30. 2

Comme nous pouvons le voir, les broches d'entrées-sorties PB0:5 sont les broches 5, 6, 7, 2, 3, 1 et la broche 1 (PB5) sert au reset.

L'IDE d'Arduino

La version 1.6.12 de l'IDE d'Arduino contient ce qui concerne le microcontrôleur ATtiny ; il suffit d'aller dans le menu Outils, puis « Type de carte » et en bas on trouve l'option ATtiny qu'il suffit de cocher.

En retournant dans Outils, on voit maintenant sous la ligne « Type de carte : ATtiny » la ligne « Processeur » qui nous permet de choisir entre l'ATtiny45 et l'ATtiny85. Sous cette ligne se trouve la ligne « Clock » qui permet de choisir entre deux fréquences internes et trois fréquences externes. Nous reviendrons à tous ces réglages lorsque nous verrons la programmation.

Les deux dernières lignes du menu Outils sont également intéressantes : « Programmateur » qui jusque là était sur AVRISP mkII sera sélectionné sur Arduino as ISP ou si vous préférez « module Arduino utilisé en tant que programmateur ». La ligne suivante « Graver la séquence d'initialisation » sera aussi utilisé si vous programmez la puce ATtiny pour la première fois.

Il est évident que le microcontrôleur ATtiny45 peut être programmé et reprogrammé aussi souvent qu'on le souhaite.

Procédure de programmation

Nous allons commencer par découvrir la procédure de programmation et vous verrez que c'est très simple. Nous utiliserons, comme premier exemple, le programme Blink que nous allons charger dans le microcontrôleur. Pour faire cette manipulation, il vous faut :

- Un module Arduino Uno et son câble USB
- Un microcontrôleur ATtiny45
- Une breadboard
- Des câbles de liaison

Commencez par ouvrir le programme Blink (donné dans les exemples) dans votre IDE et changez le numéro de la broche sur laquelle est connectée la LED. Dans le programme Blink, il s'agit de la broche 13, mais l'ATtiny45 n'en possède pas autant, il faut donc changer 13 en 3 par exemple. Vous pouvez bien sûr choisir la broche que vous voulez de 0 à 4. Enregistrez ce nouveau programme sous le nom Blink_ATtiny. Le programme est prêt pour un ATtiny45.

Pour programmer un microcontrôleur, il faut un programmateur : ce sera le module Uno, il faut donc le programmer pour qu'il devienne un programmateur. Branchez le module à votre ordinateur et ouvrez l'IDE. À partir de maintenant, il est **indispensable de bien suivre la procédure sans sauter d'étapes**.

1. Téléversez dans le module le sketch appelé ArduinoISP (pour cela, allez dans Fichier > Exemples > 11.ArduinoISP > ArduinoISP). Votre module est devenu un programmateur ; lorsque vous n'en aurez plus besoin, vous pourrez téléversez un autre programme.
2. Débranchez le module Uno de l'ordinateur.
3. Sur la breadboard, réalisez le montage avec le microcontrôleur ATtiny45 et le module Uno selon la figure 30.3.
4. Rebranchez le module Uno à l'ordinateur.
5. Ouvrez dans l'IDE le programme que vous voulez transférer dans la puce ATtiny45 (ici Blink_ATtiny).
6. Réglage de l'IDE : dans Outils, sélectionnez « Type de carte : ATtiny » et dans « Processeur », sélectionnez ATtiny45. Dans « Clock », sélectionnez 8 MHz (internal). Toujours dans Outils,

sélectionnez « Programmateur > Arduino as ISP ». L'IDE est maintenant réglé mais il ne faut rien avoir oublié ! (Vous pouvez vérifier en rappelant le menu Outils)

7. Première programmation du microcontrôleur ATtiny : si la puce est programmée pour la première fois, allez dans Outils et faites « Graver la séquence d'initiation ».
8. Vérifiez votre programme (pour éviter une erreur de syntaxe) et ensuite téléversez le. Si un message d'erreur s'affiche, ne pas en tenir compte ; votre puce ATtiny45 est maintenant programmée et prête à l'emploi et vous pouvez récupérer votre module Uno pour autre chose.

Pour vérifier votre puce, il suffit de lui brancher une LED avec sa résistance de protection (330 ohms par exemple) sur la broche 3, puis de l'alimenter avec une pile de 4,5 V (ou bien les 5 V de votre module Uno).

Comme c'est un peu dommage d'utiliser un ATtiny45 pour ne faire clignoter qu'une seule LED, je vous invite à en faire clignoter 5 à des rythmes différents en vous inspirant du programme BlinkWithoutDelay ; vous pourrez ainsi animer votre réseau avec un feu tricolore clignotant en jaune pour indiquer un danger ou des travaux, un gyrophare en bleu de voiture de pompier, une enseigne de commerçant en vert (croix de pharmacie par exemple) et les deux autres à votre choix.

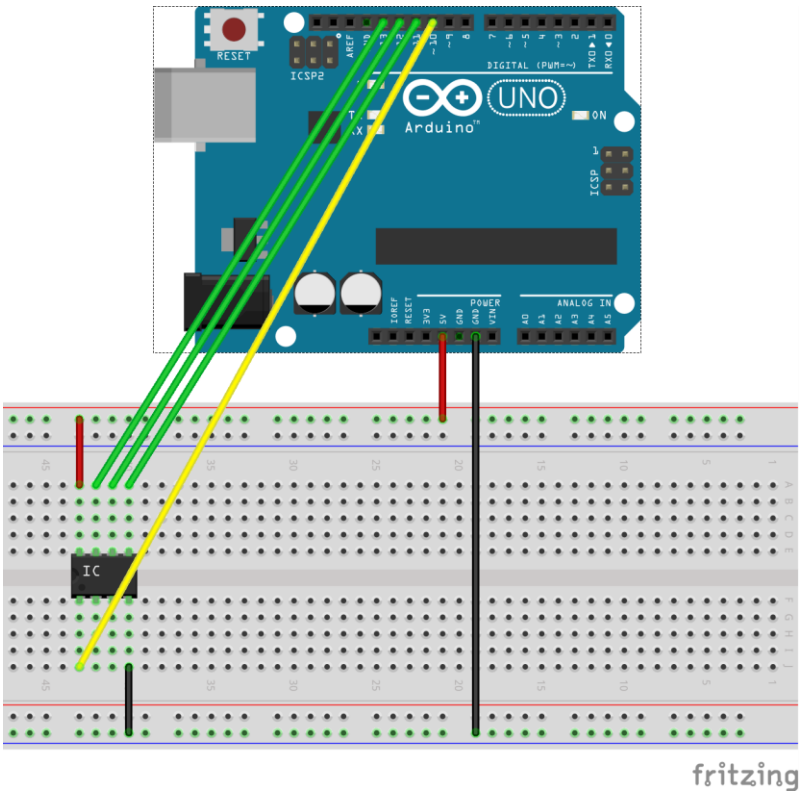


Figure 30. 3

Nota Bene : la figure 30.3 montre notre programmateur improvisé avec un module Uno. Vous pouvez remarquer un câble en jaune qui va vers la broche Reset qui est aussi la broche E/S PB5. Il y a moyen d'utiliser cette broche comme E/S et donc d'avoir 6 E/S sur le microcontrôleur, mais ceci nécessite de faire une manipulation qui prive le microcontrôleur de la possibilité de Reset externe. En conséquence, le microcontrôleur ne peut plus être programmé par ce moyen et il faut alors recourir à un vrai programmateur. Le mieux est donc de se passer de l'E/S PB5 et de conserver le Reset ou bien, si on a vraiment besoin de cette sixième E/S, être certain de ne pas avoir à reprogrammer la puce.

Exemples de montages à base d'ATtiny45

Voici pour compléter ce que vous venez d'apprendre, quelques petits montages à base de microcontrôleur ATtiny45. Je vous donne le programme et le schéma de montage sur une platine d'essai ; le programme doit être téléversé dans le microcontrôleur avant de réaliser le montage. Enfin, un lien vous permet de visualiser une vidéo montrant l'effet obtenu. Les programmes sont assez simples et n'ont pas besoin d'être commentés pour des lecteurs arrivés à ce niveau du cours.

Chenillard à 5 lampes

Schéma de montage :

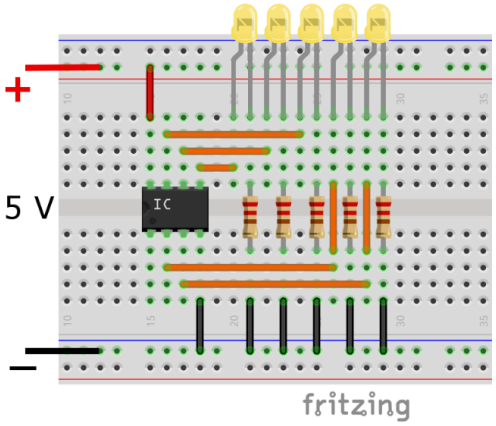


Figure 30. 4

Programme pour ATtiny45 :

Chenillard_ATTiny

```
// Chenillard_ATtiny.ino
/*
Ce programme réalise un chenillard.
Il fait flasher successivement cinq LED oranges reliées aux sorties 0 à 4
Puis il réalise une pause de 500 millisecondes avant de recommencer
*/

// Initialisation des lignes 0 à 4 en sortie
void setup () {
  pinMode (0, OUTPUT) ;
  pinMode (1, OUTPUT) ;
  pinMode (2, OUTPUT) ;
  pinMode (3, OUTPUT) ;
  pinMode (4, OUTPUT) ;
  // Extinction de toutes les LED au départ du programme
  for (byte i = 0 ; i <= 4 ; i++) {
    digitalWrite (i, LOW) ; // éteint la LED reliée à la broche i
  }
}

// Fonction loop
void loop () {
    // Boucle pour faire flasher les LED
  for (byte i = 0 ; i <= 4 ; i++) {
    digitalWrite (i, HIGH) ; // allume la LED sur broche i
    delay (50) ; // durée du flash 50 millisecondes
    digitalWrite (i, LOW) ; // éteint la LED
  }
    // délai de 500 millisecondes
  delay (500) ;
    // Recommence la séquence
}
```

Vidéo de démonstration : https://vimeo.com/144750455

Enseigne de commerçant

Schéma de montage :

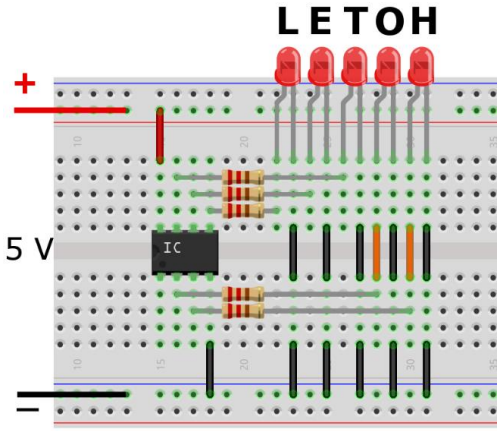


Figure 30. 5

Programme pour ATtiny45 :
L'initialisation et les fonctions setup et loop

Enseigne_ATtiny

```
int nbre_led = 5 ;     // Nombre de LED dans l'enseigne
int nbre_chen = 3 ;    // Nombre de fois pour le chenillard
int nbre_cligno = 5 ;  // Nombre de clignotement

void setup () {
// Initialisation des lignes de LED en sortie
for (byte i = 0; i <= (nbre_led - 1) ; i++)      // Moins 1 car on part de zéro
{
  pinMode ((i + led1), OUTPUT) ;
}
  // Extinction de toutes les LED au départ du programme
  for (byte i = 0 ; i <= (nbre_led - 1) ; i++)
  {
    digitalWrite ((i + led1), LOW) ; // éteint la LED reliée à la broche i + led1
  }
}

// Fonction loop
void loop () {

  // Séquence N°1 ; chenillard
  sequence_Un () ;
  delay (500) ; //delai de 500 millisecondes

  // Séquence N° 2 : cumul sur la gauche du mouvement de LED
  sequence_Deux () ;
  delay (500) ;  // délai de 500 millisecondes

  // Séquence N°3 : clignotement de l'ensemble 3 fois
  sequence_Trois () ;
  delay (2000) ;  // délai de 2 secondes

  // Séquence 4 : extinction successive de la gauche vers la droite
  sequence_Quatre () ;
  delay (2000) ;

  // Recommence la séquence
}
```

Les quatre fonctions

Enseigne_ATtiny

```
void sequence_Un ()
{
  // Séquence N°1 ; chenillard

  for (byte n = 1 ; n <= nbre_chen ; n++)
  {
    for (byte i = 0 ; i <= (nbre_led - 1) ; i++)
    {
      digitalWrite ((i + led1), HIGH) ;    // allume la LED sur broche i
      delay (200) ;                        // durée du flash 200 millisecondes
      digitalWrite ((i + led1), LOW) ;     // éteint la LED
      delay (100) ;
    }
  }
  return ;
}

void sequence_Deux ()
{
  // Séquence N° 2 : cumul sur la gauche du mouvement de LED

  for (byte n = 0 ; n <= (nbre_led - 1) ; n++)
  {
    for (byte i = led1 ; i <= ((led1 + nbre_led - 1)-n) ; i++)
    {
      digitalWrite (i, HIGH) ;   // allume la LED sur broche i
      delay (200) ;              // durée du flash 150 millisecondes
      digitalWrite (i, LOW) ;    // éteint la LED
    }
    digitalWrite ( ((led1 + nbre_led - 1) - n) , HIGH ) ; //dernière LED reste allumée
  }
  return ;
}
```

Enseigne_ATtiny

```
void sequence_Trois ()
{
  // Séquence N°3 : clignotement de l'ensemble

  for (byte j = 1 ; j <= nbre_cligno ; j++)
  {
    for (byte k = 0 ; k <= (nbre_led - 1) ; k++)
    {
      digitalWrite ((k + led1) , LOW) ;
    }
    delay (500) ;
    for (byte l = 0 ; l <= (nbre_led - 1) ; l++)
    {
      digitalWrite ((l + led1), HIGH) ;
    }
    delay (500) ;
  }
  return ;
}

void sequence_Quatre ()
{
  // Séquence 4 : extinction successive de la gauche vers la droite

  for (byte i = (led1 + nbre_led - 1) ; i >= led1 ; i--)
  {
    digitalWrite (i, LOW) ;
    delay (200);
  }
  return ;
}
```

Le programme utilise quatre fonctions définies par le programmeur, comme « sequence_Un ». Vous vous apercevez que le montage est exactement le même que pour le chenillard ; comme je vous le disais, il suffit de changer le programme pour obtenir un autre comportement, force de l'électronique programmable.

Vidéo de démonstration : https://vimeo.com/147948985

Simulateur de soudure à arc

Schéma de montage :

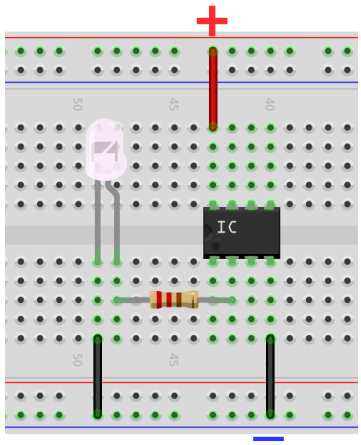


Figure 30. 6

Programme pour ATtiny45 :

L'initialisation

Sim_Arc_ATtiny

```
/* Ce programme commande une LED blanche pour simuler un poste de soudure à l'arc.
   Il fait appel à la fonction random qui génère des nombres aléatoires.
   Duree_flash est la durée d'un flash. Duree_int est la durée entre deux flashes.
   Duree_flash et Duree_int forment un événement.
   Nbre_even est le nombre d'événements au cours d'un cycle de soudure.
   P_repos est la durée entre deux cycles de soudure.
   Toutes ces données sont aléatoires mais bornées entre une valeur minimum et une
   valeur maximum. En jouant sur l'intervalle, on peut simuler au mieux l'effet de
   soudure à l'arc. Broche est la broche sur laquelle la LED est connectée, ici broche 3.
*/
// Initialisation des variables
byte Broche = 3 ;
long Duree_flash ;
long Duree_flash_mini = 10 ;
long Duree_flash_maxi = 101 ;
long Duree_int ;
long Duree_int_mini = 10 ;
long Duree_int_maxi = 31 ;
long Nbre_even ;
long Nbre_even_mini = 10 ;
long Nbre_even_maxi = 21 ;
long P_repos ;
long P_repos_mini = 1500 ;
long P_repos_maxi = 7001 ;
long i ;
```

Les fonctions setup et loop

Sim_Arc_ATtiny

```
// Fonction d'initialisation
void setup ()
{
  randomSeed (analogRead (2)) ;
  pinMode (Broche, OUTPUT) ;
}

// Corps du programme
void loop ()
{
 Nbre_even = random (Nbre_even_mini, Nbre_even_maxi) ;
 for (i = 1 ; i <= Nbre_even ; i++)
   {
     Duree_flash = random (Duree_flash_mini, Duree_flash_maxi) ;
     Duree_int = random (Duree_int_mini, Duree_int_maxi) ;
     digitalWrite (Broche, HIGH) ;
     delay (Duree_flash) ;
     digitalWrite (Broche, LOW) ;
     delay (Duree_int) ;
    }
 P_repos = random (P_repos_mini, P_repos_maxi) ;
 delay (P_repos) ;
}
```

Voici tout de même, quelques explications sur ce programme. La figure 30.7 montre un train d'impulsions émis par le programme, pour allumer la LED. Ce train d'impulsion est appelé **cycle** de soudure. Le cycle est constitué d'un certain nombre d'**événements**, chaque événement étant constitué d'un **flash** d'une certaine durée, suivi d'une **période intermédiaire** d'une certaine durée également, au cours de laquelle aucune lumière n'est émise. Entre deux cycles existe une **période de repos**, pendant laquelle le soudeur contrôle son travail ou bien positionne sa pièce autrement.

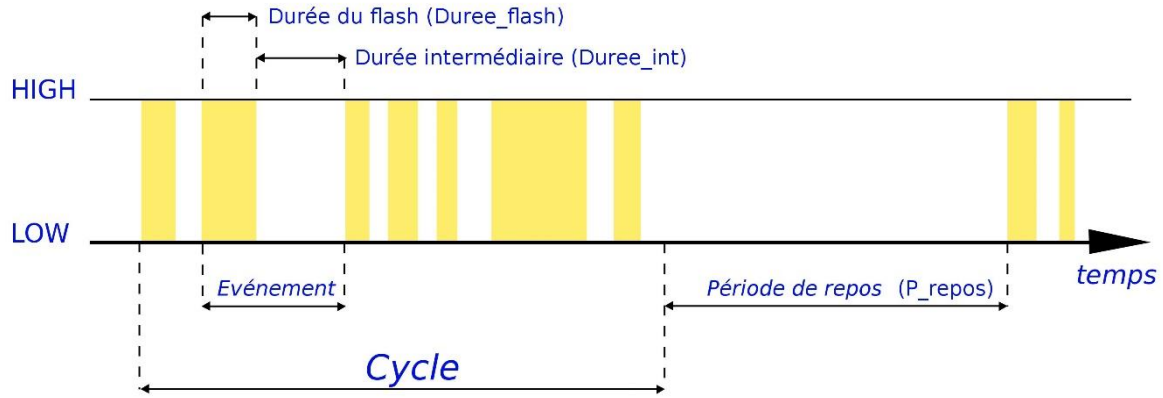


Figure 30. 7

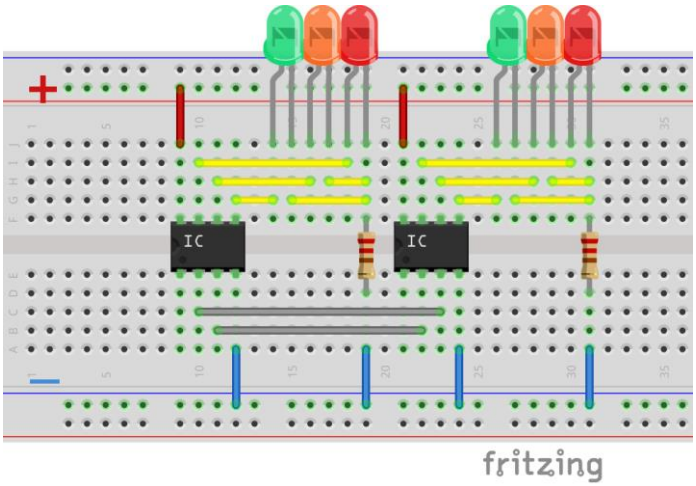
Le programme permet de **générer de façon aléatoire**, les périodes entre deux cycles, pour chaque cycle le nombre d'événements, pour chaque événement les durées d'allumage et de repos de la LED. Le résultat est un comportement complètement aléatoire des trains d'impulsions. Tout est réglable de manière à obtenir le fonctionnement le plus crédible possible lié à une activité humaine, mais cela permet aussi d'avoir des comportements différents pour plusieurs postes de soudure à arc, si le montage est reproduit en plusieurs exemplaires sur le même réseau. Chaque donnée générée par le programme de façon aléatoire, a sa valeur comprise entre une valeur minimum et une valeur maximum. C'est en jouant sur les valeurs minimum et maximum pour chaque paramètre, qu'on peut modifier le comportement de notre simulateur.

Cela vaut la peine de tester la faisabilité d'un montage avant de conclure que ce n'est pas possible. Si on se réfère au site Arduino, seules quelques fonctions sont utilisables sur un ATtiny. Pourtant, le programme du simulateur de soudure à arc fonctionne parfaitement, bien qu'il fasse appel à des fonctions générant des nombres aléatoires... En électronique câblée, il fallait trois circuits intégrés et de nombreux autres composants autour pour réaliser la même chose, en électronique programmable trois composants suffisent.

Vidéo de démonstration : https://vimeo.com/142787726

Feux tricolores de carrefour routier

Schéma de montage :



Programme pour ATtiny45 :
L'initialisation et la fonction setup

Feu_tricolore_ATtiny

```
/*
Ce programme fait fonctionner un feu tricolore A qui est
synchronise avec un feu tricolore B, chaque feu etant
commande par un µC ATtiny45.
Trois LED (verte, orange et rouge) sont reliées aux sorties 0 à 2.
L entree 3 recoit le signal d inhibition de la part de l autre feu.
La sortie 4 envoie le signal d inhibition vers l autre feu.
Trois LED (verte, orange et rouge) sont reliées aux sorties 0 à 2.
*/

// Initialisation des variables

const byte vert = 0 ;
const byte orange = 1 ;
const byte rouge = 2 ;
const byte line_IN = 3 ;
const byte line_OUT = 4 ;
const int attente_synchro = 1000 ; // 3000 pour deuxième feu B
const int attente_vert = 10000 ;
const int attente_orange = 3000 ;
const int attente_chauffard = 2000 ;

// Initialisation des lignes 0 a 4
void setup () {
  pinMode (vert, OUTPUT) ;
  pinMode (orange, OUTPUT) ;
  pinMode (rouge, OUTPUT) ;
  pinMode (line_IN, INPUT) ;
  pinMode (line_OUT, OUTPUT) ;
  // Extinction de toutes les LED au depart sauf rouge
  digitalWrite (vert, LOW) ;
  digitalWrite (orange, LOW) ;
  digitalWrite (rouge, HIGH) ;
  // decalage des sequences de chaque feu
  delay (attente_synchro) ;
}
```

La fonction loop

Feu_tricolore_ATtiny

```
// Fonction loop
void loop () {

  // Attente fin du cycle de l autre feu
  while (digitalRead (line_IN) == HIGH) {
    // Ne rien faire
  }
  // Cycle du feu tricolore

  // Ce feu prend la main
  digitalWrite (line_OUT, HIGH) ;

  // Debut de cycle
  delay (attente_chauffard) ;
  digitalWrite (rouge, LOW) ;
  digitalWrite (vert, HIGH) ;
  delay (attente_vert) ;
  digitalWrite (vert, LOW) ;
  digitalWrite (orange, HIGH) ;
  delay (attente_orange) ;
  digitalWrite (orange, LOW) ;
  digitalWrite(rouge, HIGH) ;

  // Ce feu rend la main
  digitalWrite (line_OUT, LOW);
  delay (500) ;

  // L autre feu peut travailler, line_IN sera a HIGH, ce feu va attendre

}
```

Ce programme, un peu plus compliqué que les autres, mérite quelques explications. Un des feux doit commencer son cycle le premier ; comme ils sont alimentés en même temps sur le montage, il est nécessaire que le deuxième feu soit retardé par rapport au premier. Chaque microcontrôleur reçoit le même programme, à une petite différence près, le délai de synchronisation (variable attente_synchro) qui est de 1 seconde pour le feu A et de 3 secondes pour le feu B. En conséquence, le feu B devrait commencer son cycle deux secondes après le feu A. Mais entre-temps, le feu A a pris la main et empêche le feu B de commencer son cycle. Lorsque le feu A a terminé son cycle, il autorise le feu B à faire le sien. Celui-ci empêche alors le feu A de continuer son cycle.

Un cycle est constitué uniquement des séquences feu au vert (10 s) et feu à l'orange (3 s) car dès que le feu passe au rouge, il autorise l'autre feu à travailler et maintiendra son feu au rouge puisqu'il sera empêché par l'autre feu de continuer son travail. La temporisation du chauffard (2 s) fait que les deux feux sont au rouge pendant ce court délai, ce qui permet au chauffard qui a accéléré à l'orange et passé au rouge de dégager le carrefour avant d'autoriser les automobilistes de l'autre route à redémarrer.

Les DEL sont sur les sorties 0, 1 et 2. Les E/S 3 et 4 sont utilisées pour « communiquer ». En effet, lorsqu'un microcontrôleur prend la main sur l'autre, il positionne la sortie 4 à HIGH (5 V). Cette sortie est reliée à une entrée 3 de l'autre microcontrôleur qui sait que s'il y a 5 V sur cette entrée, il ne doit rien faire. Comme vous le constatez, chaque microcontrôleur utilise 5 E/S (numérotées de 0 à 4) et la sixième est conserver pour le reset du µC.

Cycle de fonctionnement

Le setup se termine par les deux feux au rouge puis l'attente de synchronisation (1 s pour feu A et 3 s pour feu B). Le feu A commence avant le feu B.

Pour chaque feu, le cycle réalise cela :

- Vérifie qu'il peut travailler, ou attend que l'autre feu ait terminé (état de l'entrée 3)
- Prend la main pour bloquer l'autre feu
- Attend la temporisation du chauffard
- Eteint le rouge et allume le vert
- Attend délai du vert (10 s)
- Eteint le vert et allume l'orange
- Attend le délai de l'orange (3 s)
- Eteint l'orange et allume le rouge
- Passe la main à l'autre feu en mettant sa sortie 4 à LOW (qui est l'entrée 3 de l'autre feu)

Le feu B qui commence en retard, est empêché de travailler par le feu A jusqu'à ce que celui-ci lui passe la main.

Comme vous le voyez, ce montage fait appel à deux microcontrôleurs « communiquant » entre eux ; nous sommes dans le domaine du multiprocesseur. **Pensez à régler la valeur de la variable attente_synchro à 3000 ms (3 secondes) pour programmer la deuxième puce**. Ces deux valeurs ne sont pas critiques, simplement une doit être supérieure à l'autre pour que les deux feux exécutent leurs cycles à deux instants différents.

Vidéo de démonstration : https://vimeo.com/160229154

À retenir sur les microcontrôleurs ATtiny45 :

- Leurs possibilités sont énormes en regard de leur coût.
- Ils peuvent servir à gérer de petites applications qui ne demandent pas beaucoup de ressources sur un réseau de trains miniatures.
- Ils sont facilement programmables avec un simple module Arduino Uno.
- Comme pour les modules Uno, les signaux d'entrée doivent être adaptés et les signaux de sortie amplifiés.
- Certaines fonctions de programmation ne seront peut-être pas possibles sur un ATtiny45, mais cela vaut la peine d'essayer pour en être certain.

J'espère que ces quelques exemples vous auront donné envie de franchir le pas et de programmer vos puces afin de les utiliser dans des montages électroniques, forcément plus simples et moins coûteux que les mêmes montages en électronique câblée.

Liste des composants nécessaires pour :

Le chenillard :

Un ATtiny45 programmé

Cinq DEL jaunes ou oranges

Cinq résistances de 220 Ω

L'enseigne de commerçant :

Un ATtiny45 programmé

Cinq DEL rouges

Cinq résistances de 220 Ω

Le simulateur de soudure à arc :

Un ATtiny45 programmé

Une DEL blanche

Une résistance de 220 Ω

Les feux tricolores de carrefour routier :

Deux ATtiny45 programmés (deux programmes différents comme indiqué dans le texte)

Deux DEL vertes, deux oranges et deux rouges

Deux résistances de 220 Ω