Starter-kit
Flip & Click

Cher Client,

Nous vous remercions pour votre achat.

Ce starter-kit et cet ouvrage sont spécialement conçus pour vous permettre de développer rapidement et simplement de nombreuses applications à partir de la platine Flip & Click.

Nous espérons que vous vous amuserez et que vous vous instruirez au fil des différentes réalisations que nous vous avons préparé.

Bonne lecture et bon développement à tous !

Lextronic

Remerciements spéciaux à Alexandre Jr - Catherine - Nathalie - Myriane - Marijana - Tiziano - Danijela - Aleksandar

Table des matières

Introduction

Qu'est-ce qu'une plateforme compatible arduino™ ? 04
La platine « Flip & Click » 05

Détail de la platine

Description générale 06
Caractéristiques et schéma théorique 07
Correspondance du brochage de la platine 08

Détail du contenu du starter-kit

Description détaillée des différents éléments 09

Installation de l'environnement de développement

Détail de l'installation 12
Configuration de l'environnement de développement 13

Votre premier programme

Description et explications 14
Informations complémentaires 17

Applications 001 à 025

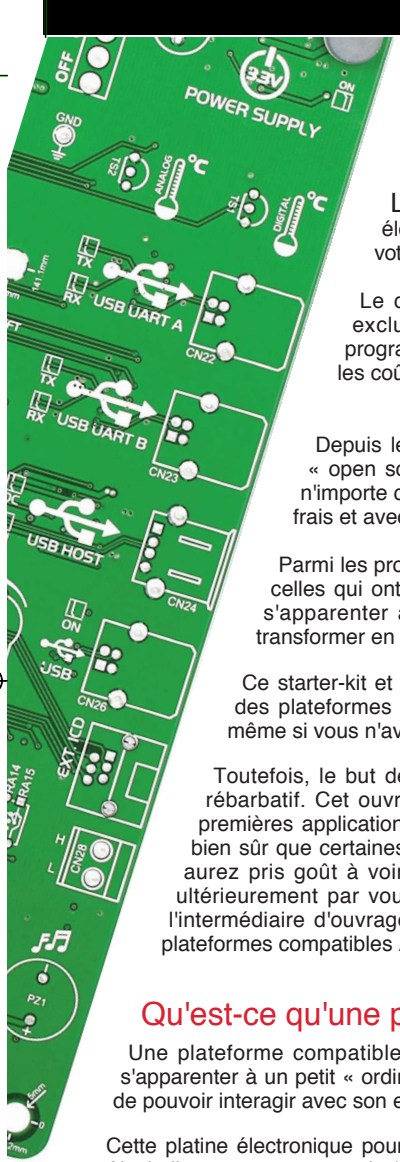
001: Leds et boutons-poussoirs 18
002: Feu tricolore / Feu tricolore à vitesse variable 20
003: Gestion des rebonds sur un bouton-poussoir 23
004: Chenillard à leds « aller - retour » 24
005: Roulette électronique 26
006: Gradateur pour leds 28
007: Interrupteur crupusculaire 32
008: La fuite d'eau infernale 34
009: Dé électronique 36

Table des matières

010: Générateur de code morse 38
011: Pilotage d'un servomoteur 42
012: Thermostat à leds 44
013: Récupération des données d'un capteur DHT22 47
014: Thermomètre sur écran LCD 48
015: Compteur/décompteur sur afficheur 7 segments à leds 50
016: Mini-centrale d'alarme 2 zones à écran LCD 52
017: Mini-centrale de mesure 58
018: Pilotage d'une matrice à leds "7 x 10 R Click" 60
019: Sablier animé avec module "7 x 10 R Click" 64
020: Jeu vidéo Tennis (1 joueur) 67
021: Jeu vidéo Tennis (2 joueurs) 72
022: Journal lumineux défilant 74
023: Petite station météorologique 78
024: Pilotage d'une led RGB 81
025: Pierre - Feuille - Ciseau avec module "7 x 10 R Click" 82

Applications 026 à 037 et plus...

026: Utilisation du module microSD Click (**MIKROE-924**) 84
027: Utilisation du module 7Seg Click (**MIKROE-1201**) 85
028: Utilisation du module BarGraph Click (**MIKROE-1423**) 86
029: Utilisation du module Tilt Click (**MIKROE-1834**) 87
030: Utilisation du module TouchKey Click (**MIKROE-1906**) 88
031: Utilisation du module Ambiant Click (**MIKROE-1880**) 89
032: Utilisation du module Button R Click (**MIKROE-1901**) 90
033: Utilisation du module 4x4 RGB Click (**MIKROE-1881**) 92
034: Utilisation du module LED ring R Click (**MIKROE-2153**) 93
035: Master Chef Démo 95
036: Breathalyzer Démo 95
037: Weather Station Démo 95
Utilisation de la platine Flip & Click avec Python™ 96



Introduction

Le monde qui nous entoure est « numérique ». L'ensemble des objets de notre quotidien renferme une électronique « intelligente ». Que l'on parle de votre réveil, de votre voiture, de votre smartphone ou de votre lave-linge, tous ces produits intègrent un cerveau (appelé microcontrôleur) qui gère leurs fonctionnements.

Le développement et la conception de tels produits étaient il y a encore quelques années de cela exclusivement réalisable que par des d'ingénieurs hautement qualifiés (en raison des techniques de programmation qu'il fallait savoir maîtriser) et également exclusivement réservées à certaines sociétés (de part les coûts prohibitifs des outils de développement qu'il fallait posséder).

Depuis les choses ont bien changé. En effet, grâce à l'arrivée de nouvelles plateformes microcontrôlées dites « open source » et de leurs outils de développement disponibles en libre accès, il est désormais possible « à n'importe qui » de pouvoir concevoir ses propres objets « intelligents » très rapidement, très simplement, à moindre frais et avec des connaissances techniques limitées.

Parmi les produits à l'origine de cette petite révolution, les plateformes compatibles Arduino™ sont incontestablement celles qui ont fait le plus progresser les choses dans ce domaine. Les plateformes compatibles Arduino™ peuvent s'apparenter à un petit cerveau que vous pourrez programmer pour qu'il devienne « intelligent » et puisse se transformer en un produit électronique utile à votre quotidien.

Ce starter-kit et l'ouvrage qui l'accompagne sont spécialement conçus pour vous permettre de découvrir le « monde » des plateformes compatibles Arduino™ en vous permettant de développer vos premières applications intelligentes.... même si vous n'avez aucune connaissance particulière en terme de programmation informatique.

Toutefois, le but de cet ouvrage n'est pas de vous proposer un cours d'électronique et de programmation complet et rébarbatif. Cet ouvrage est conçu avec une approche différente... celle de pouvoir vous permettre de développer vos premières applications immédiatement sans vous « prendre la tête » avec des notions théoriques complexes. Il en résulte bien sûr que certaines notions de base ne seront volontairement pas abordées. Nous pensons en effet qu'une fois que vous aurez pris goût à voir se concrétiser vos premiers montages, vous aurez alors envie d'approfondir vos connaissances ultérieurement par vous-même sur certains points laissés en suspens (chose que vous pourrez faire très facilement par l'intermédiaire d'ouvrages techniques spécialisés complémentaires ou par la visite des nombreux sites internet dédiés aux plateformes compatibles Arduino™).

Qu'est-ce qu'une plateforme compatible Arduino™ ?

Une plateforme compatible Arduino™ est une platine électronique (généralement livrée « nue » sans boîtier), laquelle peut s'apparenter à un petit « ordinateur ». Cette dernière est conçue pour pouvoir être programmée par vos soins afin d'avoir la capacité de pouvoir interagir avec son environnement.

Cette platine électronique pourra ainsi récupérer et analyser les informations issues des dispositifs sur lesquels elle sera raccordée. Ainsi elle pourra par exemple (grâce à différents capteurs additionnels) connaître la température ambiante de votre chambre ou encore savoir si la porte d'entrée de votre maison est ouverte ou encore vérifier si le jour se lève, etc...

Dans le même ordre d'idée, une plateforme compatible Arduino™ pourra piloter des dispositifs externes à l'aide d'interfaces de puissance additionnelles (par exemple pour allumer une lampe, faire tourner un moteur, générer des sons, etc...).

De part son intelligence embarquée, une plateforme compatible Arduino™ sera donc capable (grâce à votre programmation) de pouvoir exploiter les informations qu'elle reçoit de l'extérieur pour agir en conséquence sur des dispositifs externes.

Reprenons l'exemple de la machine à laver que nous avions cité en introduction. Une plateforme compatible Arduino™ pourrait tout à fait gérer son fonctionnement. Elle pourrait ainsi vérifier la position des boutons de commande de la machine et en fonction du mode choisi par ces boutons, actionner la rotation du tambour de la machine, puis une vanne pour l'arrivée de l'eau, puis après une certaine temporisation stopper l'ensemble et vous signaler que votre linge est propre à l'aide d'un voyant !

Il existe de nombreux modèles de platines compatibles Arduino™ qui se distinguent par leur « puissance » de calcul (leur rapidité à pouvoir exécuter des actions), ainsi que par leur capacité de stockage mémoire (c'est à dire la possibilité qu'il vous est donné de pouvoir écrire des programmes plus ou moins importants au sein de ces dernières) ainsi que par leur nombre de broches que vous pourrez utiliser pour interagir avec des dispositifs externes.

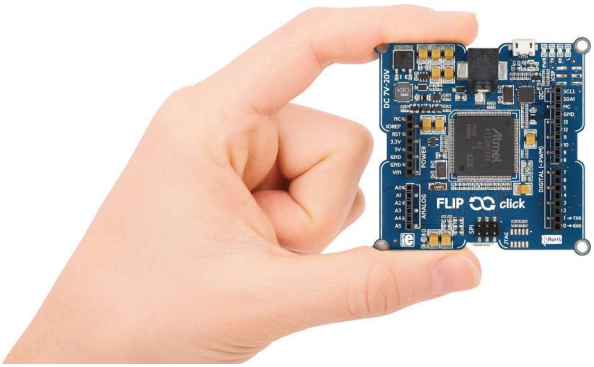
La plupart des plateformes compatibles Arduino™ disposent de connexions de raccordement généralement placées aux mêmes endroits. Ces dernières sont destinés être programmées par un ordinateur compatible PC (sous environnement Windows™ ou Linux™) ou par un Mac™. Elles se raccordent à votre ordinateur grâce une liaison USB et se programment via un environnement de développement IDE (pour Integrated Developement Environment en Anglais) librement téléchargeable via Internet. La programmation de l'application que vous développerez sur les platines compatibles Arduino™ s'effectue en langage « C » … bien qu'il soit possible d'utiliser (dans certains cas) d'autres types de langage comme Scratch™ ou Python™ (voir plus d'informations à la fin de cet ouvrage).

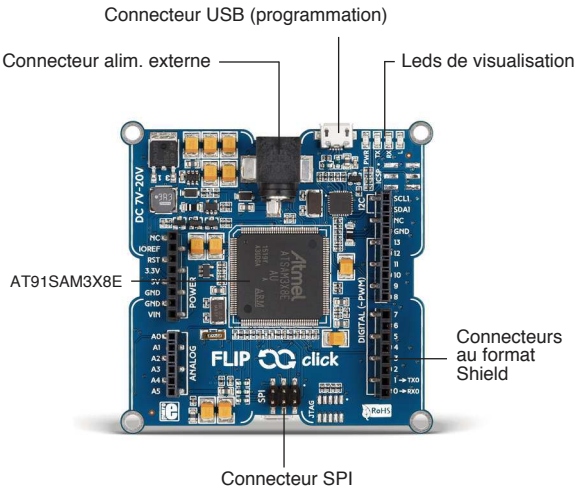
Les platines compatibles Arduino™ sont à la base directement alimentées par votre ordinateur lorsque vous les programmez (via la liaison USB) . Une fois votre programme complètement mis « au point » et terminé, vous pourrez bien sûr déconnecter les platines compatibles Arduino™ de l'ordinateur et les alimenter via une source externe (piles, batteries, adaptateurs secteur) afin de les rendre autonomes et que le programme que vous avez développé puisse s'exécuter en leur sein.

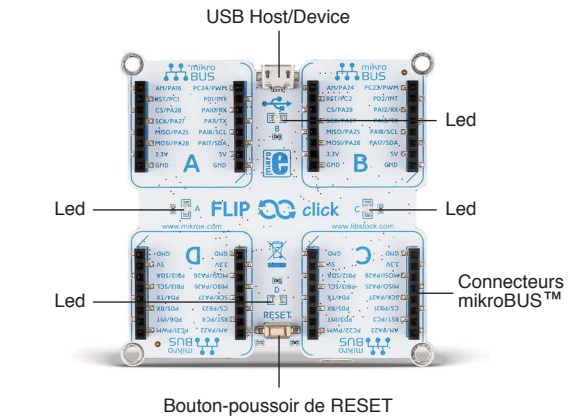
La platine « Flip & Click »

La plateforme livrée avec votre starter-kit est une platine électronique appelée « **Flip & Click** ». Celle-ci s'apparente à une platine compatible Arduino™ mais elle fait figure de version **très nettement améliorée** par rapport à la plupart des modèles que l'on rencontre généralement dans le commerce.

Compatible avec l'environnement de développement IDE des plate-formes Arduino™, elle est conçue sur la base d'un microcontrôleur beaucoup plus puissant que la moyenne (un modèle Atmel™ 32 bits MCU AT91SAM3X8E) qui permettra à cette dernière de réaliser des actions plus rapidement que sur la plupart des autres platines compatibles Arduino™ et également de pouvoir recevoir des programmes de tailles **beaucoup plus importantes** que sur la plupart des autres platines compatibles Arduino™ .







Détail de la platine

Cette dernière dispose de 2 faces exploitables. Sur la première face, vous avez accès à :

- Une connexion USB vous permettant de la relier à votre ordinateur pour la programmer (et l'alimenter durant la phase de développement de votre application).
- Des connecteurs femelles reprenant une partie des broches destinées à relier la platine « Flip & Click » à son environnement externe. Le positionnement et les dimensions de ces broches sont « standardisés » et compatibles avec la plus « célèbre » des platines arduino™ à savoir la platine arduino™ uno . Dès lors, il vous sera possible d'enficher des platines d'extensions (appelées Shield) sur la platine « Flip & Click » afin de pouvoir lui adjoindre de nouvelles possibilités. Notez toutefois que de part les niveaux de tension de sorties des broches de la platine « Flip & Click », seuls les modèles compatibles 3,3 V devront être utilisés.
- Diverses leds de visualisation.
- Un connecteur destiné à alimenter la platine via une tension externe (lorsque celle-ci n'est plus reliée à votre ordinateur).
- Un connecteur permettant une programmation SPI (pour une utilisation plus aboutie).

En retournant la platine « Flip & Click », celle-ci vous fera voir une autre de ses facettes !

A savoir :

- 4 leds reliées à certaines broches de la platine.
- 1 connecteur USB de type Host/Device.
- 4 connecteurs avec une implantation au format mikroBUS™.
- Un bouton-poussoir de RESET.

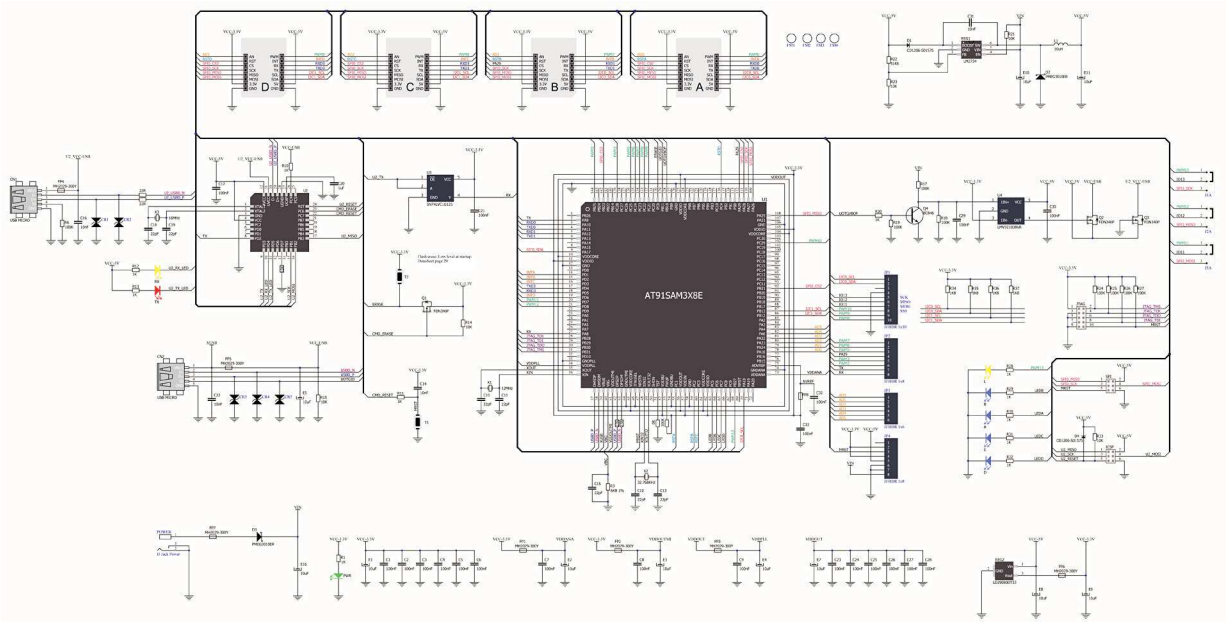
Les connecteurs au format mikroBUS™ vous permettront de pouvoir enficher et utiliser plus de **200 petits modules** d'extension optionnels appelés **Click Board™** (à acquérir séparément), lesquels vous permettront d'étendre encore plus les possibilités de votre platine « Flip & Click ».

Ainsi, en un instant, vous pourrez ajouter par exemple à votre platine « Flip & Click »:

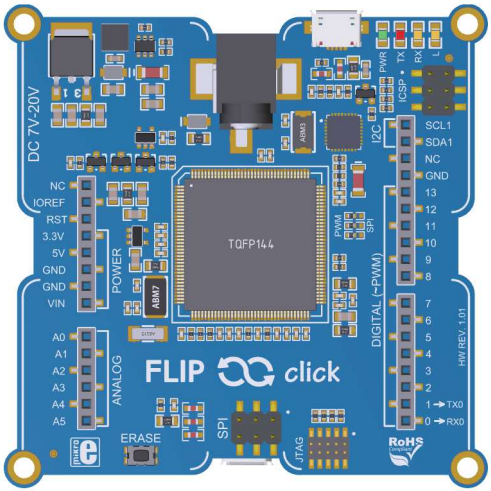
Un récepteur GPS, une matrice à leds, un module Bluetooth™ ou GSM ou WiFi, un accéléromètre 3 axes, une boussole électronique, un capteur de température ou d'humidité, une interface pour moteur "cc" ou pas-à-pas, un mini joystick, une interface RS232/485, un module de reconnaissance vocale, un bargraph à leds, des afficheurs graphiques, une interface pour bus CAN, un connecteur pour microSD™, un capteur capacitif, une horloge RTC, un capteur de gaz, un capteur de proximité, un capteur d'UV, etc... A titre d'information, votre starter-kit est livré avec 2 modules Click Board.

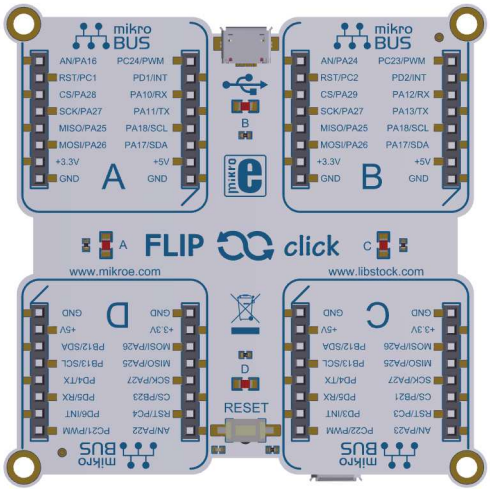
Caractéristiques techniques

- MCU 32-bit AT91SAM3X8E cadencé à 84 MHz
- 512 KB (Flash) + 100 KB (SRAM)
- Niveau logique des ports d'E/S (3,3 V)
- 1 x USB Host/Device + 1 x USB (port COM virtuel)
- 5 leds utilisables + 1 bouton-poussoir de Reset
- Connecteurs au format Shield
- 4 connecteurs au format mikroBUS ™
- Connecteur pour alimentation externe 7 à 15 Vcc
- 22 ports d'entrées/sorties (face bleue)
- 33 ports d'entrées/sorties (face blanche)
- Dimensions 73 x 73 mm



Correspondance du brochage de la platine





Support Click "A"

AN/PA16 : Port 54
RST/PC1 : Port 33
CS/PA28 : Port 77
SCK/PA27 : Port 76
MISO/PA25 : Port 74
MOSI/PA26 : Port 75

PC24/PWM : Port 6
PD1/INT : Port 26
PA10/RX : Port 15
PA11/TX : Port 14
PA18/SCL : Port 21
PA17/SDA : Port 20

Support Click "B"

AN/PA24 : Port 55
RST/PC2 : Port 34
CS/PA29 : Port 84
SCK/PA27 : Port 76
MISO/PA25 : Port 74
MOSI/PA26 : Port 75

PC23/PWM : Port 7
PD2/INT : Port 27
PA12/RX : Port 17
PA13/TX : Port 16
PA18/SCL : Port 21
PA17/SDA : Port 20

Support Click "C"

AN/PA23 : Port 56
RST/PC3 : Port 35
CS/PB21 : Port 52
SCK/PA27 : Port 76
MISO/PA25 : Port 74
MOSI/PA26 : Port 75

PC22/PWM : Port 8
PD3/INT : Port 28
PD5/RX : Port 15
PD4/TX : Port 14
PB13/SCL : Port 21
PB12/SDA : Port 20

Support Click "D"

AN/PA22 : Port 57
RST/PC4 : Port 36
CS/PA23 : Port 78
SCK/PA27 : Port 76
MISO/PA25 : Port 74
MOSI/PA26 : Port 75

PC21/PWM : Port 9
PD6/INT : Port 29
PD5/RX : Port 15
PAD4/TX : Port 14
PB13/SCL : Port 21
PB12/SDA : Port 20

Led face bleue : Port 13

Led A : Port 38
Led B : Port 37
Led C : Port 39
Led D : Port 40

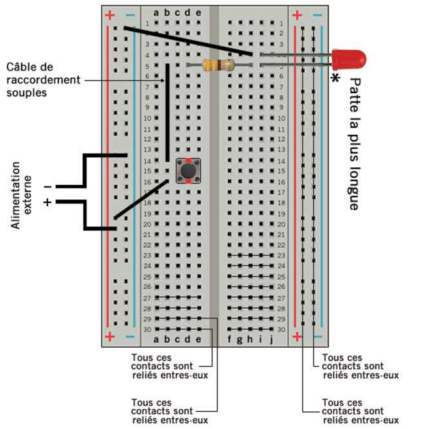
La broche 4 de la face bleue nécessite l'utilisation du port 87 pour être exploitée.

Contenu de votre starter-kit

Avant d'apprendre à utiliser la platine « Flip & Click », voyons le détail du contenu de votre starter-kit. Celui-ci est composé :

- D'une boite de rangement
- D'une platine « Flip & Click »
- D'un câble de raccordement USB
- D'un jeu de différents mini-cordons de raccordement mâle/mâle
- D'une plaque de raccordement rapide sans soudure
- D'un sachet renfermant divers composants électroniques
- D'un module d'extension « 7x10 R Click »
- D'un module d'extension « DHT22 Click »

Description détaillée des différents éléments:
A noter que suivant les approvisionnements, la représentation et les caractéristiques techniques des éléments qui composent votre starter-kit peuvent varier très légèrement par rapport aux indications de ces pages.



Exemple de câblage d'un bouton-poussoir permettant d'allumer une led

Plaque de connexion sans soudure
Ce type de plaque (encore appelé Breadboard) est spécialement conçu pour le prototypage rapide. En temps normal, un système électronique est composé d'un circuit imprimé sur lequel sont soudés différents composants électroniques.

Si vous désirez expérimenter plusieurs montages, il n'est pas concevable d'avoir à réaliser à chaque fois un tel circuit imprimé et d'avoir à y souder vos composants. Ceci vous prendrait trop de temps et reviendrait trop cher. L'utilisation de la plaque Breadboard vous permettra de relier divers composants entres eux et de réaliser de nombreux montages sans soudure. Vous pourrez en plus démonter les montages une fois terminés et récupérer vos composants afin de pouvoir les utiliser pour d'autres réalisations.

Une Breadboard est composée de multiples connecteurs femelles placés les uns à côté des autres (dans lesquels vous pourrez enficher vos composants). Plusieurs de ces connecteurs sont reliés entres eux (dans le fond de la plaque) suivant des regroupements horizontaux et verticaux. Les connexions sont symétriques d'un côté et de l'autre de la plaque avec une séparation centrale (les connexions à droite et à gauche de la plaque ne sont pas reliées entre-elles). Afin de pouvoir relier certaines de ces connexions entres elles, vous devrez utiliser les mini-câbles souples « mâle/mâle » livrés dans le starter-kit.

10 leds vertes - 10 leds rouges - 10 leds oranges

Une led (appelée Diode électroluminescente ou DEL) s'apparente à une diode qui s'illumine lorsqu'elle est alimentée. Au même titre qu'une diode, une led s'utilise selon un sens précis. Son anode (généralement repérable par la patte la plus longue) doit être reliée au «plus» et l'autre patte (la plus courte) à la «masse» avec IMPERATIVEMENT une résistance en série pour limiter le courant qui traverse la led (sous peine de destruction de la led). L'inversion du sens de la led ne pourra toutefois pas la détruire.

1 led RVB

Ce composant est en fait composé de 3 leds intégrées dans le même boitier (1 rouge - 1 verte - 1 bleue). Le modèle livré dans votre starter-kit dispose d'une cathode commune aux 3 leds. En alimentant chacune des anodes (avec une résistance en série), il sera ainsi possible d'obtenir différentes couleurs.

1 afficheur 7 segments à leds

Ce dernier est composé de 7 leds dont les emplacements vous permettront (en fonction des leds allumées) de représenter des chiffres de 0 à 9 et même quelques lettres. Le modèle livré dispose d'une cathode commune à toutes ses leds. Il est également possible d'utiliser une 8 ème led pour afficher un point de séparation.

1 afficheur alphanumérique à cristaux liquides

Doté de 2 lignes de 16 caractères et d'un dispositif de rétro-éclairage bleu, ce dernier vous permettra d'afficher des messages (textes, chiffres, caractères spéciaux). Il est livré avec un connecteur mâle qu'il vous faudra souder afin que vous puissiez l'enficher sur la Breadboard.

20 résistances

Ces composants de différentes valeurs n'ont pas de « sens » lorsque vous les utilisez. La valeur d'une résistance peut être connue grâce aux bandes de couleurs qui sont présentes sur son corps. Vous disposez ainsi de:

- 10 résistances de valeur 330 ohms (couleur: orange - orange - marron).
- 10 résistances de valeur 10 Kohms (couleur: marron - noir - orange).

3 résistances ajustables

Ce composant est une résistance dont la valeur varie en fonction de la position de son curseur. Il vous faudra un petit tournevis (non livré) pour pouvoir faire tourner le curseur des résistances ajustables de votre starter-kit.

1 Photorésistance

Aussi appelé LDR (pour Light-Dependent Resistor en Anglais), ce composant s'apparente à une résistance ayant la particularité de changer de valeur lorsque sa zone sensible est exposée à des variations de lumière.

1 buzzer

Ce composant est destiné à reproduire des sons. Le modèle livré ne dispose pas d'oscillateur intégré. Il faudra lui appliquer une succession de signaux haut et bas pour générer des sons.

4 boutons-poussoirs:
Ces derniers sont utilisés pour établir un contact lorsque vous appuyez dessus. Même s'ils n'ont pas de « sens » lorsque vous les utilisez, ils nécessitent une attention particulière lorsque vous les insérez sur la plaque de développement sans soudure. Il est impératif que vous vous basiez sur les petits carrés de couleur rouge indiqués sur les schémas de câblage (réprésentant leurs broches) pour les orienter correctement sur la plaque de développement sans soudure.

1 interrupteur:
Ce composant a la même fonction qu'un bouton-poussoir mis à part qu'il maintient l'état du contact en fonction de sa position (vous n'êtes pas obligé de laisser votre doigt appuyé dessus pour maintenir le contact). En appuyant une fois dessus, le contact est établi. En appuyant à nouveau, le contact disparaît ... et ainsi de suite.

1 servomoteur:
Ce petit module est souvent utilisé dans le modélisme pour modifier la direction des roues d'une voiture ou les ailerons d'un avion. Il est destiné à recevoir des trains d'impulsions (type signal PWM) dont la largeur des impulsions permettra de modifier la position de son palonier. Il est impératif de rappeler qu'il ne faut jamais (même si le servomoteur n'est pas alimenté) tourner manuellement l'axe d'un servomoteur avec les doigts sous risque de casser ses petits pignons internes (non pris en compte par la garantie).

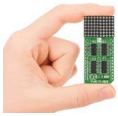
1 capteur de température de type LM35:
Ce petit composant délivre une tension proportionnelle à la température ambiante.

1 cordon USB:
Ce dernier permettra de relier la platine « Flip & Click » à votre ordinateur afin de pouvoir la programmer.

1 jeu de câbles souples mâle/mâle:
Ces derniers vous permettront d'effectuer divers raccordements entre la plaque de développement sans soudure et la platine « Flip & Click » ainsi que les composants électroniques.

1 module DHT22 Click:
Ce petit module intègre un capteur de température et d'humidité. Il est capable de mesurer le taux d'humidité de 0 à 100% et la température de -40 °C to +125 °C (dans les faits, l'enveloppe plastique du capteur ne pourra pas résister à de telles températures extrêmes. Mais le capteur sera suffisant pour effectuer des mesures au sein de votre habitation).

1 module 7x10 R Click:
Ce petit module intègre une matrice composée de 7 lignes de 10 leds. Il vous permettra de pouvoir disposer d'un excellent dispositif capable d'afficher toutes sortes de choses (lettres, chiffres, icônes...).



Installation de l'environnement de développement (IDE)

Avant de pouvoir commencer à « écrire » votre premier programme il vous faudra installer au préalable l'environnement de développement (IDE) de la platine « Flip & Click » sur votre ordinateur. Pour ce faire, **sans que la platine « Flip & Click » ne soit encore reliée au port USB** de votre ordinateur, téléchargez la dernière version de l'environnement (IDE) via Internet qui correspond à votre type d'ordinateur et à son système d'exploitation.

Vous trouverez ce programme ici : **https://www.Arduino.cc/en/Main/Software** (votre ordinateur doit donc être connecté à Internet). A noter que l'environnement de développement (IDE) évolue régulièrement (afin que ce dernier bénéficie de nouvelles possibilités). A titre indicatif, à l'heure où nous écrivons cet ouvrage la version disponible en ligne est la version : 1.6.9.

Si votre ordinateur est un compatible PC sous environnement Windows™

A la fin du téléchargement, double-cliquez sur le fichier. Si Windows™ affiche un message de sécurité, confirmez et « forcez » l'installation. Ensuite lors de l'installation, si vous acceptez la licence, validez celle-ci et choisissez le répertoire dans lequel sera installé l'IDE.

A ce stade, raccordez le câble USB sur le connecteur de programmation de la carte « Flip & Click » (face bleue - voir figure du haut page 6) et branchez l'autre extrémité du câble sur le port USB de votre PC. Le PC doit alors détecter la platine « Flip & Click » et chercher à installer les drivers qui permettront à votre ordinateur de communiquer avec elle. Dans tous les cas, ne choisissez JAMAIS une installation automatique mais indiquez à windows™ que vous allez lui préciser manuellement l'emplacement où il pourra trouver ces drivers. Puis naviguez dans les dossiers de votre ordinateur pour sélectionner le dossier « **Drivers** » (présent en tant que « sous-dossier » dans le dossier Arduino). Attention, ne sélectionnez pas le sous dossier « **FTDI USB Drivers** ».

Si en revanche, l'installation des drivers ne démarre pas automatiquement, cliquez dans le menu « **Démarrer** » et ouvrez le « **Panneau de Configuration** ». Sélectionnez ensuite le « **Gestionnaire de Périphériques** » et repérez l'icône avec un point d'exclamation indiquant un problème d'installation du driver (vous pourrez éventuellement le trouver dans les rubriques « **Autres périphériques** » ou « **Périphériques inconnus** », puis sélectionnez « **Mettre à jour le pilote** » en faisant un clic droit de souris. A ce stade, indiquez manuellement l'emplacement où le PC pourra trouver ces drivers en naviguant dans les dossiers de votre disque dur pour sélectionner le dossier « **Drivers** » (présent en tant que « sous-dossier » dans le dossier Arduino). Attention ne sélectionnez pas le sous-dossier « **FTDI USB Drivers** ».

Dans tous les cas lors de l'installation des drivers, si Windows™ affiche des messages d'alertes sur la validité des drivers, forcez l'installation en validant ceux-ci (à plusieurs reprises si nécessaire). Une fois l'installation terminée, allez dans le « Gestionnaire de périphérique » (sous Windows™) pour vérifier et notez sous la rubrique « **COM & LPT** » le numéro du port COM qui aura ainsi été créé pour la platine « Flip & Click » - Par exemple **COM5**.

Attention, la platine « Flip & Click » utilise le même microcontrôleur qu'une autre platine compatible Arduino™ (appelée Arduino DUE™). Le nom **Arduino DUE™** sera alors indiqué et pris comme référence lors de l'installation des drivers.

Pour les procédures d'installations sous environnement Linux ou sur MAC™

Consultez la documentation en ligne sur le site **https://www.Arduino.cc/en/Guide/HomePage**

Configuration de l'environnement de développement (IDE)

Une fois le driver de la carte « Flip & Click » installé, « lancez » le programme vous permettant d'avoir accès à l'environnement de développement via un double clic de souris sur l'icône Arduino™. A ce stade, la fenêtre de l'IDE doit s'afficher. Si pour une raison ou pour une autre, le texte qui s'affiche dans la fenêtre n'est pas en Français, il vous est possible de modifier ce paramètre en sélectionnant le menu « **Préférence** » (accessible dans la première colonne à gauche). A présent, Il vous faut encore réaliser quelques configurations avant de pouvoir programmer votre platine « Flip & Click ».

En premier lieu, allez dans le menu **Outils → Type de carte → Boards Manager …**

A ce stade une fenêtre s'ouvre. Sélectionnez dans celle-ci :

Arduino SAM Boards (32-bits ARM Cortex-M3) by Arduino et choisissez la dernière version, puis validez pour télécharger le complément logiciel nécessaire à la prise en compte de votre platine.

Votre ordinateur doit donc être connecté à Internet – Le téléchargement peut être plus ou moins long en fonction de la rapidité de votre connexion Internet. Une fois le téléchargement terminé, vous pouvez refermer cette fenêtre.

Sélectionnez maintenant le menu : **Outils → Type de carte → Arduino Due (programming Port)**

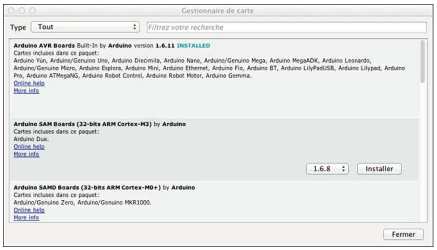
Pour rappel, la carte « Flip & Click » s'apparente à une carte compatible Arduino™ Due™

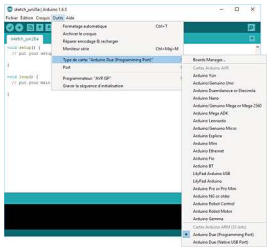
Puis sélectionnez maintenant le menu :

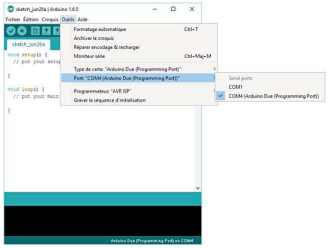
Outils → Port → COM x Arduino Due (programming Port)

ou **x** correspond au numéro du port COM virtuel qui a été créé lors de l'installation du driver de la carte.

Vous êtes désormais prêt pour pouvoir utiliser et programmer votre platine « Flip & Click » !







Votre premier programme

Afin de pouvoir tester le fonctionnement de la platine, nous allons demander à celle-ci de faire clignoter une de ses leds par le biais d'un petit programme très simple.

Pour ce premier exemple, vous n'aurez rien à saisir puisqu'il existe déjà dans l'environnement de développement.

Pour le charger à l'écran, sélectionnez le dans le menu

« **Fichier → Exemples → 01.Basics → Blink** ».

```
void setup() {                 // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the led on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the led off by making the voltage LOW
  delay(1000);              // wait for a second
}
```

A ce stade une nouvelle fenêtre s'ouvre dans laquelle le programme apparaît.

Notez que la fenêtre initialement ouverte est toujours accessible si nécessaire. Ceci vous montre qu'il est ainsi possible d'ouvrir plusieurs programmes liés à plusieurs de vos développements (très utile si vous devez récupérer une partie de code dans un projet plus ancien par exemple par un simple copier/coller). Les fenêtres des programmes non utilisés peuvent être fermées ou laissées ouvertes comme bon vous semble.

Description & Explications

Afin de simplifier la description, on peut dire que tous les programmes sont généralement scindés en 3 parties.

Une première partie dite de « **Déclaration** » dans laquelle on va indiquer à la platine le nom et la nature des variables mémoires que l'on va utiliser dans le programme (cette partie est absente de cet exemple car il s'agit d'un programme très simple ne nécessitant pas de variable).

La seconde partie (appelée « **Initialisation** ») permet d'indiquer à la platine quelles sont les broches (**que nous appellerons souvent Ports dans cet ouvrage**) que vous allez utiliser et comment vous voulez les utiliser (en entrée ou en sortie). Cette partie est comprise entre les lignes **void setup() { }** .

```
void setup() {
                ←——— Vos instructions d'initialisation devront être présentes entre les 2 accolades { }
}
```

La troisième partie correspond au **Programme Principal**. Cette partie est comprise entre les lignes **void loop( ) { }** .

```
void loop( ) {
                ←——— Les instructions du programme principal devront être présentes entre les 2 accolades { }
}
```

Description & Explications (suite)

Comme vous pourrez le constater chaque instruction présente dans la partie initialisation ou dans le programme principal, se termine par un caractère **;**

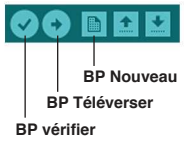
Une des erreurs que vous ferez souvent (tout au moins au début) sera d'oublier d'ajouter ce point virgule à la fin de vos instructions. Mais ne vous inquiétez pas... l'environnement de développement vous le signalera au moment ou vous voudrez envoyer votre programme dans la platine « Flip & Click ».

La dernière remarque concerne la présence de lignes de commentaires qu'il vous est possible d'ajouter dans votre programme. Ces lignes sont repérées par les 2 caractères **//** au début de celles-ci. N'hésitez pas à user et à abuser des lignes de commentaires dans vos programmes pour décrire précisément le fonctionnement de celui-ci. Ceci permettra à une personne n'ayant pas écrit votre programme de comprendre comment il fonctionne et aussi pourquoi vous avez utilisé telle ou telle instruction. Ces lignes vous seront également très utiles à vous aussi pour vous « replonger » dans le fonctionnement et la compréhension de votre programme en vous évitant d'avoir tout oublié de votre phase de développement.

Revenons maintenant à l'explication du fonctionnement de ce premier programme. Celui-ci est relativement simple à comprendre. Dans la partie initialisation, le programme définit la broche 13 de la platine « Flip & Click » en tant que sortie. Cette broche est reliée à une led avec une résistance en série. Cette led est présente sur la platine « Flip & Click » (il s'agit de la led orange qui est tout à fait à droite du câble de programmation sur la face bleue de la platine - voir page 6 ou 8).

Le programme principal propose quand à lui une boucle sans fin dans laquelle la platine « Flip & Click » positionne un niveau logique haut sur la led (afin de l'allumer), puis attend 1 seconde avant d'appliquer un niveau logique « bas » sur la led (pour l'éteindre), puis attend à nouveau 1 seconde... et ainsi de suite.

Passons maintenant au test « réel » du programme. Cliquez sur le bouton « **Téléverser** ». A ce stade, l'éditeur de l'Arduino procède à la compilation de votre programme (un message « **Compilation du croquis** » apparaît en bas de page). L'éditeur va vérifier lors de cette compilation la syntaxe de votre programme (en vous signalant par exemple si vous avez utilisé une instruction qu'il ne connaît pas ou si une instruction est mal orthographiée ou si vous essayez d'utiliser une variable non déclarée, etc...).



Cette étape prend plus ou moins de temps en fonction de la taille de votre programme et de la puissance de votre ordinateur. En cas d'erreur de syntaxe détectée, l'éditeur vous affichera la nature de l'anomalie dans la fenêtre au bas de l'écran. Si aucune erreur n'est détectée, le programme ainsi « compilé » sera transféré dans la mémoire Flash de la carte « Flip & Click » pour être ensuite immédiatement exécuté. Vérifiez donc le fonctionnement de celui-ci en vous assurant de voir la led clignoter.

A noter qu'il vous est également possible de vérifier la syntaxe de votre programme sans que ce dernier ne soit téléversé dans la mémoire de la platine « Flip & Click ». Pour se faire, il suffit de cliquer sur le bouton à gauche du bouton « **Téléverser** ».

A titre indicatif, lorsque vous téléversez un programme dans la carte « Flip & Click », l'ancien programme présent dans la mémoire de la carte sera « effacé » au profit du nouveau programme téléversé.

Ce que l'on a appris

L'instruction **pinMode( );** permet de définir le fonctionnement d'une broche (qu'on appelle **port**) de la platine « Flip & Click ».

Cette instruction doit être impérativement présente dans la partie initialisation de votre programme avant de pouvoir utiliser un port. Le premier paramètre de cette instruction correspond au numéro du port (dans notre programme il s'agit du port 13 sur lequel est relié la led) et le second paramètre **OUTPUT** ou **INPUT** définit si on veux utiliser le port en **sortie** (OUTPUT) ou en **entrée** (INPUT).

L'instruction **digitalWrite( );** permet d'appliquer un niveau logique haut ou bas sur un port de la platine « Flip & Click ».

Le premier paramètre présent dans la parenthèse de l'instruction détermine le numéro du port de la platine « Flip & Click ». Dans notre exemple il s'agit du port 13 sur lequel est relié la led.

Le second paramètre **HIGH** ou **LOW** détermine si on désire appliquer un niveau logique « **Haut** (HIGH) » (dans notre cas pour allumer la led) ou « **Bas** (LOW) » (pour éteindre la led).

L'instruction **delay( );** permet à la platine « Flip & Click » d'effectuer une temporisation.

La valeur que l'on met dans la parenthèse est en milliseconde. Par exemple **delay (1000);** permettra à la platine « Flip & Click » d'effectuer une temporisation d'une seconde.

Si vous voulez aller plus loin...

Changez les valeurs entre les parenthèses des instructions **delay ( );** et observez le résultat. Essayez par exemple de mettre la valeur 500 au lieu de 1000 (pour les 2 instructions). Comme vous pourrez le constater, le clignotement de la led sera plus rapide. Maintenant, laissez la valeur de la première instruction à 500 et changez la seconde valeur de l'instruction **delay ( );** avec la valeur 5. Téléversez le programme et regardez ce qu'il se passe.

La led reste allumée ! ... Pourquoi ?

Pas de panique, il ne s'agit pas d'un Bug, mais simplement d'une limitation de notre système visuel qui ne permet pas de visualiser le moment (trop court) pendant lequel la led est éteinte. La persistance rétinienne vous fera ainsi croire que la led reste tout le temps allumée.

Allez un dernier petit essai « pour la route » !

Retournez votre platine « Flip & Click ». Vous trouverez au dos 4 leds marquées A, B, C et D. Ces leds sont racordées aux ports 38, 37, 39 et 40. Modifiez le programme en remplaçant la led de la broche 13 par une des leds A, B, C ou D.

Maintenant essayez d'allumer et d'éteindre plusieurs leds en même temps en ajoutant autant d'instructions que nécessaires.

Informations complémentaires

La suite de cet ouvrage va maintenant vous montrer comment développer vos premières applications.

A propos des programmes

Vous avez le choix de pouvoir saisir manuellement ces derniers dans l'environnement de développement de la platine Flip & Click ou bien de les télécharger (à l'adresse indiquée ci-après) afin de les récupérer directement sur l'écran de votre ordinateur.

Nous ne serions trop vous conseiller dans un premier temps de saisir manuellement les programmes les plus simples afin de vous familiariser avec le langage C et l'environnement de développement de la platine.

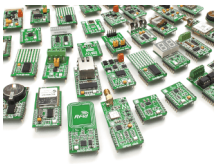
Noter que faute de place suffisante dans ces pages, nous avons dû raccourcir la plupart des commentaires présents dans les programmes imprimés dans cet ouvrage. En revanche, vous disposerez de commentaires beaucoup plus complets et explicites dans les programmes disponibles en téléchargement.

A propos des schémas de connexion

Bien que vous disposez dans cet ouvrage des schémas de connexion détaillés pour chacune des applications, nous proposons également ces schémas en téléchargement afin que vous puissiez les afficher sur l'écran de votre ordinateur, pour pouvoir repérer encore plus facilement et avec une plus grande précision les emplacements des différents composants et fils de liaison. En cas de doute sur un raccordement présent sur le schéma de cet ouvrage, consultez systématiquement le schéma sur l'écran de votre ordinateur afin d'éviter une erreur de câblage.

Concernant le câblage, notez enfin que vous n'êtes pas obligé d'utiliser des straps souples de mêmes couleurs que ceux que nous avons représenté sur les schémas de câblage. Les couleurs des fils utilisés sur nos schémas ont été choisies pour assurer une lisibilité optimale. En dehors des connexions aux sources d'alimentation (pour lesquels il est généralement conseillé d'utiliser par standardisation des fils rouges pour le + et des fils noirs pour la masse), il n'y a pas de véritables règles établies pour les autres signaux.

Téléchargez les programmes et les schémas des applications ici: **http://www.lextronic.fr/lextronic_doc/BOOK.zip**

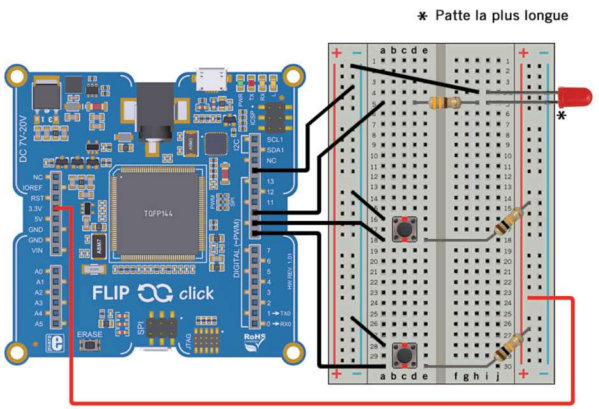


A propos des modules d'extensions Click Board

2 modules Click Board sont livrés de base avec votre starter-kit. Vous disposez par ailleurs de plus de **200 autres modules Click board** qu'il vous est possible d'ajouter **en option** à votre platine Flip & Click afin d'augmenter les possibilités de votre starter-kit !

Les applications 26 à 36 de cet ouvrage sont conçues sur la base de certains de ces modules optionnels. Connectez-vous sur le site **www.lextronic.fr** pour découvrir et éventuellement acheter les modules d'extension Click Board optionnels.

Application N° 001 Leds et boutons-poussoirs




```
const byte BP1 = 8;     // Attribution du BP N° 1 au port 8
const byte BP2 = 9;     // Attribution du BP N° 2 au port 9
const byte Led = 10;    // Attribution de la led au port 10

void setup()
{
  pinMode(BP1, INPUT);          // Configure port BP 1 en entrée
  pinMode(BP2, INPUT);          // Configure port BP 2 en entrée
  pinMode(Led, OUTPUT);         // Configure Port de la led en sortie
  digitalWrite(Led, LOW);       // Eteint la led
}

void loop()
{
  if (digitalRead(BP1) == LOW)  digitalWrite(Led, HIGH);
  if (digitalRead(BP2) == LOW)  digitalWrite(Led, LOW);
}
```

Cette première application très simple va vous permettre d'apprendre à récupérer les états de 2 boutons-poussoirs et en fonction de ces derniers, d'allumer ou d'éteindre une led. En sollicitant le premier bouton-poussoir vous allumerez la led. En sollicitant le second bouton-poussoir, vous éteindrez la led.

Réalisation du câblage & interprétation

Comme nous n'aurons de cesse de le répéter tout au long de cet ouvrage, prenez l'habitude **de déconnecter** la platine « Flip & Click » du port USB de votre ordinateur **avant chaque nouveau montage** afin d'éviter de réaliser un court-circuit au niveau de l'alimentation de votre ordinateur en cas d'erreur de câblage ou de mauvaise manipulation.

Laissez la prise USB branchée au niveau de la platine « Flip & Click » (mais débranchez la du côté de votre ordinateur).

Réalisez ensuite le montage de la figure de gauche. Celui-ci présente 2 boutons-poussoirs associés à des résistances ramenées au 3,3 V. Repérez bien l'orientation des boutons-poussoirs (**suivez les repères carrés rouges**). Vérifiez également la valeur des résistances de 10 kohms (marron – noir – orange). **ATTENTION**, les résistances sont reliées au 3,3 V (PAS au 5 V sous peine de destruction de la platine « Flip & Click »).

Le rôle de ces résistances (dites **de tirage**) est de forcer les ports de la platine « Flip & Click » à un niveau logique « haut » lorsque le bouton-poussoir n'est pas sollicité. A l'inverse, le port de la platine « Flip & Click » sera forcé à un niveau logique « bas » lorsque vous appuyez sur le bouton-poussoir.

En résumé, le port de la platine verra un « 1 » au repos et un « 0 » si vous appuyez sur le bouton-poussoir.

Vous devrez également utiliser une led (attention au sens de celle-ci) et une résistance de 330 ohms (orange - orange – marron) à câbler en série avec la led.

Description et explications

Saisissez alors le programme (voir dans l'encadré de la page de gauche). Raccordez la platine à votre ordinateur et téléversez le programme dans la platine « Flip & Click ». Sollicitez ensuite le bouton-poussoir N° 1 puis le bouton-poussoir N° 2 et regardez comment réagit la led.

Le programme de cette première application est relativement simple à comprendre. Dans la première partie de la déclaration, on utilise l'instruction **const** qui nous permet d'attribuer le N° du port 8 au « mot » BP1 (BP1 pour bouton-poussoir N° 1). Nous avons fait de même pour le port 9 et le bouton-poussoir N° 2 et pour la led et le port 10. Cette méthode offre plusieurs avantages. En premier lieu, elle permet une plus grande « lisibilité » et compréhension de votre programme. Il est en effet plus facile de comprendre que le « mot » BP1 correspond au bouton-poussoir N° 1 qu'un nombre correspondant à un numéro de port de votre carte.

L'autre avantage est de pouvoir modifier très simplement l'attribution des ports dans votre programme. Imaginons que vous n'avez pas utilisé cette méthode de déclaration et que votre programme soit composé de très nombreuses lignes de codes mettant le port 10 de la platine « Flip & Click » au niveau « Haut » via l'instruction **digitalWrite(10, HIGH);** Si pour une raison ou pour une autre, vous décidez de ne plus utiliser le port 10 pour piloter la led, mais le port 12 par exemple, il vous faudra alors modifier toutes les lignes de code de votre programme faisant référence au port 10 afin de les remplacer par le port 12.

Avec la méthode de déclaration que nous avons utilisé, il nous suffira juste de modifier en début de programme une seule ligne **const byte led = 10;** par **const byte led = 12;** pour que tout le reste du programme soit « informé » que le port attribué à la led est le port 12.

Note: L'éditeur dans lequel vous écrivez le programme de votre platine « Flip & Click » est sensible à la « casse ». C'est à dire qu'il fait la distinction entre les majuscules et les minuscules lorsque vous définissez le nom de vos variables. Dès lors si vous définissez une variable avec le nom led1 dans la phase d'initialisation, l'éditeur générera une erreur si vous essayez d'utiliser une variable avec le nom Led1 par la suite.

Dans la seconde partie d'initialisation, le programme définit les ports en entrée (sur lesquels sont reliés les boutons) et le port sur lequel est relié la led en sortie. Au passage on éteint également cette led en appliquant un niveau logique bas sur le port qui la pilote. Dans la boucle du programme principal, on va tester très simplement le niveau logique présent sur le port attribué au bouton-poussoir N° 1. Si le niveau logique est « bas » (si on a donc appuyé sur le bouton-poussoir) alors on allumera la led en appliquant un niveau logique « haut » sur le port qui la pilote. Ce test est réalisé grâce à l'instruction **if ( )**

if ( ) est une instruction impliquant une condition que l'on exprime entre la parenthèse... if veut dire « si » en anglais.

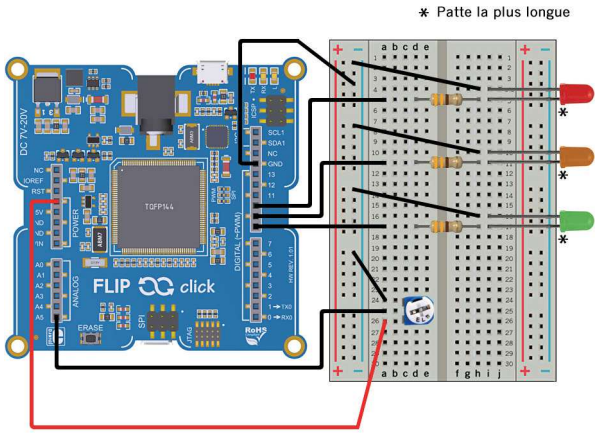
Dans notre cas, on demande au programme de vérifier que **if digitalRead(BP1) == LOW** (si on a appuyé sur le bouton-poussoir N° 1) alors on allume la led au moyen de l'instruction qui est juste après le if ( ) à savoir **digitalWrite(led, HIGH);**

On réalise exactement le même test pour le bouton-poussoir N° 2 mais en demandant cette fois-ci d'éteindre la led si le bouton-poussoir N°2 est sollicité: **if digitalRead(BP2) == LOW digitalWrite(led, LOW);**

L'instruction **if ( )** est une instruction très utilisée en programmation. Nous avons choisi pour ce premier exemple la façon la plus simple de mettre en oeuvre ce type d'instruction. Nous verrons plus tard dans d'autre exemples, d'autres façons plus complexes d'utiliser l'instruction **if ( )** .

Application N° 002 Feu tricolore

Cette application va nous permettre de simuler le fonctionnement d'un feu tricolore routier (lequel passe du vert... à l'orange avant de passer au rouge... et ainsi de suite). Pour ce faire, nous utiliserons 3 leds de couleurs différentes pour simuler le feu tricolore.




```
const byte Vert = 8;              // Attribution de la led Verte au port 8
const byte Orange = 9;            // Attribution de la led Orange au port 9
const byte Rouge = 10;            // Attribution de la led Rouge au port 10

Void setup()
{
  pinMode(Vert, OUTPUT);          // Configure port led Verte en sortie
  pinMode(Orange, OUTPUT);        // Configure port led Orange en sortie
  pinMode(Rouge, OUTPUT);         // Configure port led Rouge en sortie
}

void loop()  {
  digitalWrite(Vert, HIGH);       // Allume la led Verte
  digitalWrite(Orange, LOW);      // Eteint la led Orange
  digitalWrite(Rouge, LOW);       // Eteint la led Rouge
  delay(3000);                    // Attend 3 secondes
  digitalWrite(Vert, LOW);        // Eteint la led Verte
  digitalWrite(Orange, HIGH);     // Allume la led Orange
  delay(1500);                    // Attend 1,5 secondes
  digitalWrite(Orange, LOW);      // Eteint la led Orange
  digitalWrite(Rouge, HIGH);      // Allume la led Rouge
  delay(3000);                    // Attend 3 secondes
}
```

Réalisation du câblage

Déconnectez la platine « Flip & Click » du port USB de votre ordinateur, puis câblez les 3 leds et les 3 résistances sur les ports de la platine « Flip & Click » comme indiqué sur le schéma (attention au sens des leds). Câblez également la résistance ajustable (bien que celle-ci ne soit pas utilisée tout de suite dans ce premier programme). Le curseur de la résistance ajustable sera relié sur l'entrée de conversion « analogique/numérique » A5 de la platine « Flip & Click » tandis que les 2 contacts extrêmes de la résistance ajustable seront reliés à la masse pour l'un et au 3,3 V pour l'autre. ATTENTION, utilisez bien la broche 3,3 V (PAS la broche 5 V... sous peine de destruction de la platine « Flip & Click »). Ainsi en tournant la résistance ajustable on disposera sur son curseur d'une tension qui variera entre 0 et 3,3 V.

Saisissez alors le programme ci-dessus, raccordez la platine à votre ordinateur et téléversez-le programme pour tester

Description et explications

Dans la partie déclaration du programme, on attribue la couleur de chaque led aux ports 8, 9 et 10 de la platine « Flip & Click ».

Dans la partie initialisation, on définit ces ports en tant que sorties.

Le programme principal propose alors une boucle sans fin dans laquelle la platine « Flip & Click » positionne un niveau logique haut sur la led verte (port 8) afin de l'allumer et un niveau logique « 0 » sur les 2 autres leds (afin de les éteindre), puis le programme attend 3 secondes avant d'appliquer un niveau logique « haut » sur la led orange (en éteignant les autres leds), puis après 1,5 seconde, le programme allume la led rouge pendant 3 secondes (en éteignant les autres) et ainsi de suite.

Vous pouvez vous amuser à modifier les valeurs des temporisations des instructions **delay ( )** afin de modifier la « vitesse » du feu tricolore.

Feu tricolore à vitesse variable

Saisissez le programme proposé dans l'encadré de droite.

Cette variante du programme va nous permettre de modifier la vitesse de fonctionnement du feu tricolore au moyen de la résistance ajustable.

Pour ce faire nous effectuerons une conversion « analogique/numérique » de la tension présente sur le curseur de la résistance ajustable (via le port **A5**) avec l'instruction **analogRead( );**

Cette instruction ne vous permet pas de récupérer directement la valeur de la tension présente sur le curseur de la résistance ajustable, mais une valeur (comprise entre **0** et **1023**) qui est proportionnelle à cette tension.

Ainsi vous récupèrerez la valeur 0 lorsque le curseur de la résistance ajustable sera à fond vers la masse et la valeur 1023 lorsqu'il sera à fond vers le 3,3 V... et par déduction la valeur 512 lorsque le curseur sera à mi-course.

```
const byte Vert = 8;                // N° du port sur lequel est relié la led Verte
const byte Orange = 9;              // N° du port sur lequel est relié la led Orange
const byte Rouge = 10;              // N° du port sur lequel est relié la led Rouge
const byte Rajustable = A5;         // N° du port relié la résistance ajustable
int Temporisation = 0;              // Durée de la temporisation

void setup()
{
  pinMode(Vert, OUTPUT);          // Port sur lequel est relié la led verte en sortie
  pinMode(Orange, OUTPUT);        // Port sur lequel est relié la led Orange en sortie
  pinMode(Rouge, OUTPUT);         // Port sur lequel est relié la led Rouge en sortie
}

void loop() 0
{
  Temporisation = analogRead(Rajustable);   // Lecture tension résistance ajust.
  digitalWrite(Vert, HIGH);           // Allume la led Verte
  digitalWrite(Orange, LOW);          // Eteint la led Orange
  digitalWrite(Rouge, LOW);           // Eteint la led Rouge
  delay(Temporisation*10);            // Temporisation d'allumage de la led Verte
  Temporisation = analogRead(Rajustable);   // Lecture tension résistance ajust.
  digitalWrite(Vert, LOW);            // Eteint la led Verte
  digitalWrite(Orange, HIGH);         // Allume la led Orange
  delay(Temporisation*3);             // Temporisation d'allumage de la led Orange
  Temporisation = analogRead(Rajustable);   // Lecture tension résistance ajust.
  digitalWrite(Orange, LOW);          // Eteint la led Orange
  digitalWrite(Rouge, HIGH);          // Allume la led Rouge
  delay(Temporisation*10);            // Temporisation d'allumage de la led Rouge
}
```

Description et explications (Feu tricolore à vitesse variable)

On utilise alors cette valeur pour déterminer la durée d'allumage des leds rouge et verte du feu tricolore (en la multipliant par 10) au sein de l'instruction **delay (Temporisation*10);** Pour la led orange, on appliquera un coefficient multiplicateur de 3 afin que celle-ci reste allumée moins longtemps. Ainsi par exemple lorsque la résistance ajustable sera placée à mi-course, la durée d'allumage des leds rouge et verte sera de 512 x 10 (soit environ 5,12 secondes), tandis que celle de la led orange sera de 3 x 512 = 1,536 secondes env.

Vous pourrez également remarquer que nous avons utilisé l'instruction **analogRead( );** par 3 fois dans notre programme (en fait après chaque changement d'état du feu tricolore). Dans les faits, nous aurions pu utiliser cette instruction qu'une seule fois. La limitation qui en découlerait serait alors que le changement de vitesse du feu tricolore (suite à la variation du curseur de la résistance ajustable) ne serait opérationnel que lorsque les 3 leds auront effectué un cycle d'allumage/ extinction complet. L'usage de la résistance ajustable serait alors moins réactif. En faisant une lecture de la résistance ajustable à chaque transition du feu tricolore, la modification de la position du curseur de la résistance ajustable est donc bien plus réactive.

Si vous voulez aller plus loin...

Prêtez attention à la façon dont nous avons effectué les déclarations au tout début du programme. Nous avons utilisé la syntaxe **const byte Vert = 8 ;** Par contre quelques lignes après, nous avons utilisé une autre syntaxe pour définir la variable de la temporisation. A savoir **int Temporisation = 0;**

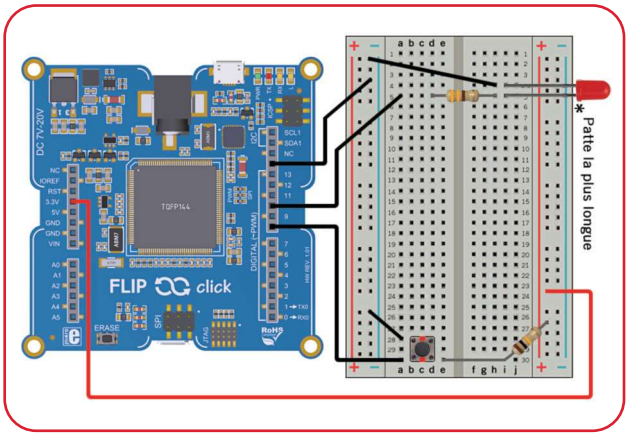
Le terme **byte** désigne une donnée pouvant prendre comme valeur 0 à 255 (donnée sur 8 bits soit 1 octet). Le terme **int** désigne quand à lui une donnée pouvant prendre comme valeur -32768 à 32767 (donnée sur 16 bits soit 2 octets). Pour être plus exacte, la platine « Flip & Click » utilise un microcontrôleur de nouvelle génération pour lequel les données de type **int** sont en fait stockées sur 32 bits (4 octets). Vous pourrez donc stocker des valeurs comprises entre -2147483648 et 2147483647. Dans le premier cas, nous avons utilisé une donnée de type **byte** car les numéros des ports ne dépassent pas 255.

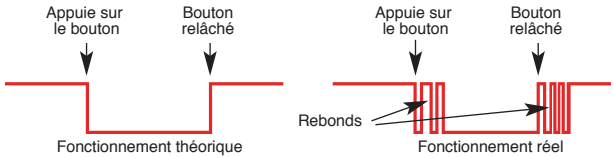
Dans le second cas, nous avons utilisé une donnée de type **int** car la valeur de la temporisation dans notre programme peut être supérieure à 255. Ces variables sont stockées dans une mémoire SRAM d'une capacité de 96 KB au sein de la platine « Flip & Click ». Il est donc important de toujours choisir la nature de ses données avec attention afin de ne pas « gaspiller » de l'espace mémoire SRAM (ceci n'est bien sûr absolument pas critique dans le cadre de la taille « ridicule » de notre programme).

Nous avons utilisé l'instruction **const** lors de l'association des ports avec les couleurs des leds alors que nous n'avons pas utilisé celle-ci lors de la déclaration de la variable de temporisation. **const** indique au programme que l'on attribue une constante qui ne pourra par la suite pas prendre d'autre valeur dans toute l'exécution de votre application (dans notre cas le mot vert sera toujours associé au port 8 de la platine). Par contre la variable **Temporisation** pourra elle changer de valeur au cours de l'exécution de notre application.

Autre info utile: Pour connaître la valeur réelle de la tension suite à une conversion « analogique/numérique », il vous suffira de multiplier la valeur retournée par l'instruction **analogRead( );** par 3,3 et de diviser le résultat par 1023. Ainsi suite à une conversion « analogique/numérique », si vous obtenez 512 comme résultat, vous pourrez en déduire que la tension mesurée est de 512 x 3,3 = 1689,60 / 1023 = 1,6516... soit 1,65 V (en arrondissant).

Application N° 003 Gestion des rebonds sur un bouton-poussoir






```
const byte BP = 8;              // Attribution du bouton-poussoir au port 8
const byte Led = 10;            // Attribution de la led au port 10
volatile byte Sortie = 0;       // Etat initial de la variable Sortie

void setup()  {
  pinMode(BP, INPUT);           // Port relié le bouton-poussoir en entrée
  pinMode(Led, OUTPUT);   // Port relié la led en sortie
  digitalWrite(Led, LOW); }    // Eteint la led

void loop()  {
 while (digitalRead(BP) == HIGH) {  }      // Attend appui sur BP
 // delay (10);
 while (digitalRead(BP) == LOW) {  }      // Attendre relachement BP
 //delay (10);
 Sortie = !Sortie;              // inverse l'état de la variable sortie
 digitalWrite(Led, Sortie);} // Pilote led selon valeur variable Sortie
```

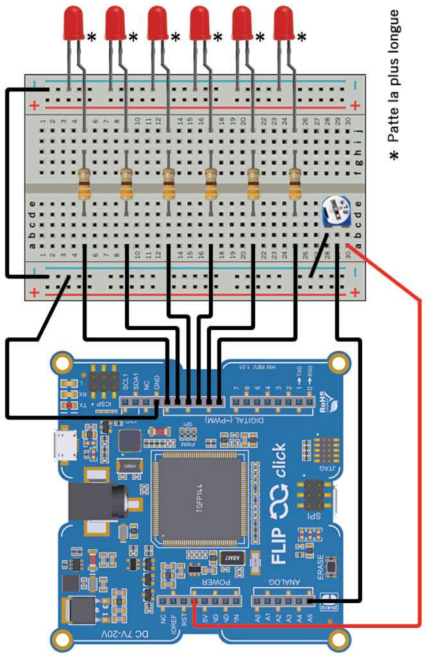
Cette application va vous permettre de mettre en lumière certains problèmes que l'on rencontre lorsqu'on utilise des boutons-poussoirs. L'application consiste à utiliser un bouton-poussoir qui à chaque sollicitation fera changer l'état d'une led.... Simple non ?

Réalisation - Description et explications

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre. Raccordez la platine à votre ordinateur, saisissez le programme et téléversez le afin de tester ce dernier en appuyant plusieurs fois de suite sur le bouton pour que la led change d'état. Comme vous pourrez le constater... cela ne se passe pas comme prévu. De temps en temps la led ne change pas d'état ou s'allume et s'éteint aussitôt.

Analysons ce qu'est censé faire le programme. Après les initialisations et déclarations habituelles, le programme entre dans une boucle **while (digitalRead(BP) == HIGH) { }** qui consiste à attendre tant que l'on détecte un niveau logique haut sur l'entrée reliée au bouton (donc tant qu'on appuie pas sur le bouton). A ce stade, le programme entre dans une seconde boucle **while (digitalRead(BP) == LOW) { }** qui attend tant qu'on détecte un niveau bas (tant que le bouton n'est pas relâché). Si tel est le cas, on inverse l'état d'une variable **Sortie = !Sortie;** que l'on va utiliser pour mettre à jour le port sur lequel est relié la led. En théorie, celà devrait fonctionner ! Oui mais en pratique, un bouton-poussoir ne délivre pas un signal aussi stable que voulu. Lorsque vous appuyez et relâchez ce dernier, vous n'obtiendrez pas un signal bas franc, puis un signal haut franc sur le port 8 de la platine, mais une suite de transitions (appelées rebonds) dont la durée est généralement liée à la qualité du bouton. Pour remédier à ce problème, nous avons ajouté 2 petites temporisations permettant au programme de patienter le temps de ces rebonds afin de prendre en compte uniquement des signaux stables. Retirez les **//** des 2 lignes de commentaires **//delay (10);** et essayez à nouveau le programme. Tout devrait fonctionner bien mieux désormais.

Application N° 004 Chenillard « aller-retour »



Cette application va vous permettre de réaliser un petit chenillard à leds, lequel sera similaire à celui qu'on pouvait voir à l'avant d'une célèbre voiture intelligente issue d'une série télévisée à succès diffusée dans les années 80 !

```
const byte Rajustable = A5;       // Port relié à la résistance ajustable
int Temporisation = 0;            // Durée de la temporisation

void setup()  {
  for (byte i = 8 ; i < 14; i++) {   // Initialisation ports 8 à 13 en sortie
  pinMode(i, OUTPUT);             // Attribution des ports en sortie
  digitalWrite(i, LOW);           // Eteint les leds du chenillard
  } }

void loop()  {
  Temporisation = analogRead(Rajustable);   // Lecture résistance aj.
  digitalWrite(8, HIGH);          // Allume la première led du chenillard
  delay(Temporisation*2);         // Temporisation
  digitalWrite(8, LOW);           // Eteint la première led du chenillard
  for (byte i = 9 ; i < 13; i++)  // Déplacement leds 9 à 12 du chenillard
  {
    digitalWrite(i, HIGH);        // Allume led du chenillard
    delay(Temporisation*2);       // Temporisation
    digitalWrite(i, LOW);         // Eteint led du chenillard
  }
  digitalWrite(13, HIGH);         // Allume la dernière led du chenillard
  delay(Temporisation*2);         // Temporisation
  digitalWrite(13, LOW);          // Eteint la dernière led du chenillard
  Temporisation = analogRead(Rajustable);   // Lecture résistance aj.
  for (byte i = 12 ; i > 8; i--)  // Déplacement leds 12 à 9 du chenillard
  {
    digitalWrite(i, HIGH);        // Allume led du chenillard
    delay(Temporisation*2);       // Temporisation
    digitalWrite(i, LOW);         // Eteint led du chenillard
  }  }
```

Réalisation du câblage - Description & Explications

Déconnectez la platine du port USB de votre ordinateur, puis réalisez le montage ci-dessus, saisissez le programme et téléversez-le dans la platine. Ce programme est également relativement simple à comprendre.

Dans la partie déclaration, on associe l'entrée de conversion « analogique/numérique » **A5** de la platine « Flip & Click » au mot **Rajustable** (diminutif de Résistance ajustable). On déclare également la variable **Temporisation**. Dans la partie initialisation, il va nous falloir configurer les 6 ports de la platine « Flip & Click » en tant que sortie et éteindre dans un premier temps les 6 leds qui sont reliées sur ces sorties.

Description et explications (suite)

Si on se rappelle ce que l'on a appris dans les programmes précédents, il nous faudrait utiliser les instructions **pinMode ( )** et **digitalWrite ( )** pour chacun des 6 ports... ce qui ferait alors 12 lignes de codes à écrire. Pour simplifier notre programme, nous allons avoir recours à une nouvelle instruction **for ( )** Avec cette instruction, il vous sera ainsi possible de répéter toutes les instructions qui seront comprises entre ses accolades.

```
for ( )
 {
  instructions à répéter
 }
```

Ces instructions seront répétées un « certain »nombre de fois selon des conditions que l'on définira via une variable à l'intérieur des parenthèses de l'instruction for ( ).

Ainsi avec l'instruction **for (byte i = 8 ; i < 14; i++)** on indique que l'on va utiliser la variable **i** de type **byte** (donc sur un octet), laquelle aura pour valeur initiale **8**. On indique ensuite que les instructions entre les accolades devront être répétées tant que la variable **i** reste inférieure à la valeur **14**. Pour finir, on indique que la variable **i** doit être incrémentée d'une unité (**i ++**) à chaque fois que l'on a réalisé toutes les instructions entre les accolades.

Étudions maintenant les instructions qui sont entre les accolades : **pinMode(i, OUTPUT);** et **digitalWrite(i, LOW);**

Ce sont des instructions que vous connaissez déjà. La première permet de définir un port de la platine « Flip & Click » en tant que sortie. La seconde permet d'appliquer un niveau logique « bas » sur un port de la platine. La différence avec la syntaxe utilisée dans les programmes précédents est que cette fois-ci on a remplacé le numéro du port de la platine « Flip & Click » par la variable **i**. Voici maintenant ce qu'il se passe lorsque le programme exécute ces instructions.

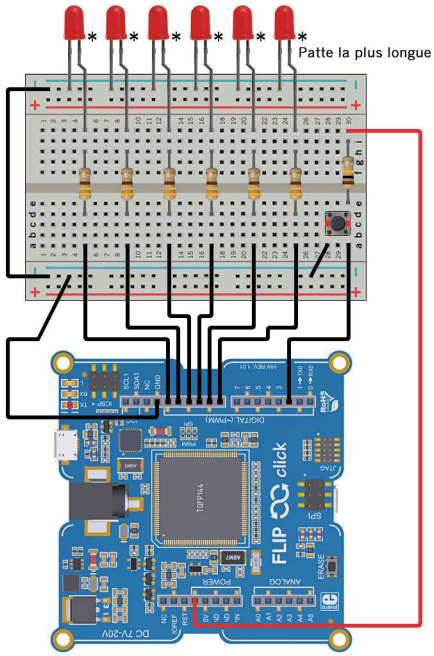
Au premier passage sur l'instruction for ( ), la variable i est initialisée avec la valeur 8. Ensuite le programme exécute les instructions. En configurant ainsi le port 8 en sortie et en y appliquant un niveau logique bas (pour éteindre la led qui est reliée dessus). A ce stade, la boucle for ( ) incrémente la variable i (i devient égal à la valeur 9). Le programme va alors à nouveau initialiser le port 9 et y placer un niveau logique bas... et ainsi de suite jusqu'au port 13 y compris. A ce stade, la variable i est de nouveau incrémentée mais comme elle n'est désormais plus inférieure à la valeur 14, le programme sort de la boucle for ( ) et continue la suite des instructions.

Des lors, on « entre » dans la boucle principale du programme. Celle-ci commence (via le port A5) par récupérer une valeur comprise entre 0 et 1023 via une conversion « analogique/numérique » de la tension présente sur le curseur de la résistance ajustable (laquelle servira à sélectionner la vitesse de déplacement des leds du chenillard). A ce stade, le programme va allumer puis éteindre la première led (du port 8) puis via une nouvelle boucle for ( ), nous allons allumer, puis réaliser une pause, puis éteindre les leds en partant des ports 9 à 12. Une fois la boucle terminée, le programme va allumer, puis éteindre la led du port 13 pour ensuite dans une seconde boucle for ( ) allumer puis éteindre les leds des ports 12 à 9 afin de créer l'effet « retour » des leds.

Étudiez la syntaxe utilisée dans cette boucle for ( ). Cette fois-ci on initialise la variable i avec la valeur 12 avec comme condition de sortie le fait que i devra être supérieur à la valeur 8 et cette fois-ci on indique qu'à chaque exécution de la boucle i devra être décrémenté (**i--**) d'une unité. Comme vous pouvez le voir, on ne gère pas l'allumage et l'extinction des 2 leds extrêmes du chenillard dans les boucles for ( ) afin d'obtenir un réel effet d'aller-retour en évitant qu'une fois arrivé à une extrémité du chenillard on se retrouve avec une configuration dans laquelle toutes les leds sont éteintes.

Application N° 005 Roulette électronique

Cette application va vous permettre de simuler un jeu de roulette électronique. Faites vos jeux... Rien ne va plus...




```
int Temporisation;          // Variable gérant la vitesse de la roulette
const byte BP = 2;          // Attribution du port 2 au bouton-poussoir
byte j,deplace;             // Variables à usage général

void setup() {
  for (byte i = 8 ; i < 14; i++)   // Initialisation des ports 8 à 13 en sortie
  {
  pinMode(i, OUTPUT);       // Attribution des ports en sortie
  digitalWrite(i, LOW);     // Eteint les leds de la roulette
  }
  pinMode(BP, INPUT);             // Port relié au BP en entrée
  randomSeed(analogRead(0));  // Réinitialise la séquence aléatoire
}

void loop()  {
  if (digitalRead(BP) == LOW) {  // Test la sollicitation du BP
    Temporisation = 10;          // Vitesse initiale (rapide) des leds
    deplace = random (40,46);    // Définit led qui devra rester alumée
    j = 8;                       // Définit la première led à allumer
    for (byte i = 0 ; i < deplace; i++)
    {
     digitalWrite(j, HIGH);      // Allume led de la roulette
     delay(Temporisation);       // Temporisation
     digitalWrite(j, LOW);       // Eteint led de la roulette
     j++;                        // Sélectionne led suivante
     if (j > 13) j = 8;
     Temporisation = Temporisation + 10;   // Ralentissement des leds
    }

    for (byte i = 0 ; i < 5; i++)     // Boucle clignotement de la led
    {
    digitalWrite(j, HIGH);      // Allume la led de la roulette
    delay(100);
    digitalWrite(j, LOW);       // Eteint la led de la roulette
    delay(100);
    }
   digitalWrite(j, HIGH);       // Allume led de la roulette
} }
```

Réalisation du câblage - Description & Explications

Déconnectez la platine « Flip & Click » du port USB de votre ordinateur, puis réalisez le montage ci-contre (attention au sens des leds). Attention à la valeur de 3,3 V à relier sur la résistance de tirage du bouton-poussoir. Saisissez ensuite le programme ci-dessus, puis reconnectez la platine « Flip & Click » et téléversez le programme dans celle-ci . Sollicitez alors le bouton-poussoir pour voir la roulette s'élancer rapidement (ceci est matérialisé par l'allumage successif des différentes leds)... puis ralentir avant de stopper sur une des 6 leds (qui se met à clignoter rapidement avant de rester allumée).

Description et explications (suite)

Le programme commence par définir 3 variables qui seront utilisées au cours du programme: **BP**, **J**, **deplace** Puis dans la partie initialisation, on définit les ports 8 à 13 de la platine « Flip & Click » en sortie et on éteint les leds qui y sont reliées. Ensuite, on définit le port 2 en entrée (c'est sur ce port qu'est relié le bouton-poussoir et sa résistance de tirage au 3,3 V). Nous reviendrons plus tard sur l'instruction **randomSeed(analogRead(0));**

Dans le programme principal, on teste la sollicitation du bouton-poussoir (si le niveau logique du port 2 est à zéro... c'est que l'on a appuyé sur le bouton-poussoir). Si tel est le cas, on initialise la valeur de la temporisation qui nous servira à déterminer la vitesse de déplacement de la roulette avec la valeur 10.

Puis on utilise une nouvelle instruction **random ( );** qui est destinée à choisir un nombre aléatoire compris entre 40 et 45 (il faut bien utiliser la syntaxe random (40,46); afin de donner un nombre aléatoire compris entre 40 et 45). Ce nombre est stocké dans la variable **deplace**.

Le programme entre alors dans une boucle **for ( )** dont la variable **i** sera incrémentée (de 0 jusqu'à la valeur aléatoire stockée dans la variable **deplace**). Ceci permet en fait de déterminer la led sur laquelle la roulette va s'arrêter.

Au cours de cette boucle, on utilise la variable **j** que l'on incrémente tour à tour pour allumer et éteindre (avec une temporisation intermédiaire) les différentes leds de la roulette. Un test avec une instruction **if ( )** permet de reconfigurer la variable j avec la valeur 8 (donc sur la première led de la roulette) si on est arrivé sur la dernière led de la roulette. Enfin vous pourrez remarquer qu'à chaque tour de la boucle, on augmente la valeur de la temporisation de 10 unités pour que le la vitesse de déplacement de la roulette ralentisse petit à petit.

Une fois la boucle arrivée sur la led choisie aléatoirement, on effectue une nouvelle boucle **for ( )** pendant laquelle on va faire clignoter rapidement 5 fois de suite la led qui a été sélectionnée avant de la laisser enfin allumée.

Si vous voulez aller plus loin...

Comme vous pouvez le voir, lorsque vous utilisez votre roulette, à chaque sollicitation du bouton-poussoir la platine « Flip & Click » va générer un nombre aléatoire compris entre 40 et 45.

Toutefois la séquence des nombres générés n'est pas vraiment aléatoire !

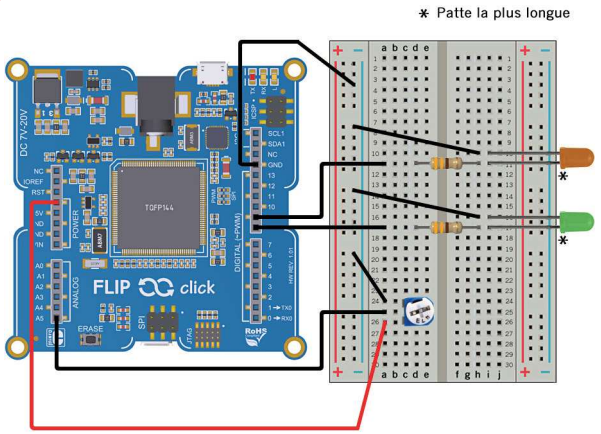
Ceci est en partie normal et expliqué sur le site de référence lié aux Arduino™. En effet l'instruction ramdom génère une longue suite de chiffres aléatoires dont la séquence se répète à chaque mise sous tension.

Pour remédier à ce problème, il faut avoir recours à une instruction spéciale **randomSeed ( );** laquelle permet d'initialiser le point de départ de la séquence aléatoire. La solution proposée dans la documentation de l'Arduino™ consiste à utiliser la valeur de la tension présente sur une entrée de conversion « analogique/numérique » restée en l'air (donc reliée à rien du tout) et dont la valeur de la tension sera alors incertaine et elle-même aléatoire.

Dans notre exemple, nous utilisons le port **A0**.
C'est donc pour cela que l'on a eu recours à l'instruction **randomSeed(analogRead(0));** en début de programme.

Application N° 006 Gradateur pour leds

Cette application va vous permettre de modifier la luminosité de 2 leds au moyen d'une résistance ajustable. En tournant son curseur dans un sens, la première led s'éclairera en étant de plus en plus « brillante », tandis que l'autre led verra son intensité lumineuse se réduire jusqu'à s'éteindre. En tournant la résistance ajustable dans l'autre sens, l'effet inverse se produira sur les leds.




```
const byte Led1 = 8;                  // Attribution de la led N° 1 au port 8
const byte Led2 = 9;                  // Attribution de la led N° 2 au port 9
int Rajustable = A5;                  // Port relié à la résistance ajustable
int Rcycle = 0;                       // Variable stockage rapport cyclique


void setup()
{
  Serial.begin(9600);                 // initialise port com série à 9600 bds
  pinMode(Led1, OUTPUT);              // Configure port led 1 en sortie
  pinMode(Led2, OUTPUT);              // Configure port led 2 en sortie
}


void loop()
{
  Rcycle = analogRead(Rajustable);    // Lecture résistance ajustable
  Serial.print("Rcycle a pour valeur : ");
  Serial.println(Rcycle);             // Envoi valeur sur le port série
  delay (10);                         // Temporisation
  analogWrite(Led1, Rcycle/4);        // Génére signal PWM led 1
  analogWrite(Led2, (1023-Rcycle)/4); // Génére signal PWM  led 2
}
```

Réalisation du câblage - Description & Explications

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre. Les ports 8 et 9 de la platine pilotent les 2 leds tandis que les 2 extrémités de la résistance ajustable sont respectivement connectées à la masse (GND) et sur la sortie 3,3 V (ATTENTION ne confondez pas avec la sortie 5 V). Le curseur de la résistance ajustable est quand à lui relié sur le port de conversion « analogique/numérique » **A5** de la platine « Flip & Click ». Saisissez alors le programme suivant, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier. Tournez le curseur de la résistance ajustable et regardez l'effet qui s'opère sur les 2 leds.

Afin de pouvoir faire varier l'intensité lumineuse des leds, nous allons alimenter celles-ci par intermittence via un signal dont la largeur d'impulsion sera modulée en fonction de la position du curseur du potentiomètre.

Ce type de signal (appelé signal **PWM** - en anglais : **P**ulse **W**idth **M**odulation) peut également être utilisé pour piloter des moteurs (avec une interface de puissance intermédiaire) afin de déterminer leur vitesse de rotation.

Description et explications (suite)

Figure 1: Rapport cyclique 25 %

Figure 2 : Rapport cyclique 75 %

Un signal PWM s'apparente à un signal variant rapidement en permanence d'un niveau haut vers un niveau bas et inversement avec une fréquence fixe, mais avec un rapport cyclique variable. C'est en fonction de ce rapport cyclique que la « moyenne » du signal généré sera plus ou moins importante et que la led s'éclairera plus ou moins.

Si on étudie la **figure 1**, on peut voir que le signal est au niveau haut pendant **25 %** du temps (la led sur une période donnée ne sera donc alimentée que **25 %** du temps) . Celle-ci ne s'éclairera donc pas beaucoup.

Comme nous l'avons déjà vu dans le tout premier programme de test qui nous permettait de faire clignoter une led, du fait que la transition entre les niveaux « haut » et « bas » du signal est très rapide, il vous semblera que la led est allumée.. mais avec une lueur très faible (voir page 16 - explication sur la persistance rétinienne - rubrique pour aller plus loin)..

Dans le cas de la **figure 2**, on peut voir que le signal est au niveau « haut » pendant **75 %** du temps (la led sur une période donnée sera donc alimentée plus « longtemps » que dans l'autre exemple). Il en résulte donc qu'elle s'allumera avec une intensité lumineuse plus importante.

Il serait possible par programmation de demander à la platine « Flip & Click » de modifier l'état d'un port pour recréer la forme des signaux ci-dessus. Le problème est que ceci serait très contraignant puisque votre programme serait attelé à cette tâche et qu'il serait difficile de demander à la platine de faire autre chose en même temps. C'est pourquoi certaines broches de la platine « Flip & Click » sont spécialement prévues pour pouvoir générer ce type de signal (en tâche de fond) au moyen d'une instruction dédiée : **analogWrite( );**

C'est-à-dire qu'après avoir utilisé cette instruction, la broche générera alors en permanence le signal PWM et votre platine pourra continuer le reste de votre programme afin de pouvoir réaliser d'autres tâches.

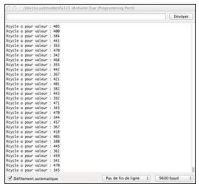
Notes : Les ports 2 à 13 sont des ports qui peuvent être utilisés comme des ports d'entrée/sortie standards ou comme des sorties PWM (si on leur associe une instruction **analogWrite ( )**).

On retrouve en début de programme les habituelles déclarations et l'attribution déjà vues précédemment. Toutefois vous pouvez voir l'utilisation d'une nouvelle instruction **Serial.begin( );** Nous reviendrons sur son rôle dans les pages suivantes. Dans le programme principal on commence par récupérer une valeur comprise entre 0 et 1023 en fonction de la tension présente sur le curseur de la résistance ajustable (laquelle est stockée dans la variable **Rcycle**). Cette valeur nous servira pour modifier le paramètre de génération des signaux PWM des 2 leds afin de faire varier leur niveau d'éclairement.

Description et explications (suite)

La suite du programme utilise une nouvelle instruction **Serial.println( );** Il s'agit d'une instruction extrêmement utile qui vous permettra « d'envoyer » des informations depuis la platine « Flip & Click » vers votre ordinateur afin que ces informations puissent être visualisées sur la fenêtre de l'interface de développement. Ceci est primordial pour débugger votre application (c'est-à-dire pour trouver les problèmes liés à une mauvaise programmation et y remédier).

Accès fenêtre du moniteur



Comme vous pourrez vous en rendre compte assez rapidement lorsque vous essayerez d'écrire vous-même vos propres applications et que vous ne vous contenterez plus de recopier des exemples « tout fait », vous allez écrire un « bout » de code censé réaliser une certaine action et lorsque vous demanderez à la platine « Flip & Click » de tester votre application... il y a des « chances » que cela ne se passe pas comme vous l'aviez prévu. Pour la simple et bonne raison qu'au début, il n'est pas possible de penser à tout et que vous allez oublier de prendre en compte certains paramètres.

Par exemple, si vous voulez créer un jeu dans lequel vous comptabilisez des points, vous pourrez avoir déclaré la variable mémorisant le score en tant que byte. Dès lors, votre score ne pourra pas dépasser 255 et cela posera des problèmes pour son affichage. Pour arriver à trouver plus facilement l'origine de ces petites erreurs de programmation, n'hésitez pas à user (et à abuser) de l'instruction Serial.println( ); ainsi en renvoyant la valeur de cette variable vers l'écran de votre ordinateur, vous pourrez vous rendre compte que celle-ci passera à zéro à chaque foit qu'elle dépassera la valeur 255 (en vous permettant ainsi de voir cette erreur et de la déclarer en **int** pour que vous puissiez afficher des scores supérieurs à 255).

C'est grâce au port série de la platine « Flip & Click » (ports 0 et 1) que ces informations seront envoyées vers votre ordinateur. Pour rappel le port série de la platine « Flip & Click » est également connecté au port de programmation USB de la platine. Afin de pouvoir utiliser ce port série, il vous faut impérativement initialiser la communication série grâce à la « fameuse » instruction **Serial.begin(9600);** dont nous vous avons parlé au début de cette description. Le paramètre entre les parenthèses détermine la vitesse de communication série... ici 9600 bauds.

Ceci va nous permettre d'envoyer la valeur de la conversion «numérique/analogique» de la tension mesurée sur la résistance ajustable vers votre ordinateur via l'instruction **Serial.println(Rcycle);** Cette valeur ne s'affiche pas directement dans la fenêtre dans laquelle se trouve votre programme. En effet, pour voir les valeurs retournées par la platine « Flip & Click », il vous faut cliquer sur le bouton « Moniteur Série » matérialisé par la petite loupe en haut à droite de l'écran.

A ce stade, vous pourrez voir les valeurs défiler à l'écran dans une fenêtre secondaire. Tournez la résistance ajustable et observez la variation des valeurs reçues. A noter que pour pouvoir fonctionner correctement, il vous faudra configurer correctement le débit de communication (via le bouton en bas à droite de la fenêtre) pour que celui-ci corresponde à celui que vous avez choisi dans la programme de la platine « Flip & Click ».

Note: comme indiqué ci-avant, la platine « Flip & Click » utilise les ports 0 et 1 pour communiquer avec la fenêtre « moniteur série ». Il est également possible à la platine « Flip & Click » de recevoir des informations en provenance de cette fenêtre (nous y reviendrons plus tard pour les besoins d'une autre application). Nous insistons sur ces points pour que vous gardiez à l'esprit que si votre application utilise aussi les ports 0 et 1 pour d'autres fonctions... vous risquez d'avoir un conflit de communication avec pour conséquence de ne plus pouvoir utiliser correctement l'instruction **Serial.println( );**

Description et explications (suite)

Revenons maintenant à la suite de notre programme. Après une petite temporisation (conseillée afin de ne pas envoyer consécutivement et trop rapidement les informations vers votre ordinateur), nous allons maintenant générer le signal PWM de la première led au moyen de l'instruction **analogWrite( );** Cette instruction nécessite 2 paramètres.

Le premier correspond au numéro du port sur lequel on doit générer le signal PWM.

Le second paramètre correspond à la valeur du rapport cyclique devant être généré. Cette valeur doit être comprise entre 0 (la led sera complètement éteinte) et 255 (le led sera éclairée au maximum).

Dans notre cas, la variable dans laquelle est stockée la valeur de la conversion « analogique/numérique » de la tension de la résistance ajustable correspond à un nombre compris entre 0 et 1023 (donc supérieur à 255). C'est la raison pour laquelle on divise la valeur par 4 dans notre instruction **analogWrite(led1, Rcycle/4);** afin de retomber dans la plage de 0 à 255.

Afin de réaliser un éclairement inversé de la led numéro 2, on utilise l'instruction **analogWrite(led2, (1023-Rcycle)/4);** dans laquelle on retranche la valeur mesurée à 1023 avant de diviser le tout par 4.

Si vous voulez aller plus loin...

Dans l'instruction **Serial.println(Rcycle);** les lettres **ln** signifie que l'affichage dans la fenêtre du moniteur série sautera une ligne après avoir affiché les informations. Si nous avions écrit **Serial.print(Rcycle);** les données s'afficheraient alors à l'écran les unes derrière les autres (sans sauter de ligne... ce qui rendrait l'affichage des informations moins lisibles).

Il est également possible d'envoyer un « texte » vers la fenêtre du moniteur.

Ajoutez par exemple l'instruction **Serial.print("Rcycle à pour valeur : ");** et téléversez à nouveau le programme.

Cette nouvelle instruction vous permettra d'obtenir l'affichage suivant dans la fenêtre du moniteur série.

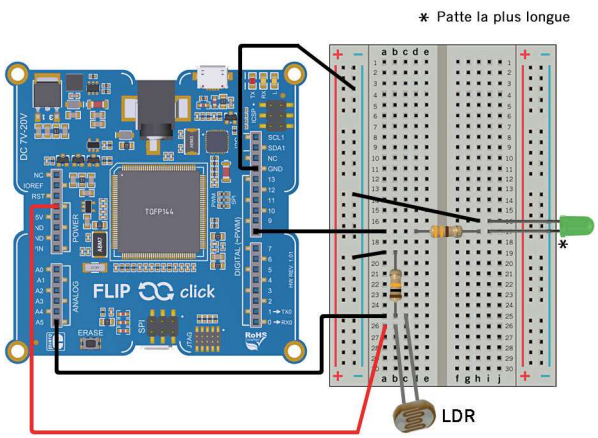
Rcycle à pour valeur : 524 (si bien sûr Rcycle a pour valeur 524)
Rcycle à pour valeur : 535 (si bien sûr Rcycle a pour valeur 535)

Ainsi l'instruction **Serial.print("Rcycle à pour valeur : ");** va afficher Rcycle à pour valeur : (sans sauter de ligne) et l'autre instruction Serial.println(Rcycle); va afficher juste dernière la valeur de Rcyle en sautant cette fois-ci une ligne.

Encore une fois n'hésitez pas à avoir recours à cette instruction pour vous aider à trouver la source à vos problèmes si votre programme ne réagit pas comme prévu. Il peut être intéressant par exemple de remonter un petit texte du style « Je suis ici » vers la fenêtre du moniteur série pour vérifier par exemple que votre programme passe bien à un endroit précis de votre code. Si vous avez par exemple programmé une condition sur l'état d'une variable qui est censée générer le passage sur le code et que le message ne s'affiche pas, vous aurez alors un élément de recherche sur la cause du disfonctionnement de votre application.

Application N° 007 Interrupteur crépusculaire

Cette application va vous permettre de détecter la tombée du jour (ce qui aura pour conséquence d'allumer une led). Cette led s'éteindra dès que la lumière du jour reviendra.




```
const byte Led = 8;   // Attribution de la led au port 8
int LDR = A5;         // N° du port sur lequel est relié la LDR
int Lumiere = 0;      // Variable dédiée à la mesure de la luminosité

void setup()    {
  Serial.begin(9600);           // Initialise le port Com à 9600 bds
  pinMode(Led, OUTPUT);     // Configure le port de la led en sortie
}

void loop()
{
  Lumiere = analogRead(LDR);      // Lecture tension LDR
  Serial.println(Lumiere);        // Envoi la valeur sur port Com
  delay (10);                     // Temporisation
  if( Lumiere > 180 )             // Test si valeur > 180
    {
    digitalWrite(Led,LOW);        // alors Eteint la led
    }
    else                          // Sinon
    {
    digitalWrite(Led,HIGH);       // Allume la led
    }
}
```

Réalisation du câblage - Description & Explications

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre . Vérifiez les couleurs des résistances orange – orange – marron pour celle de la led et marron – noir – orange pour celle raccordée à la photorésistance (laquelle ne dispose d'aucun sens de câblage).

Saisissez alors le programme ci-dessus, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Pour réaliser cette application, nous avons utilisé un nouveau composant: **La photorésistance**

Aussi appelée **LDR** (pour **L**ight-dependent **R**esistor en **A**nglais). Ce composant s'apparente à une résistance ayant la particularité de changer de valeur lorsque sa zone sensible est exposée à des variations lumineuses.

Description et explications (suite)

Son utilisation est très simple puisqu'il suffit de l'utiliser au sein d'un pont diviseur pour pouvoir récupérer une tension qui varie en fonction du niveau de lumière qui frappe la LDR. Cette tension pourra être mesurée au moyen d'une entrée de conversion « analogique/numérique » de la platine « Flip & Click ».

Ce programme est très facile à comprendre. Après une phase d'initialisation et de déclaration (du port de pilotage de la led, du port **A5** dédié à la LDR et du port de communication série de la platine « Flip & Click »), le programme principal commence par effectuer une conversion « analogique/numérique » de la tension au niveau du pont de résistance de la LDR.

La valeur de la conversion est ensuite envoyée vers le moniteur série de votre ordinateur avec l'instruction **Serial.println(Lumiere);** A ce titre, activez la fenêtre du moniteur série dans l'environnement de développement (bouton «loupe» en haut à droite) et déplacez votre main devant la LDR ou recouvrez la LDR avec votre main pour voir comment évoluent les valeurs retournées.

La suite du programme utilise une instruction de test déjà vue précédemment. Toutefois, dans le cas présent, l'instruction est plus évoluée. Il s'agit de l'instruction **if ( ) else** . Cette dernière se compose d'une condition, d'une action à réaliser si la condition est remplie et d'une autre action à réaliser si la condition n'est pas remplie. Sa syntaxe est la suivante :

```
if ( condition)
{
  // Action à réaliser si la condition est remplie
}
else
{
  // Action à réaliser si la condition n'est PAS remplie
}
```

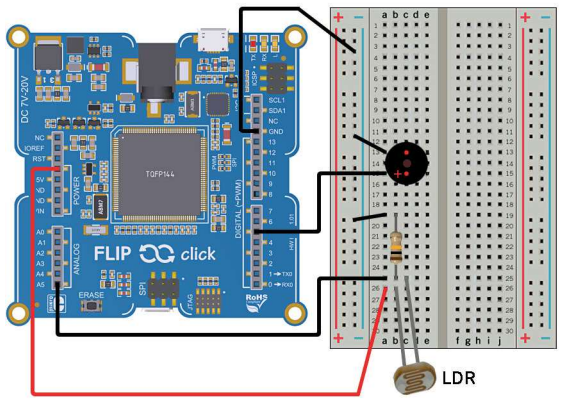
La syntaxe est simple à assimiler (attention toutefois à ne pas se tromper dans le positionnement des accolades).

Après le **if ( )**, on détermine la condition à remplir. Dans notre cas, on teste si la valeur retournée par l'entrée de conversion « analogique / numérique » est supérieure à 180.

Si tel est le cas, l'action à remplir consiste à éteindre la led. On ajoute ensuite le mot **else** qui marque la transition dans l'instruction et ensuite l'action à remplir si la valeur n'est pas supérieure à 180 (qui consiste dans notre cas à allumer la led). Suivez rigoureusement la syntaxe **if ( ) else** avec les différentes accolades.

Ainsi en fonction du niveau de lumière appliqué sur la surface de la LDR, vous obtiendrez l'allumage (en cas de faible lumière sur la LDR) ou l'extinction de la led (si l'éclairement sur la LDR est plus important). A noter qu'en raison du fait que la LDR dispose d'une certaine tolérance et que votre niveau d'éclairage peut également être différent du nôtre, il vous faudra probablement modifier la valeur de seuil **180** que nous avons utilisé. C'est la raison pour laquelle il vous faut contrôler à l'aide du moniteur série quelles sont les valeurs retournées en plein éclairage, ainsi que celles obtenues lorsque vous recouvrez la LDR avec votre main afin de déterminer vous-même la valeur de seuil à mettre dans l'instruction **if ( )**.

Application N° 008 La fuite d'eau infernale



Cette application va vous permettre de faire une farce à votre entourage en leur faisant croire qu'une fuite d'eau se déclare dans leur chambre au moment où ils s'apprêtent à s'endormir.

Ainsi après avoir éteint la lumière ces derniers vont entendre (après une durée aléatoire) des gouttes d'eau tomber (à intervalles également aléatoires). Le problème (pour eux) est que lorsqu'ils vont allumer la lumière pour essayer de localiser cette fuite, les bruits vont s'arrêter immédiatement !

Après quelques minutes de recherches infructueuses, ces derniers vont éteindre à nouveau la lumière pour essayer de dormir... et bien sûr la fuite va reprendre de plus belle !

Votre montage devra être placé au dessus d'un meuble afin qu'il ne puisse pas être vu facilement. Enfin, ne laissez pas votre entourage « en peine » trop longtemps...

Après un petit moment avouez leur la supercherie... les blagues les plus courtes étant toujours les meilleures !

```
const byte Buzzer = 5;          // Attribution du buzzer au port 5
int LDR = A5;                   // Attribution de la LDR au port A5
int Lumiere = 0;                // Variable pour mesure de la luminosité

void setup()  {
  randomSeed(analogRead(0));    // Réinitialise la séquence aléatoire
  // Serial.begin(9600);   }

void loop()    {
  Lumiere = analogRead(LDR      // Lecture tension LDR
 // Serial.println(Lumiere);
  if( Lumiere < 300 )           // Test si on a éteint la lumière
    {
    delay (1000 * random(30,51));     // Durée aléatoire 30 - 50 sec.
    while( Lumiere < 300 )          // Tant qu'on a pas allumé la lumière
      {
      for (byte i=50; i<150; i++)         // Génère le "ploc" de la goutte
        { analogWrite(Buzzer, i);
          analogWrite(Buzzer, i);
          analogWrite(Buzzer, i);  }
      analogWrite(Buzzer, 0);       // Stop signal PWM sur le buzzer
      delay (1000 * random(1,11));          // Durée aléatoire 1 - 10 sec.
      Lumiere = analogRead(LDR);            // Lecture tension LDR
     // Serial.println(Lumiere);   }   }   }
```

Réalisation du câblage & interprétation

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre .

Saisissez le programme suivant, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Ce montage fait appel à la photorésistance que vous connaissez déjà ainsi qu'à un nouveau composant. **Le buzzer piézoélectrique**.

Ce modèle de buzzer est dépourvu d'oscillateur. Il est conçu pour restituer des sons. Un peu à l'image d'un haut-parleur, il est muni d'une membrane qui vibre sous l'impulsion de signaux électriques (laquelle génère des sons de par l'air qu'elle déplace).

Description et explications (suite)

Ce programme est relativement simple et fait appel aux connaissances précédemment acquises. Comme pour le montage de l'interrupteur crépusculaire précédent, on va récupérer la tension issue du pont de résistances au niveau de la LDR pour détecter l'extinction de la lumière via l'instruction **if(Lumiere < 300)**. Vous serez peut-être amené à modifier la valeur **300** en fonction des conditions d'éclairement de la pièce où sera placé le montage. Pour ce faire, vous trouverez des instructions en rouge dans le listing de gauche (actuellement placées en tant que commentaires avec les caractères **//** devant celles-ci). Retirez les 2 caractères // afin de rendre ces instructions actives et ainsi avoir la possibilité de visualiser les valeurs mesurées par la LDR dans la fenêtre du « Moniteur Série » de l'environnement de développement. Allumez et éteignez la lumière de la pièce et repérez la valeur de seuil qui correspond à votre cas.

Lorsque l'exctinction de la lumière est détectée, la platine « Flip & Click » va générer une temporisation aléatoire comprise entre 30 et 50 secondes avant la génération de la chute des gouttes d'eau. Ces gouttes d'eau seront générées à l'intérieur d'une boucle spéciale réalisée par une nouvelle instruction

```
while ( )
{
}
```

Ces instructions seront répétées tant que la condition présente entre les parenthèses de l'instruction while est remplie.

Cette instruction permet de réaliser une suite d'instructions (à placer entre les accolades) tant qu'une condition (à mettre entre les parenthèses) est remplie.

Ici l'instruction sera **while (Lumiere < 600)** que nous pouvons en quelque sorte, dans le contexte du programme traduire par: « tant que la lumière est toujours éteinte »... on génère la chute des gouttes d'eau.

Pour générer le son des gouttes d'eau, on va appliquer un signal PWM (dont le rapport cyclique va varier très rapidement sur le buzzer) via une boucle habituelle **for ( )**. La génération du « ploc » de la goutte d'eau est rendu possible en utilisant plusieurs fois de suite l'instruction **analogWrite(Buzzer, i);** afin d'obtenir un bruit aussi « proche » que possible d'une goutte d'eau.

Une fois le « ploc » généré, on stoppe immédiatement la génération du signal PWM puis en réalise une temporisation aléatoire comprise entre 1 et 10 secondes entre chaque « ploc » avant de mesurer à nouveau la tension au niveau de la LDR (afin de vérifier si la lumière n'est pas revenue entre temps).

Si ce n'est pas le cas, les instructions à l'intérieur de la boucle **while ()** sont à nouveau exécutées.

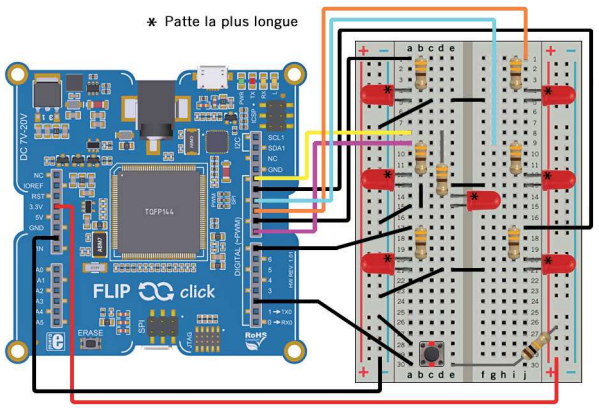
Si par contre la lumière revient, le programme « sort » de la boucle while pour revenir dans la boucle initiale dans l'attente d'une nouvelle extinction de la lumière.

Vous pouvez bien sûr modifier la durée de la génération des gouttes ainsi que la durée des silences entre chaque goutte.

Note: Le son généré par le buzzer pour simuler la goutte d'eau est très faible. Mais ce dernier sera suffisamment audible (dans une pièce avec un environnement silencieux) pour attirer l'attention de la personne que vous allez piéger tout en évitant que celle-ci ne repère trop facilement l'endroit d'ou provient le son.

Application N° 009 Dé électronique

Cette application va vous permettre de réaliser un « Dé » électronique qui, sur sollicitation d'un bouton-poussoir, va se mettre à « rouler » avant de s'arrêter aléatoirement sur une de ses faces. Un vrai dé possède 6 faces avec pour chaque face la représentation d'un nombre de 1 à 6 (réalisée avec des points). Dans notre cas, nous utiliserons 7 leds agencées de telle sorte que l'on pourra représenter les 6 faces d'un Dé en allumant (l'une après l'autre) certaines d'entre-elles.




```
byte Face;                    // Variable mémorisant la face du Dé
const byte Bouton = 2;        // Association du bouton-poussoir au port 2
const byte De[6][7] =         // Configuration des 7 leds à allumer du Dé
{
 {0,0,0,0,0,0,1},             // Dé = 1
 {0,0,1,0,0,1,0},             // Dé = 2
 {0,0,1,0,0,1,1},             // Dé = 3
 {1,0,1,1,0,1,0},             // Dé = 4
 {1,0,1,1,0,1,1},             // Dé = 5
 {1,1,1,1,1,1,0} };           // Dé = 6

void setup()    {
 for (byte i = 7 ; i < 14; i++)       // Initialisation ports 7 à 13 en sortie
 {
 pinMode(i, OUTPUT);                  // Attribution des ports en sortie
 digitalWrite(i, LOW);                // Eteint toutes les leds du Dé
 }
 pinMode(Bouton, INPUT);              // Le port relié au BP en entrée
 randomSeed(analogRead(0));           // Réinitialise la séquence aléatoire
}

void loop()  {
 if (digitalRead(Bouton) == LOW) {  // Teste la sollicitation du BP
   for (byte i = 1 ; i < 11; i++)           // Le Dé roule 10 fois
   {
     Face = random(0,6);                    // Choisi un nombre aléatoire
     for(byte j = 7; j < 14 ; j++)          // Affichage des Leds du Dé
     {
       digitalWrite(j, De[Face][j-7]);      // Récupère les leds à afficher
     }
    delay(150);                             // Temporisation
}   }   }
```

Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-dessus. Prenez votre temps car il y a beaucoup de câblage à faire et repérez bien le sens des leds et la valeur des résistances. Saisissez le programme ci-dessus, raccordez la platine à votre ordinateur et téléversez le programme dans la platine « Flip & Click ». Sollicitez alors le bouton-poussoir pour voir le Dé « s'élancer » puis s'arrêter sur une face.

Description et explications

Ce programme fait appel à des notions déjà acquises dans les applications précédentes ainsi qu'à un nouveau type de variable de type tableau (**Array** en Anglais). Ce type de variable permet de stocker une suite de valeurs que l'on pourra ensuite récupérer grâce à un numéro d'index.

Pour la réalisation de cette application, nous aurions pu générer un nombre aléatoire et en fonction de ce nombre allumer ou éteindre chacune des leds du Dé. Le « problème » est que si nous avions utilisé cette solution, ceci aurait nécessité beaucoup de ligne de codes assez répétitives. C'est pourquoi nous allons avoir recours à une variable de type tableau (dit à 2 dimensions dans notre cas) pour mémoriser l'état des 7 leds du Dé. Cette variable que nous avons appelé **De** est repartie sur plusieurs lignes. La première ligne est composée des valeurs 0,0,0,0,0,0,1 , ceci correspond en fait à l'état que devront prendre les ports 7 à 13 dans le cas ou le dé a pour valeur 1 (seule la led centrale (port 13) est allumée).

Pour la deuxième ligne la variable **De** à pour valeurs 0,0,1,0,0,1,0 (dans ce cas, seul les leds des ports 9 et 12 sont allumées pour afficher le chiffre 2) et ainsi de suite pour les 6 faces du Dé.

Comme vous pouvez le voir, il y a 2 crochets au niveau de la définition de la variable de type tableau **De[6][7]**
6 correspond au nombre max. de lignes (ici 6 pour les 6 faces) et 7 correspond au nombre d'éléments max. présents dans chaque ligne (ici 7 pour les 7 leds constituant la face du Dé). L'intérêt d'utiliser ce type de variable est que l'on peu ensuite avoir accès à n'importe quelle donnée du tableau en « jouant » sur les valeurs des nombres entre les crochets. On gardera à l'esprit que les index vous permettant de repérer les données dans le tableau commencent par 0. Ainsi par exemple, on obtiendra la valeur 1 pour la variable De[0][6] (0 sélectionne la première ligne du tableau et 6 pour sélectionne le 7 ème élement de cette ligne (en partant de 0) ... soit la valeur 1.

Reprenons à présent la description de l'application. Après les déclarations et initialisations habituelles, la boucle principale du programme teste la sollicitation du bouton-poussoir. Si tel est le cas, le programme génère une boucle dans laquelle 10 coups de Dé vont être tirés aléatoirement. Une fois la face du Dé choisie, le programme va afficher les leds correspondantes à la valeur du Dé grâce à la boucle :

```
for(byte j = 7; j < 14 ; j++)
{
 digitalWrite(j, De[Face][j-7]);
}
```

Au cours de cette boucle, la variable **j** évoluera des valeurs 7 à 13 (correspondant aux ports de la platine « Flip & Click » sur lesquels sont reliés les leds.

L'instruction **digitalWrite** va donc piloter chacune des leds de ces ports. Le paramètre dans la première parenthèse de la variable **De** correspond au nombre aléatoire (0 à 5) choisi par le programme, permettant ainsi de sélectionner la bonne ligne de la variable tableau. Pour le paramètre présent dans la seconde parenthèse de la variable De on utilise **j-7** afin que ce paramètre évolue de 0 à 6 (lorsque j évolue de 7 à 13) afin de pouvoir récupérer un à un l'état que devront prendre les ports de la platine pour allumer les différentes leds. Le programme génère enfin une petite temporisation entre chaque affichage afin que vous puissiez avoir le temps de visualiser les transitions des faces du dé.

Application N° 010 Générateur de code morse

```
const byte morse[]={5,0b11111000,5,0b01111000,5,0b00111000,
5,0b00011000,5,0b00001000,5,0b00000000,5,0b10000000,5,0b110
00000,5,0b11100000,5,0b11110000,  // Définition chiffres 0 à 9
2,0b01000000,4,0b10000000,4,0b10100000,3,0b10000000,1,0b000
00000,4,0b00100000,3,0b11000000,4,0b00000000,2,0b00000000,
// Définition lettres A à I
4,0b01110000,3,0b10100000,4,0b01000000,2,0b11000000,2,0b100
00000,3,0b11100000,4,0b01100000,4,0b11010000,3,0b01000000,
// Définition lettres J à R
3,0b00000000,1,0b10000000,3,0b00100000,4,0b00010000,3,0b011
00000,4,0b10010000,4,0b10110000,4,0b11000000};
// Définition lettres S à Z

const byte LedC = 39;     // Association de la Led C au port 39
byte k,pointrait;         // Variables d'usage général
char recu[13];            // Buffer de reception des messages séries

void setup()   {
  Serial.begin(9600);              // Initialisation du port série
  pinMode(LedC, OUTPUT);           // Port Led C en sortie
  digitalWrite(LedC, LOW);         // Eteint la led
}

void loop()
 {
   k = 0 ;                 // Attente et récupération des messages séries
   while (Serial.available() > 0) {
      recu[k] = Serial.read();
      k++;
    delay(10); }

  if (k > 0)               // Regarde si on a reçu un message à envoyer
    {
    for (byte j=0; j<k; j++) {  // Envoie un à un les caractères en morse
        if (recu[j] >= 48 & recu[j]<=57){     // Regarde si chiffre 0 à 9
          pointrait = morse[((recu[j]-48)*2)+1];   // Récupère codification
          for (byte l=0; l<morse[(recu[j]-48)*2]; l++) {
             if (bitRead(pointrait,7) == 0)       // On envoie un point ?
               {point();}                  // Oui -> Génère un point (via la led)
             else
               {trait();}                  // Non -> Génère un trait (via la led)
             pointrait = pointrait << 1;   // Récupère codification suivante
          }   }

        if (recu[j] >= 65 & recu[j]<=90){     // Regarde lettre A à Z
          pointrait = morse[((recu[j]-65)*2)+21]; // Récupère codification
          for (byte l=0; l<morse[((recu[j]-65)*2)+20]; l++) {
             if (bitRead(pointrait,7) == 0)     // Regarde si envoie point ?
               {point();}                       // Oui -> Génère un point
             else
               {trait();}                       // Non -> Génère un trait
             pointrait = pointrait << 1;    // Récupère codification suivante
          }
        }
      delay(400);   // Temporisation entre caractères morse envoyés
      } }
}

//-------------------------------------- Fonctions ------------------------------

    void point() {
    digitalWrite(LedC, HIGH);               // Génération d'un "Point"
    delay(230);
    digitalWrite(LedC, LOW);                // Eteint la led
    delay(250);
    }

    void trait() {
    digitalWrite(LedC, HIGH);               // Génération d'un "Trait"
    delay(900);
    digitalWrite(LedC, LOW);                // Eteint la led
    delay(250);
    }
```

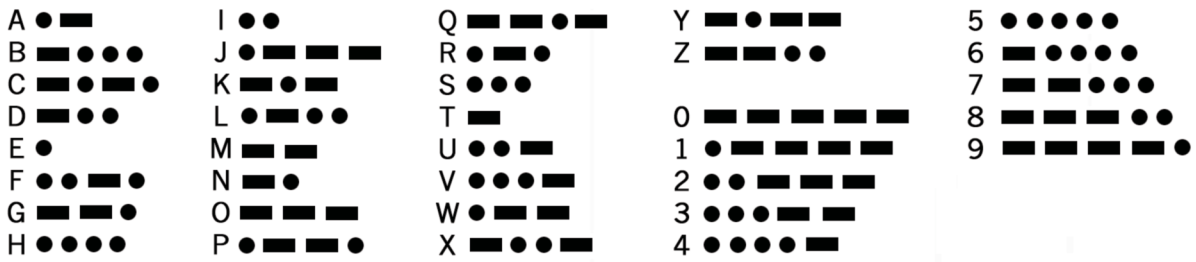
Considéré en quelque sorte comme le précurseur des communications numériques, le code morse inventé en 1832 permettait de transmettre des messages à distance à l'aide de séries d'impulsions courtes et longues retransmises sous forme lumineuse ou sonore.

Pour ce faire, chaque caractère (lettre ou chiffre) était codé en une suite de 1 à 5 impulsions (courtes ou longues – appelées généralement point et trait). Par exemple la lettre A était codée en . - et la lettre B en - . . . (retrouvez l'ensemble du code morse sur la page de droite).

Dès lors il fallait que les opérateurs qui devaient envoyer les messages connaissent la codification de chaque caractère avant de pouvoir les générer.

Notre application va vous permettre de saisir un message sur votre ordinateur depuis la fenêtre du « moniteur série » de l'environnement de développement et de l'envoyer vers la platine « Flip & Click » via le port USB afin que celle-ci le retranscrive immédiatement en code morse !

Détail de la codification du code morse



Réalisation du câblage - Description et explications

Déconnectez la platine « Flip & Click » de votre ordinateur et retirez toutes les connexions précédemment réalisées. Pour cette application, nous n'utiliserons que la led C présente au dos de la platine. Vous n'aurez donc rien à faire comme montage. Saisissez alors le programme de la page de gauche, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Voici une application un peu plus complexe et plus difficile à comprendre, laquelle fait également appel à de nombreuses nouvelles notions. Pour ce programme, nous allons utiliser une variable de type tableau pour stocker les différentes codifications morse des chiffres et des lettres. La première remarque est que contrairement au programme précédent, nous allons utiliser une variable de type tableau à une seule dimension et comme vous pourrez le constater, nous n'avons pas indiqué la valeur max. entre les crochets lors de notre déclaration. Cette syntaxe est aussi autorisée par l'environnement de développement de votre platine.

Pour codifier chaque caractère en morse, nous sommes obligés d'utiliser 2 informations : la première correspond au nombre de points/traits composant le caractère et la seconde correspond justement aux points/traits à générer. Notre tableau commence par la définition du chiffre 0. En code morse, celui ci est représenté par 5 traits. La première valeur que nous avons stocké dans le tableau est donc le chiffre 5. Ensuite nous avons stocké le chiffre 0b11111000 (que nous avons représenté sous sa forme binaire afin de simplifier les choses). Nous ne détaillerons pas ici le principe de la représentation binaire et n'allons pas vous faire un cours sur ce thème (vous trouverez énormément d'informations à ce sujet sur Internet). Le 0b du début indique que le nombre est représenté sous sa forme binaire. Ensuite viennent les 8 bits qui représentent le nombre proprement dit. On utilisera un 1 pour représenter un trait et un 0 pour représenter un point. On retrouve bien les 5 traits (via cinq « 1 »). Nous avons mis les autres bits d'office à 0 afin de compléter l'octet et d'arriver à 8 bits.

Les 2 valeurs suivantes du tableau 5,0b01111000 codifient le chiffre 1 (représenté en morse par un point et 4 traits. Nous avons donc bien la valeur 5 pour le nombre total de points et de traits représentant le chiffre 1 et ensuite 01111 pour le point et les 4 traits (et ensuite toujours des zéro à la fin pour arriver à une représentation sur 8 bits).

Il en est de même pour la codification du reste des chiffres et des lettres de A à Z.

Description et explications (suite)

La suite des déclarations et initialisations est similaire à ce que nous avons déjà vu dans les autres applications, mis à part la déclaration d'une seconde variable de type tableau **char recu[13];** laquelle est prévue pour le stockage de données de type caractère. Toutefois contrairement aux cas précédents, nous n'initialiserons pas les données de ce tableau et allons nous contenter de déclarer le tableau afin de pouvoir l'utiliser dans le cœur de notre programme.

```
 k = 0 ;
while (Serial.available() > 0) {
   recu[k] = Serial.read();
   k++;
   delay(10); }
```

Voici maintenant une nouvelle partie du code très intéressante. Comme nous l'avons déjà indiqué il est possible à la platine « Flip & Click » d'échanger (via son port série) des informations avec votre ordinateur au travers de la connexion USB. A ce titre, nous avons déjà vu comment envoyer des informations depuis la platine « Flip & Click » vers la fenêtre du moniteur série de l'environnement de développement.

Les lignes de codes ci-dessus permettent à la platine de récupérer des caractères que vous pourrez saisir dans la fenêtre du moniteur. Pour ce faire, une boucle **while ( )** récupère les données présentes dans le buffer de réception du port série de la carte et les stocke une à une dans la variable tableau **recu[ ]**. La variable **k** est utilisée pour incrémenter l'index du tableau (cette variable nous permettra également à la sortie de la boucle **while ( )** et de vérifier si des données ont été réceptionnées via le test **if (k > 0)**

Pour pouvoir envoyer des données depuis la fenêtre du moniteur de l'environnement de développement, il suffit d'activer la fenêtre en question sur votre ordinateur (bouton « loupe » en haut à droite) et de cliquer dans la ligne en haut de cette fenêtre et de saisir des caractères avec votre clavier et enfin de solliciter la touche « Return ». C'est sur cette dernière action que la fenêtre du moniteur envoie toute la chaîne de caractères saisie vers le port série de la platine « Flip & Click ».

Reprenons la description de notre programme qui est ensuite structurée sur la base de 2 boucles qui se suivent, lesquelles vont respectivement vérifier que les caractères reçus depuis votre ordinateur correspondent bien aux chiffres 0 à 9 ou aux lettres (en majuscules) A à Z. Pour ce faire, le programme teste la valeur ASCII des données reçues. Chaque chiffre et chaque lettre disposent d'une représentation ASCII (une valeur qu'on leur donne pour les identifier). Par exemple le chiffre 0 est représenté par le nombre décimal 48, le chiffre 1 par la valeur 49. Le programme vérifie donc que les valeurs retournées par votre ordinateur soient comprises entre 48 et 57 (représentation des chiffres 0 à 9) et comprise entre 65 et 90 (représentation des lettres majuscules A à Z). Si tel n'est pas le cas, aucun code morse ne sera généré pour les caractères reçus. Ensuite on utilise la variable **pointrait** pour récupérer la représentation (en points et en traits) du caractère que l'on a reçu via la variable de type tableau **morse**.

A noter que suivant les cas, on soustrait la valeur 48 à la valeur ASCII reçue et on multiple ce résultat par 2 pour pouvoir se placer en « phase » sur l index du tableau par rapport aux caractères reçus. Le tableau morse commence par la représentation du chiffre 0, puis 1... Si par exemple on reçoit le chiffre 1, sa représentation ASCII est 49, on sera alors bien placé sur (49 – 48) = 1 * 2 = 2 donc sur la donnée du nombre de points et de traits du chiffre 1. On ajoute +1 à ce résultat pour le calcul de l'index de la représentation des points/traits du caractère récupéré.

On fait de même avec la représentation des lettres A & Z (sauf qu'on additionne en plus le chiffre 20 car on prend en compte les 20 premières données nécessaires au stockage des représentations des chiffre 0 à 9 en début de tableau).

On va ensuite réaliser une boucle dont le nombre d'itération sera égal au nombre de points et de traits sur lequel le caractère reçu est codé (cette information est aussi récupérée via la variable de type tableau morse).

Description et explications (suite)

Au cours de cette boucle, le programme va récupérer le bit de poids fort (le bit le plus à gauche) de la variable **pointrait** afin de savoir s'il s'agit d'un point(chiffre 0) ou d'un trait (chiffre 1).

On récupère cette information via l'instruction **bitRead( )** Le premier paramètre de cette instruction correspond à la variable sur laquelle on veut connaître la valeur d'un bit. Le deuxième paramètre correspond au positionnement de ce bit. Ici on met le chiffre 7 car on veut lire le bit de poids fort (celui qui est complètement à gauche). On test ensuite la valeur du bit pour voir si elle est égale à 1 (attention lorsqu'on teste une égalité vis à vis d'une valeur on utilise 2 fois le caractère ==). Si tel est le cas, on va générer visuellement un trait par un « flash » plus long sur la led. Si tel n'est pas le cas on générera un « flash » plus court sur la led (pour représenter un point).

Pour ce faire, on fait appel à des **fonctions**. Les fonctions sont des « petits morceaux » de programmes (généralement placés au bas du programme général). Ces « petits morceaux » de programmes peuvent être exécutés à tout moment depuis le programme principal. L'intérêt de ces derniers est qu'ils vous serviront à réaliser plusieurs fois les mêmes opérations sans avoir besoin d'avoir le même code à plusieurs endroits de votre programme et ainsi vous faire gagner de la place mémoire.

Dans notre cas, nous avons créé 2 fonctions **point ( )** et **trait ( )**
Vous pouvez donner le nom que vous voulez à vos fonctions (à condition que le mot ne soit pas déjà réservé pour une instruction propre à la platine « Flip & Click »). La structure doit être sous la forme :

```
void point()
  {
  // Instructions à réaliser par la fonction
  }
```

Dans notre cas, la fonction point genère un « flash » court sur la led et la fonction « trait » génère un « flash » plus long sur la led.

Nous pouvons désormais (depuis n'importe quel endroit de notre programme principal) écrire la ligne de code **trait();** pour « appeler » la fonction et faire en sorte que le programme principal exécute le « petit morceau » de programme de cette fonction puis continue la suite du programme principal. Si nous n'avions pas utilisé ces fonctions, il nous aurait fallu recopier le contenu de leur code à plusieurs endroits dans notre programme principal et au final la taille de notre programme aurait été plus importante.

Revenons à la description du déroulement du notre programme. Une fois la génération du point ou du trait effectuée, on utilise une nouvelle instruction **pointrait = pointrait << 1;** Les 2 << sur la variable **pointrait** permettent de décaler l'ensemble de ses bits vers la gauche **(**afin de pouvoir récupérer la prochaine codification du point ou du trait à envoyer). Une fois que le nombre de traits et de points a été généré par la led (pour le caractère reçu), le programme genère une légère pause afin de pouvoir différencier le code morse émis par chaque caractère.

L'opération se répète alors pour tous les autres caractères reçus dans le buffer de réception série.

Ainsi si vous saisissez la chaîne de caractères SOS (suivi de la touche « Return » dans la fenêtre du moniteur série de votre ordinateur), la led émettra 3 « flash » courts, puis 3 « flash » longs puis à nouveau 3 « flash » courts.

Application N° 011 Pilotage d'un servomoteur

Cette application va vous permettre de piloter la position d'un petit servomoteur à l'aide d'une résistance ajustable. Ce type de servomoteur est souvent utilisé dans le modélisme pour modifier la direction des roues d'une voiture ou les ailerons d'un avion. Il est constitué d'un petit moteur électrique associé à une électronique de commande ainsi qu'à des pignons avec une sortie sur un axe (capable généralement de tourner sur 180 °). Ce type de servomoteur dispose de 3 fils.

Rouge : Alimentation (généralement 5 V)
Marron (ou noir): Masse
Orange (ou jaune ou blanc) : Signal de commande

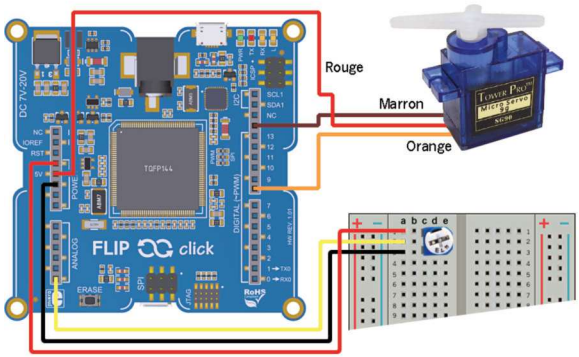
Ce fil est destiné à recevoir des trains d'impulsions (type signal PWM) dont la largeur des impulsions doit être généralement comprise entre 1 et 2 ms (ces valeurs correspondent aux 2 positions extrêmes que peut prendre l'axe du servomoteur).

Il est impératif de rappeler qu'il ne faut jamais (même si le servomoteur n'est pas alimenté) tourner manuellement l'axe d'un servomoteur avec la main, sous risque de casser ses petits pignons internes (non pris en compte par la garantie).

Comme indiqué ci-avant l'angle de rotation d'un servomoteur est d'environ 180°. Ce dernier dispose donc de butées internes interdisant d'aller au delà de cette variation. Pour les mêmes raisons, il est impératif de ne pas forcer la rotation du servomoteur au delà de ces butées (de façon manuelle ou par programmation).

Notre application va consister à générer des impulsions (signal PWM) sur l'entrée du servomoteur afin de pouvoir modifier sa position par rapport à celle du curseur d'une résistance ajustable.

```
#include <Servo.h>       // Fait appel à la librairie "Servo"
Servo Monservo;          // Création objet pour contrôler le servomoteur

const byte Rajustable = A5;    // N° du port relié à la résistance aj.
int valeur;                    // Variable utilisée résistance ajustable

void setup()  {
  Monservo.attach(8);          // Servomoteur sur broche 8
}

void loop()   {
  valeur = analogRead(Rajustable);    // Conversion analog./numérique
  valeur = map(valeur, 0, 1023, 0, 179);    // Mise à l'échelle
  Monservo.write(valeur);                   // Positionne le servomoteur
  delay(20);                                // Temporisation
}
```



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-dessus (attention à ne pas intervertir les 2 signaux correspondant au 3,3 et 5 V). Fixez un palonnier sur l'axe du servomoteur. Il n'est pas nécessaire de le visser pour vos tests. Saisissez alors le programme ci-dessus, raccordez la platine à votre ordinateur et téléversez le programme dans la platine « Flip & Click ».

Description et explications

Cette application relativement simple à comprendre fait appel à de nouvelles notions.

En premier lieu la notion de librairie. Les librairies s'apparentent à des « morceaux » de programmes réalisant une action précise, lesquels pourront être utilisés par votre application (sans que leur code n'apparaisse dans votre programme). Cette possibilité apporte de très nombreux avantages.

Vous allez par exemple pouvoir utiliser des librairies (développées par d'autres personnes) qui vont remplir une certaine tâche sans que vous ayez à vous pencher sur la façon dont ces dernières fonctionnent. Ceci vous permet ainsi de pouvoir utiliser de nouvelles possibilités... sans trop vous « casser » la tête.

Dans notre cas, afin d'éviter d'avoir à générer un signal PWM avec des valeurs d'impulsions « caractéristiques » nécessaires au pilotage de notre servomoteur, nous allons faire appel à la librairie **Servo** par le biais de l'instruction :

```
#include <Servo.h>
```

Cette librairie va permettre à votre programme de reconnaître de nouvelles instructions, comme celle associant la broche **8** de la platine « Flip & Click » au pilotage du servo **Monservo.attach(8);**

Dans la boule principale du programme on effectue la conversion « analogique/numérique » de la tension présente sur le curseur de la résistance ajustable, puis via une nouvelle instruction **map( );** nous allons modifier l'échelle de la variable permettant de piloter le servomoteur.

Pour piloter votre servo, il existe (grâce à l'usage de la librairie externe) une instruction dédiée : **Monservo.write( );**

Le problème est que la variable à envoyer au servomoteur doit correspondre à sa valeur en degré (de 0 à 180). Hors la variable récupérée après la conversion « analogique/numérique » se présente sous la forme d'un nombre compris entre 0 et 1023.

C'est pourquoi l'instruction **map( );** vous permettra de réaliser une mise à l'échelle très facilement. Le premier paramètre de l'instruction que nous avons utilisé : **map(valeur, 0, 1023, 0, 179);** correspond au nom de la variable à mettre à l'échelle, ensuite vous devez indiquer les 2 valeurs extrêmes de la première variation et les 2 variables extrêmes de la seconde variation afin que le programme s'occupe de tout !

Maintenant tournez la résistance ajustable et observez le palonnier du petit servomoteur.

Attention si se dernier se met à vibrer, c'est qu'il arrive en fin de course et qu'il est stoppé par une de ses butées.

Évitez dans la mesure du possible cette position. Et une nouvelle fois, ne tournez pas le palonnier à la main (que le servomoteur soit alimenté ou non).

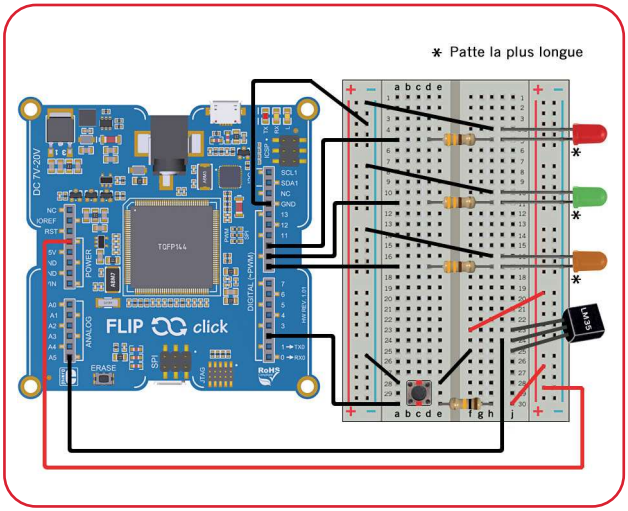
Application N° 012 Thermostat à leds

Cette application va vous permettre de réaliser un petit thermostat à led. Ce dernier met en œuvre 3 leds (rouge, verte et orange) associées à un capteur de température de type LM35.

Ce composant se présente sous la forme d'un petit boîtier (type transistor au format TO92) fournissant une tension comprise entre 0 et 1 V proportionnelle à la température ambiante. Le thermostat disposera de 3 leds permettant de surveiller une température de consigne.

A la mise sous tension, le thermostat va mesurer la température ambiante et l'assimiler à une température de consigne qui sera surveillée. Si la température monte et dépasse cette valeur de consigne, alors la led verte initialement allumée s'éteindra au profit de la led rouge (qui signalera un dépassement « haut »).

Au contraire si la température descend en dessous de la valeur de consigne, la led verte s'éteindra et la led orange s'allumera pour indiquer un dépassement « Bas ».




```
byte Temperature;      // Variable pour la mesure de la température
byte Consigne;         // Variable mémoire valeur de consigne
byte Etat;             // Variable dans laquelle on mémorise l'état du
thermostat
const byte LM35 = A5;   // Attribution de la sortie du LM35 au port A5
const byte LedR = 10;   // Attribution du port 10 à la led rouge
const byte LedV = 9;    // Attribution du port 9 à la led verte
const byte LedO = 8;    // Attribution du port 8 à la led orange
const byte BP = 2;      // Attribution du port 2 au bouton-poussoir

void setup()  {
 for (byte i = 8 ; i < 11; i++)   // Initialisation des ports 8 à 10 en sortie
 {
 pinMode(i, OUTPUT);         // Attribution des ports en sortie
 digitalWrite(i, LOW);       // Eteint la led
 }
 pinMode(BP, INPUT);                // Configure port du BP en entrée
 Consigne = analogRead(LM35);  // Mémorise la valeur de consigne
}

void loop()    {
  if (digitalRead(BP) == LOW)  Consigne = analogRead(LM35);
Temperature = analogRead(LM35);          // Mesure sortie du LM35

  if (abs(Consigne- Temperature) <= 3 ) Etat = 0;

  if (Consigne > Temperature)
    { if ((Consigne-Temperature) > 3 ) Etat = 1;}
  else
    { if ((Temperature-Consigne) > 3 ) Etat = 2;}

 switch (Etat) {     // Allume leds thermostat fonction valeur consigne
  case 1:            // Température en dessous valeur de consigne
      digitalWrite(LedR, LOW);  // Eteint tout sauf la led orange
      digitalWrite(LedV, LOW);
      digitalWrite(LedO, HIGH);
   break;
  case 2:            // Température au dessus valeur de consigne
      digitalWrite(LedR, HIGH);  // Eteint tout sauf la led rouge
      digitalWrite(LedV, LOW);
      digitalWrite(LedO, LOW);

    break;
  default:           // Cas où la valeur de consigne est respectée
      digitalWrite(LedR, LOW);   // Eteint tout sauf la led verte
      digitalWrite(LedV, HIGH);
      digitalWrite(LedO, LOW);
  break;
 }
 delay(10);
}
```

Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage de la page de gauche. Vérifiez le sens du capteur LM35 (avec son méplat).

Saisissez alors le programme, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Description et explications

Dans la partie initialisation et déclaration du programme on va définir les ports de la platine « Flip & Click » utilisés pour: piloter les différentes leds, ainsi que pour raccorder le bouton-poussoir et le capteur LM35.

On va également déclarer des variables pour stocker : la valeur de consigne, l'état du thermostat et le résultat de la conversion « analogique/numérique » de la tension présente en sortie du LM35 (que nous appellerons « injustement» **Temperature**).

On en profite à ce titre pour utiliser la variable **Consigne** pour commencer par stocker le résultat de la conversion «analogique/numérique» de la tension en sortie du LM35 qui correspondra à la « température normale » du thermostat.

Le programme principal va ensuite vérifier si on sollicite le bouton-poussoir. Si tel est le cas, on affecte à nouveau la valeur de consigne à la dernière mesure du LM35. Ceci permet donc à tout moment de « calibrer » le thermostat avec une nouvelle valeur de consigne si nécessaire.

Ensuite le programme soustrait la valeur de consigne à la valeur mesurée et vérifie que la valeur absolue associée soit inférieure ou égale à 3 unités. Pour ce faire, on fait appel à une nouvelle instruction **abs( )**. On utilise une valeur absolue car on ne peut pas savoir si la valeur de consigne est inférieure ou supérieure à la valeur mesurée.

Attention, on ne teste pas un écart de 3°C ! pour rappel, on ne mesure pas directement une température en sortie du LM35... mais une tension... elle même convertie en un chiffre compris entre 0 et 1023. Dans les faits, la valeur retournée par la conversion « analogique/numérique » est bien inférieure à 1023 puisque la tension max. en sortie du LM35 ne dépasse jamais 1 V.

On teste donc que le résultat de la conversion « analogique/numérique » de la tension du LM35 soit « stable » et ne dépasse pas 3 unités. Si tel est le cas, on initialise la variable **Etat** avec la valeur 0.

Dans un deuxième temps, on teste cette fois-ci si la valeur de consigne est supérieure à la valeur mesurée, si tel est le cas, on vérifie si l'écart entre les 2 valeurs est supérieur à 3 unités.

Si tel est le cas, on en déduit que la température est en dessous de la valeur de consigne (dépassement « bas » du thermostat) et on initialise la variable Etat avec la valeur 1.

Dans l'autre cas (si la valeur de consigne était inférieure à la valeur ambiante mesurée), on teste à nouveau l'écart entre les 2 et si cet écart est supérieur à 3 unités on affecte la valeur 2 à la variable **Etat** pour mémoriser un dépassement « haut » du thermostat.

Description et explications (suite)

La suite du programme va nous permettre de visualiser l'état du thermostat sur les 3 leds grâce à une nouvelle instruction

```
switch (variable) {
   case 1:
     // Actions à réaliser si la variable est égale à 1
     break;
   case 2:
     // Actions à réaliser si la variable est égale à 2
     break;
   default:
     // Actions à réaliser si la variable ne correspond à aucune des autres valeurs
   break;
 }
```

Cette instruction permet de réaliser des actions en fonction de différentes valeurs. On utilisera la variable **Etat** avec l'instruction **switch ( )**. La syntaxe de cette instruction est relativement simple. On indique la valeur de la variable à associer aux actions avec les instructions **case**.

Dans notre programme, nous allumerons alors uniquement la « bonne » led en fonction de la valeur de la variable **Etat**.

Par exemple, si la variable **Etat** est à 1, on allumera que la led orange pour montrer que le thermostat est en dépassement « bas ». A noter que la dernière instruction **default** permet de gérer les autres valeurs par défaut (dans notre cas lorsque la variable Etat n'est ni à 1, ni à 2)... donc forcément dans le cas ou sa valeur sera 0.

Nous aurions également pu ajouter une instruction **case 0** pour traiter le cas ou la variable Etat est à 0 et ne pas utiliser l'instruction **default**.

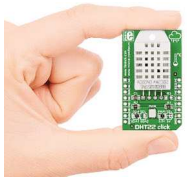
Après le téléversement, seule la led verte doit être allumée. Approchez vous au plus près du capteur LM35 et soufflez dessus afin de générer de l'air chaud (ou placez vos doigts sur le capteur). A ce stade, la led verte doit s'éteindre et la led rouge doit s'allumer pour signaler un dépassement « haut » de la valeur de consigne.

Après quelques secondes, la led rouge va s'éteindre et la led verte va se rallumer. Il peut y avoir une période d'instabilité pendant laquelle les 2 leds s'allument alternativement lorsque la température mesurée « tourne » autour de la température de consigne. Maintenant, mouillez un linge avec de l'eau très froide et appliquez le sur le capteur LM35.

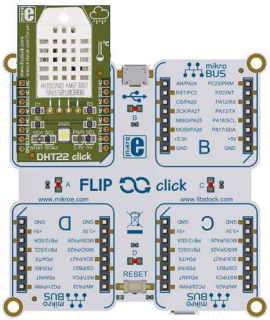
A ce stade, la led verte s'éteindra et la led orange s'allumera pour signaler un dépassement « bas » du thermostat. Eloignez le linge du LM35 et après quelques secondes, la led verte s'allumera à nouveau tandis que la led orange s'éteindra.

Vous pouvez à tout moment réinitialiser la valeur de consigne (avec la dernière valeur mesurée) en sollicitant le bouton-poussoir.

Application N° 013 Récupération des données d'un capteur DHT22



Le module « DHT22 Click » livré avec votre starter-kit intègre un capteur de température et d'humidité. Ce dernier est capable de mesurer le taux d'humidité de 0 à 100% et la température de -40 °C to +125 °C (dans les faits, l'enveloppe plastique du capteur ne pourra pas résister à de telles températures extrêmes. Mais le capteur sera suffisant pour effectuer des mesures au sein de votre habitation).



Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «DHT22 click » sur le support **A** (attention au sens). Connectez ensuite à nouveau la platine « Flip & Click » à votre ordinateur.

La communication avec le capteur «DHT22 » nécessite un dialogue via une liaison numérique série. Pour vous simplifier la « vie », il existe également une excellente librairie qui s'occupera de toute la partie communication. Afin de pouvoir utiliser cette librairie, il vous faut quitter complètement l'environnement IDE, puis la télécharger (via le bouton «Download ZIP») à partir de cette adresse:

https://github.com/adafruit/DHT-sensor-library

Une fois téléchargé, décompressez le fichier (qui se présente sous la forme d'un dossier) dans le répertoire:
C:\Program Files (x86)\Arduino\libraries ... si bien sûr vous avez installé l'environnement IDE dans le dossier C:\Program Files (x86).

A ce stade exécutez l'environnement de développement, puis chargez l'exemple :

Fichier → Exemples → DHT Sensor Library → DhtTester

Modifiez la ligne : **#define DHTPIN 2** par **#define DHTPIN 77** afin d'indiquer au programme avec quelle broche vous pilotez le capteur.

Téléversez ensuite le programme, puis une fois celui-ci transféré dans la platine « Flip&Click », lancez la fenêtre du moniteur série dans l'environnement IDE et consultez le résultat des mesures à l'écran (voici un exemple de ce que pourra vous retourner le capteur):

DHTxx test!
Humidity: 38.40 % Temperature: 22.40 *C 72.32 *F Heat index: 21.70 *C 71.06 *F
Humidity: 38.60 % Temperature: 22.40 *C 72.32 *F Heat index: 21.70 *C 71.07 *F
Humidity: 38.80 % Temperature: 22.40 *C 72.32 *F Heat index: 21.70 *C 71.07 *F

Soufflez maintenant au plus près du capteur pour voir évoluer le taux d'humidité à la hausse (et éventuellement la température). Secouez ensuite la platine un petit moment pour que le taux d'humidité redescende. Notez que le capteur réagit avec une certaine inertie. Attention ce dernier n'est pas étanche... ne projetez pas d'eau dessus pour le tester !

Application N° 014 Thermomètre sur écran LCD

Cette application va vous permettre de réaliser un thermomètre numérique avec affichage sur un écran LCD. Ce dernier met en œuvre le capteur de température LM35 (que nous avons déjà utilisé précédemment). Nous utiliserons également un afficheur LCD alphanumérique de 2 lignes de 16 caractères. Ce modèle est normalement prévu pour fonctionner sous une tension de 5 V. Nous utiliserons donc la sortie 5 V de la platine pour alimenter l'afficheur et bien que les ports de l'afficheur soient normalement conçus pour fonctionner sous 5 V, nous pourrons les piloter par les ports (d'un niveau logique de 3,3 V) de la platine « Flip & Click » car les broches utilisées sur l'afficheur ne sont que des entrées. Dès lors, on n'appliquera pas de tension 5 V sur les ports de la platine « Flip & Click » mais du 3,3 V (issus de la platine « Flip & Click » vers l'afficheur, ce qui d'un point de vue électrique ne risque rien).

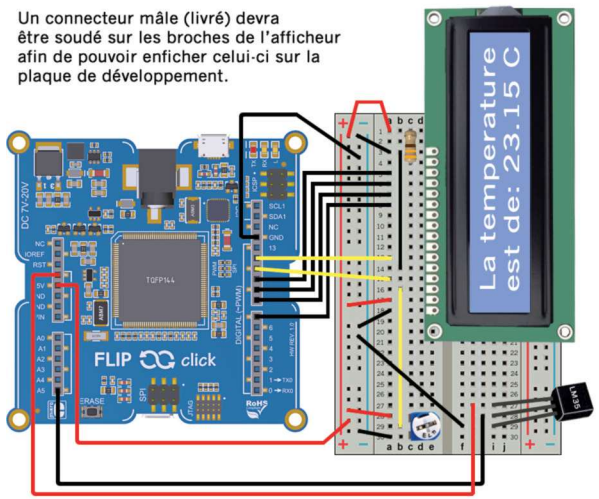
Toutefois, gardez toujours à l'esprit (comme déjà indiqué dans cet ouvrage) qu'il ne faut jamais appliquer de tension supérieure à 3,3 V sur les ports de la platine « Flip & Click ».

Pour en revenir à l'afficheur, ce dernier fonctionnant en 5 V, ses entrées seront considérées à un niveau logique « haut » pour une tension supérieure à 2,5 V. Donc les ports de la platine « Flip & Click » qui sont en 3,3 V seront bien tolérés et « compris » par l'afficheur. L'afficheur dispose de 3 broches de commande (dont une que l'on « force » à la masse) et de 4 broches destinées à recevoir les données en provenance de la platine « Flip & Click ».

Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre. Prenez soin de ne pas vous tromper dans le câblage de l'afficheur. La résistance de 330 ohms sert à limiter le courant du rétro-éclairage de l'afficheur. La résistance ajustable sert à régler le contraste de l'afficheur (tournez celle-ci dans un sens ou dans l'autre pour obtenir un affichage correct). Vérifiez également le sens du capteur LM35 (avec son méplat). Saisissez alors le programme ci-dessus, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

```
#include <LiquidCrystal.h>        // Ajout librairie gestion afficheur LCD
LiquidCrystal lcd(11, 12, 7, 8, 9, 10);     // Attribution broches afficheur
int Temperature;                            // Variable mesure de la température
const byte LM35 = A5;     // Attribution de la sortie du LM35 au port A5

void setup()    {
lcd.begin(16, 2);   // Déclaration afficheur LCD 2 lignes de 16 car.
lcd.clear();            // Efface écran et place le curseur en haut à gauche
lcd.print("La temperature  ");     // Affichage message "La température"
lcd.setCursor(0, 1);    // Place curseur sur 2ème ligne en bas à gauche
lcd.print("est de:       C ");          // Affichage du message "est de:   C"
}

void loop()    {
Temperature = analogRead(LM35);    // Mesure tension sortie du LM35
float millivolt = (Temperature/1023.0)*3200;   // Conv. tension -> temp.
float celcius = millivolt/10;
lcd.setCursor(8, 1);      // Place curseur 2ème ligne au milieu de l'écran
lcd.print(celcius);         // Affiche la température sur l'écran LCD
delay(1000);                // Temporisation entre 2 mesures
}
```



Description et explications

Le capteur LM35 est idéal car il est très économique et facile à utiliser. Pourtant sa principale limitation est que sa tension de sortie ne varie que de 0 à 1 V (alors que les entrées de conversion « analogique/numérique » peuvent recevoir des tensions de 0 à 3,3 V. Il en résulte que la mesure ne se fera pas à pleine échelle et que nous ne pourrons pas profiter de la pleine résolution des convertisseurs et que la précision de la mesure ne sera pas très bonne ... mais largement suffisante pour notre projet).

Concernant l'afficheur LCD, nous allons utiliser une librairie spécialement adaptée à ce dernier, laquelle va vous simplifier grandement la vie ! Celle-ci est appelée via l'instruction

#include <LiquidCrystal.h>

La ligne suivante **LiquidCrystal lcd(11, 12, 7, 8, 9, 10);** permet d'attribuer et de configurer le numéro des ports de la platine « Flip & Click » qui devront être utilisés pour piloter l'afficheur (dans notre cas, il s'agit des broches 7 à 12).

On retrouve ensuite les déclarations habituelles, plus une nouvelle déclaration dédiée à l'afficheur

lcd.begin(16, 2);

Laquelle vous permet d'indiquer au programme que l'afficheur dispose de 2 lignes de 16 caractères. Le programme utilise ensuite encore plusieurs nouvelles instructions spécialement dédiées au pilotage de l'afficheur

lcd.clear();

Permettant d'effacer le contenu de l'afficheur et de placer son curseur en haut à gauche de l'écran.

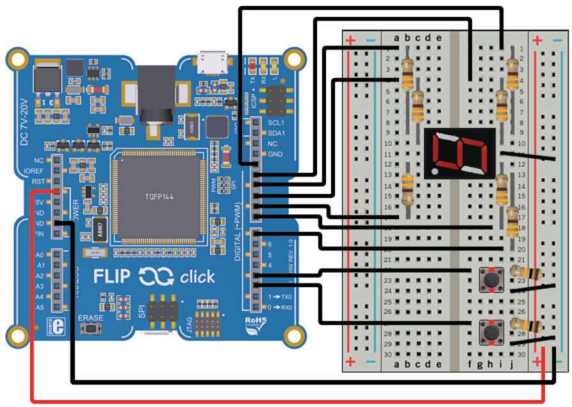
lcd.print("La temperature "); Permet d'afficher un message sur l'écran de l'afficheur (à partir de la position du curseur). Ici on affiche « La temperature ». Attention comme vous pouvez le constater l'afficheur n'accepte pas les caractères avec les accents.

L'instruction suivante **lcd.setCursor(0, 1);** permet de placer le curseur à gauche sur la deuxième ligne. Le premier paramètre correspond à la coordonnée X et le second paramètre à la coordonnée Y de la position du curseur.

Au cours de la boucle principale, le programme va effectuer une conversion « analogique/numérique » de la tension présente en sortie du LM35. Les 2 lignes qui suivent dans le programme permettent de convertir la tension du LM35 en température via un calcul d'échelle dans lequel on utilise l'instruction **float** permettant d'obtenir un résultat sur un nombre à virgule.

Le résultat ainsi obtenu sera alors affiché sur l'écran de l'afficheur. Enfin une temporisation permet d'espacer les mesures toutes les secondes.

Application N° 015 Compteur / décompteur sur affichageur 7 segments



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-dessus. Prenez soin de ne pas vous tromper dans le câblage des différentes résistances à associer avec l'afficheur.

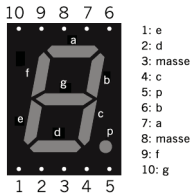
Saisissez alors le programme présenté en page suivante puis raccordez la platine à votre ordinateur et téléversez-y le programme pour tester ce dernier.

Sollicitez alors chacun des deux boutons-poussoirs pour voir comment réagit l'affichage.

Cette application va vous permettre de réaliser un compteur / décompteur sur un afficheur 7 segments à leds. Ce type d'afficheur est composé de 8 leds.

7 d'entre-elles (à la forme rectangulaire) sont agencées de telle sorte qu'elles représentent des segments capables (suivant les leds allumées ou non) d'afficher les chiffres **0** à **9** ainsi que les lettres **A**, **b**, **C**, **c**, **d**, **E** et **F**.

La 8ème led permet d'allumer un point (généralement utilisé pour réaliser une virgule de séparation lorsque plusieurs afficheurs sont utilisés les uns à côté des autres).



Le modèle d'afficheur livré avec votre starter-kit dispose d'une cathode commune à toutes les leds (qu'il vous faudra raccorder à la masse).

Le schéma ci-dessus donne la correspondance entre les connexions de l'afficheur et les différentes anodes de ses leds (qu'il vous faudra alimenter chacune au travers d'une résistance).

Pour notre application, nous disposerons de 2 boutons-poussoirs. Le premier vous permettra d'incrémenter le compteur d'une unité. le second bouton-poussoir vous permettra de décompter le compteur d'une unité. Le compteur est initialisé avec la valeur 0.

En utilisant successivement le premier bouton-poussoir, vous obtiendrez ainsi l'affichage de la valeur 1 puis 2 puis 3... jusqu'à 9. En sollicitant à nouveau le bouton-poussoir, le compteur repassera à la valeur 0, puis 1 puis 2... etc...

Si le compteur est à 0 et que vous utilisez successivement le second bouton-poussoir (celui qui décompte), alors le compteur repassera à la valeur 9, puis 8, puis 7... etc ...

Le programme qui gère cette application est relativement simple à comprendre. Ce dernier fait appel à de nombreuses notions déja vues au cours des applications précédentes.

```
const byte BPC = 2;           // Attribution port 2 au BP de comptage
const byte BPD = 3;           // Attribution port 3 au BP de décomptage
byte Compteur = 0;            // Variable pour le compteur

const byte SEG[10][7] =       // Configuration des 7 leds de l'afficheur
{
  {1,1,1,1,1,1,0},            // Chiffre = 0
  {1,0,1,0,0,0,0},            // Chiffre = 1
  {1,1,0,1,1,0,1},            // Chiffre = 2
  {1,1,1,1,0,0,1},            // Chiffre = 3
  {1,0,1,0,0,1,1},            // Chiffre = 4
  {0,1,1,1,0,1,1},            // Chiffre = 5
  {0,1,1,1,1,1,1},            // Chiffre = 6
  {1,1,1,0,0,1,0},            // Chiffre = 7
  {1,1,1,1,1,1,1},            // Chiffre = 8
  {1,1,1,0,0,1,1}             // Chiffre = 9
};

void setup() {
  for (byte i = 7 ; i < 14; i++)  // Initialisation des ports 7 à 13
  {
  pinMode(i, OUTPUT);         // Attribution des ports en sortie
  digitalWrite(i, LOW);       // Eteint les leds de l'afficheur 7 segments
  }

  pinMode(BPC, INPUT);        // Le port BP comptage en entrée

  pinMode(BPD, INPUT);        // Le port BP décomptage en entrée
}

void loop()
{
  if (digitalRead(BPC) == LOW) {   // Test BP de comptage
    Compteur++;                    // Incrémente le compteur
    delay (10);
    while (digitalRead(BPC) == LOW) {  }  // Attendre relâchement BP
    delay (10);

  }
  if (digitalRead(BPD) == LOW) {       // Test BP de décomptage
    Compteur--;                        // Décrémente le compteur
    delay (10);
    while (digitalRead(BPD) == LOW) {  }   // Attendre relâchement BP
    delay (10);
  }

  if (Compteur == 10) {Compteur = 0;}      // test si dépassement haut

  if (Compteur == 255) {Compteur = 9;}     // test si dépassement bas

  for(byte j = 7; j < 14 ; j++)      // Affichage du compteur sur l'afficheur
    {
      digitalWrite(j, SEG[Compteur][j-7]);     // Récupère leds à afficher
    }
}
```

Description et explications

Dans la phase d'initialisation du programme, on attribue les ports 2 et 3 de la platine « Flip & Click » aux mots **BPC** et **BPD** (diminutif de Bouton-Poussoir Compteur et de Bouton-poussoir Décompteur). On déclare ensuite la variable **Compteur** qui nous servira bien évidemment à stocker la valeur... du compteur !

On déclare ensuite une variable de type tableau à 2 dimensions **SEG[10][7]** dans laquelle on va stocker l'état que devront prendre les ports 7 à 13 pour représenter respectivement les chiffres 0 à 9.

Par exemple pour représenter le chiffre 0, les ports 7 à 12 seront à 1 et le 13 port sera à 0 **{1,1,1,1,1,1,0}** afin que le segment **g** au milieu de l'afficheur (voir figure sur la page de gauche) ne soit pas allumé. On a déjà utilisé ce principe de mémorisation dans l'application N° 9 du dé électronique pour mémoriser les différentes faces du dé.

Dans la partie initialisation, on va ensuite indiquer à la platine « Flip & Click » que les ports 7 à 13 seront utilisés en sortie (en prenant soins d'y placer un niveau logique bas afin qu'aucun segment de l'afficheur ne soit éclairé). On initialise aussi les ports 2 et 3 en entrée (afin de pouvoir les relier aux boutons-poussoirs).

Le programme principal effectue une boucle sans fin dans laquelle on commence par tester si on appuie sur le bouton-poussoir de comptage, puis par tester si on appuie sur le buton-poussoir de décomptage. Suivant les cas, on mettra à jour la valeur de la variable **Compteur** (en l'incrémentant ou en la décrémentant). Dans les 2 cas, on utilisera des temporisations permettant de gérer l'effet du rebond des boutons-poussoirs (déjà vu dans l'application N° 3) et on attendra que les boutons-poussoirs soient relâchés. Ensuite on testera les dépassements haut et bas de la variable **Compteur** en la remettant à jour si nécessaire.

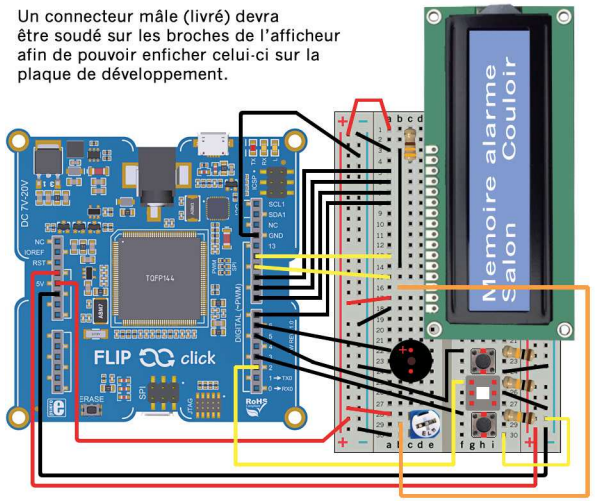
La dernière étape de la boucle principale consiste à mettre à jour l'état des ports 7 à 13 pour allumer les leds de l'afficheur en fonction de la valeur de la variable **Compteur**.

Application N° 016 Mini-centrale d'alarme 2 zones à écran LCD

Cette application va vous permettre de réaliser une mini-centrale d'alarme extrêmement complète. Cette dernière dispose de 2 zones de surveillance (une zone à déclenchement instantanée et une zone à déclenchement retardée). La centrale se met en service par un interrupteur (ce dernier est censé représenter une clef de mise en service de type « Marche/Arrêt »).

La centrale dispose d'un système d'affichage sur un écran LCD de 2 lignes de 16 caractères et d'un buzzer faisant office de sirène. A l'image d'une vraie centrale d'alarme, ce système dispose de différentes temporisations de sortie, d'entrée et d'alarme ainsi que d'une mémoire alarme (avec affichage « en clair » du nom des zones ayant créée un déclenchement).

Note : Bien que n'ayant rien à envier à un « vrai » système d'alarme en terme de possibilités, gardez à l'esprit que cette application reste une « maquette » de prototypage et d'expérimentation et qu'il ne faut pas l'utiliser tel quel pour protéger votre maison. Il n'est pas possible de raccorder directement des capteurs d'ouverture sur la platine car sur un vrai système d'alarme, l'électronique utilisée dispose de systèmes de protection additionnels sur les entrées (pour éviter de les endommager en cas de surtensions ou de perturbations extrêmes) ainsi que des sorties sur relais pour pouvoir piloter des sirènes et bien sûr d'une alimentation secourue par batterie.



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-dessous Prenez soin de ne pas vous tromper dans le câblage de l'afficheur et dans le sens des boutons et de l'interrupteur (**suivez bien les repères carrés rouges**). Saisissez alors le programme proposé dans les pages suivantes, raccordez la platine à votre ordinateur et téléversez-le dans la platine « Flip & Click ».

Description et explications

La partie déclaration du programme reprend celle de l'afficheur que nous avons déjà utilisé précédemment (nous n'y reviendrons pas). Vous trouverez ensuite la déclaration de tous les paramètres de fonctionnement de la centrale d'alarme (que vous pourrez modifier à volonté).

A savoir : les temporisations de sortie et d'entrée (que vous pouvez modifier de 1 à 99 secondes), la temporisation d'alarme (modifiable de 1 à 999 secondes) ainsi que le nom des 2 zones de votre alarme (chaque nom ne doit pas dépasser 8 caractères).

```
#include <LiquidCrystal.h>              // Ajout de la librairie afficheur LCD
LiquidCrystal lcd(12, 11, 10, 9, 8, 7); // Attribution broches afficheur
const byte Clef = 2;                    // Clef "M/A" sur le port 2
const byte ZoneInst = 3;                // Zone instantanée sur le port 3
const byte ZoneRet = 5;                 // Zone retardée sur le port 5
const byte Buzzer = 6;                  // Buzzer sur le port
byte Tempo;               // Variable générale gestion des temporisations
byte MemZoneI = 0;    // Memoire alarme zone instantanée
byte MemZoneR = 0;  // Memoire alarme zone Retardée
int Compteur;             // Variable génération des bips sonores fugitifs
const byte TempoSortie = 15;      // Durée de la temporisation de sortie
const byte TempoEntree = 10;    // Durée de la temporisation d'entrée
const byte TempoAlarme = 180;  // Durée de la temporisation d'alarme
const char NomZoneI[] = "Salon";        // Nom de la zone instantanée
const char NomZoneR[] = "Couloir";    // Nom de la zone retardée

void setup()  {
  lcd.begin(16, 2);         // Déclaration afficheur LCD  2 x 16 caractères
  pinMode(Clef, INPUT);           // Port relié à la Clef en entrée
  pinMode(ZoneInst, INPUT);   // Port de la zone instantanée en entrée
  pinMode(ZoneRet, INPUT);   // Port de la zone retardée en entrée
  Arret ();                       // Affficher le message "centrale à l'arrêt"
}

void loop() {
  Compteur = 0;      // Initialise compteur bips sonores mémoire alarme
  while (digitalRead(Clef) == HIGH) // Tant que centrale pas en marche
  {
  if (MemZoneI == 1 | MemZoneR ==1)     // Test mémoire alarme ?
  {
    if (Compteur > 2000000)
      {
        analogWrite(Buzzer, 80);                // Bip sonore fugitif
        delay(10);
        analogWrite(Buzzer, 0);
        Compteur = 0;       // Réinitialisation compteur bip sonores fugitifs
      }
  }
    Compteur ++;  // Incrémente compteur génération de Bip sonores
 }
  delay(50);        // Temporisation intégration du rebond de la clef "M/A"
  lcd.clear();       // Efface l'écran et place le curseur en haut à gauche
  lcd.print("Temporisation de");    // Affichage activation tempo de sortie
  lcd.setCursor(0, 1);                  // Place le curseur sur la 2ème ligne
  lcd.print("sortie active   ");
  MemZoneI = 0;    // Initialise mémoire alarme de la zone instantanée
  MemZoneR = 0;  // Initialise mémoire alarme de la zone retardée
  Tempo = TempoSortie;      // Récupère valeur temporisation de sortie

  while (digitalRead(Clef) == LOW  &  Tempo !=255)
  {
    lcd.setCursor(14, 1);    // Place curseur 2ème ligne affichage tempo
    lcd.print(Tempo);     // Affiche la valeur de la temporisation de sortie
    lcd.print(" ");           // Affiche caractère vide (efface "anciens" digits)
    analogWrite(Buzzer, 80);                // Bip sonore
    delay(80);
    analogWrite(Buzzer, 0);
    Tempo--;                    // Décompte la durée temporisation de sortie
    delay(920);
  }

  if (digitalRead(Clef) == LOW) Actif (); //

//------------------------- La centrale est armée et surveille ses 2 zones

  while (digitalRead(Clef) == LOW)      {
    if (digitalRead(ZoneInst) == LOW)   // Test intrusion zone Inst.
     {
       MemZoneI = 1;       // Mémorise alarme sur la zone instantanée
       Sirene() ;              // Active la sirène
     }
    if (digitalRead(ZoneRet) == LOW)  Entree();
  }

//---------------------------------------------- La centrale vient d'être désarmée

  if (MemZoneI == 0 & MemZoneR ==0)   // Test mémoire alarme ?
   {Arret ();}
    else
   {
     lcd.clear();      // Efface écran et place le curseur en haut à gauche
     lcd.print("Memoire alarme  "); // Affichage message mém. alarme
     if (MemZoneI == 1)     // Test mémoire alarme zone instantanée ?
      {
       lcd.setCursor(0, 1);            // Place le curseur sur la 2ème ligne
       for (byte i=0; i<5; i++)        // 5 = nombre lettres nom zone instant.
        {
         lcd.print(NomZoneI[i]);    // Affiche nom zone instantanée
        }
      }

     if (MemZoneR == 1) // Test mémoire alarme sur la zone retardée ?
      {
       lcd.setCursor(8, 1);          // Place le curseur 2ème ligne au milieu
       for (byte i=0; i<7; i++)      // 7 = nombre lettres nom zone retardée
        {
         lcd.print(NomZoneR[i]);    // Affiche nom zone retardée
        }   }    }   }

//---------------------------------------- Fonctions ----------------------------------

   void Entree()
   {
     Tempo = TempoEntree;    // Récupère valeur temporisation entrée
     lcd.clear();     // Efface l'écran et place le curseur en haut à gauche
     lcd.print("Detection zone  ");   // Affichage du message
     lcd.setCursor(0, 1);                  // Curseur sur la 2ème ligne
     lcd.print("Tempo Entree    ");
     while (digitalRead(Clef) == LOW  &  Tempo !=255)
     {
```

```
  lcd.setCursor(13, 1);          // Place le curseur sur la 2ème ligne
        lcd.print(Tempo);        // Affiche la valeur  temporisation d'entrée
        lcd.print("  ");         // Affiche caractère vide pour effacer digit
        Tempo--;                 // Décompte durée  temporisation d'entrée
        delay(1000);             // Temporisation d'une seconde
      }
      if (Tempo ==255)           // Regarde si fin de la tempo d'entrée ?
       {
        MemZoneR = 1;            // Si tel est le cas, on mémorise l'alarme et
        Sirene() ;               // on active la sirène
      }
}

//---------------------------------------------

    void Sirene()
    {
     Tempo = TempoAlarme;        // Récupère valeur tempo. alarme
     lcd.clear();     // Efface l'écran et place le curseur en haut à gauche
     lcd.print("Alarme Intrusion");  // Affichage du message
     lcd.setCursor(0, 1);            // Place le curseur sur la 2ème ligne
     lcd.print("Temp Alarme     ");
     while (digitalRead(Clef) == LOW  &  Tempo !=255)
     {
       lcd.setCursor(13, 1);      // Place curseur 2ème ligne
       lcd.print(Tempo);          // Affiche la valeur temporisation d'alarme
       lcd.print("  ");           // Affiche un caractère à "vide"
       Tempo--;                   // Décompte durée temporisation d'alarme
       for (byte i=5; i<100; i++)
        {
          analogWrite(Buzzer, i);          // Génère bruit de la sirène
          delay(10);
        }
          delay(750);
          analogWrite(Buzzer, 0);
          delay(100);
     }
     Actif();       // Affiche de nouveau que le systeme d'alarme est actif
  }

  void Arret ()
  {
    lcd.clear();     // Efface l'écran et place le curseur en haut à gauche
    lcd.print("Systeme d'alarme");    // Affichage état centrale d'alarme
    lcd.setCursor(0, 1);                // Place le curseur sur la 2ème ligne
    lcd.print("   a l'arret    ");
  }

   void Actif()
  {
    lcd.clear();     // Efface l'écran et place le curseur en haut à gauche
    lcd.print("Systeme d'alarme");    // Affichage état centrale d'alarme
    lcd.setCursor(0, 1);                // Place le curseur sur la 2ème ligne
    lcd.print("     ACTIF      ");
  }
```

Description et explications (suite)

Dans la partie initialisation, le programme va également attribuer les ports de la platine « Flip & Click » aux entrées de détection (instantanée / retardée), à la clef « M/A » et au buzzer de la centrale et ensuite appeler une fonction (présente à la fin du programme) permettant d'afficher à l'écran que la centrale est à l'arrêt.

A ce stade, le programme principal entame une première boucle **while ( )** qui teste si la clef « M/A » est toujours sur la position « arrêt ».

A l'intérieur de cette boucle, on effectue un test sur 2 variables qui correspondent à la mémoire alarme des 2 zones de surveillance (instantanée et retardée).

Si une des variables n'est pas nulle (c'est à dire qu'il y a eu une alarme de mémorisée) alors la centrale va générer à intervalles espacés des « bip sonores » fugitifs sur le buzzer pour attirer votre attention (idéal pour vous signaler qu'il y a eu une tentative d'intrusion pendant votre absence lorsque vous rentrez chez vous et que vous stoppez votre alarme).

La gestion des bips sonores est prise en compte par la variable **Compteur** qui permet de déterminer la durée des espacements entre les « bip sonores ». Si aucune alarme n'est mémorisée, le buzzer restera silencieux.

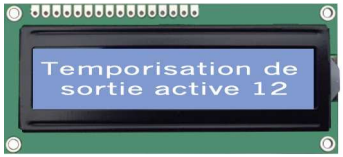
Si on active la clef « M/A » de la centrale, le programme quitte alors la première boucle **while ()** et l'écran de la centrale affiche le décompte de la temporisation de sortie avec un « Bip sonore » lancinant.

Cette temporisation de sortie vous permet de quitter votre maison avant que le système passe sous surveillance.

Description et explications (suite)

A noter au passage que nous profitons de cette étape pour initialiser les variables de mémoire alarme des 2 zones (afin d'indiquer à la centrale qu'aucune mémoire alarme n'est mémorisée à ce stade).

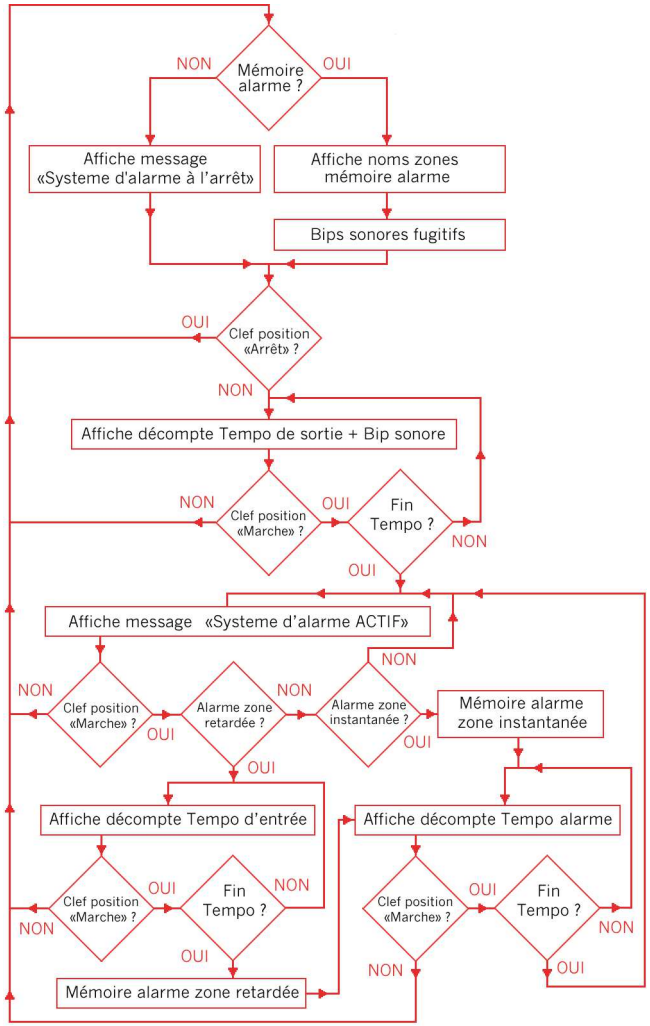
Durant la phase de temporisation de sortie , la centrale vous laisse le temps de sortir tranquillement de chez vous sans surveiller les 2 zones.



Les bips sonores générés par le buzzer durant cette phase vous permettent de disposer en plus d'une indication sur l'état du système. Ainsi, une fois arrivé à l'extérieur de votre maison, lorsque les bips sonores cessent.... vous aurez la certitude que votre système s'est bien activé et que votre habitation est désormais sous surveillance.

Si vous stoppez la centrale (à l'aide de la clef « M/A ») avant la fin de la temporisation de sortie, la centrale retournera dans la boucle initiale en indiquant que celle-ci est à l'arrêt.

Si en revanche vous laissez la temporisation arriver à terme, la centrale affichera un message indiquant qu'elle est sous surveillance avant d'entrer dans une nouvelle boucle **while ( )** pendant laquelle cette dernière va maintenant surveiller ses 2 zones.



Description et explications (suite)

A ce stade, si la zone de surveillance à déclenchement instantanée est sollicitée (c'est à dire s'il y a une intrusion sur cette zone), alors le programme mémorise une alarme dans la variable **MemZonel** et appelle la fonction **Sirène ( )** - voir en fin de programme.

Cette fonction affiche une information liée à cette intrusion avec un décompte de la temporisation d'alarme et le déclenchement du buzzer en « mode sirène ».

Si la zone retardée est à l'origine de l'intrusion, la centrale appelle la fonction **Entrée ( )** - voir en fin de programme – laquelle affiche le décompte de la temporisation d'entrée (pendant laquelle les 2 zones ne seront plus surveillées, ce qui vous laissera le temps de venir stopper l'alarme avec la clef « M/A »).

La zone retardée est donc généralement destinée à surveiller la porte d'entrée principale de votre logement afin de détecter votre retour dans l'habitation. En tant que propriétaire des lieux, vous aurez le temps de venir stopper le système d'alarme avant que celui-ci n'active les sirènes.

Si en revanche, il s'agit d'une intrusion frauduleuse sur cette zone retardée et que la centrale d'alarme n'est pas stoppée au terme de la durée de la temporisation d'entrée, la centrale considèrera qu'il s'agit d'une réelle intrusion et mémorisera l'information dans la variable **MemZoneR**.

A ce stade, le programme appelera la fonction **Sirène ( )** pour déclencher le buzzer en « mode sirène » et activer le décompte de la temporisation d'alarme (comme vu ci-avant).

Dans le cas d'une réelle tentative d'intrusion (sur la zone retardée ou instantanée), le cambrioleur n'a pas les clefs de l'installation permettant sa mise hors service, il en résultera que le décompte de la temporisation d'alarme finira par arriver à terme .

Le buzzer va alors stopper son mode « sirène » et la centrale passera à nouveau sous surveillance pour pouvoir se déclencher encore si une nouvelle intrusion survient sur une de ses 2 zones (à condition que les zones aient été refermées).

En effet, comme indiqué en introduction, ce système d'alarme est un modèle expérimental et pédagogique.

Afin de simplifier le programme qui gère celui-ci, nous avons réalisé une gestion simplifiée basée sur une détection de type impulsionnel (pour simuler une tentative d'intrusion, vous devez appuyer sur un des 2 boutons-poussoirs et le relâcher aussitôt).

Description et explications (suite)

A titre de comparaison, sur un vrai système d'alarme les zones sont généralement reliées à des contacts d'ouvertures (placés sur les portes et les fenêtres de l'habitation) ou sur les relais de détecteurs de mouvements. Nos boutons-poussoirs simulent le fonctionnement des relais d'un vrai détecteur de mouvement (dont le relais s'ouvre un court instant et se referme aussitôt) - comme le bouton-poussoir de notre alarme qu'on sollicite puis relâche aussitôt. Par contre, notre alarme n'est pas conçue pour simuler l'utilisation d'un contact d'ouverture qui peut rester ouvert en permanence après une tentative d'intrusion. Dès lors, si vous restez appuyé en permanence sur les boutons-poussoirs simulant les zones, le cycle d'alarme se répètera en boucle puisque la centrale verra les zones à nouveau en défaut au terme de la durée d'alarme lorsqu'elle repassera sous surveillance.

Dans tous les cas, si vous stoppez la centrale d'alarme après qu'il y ait eu une intrusion (pendant le cycle de temporisation d'alarme - si par exemple vous avez créée une alarme intempestive ou après le cycle de la temporisation d'alarme - s'il y a eu une tentative d'intrusion pendant votre absence), celle-ci passe par une dernière étape dans laquelle elle va vérifier l'état des 2 variables **MemZonel** et **MemZoneR**.

En fonction de leurs états, elle va afficher un message indiquant une mémoire d'alarme en inscrivant en plus à l'écran le nom de la zone responsable du déclenchement. Le nom de la zone instantanée s'affichera en bas à gauche de l'écran (si c'est elle qui est responsable d'une alarme), tandis que le nom de la zone retardée s'affichera au milieu en bas de l'écran si c'est elle qui est à l'origine de l'alarme. Si les 2 zones sont responsables d'une alarme... les 2 noms s'afficheront. Le programme reprend alors à l'étape de la première boucle (qui teste la mise en service de la centrale via sa « Clef » et la présence d'une mémoire alarme).

Comme indiqué ci-avant, il en résultera donc que suite à une ou à plusieurs alarmes en votre absence, la centrale d'alarme attirera votre attention par des « bip sonores » fugitifs lorsque vous la désarmerez en vous indiquant le nom des zones responsables des alarmes. Pour stopper cet affichage, il suffit de mettre la centrale en marche puis de l'arrêter aussitôt à l'aide de sa clef.

A noter que lors de l'affichage des temporisations le programme affiche systématiquement quelques caractères « vides » juste derrière. Ceci permet d'effacer les caractères pouvant rester à droite lorsque la temporisation passe des centaines aux dizaines ou des dizaines aux unités (par exemple lorsque la temporisation passe de 10 à 9). En l'absence de cette « astuce », l'écran afficherait 12...11....10, puis 90 (au lieu de 9) car la temporisation affichée est collée vers la gauche et le 0 du 10 resterait à l'écran.

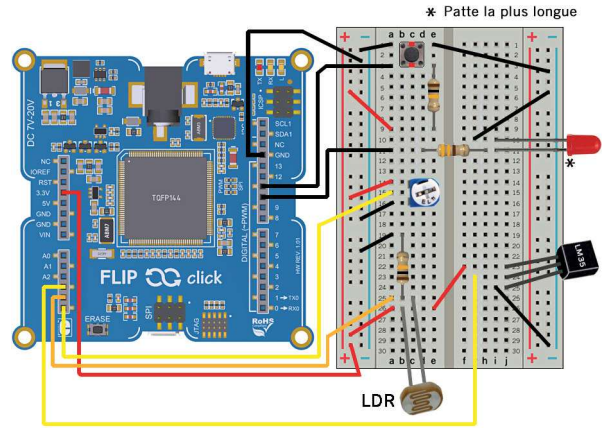
Si vous voulez aller plus loin...
Vous disposez d'un beau projet que vous pouvez améliorer en ajoutant par exemple plus de zones de détection ou en modifiant celui-ci pour être capable de pouvoir gérer des contacts d'ouvertures suceptibles de rester ouvert après une intrusion ou encore en ajoutant un capteur de température ou une horloge vous permettant d'horodater les alarmes et les mises « en/hors » services.

Application N° 017 Mini-centrale de mesure

Cette application va vous permettre de réaliser une petite centrale de mesure pilotée par votre ordinateur au moyen d'ordres que vous pourrez envoyer via la fenêtre du « moniteur série » (la centrale de mesure devra donc être reliée à votre ordinateur via la liaison USB).

Ainsi il vous sera possible d'afficher « sur commande »:

- La valeur de la tension présente sur le curseur d'une résistance ajustable.
- La valeur de la mesure effectuée sur une LDR.
- La valeur de la température retournée par un capteur de température LM35
- L'état d'un bouton-poussoir (appuyé ou relâché) .
- Vous pourrez également piloter l'état d'une led (pour l'allumer ou l'"éteindre) depuis votre ordinateur.




```
const byte Led = 10;                // Association de la Led au port 10
const byte BP = 11;                 // Association du BP au port 11
const byte LDR = A4;                // N° du port sur lequel est relié la LDR
const byte Rajustable = A5;  // N° du port relié à la résistance ajustable
const byte LM35 = A3;               // N° du port relié au capteur LM35
byte k;                     // Variable pour réception des ordres de l'ordinateur
int Temperature;            // Variable pour la mesure de la température
char recu[3];               // Buffer de réception des ordres de l'ordinateur

void setup() {
  Serial.begin(9600);               // Initialisation du port série
  pinMode(Led, OUTPUT);  // Le port de la led est en sortie
  digitalWrite(Led, LOW);     // Eteint led
  pinMode(BP, INPUT);         // Le port du bouton-poussoir est en entrée
}

void loop()  {
   k = 0 ;                                                  // Initialisation des variables
   recu[0] = 0;
   float millivolt = (analogRead(LM35)/1023.0)*3200;    // Mesure temp.
   float celcius = millivolt/10;

   while (Serial.available() > 0)  {          // Attente ordres de l'ordinateur
     recu[k] = Serial.read();
     k++;
     delay(10);    }

   switch (recu[0]) {                          // Regarde quel ordre a été reçu ?
    case 65:             // Code ASCII reçu = 65 (lette A) -> Allume la Led
        digitalWrite(Led, HIGH);        // Allume la led
     break;

    case 66:   // Code ASCII reçu = 66 (lette B) -> Demande l'état BP
        if (digitalRead(BP) == LOW)    // Envoi l'état du bouton-poussoir
        {Serial.println("Le bouton-poussoir est sollicité");}
        else
        {Serial.println("Le bouton-poussoir n'est pas utilisé");}
     break;
    case 69:             // Code ASCII reçu = 69 (lette E) -> Eteint la Led
        digitalWrite(Led, LOW);                                  // Eteint la led
     break;
    case 76:       // Code ASCII reçu = 76 (lette L) -> Mesure de la LDR
        Serial.print("Valeur retournée par la LDR: ");     // Envoi la valeur
        Serial.println(analogRead(LDR));
     break;
     case 82:     // Code ASCII reçu = 82 (lette R) -> Mesure R ajustable
        Serial.print("Valeur retournée par la résistance ajustable: ");
        Serial.print(analogRead(Rajustable));
        Serial.print(" -> soit: ");
        Serial.print((analogRead(Rajustable)*3.3)/1023);
        Serial.println(" Vcc");
```

```
      break;

 //--------------------

      case 84:  // Code ASCII reçu = 84 (lette T) -> Mesure température
         Serial.print("Valeur dela temperature: ");    // Renvoie la valeur
         Serial.print(celcius);
         Serial.println(" C");
       break;

 //--------------------

     default:                                          // Ordre non reconnu
         if (k != 0) Serial.println("Ordre non reconnu !");    // On le signale
 break;
   }
 }
```

Voici la liste des commandes qui seront reconnues par la platine « Flip & Click» (lesquelles doivent être suivies de la touche « return » pour pouvoir être envoyées par la fenêtre du « moniteur série » de votre ordinateur):

A → Allume la led de la platine « Flip & Click».

E → Eteint la led de la platine « Flip & Click».

R → Retourne la valeur de la conversion « analogique/ numérique » et de la tension présente sur le curseur de la résistance ajustable

T → Retourne la valeur de la température mesurée par le LM35

B → Retourne l'état du bouton-poussoir (sollicité ou non)

L → Retourne la valeur de la conversion « analogique/numérique » de la tension présente au niveau de la LDR

Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage de la page précédente. Saisissez alors le programme, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Activez la fenêtre du moniteur série dans l'environnement de développement de la platine « Flip & Click » et essayez les différents ordres pour voir comment réagit la platine (pensez à solliciter la touche « Return » après chaque ordre).

Description et explications

Ce programme est relativement simple à comprendre. Il représente une synthèse de tout ce que vous avez appris précédemment.

Celui-ci repose sur la récupération des données qui arrivent sur le port série de la platine « Flip & Click ».

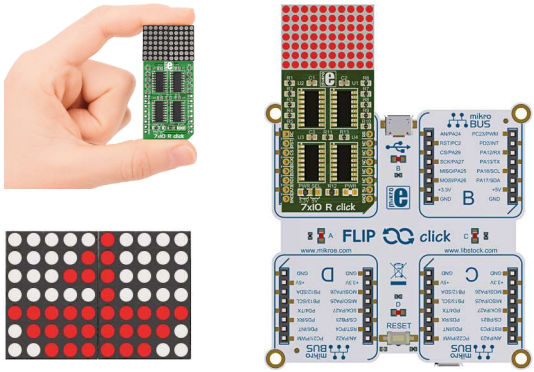
Puis par l'intermédiaire de l'instruction **switch ( )**, le programme teste un à un les codes ASCII correspondants aux ordres devant être reconnus. L'instruction **default:** prévoit également de renvoyer un message à votre ordinateur si le caractère ASCII reçu ne correspond à aucun ordre connu.

Le programme va ensuite, au cas par cas, envoyer un message à votre ordinateur avec les informations demandées.

A noter, que le calcul permettant de récupérer la valeur de la tension au niveau du curseur de la résistance ajustable consiste à multiplier la valeur de la conversion « analogique/numérique » par 3,3 et à diviser le tout par 1023 (qui correspond à la résolution du convertisseur « analogique/numérique » de la platine « Flip & Click »). Ceci a déjà fait l'objet d'une explication en page 22.

Application N° 018 Pilotage d'une matrice à leds "7 x 10 R Click"

Cette application va vous permettre d'apprendre à piloter la matrice à led "7x10 R Click" livrée avec votre starter-kit. Celle-ci se compose de 7 lignes de 10 leds. Elle vous permettra de pouvoir disposer d'un excellent petit dispositif capable d'afficher toutes sortes de choses (lettres, chiffres, icônes...). Dans le cadre de notre application, nous allons utiliser la matrice pour y afficher le dessin d'un petit bateau à voile. Ce dernier sera composé de plusieurs points (leds allumées).



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur, puis insérez le module «7x10 R click » sur le support **A** (attention au sens) et deconnecter tous les autres composants potentiellement raccordés sur la platine.

Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-contre et téléversez-le dans la platine.

Avant de détailler ce programme, il conviendra dans un premier temps d'étudier le schéma théorique du module «7x10 R click » afin d'en comprendre le fonctionnement.

```
#include <SPI.h>                    // Intégration de la librairie SPI
#define CS0 77                      // Attribution du signal CS0 au port 77
#define RCLK 54                     // Attribution du signal RCLK au port 54
#define MR 33                       // Attribution du signal MR au port 33
#define RST 6                       // Attribution du signal RST au port 6

// Définition du dessin du bateau
const byte Matrice[2][7] ={{0b00000001,0b00000001,0b00000001,
0b00000001,0b00011111,0b00001111,0b00000111},

{0b00000000,0b00010000,0b00011000,0b00000000,0b00011111,0b0
0011110,0b00011100}};

void setup() {
  pinMode(CS0, OUTPUT);             // La broche CS0 est une sortie
  pinMode(RCLK, OUTPUT);            // La broche RCLK est une sortie
  pinMode(MR, OUTPUT);              // La broche MR est une sortie
  pinMode(RST, OUTPUT);             // La broche RST est une sortie
  digitalWrite(MR, HIGH);           // Force la broche MR au niveau
logique haut
  digitalWrite(RCLK, LOW);          // Niveau logique bas horloge 4017
  digitalWrite(RST, LOW);           // Niveau logique bas RESET 4017
  SPI.begin();                      // Initialiation du port SPI
}

void loop() {                       // Fonction permettant de rafraichir la matrice
   digitalWrite(RST, HIGH);         // Reset du 4017 du module 7x10 R click
   digitalWrite(RST, LOW);
   for (byte i = 0 ; i < 7; i++){   // Boucle générant 7 coups d'horloge
     digitalWrite(CS0, LOW);        // Sélection broche CSO
     SPI.transfer(Matrice[0][i]);   // Envoi données première matrice
     SPI.transfer(Matrice[1][i]);   // Envoi données deuxième matrice
     digitalWrite(CS0, HIGH);       // Désélection de la broche CS0
     delay(2);
     digitalWrite(MR, LOW);         // Vide registre à décalage 74HC595
     digitalWrite(MR, HIGH);
     // delay(50)    // A remettre pour comprendre le fonctionnement
     digitalWrite(RCLK, HIGH);      // Coup d'horloge sur le 4017
digitalWrite(RCLK, LOW);}
     digitalWrite(CS0, LOW);        // Sélection broche CSO
     SPI.transfer(Matrice[0][0]);   // Envoi données première matrice
     SPI.transfer(Matrice[1][0]);   // Envoi données deuxième matrice
     digitalWrite(CS0, HIGH);       // Désélection de la broche CS0
}
```

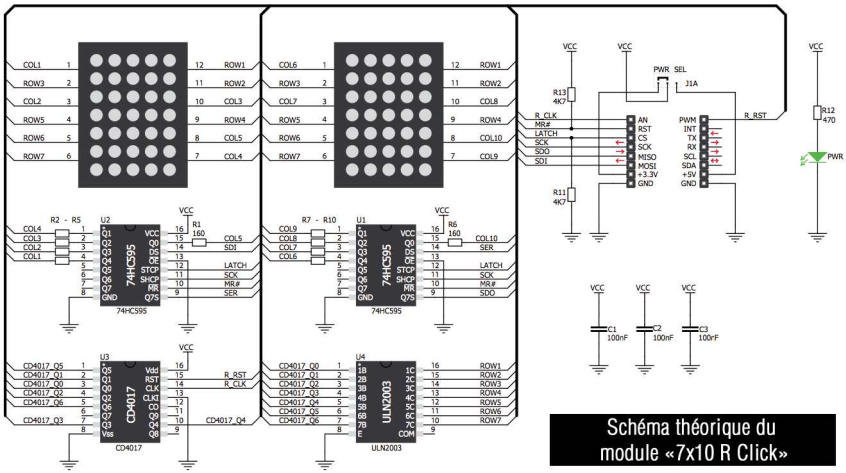


Schéma théorique du module «7x10 R Click»

Description et explications

Le module «7x10 R click » repose sur l'utilisation de 2 mini-matrices à leds (5 x 7 leds) pilotées (via un driver de puissance ULN2003) par 2 circuits intégrés 74HC595 (eux-même associés à un circuit intégré 4017).

Le circuit intégré 74HC595 est un registre à décalage destiné à être piloté via une liaison série de type SPI. Pour l'utiliser, il vous suffira d'activer celui-ci via une broche spéciale, puis de lui envoyer une donnée sur 8 bits (via ce port série), puis de le désactiver. A ce moment, chacun des bits de la donnée reçue par le 74HC595 seront « dispatchés » sur ses sorties parallèles.

Son utilisation première est de pouvoir par exemple ajouter 8 sorties à votre microcontrôleur au moyen d'une liaison série. Afin de pouvoir augmenter le nombre de sorties, il est possible de cascader 2 circuits intégrés 74HC595 (c'est ce qui est d'ailleurs fait sur le module «7x10 R click »).

Dans cette configuration, on devra envoyer 2 octets (l'un après l'autre) via le port série SPI vers les 74HC595, lesquels à leur désactivation, feront apparaître (à eux deux) les 16 bits (des 2 octets envoyés) sur chacunes de leurs sorties.

C'est cette même configuration qui est utilisée pour piloter la matrice à led de votre module (laquelle est constituée de 7 lignes comportant 10 leds chacunes). En regardant de plus près, cette matrice est en fait constituée de 2 mini-matrices placées l'une contre l'autre (elles-mêmes composées de 7 lignes de 5 leds chacunes).

L'étude du schéma théorique du module «7x10 R click » montre que les 74HC595 pilotent chacun 5 leds de chacune des mini-matrices. Dès lors, il en résulte qu'on ne pourra allumer que les leds de la première ligne de la matrice principale !

Description et explications (suite)

C'est à ce moment qu'intervient le circuit intégré 4017. Ce compteur va nous permettre en fait de sélectionner une à une les 7 lignes de la matrice pour que les 74HC595 allument également tour à tour leurs leds.

La méthode générale va donc consister à sélectionner la première ligne de 10 leds (via le compteur 4017), à y allumer les leds nécessaires (via les 74HC595 en envoyant via le port SPI les 2 x 5 leds à allumer), puis via le 4017, à passer à la deuxième ligne, à allumer à nouveau les leds de cette 2ème ligne via les 74HC595 et ainsi de suite jusqu'à la 7ème ligne.

Le problème majeur est que lorsque le 4017 va passer d'une ligne à l'autre, seules les leds de la ligne sélectionnée seront allumées ! … Mais pas de panique, grâce au principe de la persistance rétinienne déjà évoqué à plusieurs reprises dans cet ouvrage, en balayant très rapidement les lignes de la matrices, votre œil verra les leds de toutes les lignes allumées en même temps.

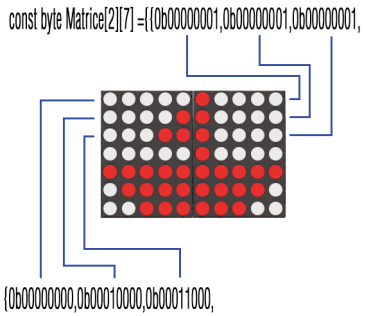
Voyons maintenant en pratique comment cela se traduit pour la programmation de la platine « Flip & Click »

L'utilisation du bus de communication série SPI est relativement simple à mettre en œuvre grâce à une librairie spécialisée qui est intégrée en début de programme.

```
#include <SPI.h>
```

Une liaison SPI nécessite plusieurs broches de communications, lesquelles sont déclarées et associées à certaines broches de votre platine en début de programme. Les broches dédiées au signal d'horloge et au Reset du compteur 4017 sont également déclarées à cet endroit.

La suite du programme fait appel à la déclaration d'une variable de type tableau (notion déjà abordée au cours d'une réalisation précédente). Ce tableau va nous permettre de mémoriser le dessin du petit bateau à voile que nous désirons faire afficher par la matrice.



La première ligne de données concerne l'état que devront prendre les 7 lignes (de 5 leds) de la mini-matrice à led de droite

La deuxième ligne de données concerne l'état que devront prendre les 7 lignes (de 5 leds) de la mini-matrice à led de gauche. Ces données étant sur 8 bits, il en résulte que les 3 premiers bits de poids forts seront toujours à 0.

La suite du programme concerne les initialisations et attributions de fonctionnement (entrée ou sortie) des différents ports utilisés pour l'application.

Vient ensuite la boucle principale dans laquelle on va venir rafraîchir en permanence chacune des lignes de la matrice principale.

Description et explications (suite)

L'opération consiste en premier lieu à effectuer un RESET du compteur 4017 (afin qu'il sélectionne la première ligne de la matrice).

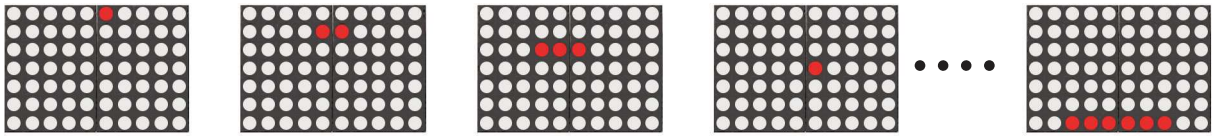
Le programme se poursuit avec une boucle **for (byte i = 0 ; i < 7; i++)** au cours de laquelle on va tour à tour sélectionner les 74HC595, leur envoyer les données via la variable **Matrice[0][i]**, puis désélectionner les 74HC595 afin que les données envoyées activent alors les leds de la ligne en cours de sélection.

Le programme effectue une légère temporisation via l'instruction **delay(2);** afin d'éviter un phénomène de scintillement (toujours lié à la persistance rétinienne) et obtenir un affichage stable.

Le programme se poursuit en vidant les buffers des 74HC575 afin qu'ils soient prêts à recevoir une nouvelle salve de données.

A ce stade, un coup d'horloge est généré sur le 4017 afin de passer à la ligne suivante... et ainsi de suite jusqu'à la 6 ème ligne.

La même opération est effectuée à la sortie de la boucle pour la 7 ème ligne (laquelle a été gérée de la sorte pour éviter que le 4017 ne passe sur la 8 ème ligne (qui n'existe pas) suite à un nouveau coup d'horloge.



La boucle principale recommence alors indéfiniment son cycle avec de nouveau l'affichage sur la ligne 1, puis sur la ligne 2... et ainsi de suite.

Afin que vous compreniez bien le fonctionnement de l'ensemble, nous avons ajouté une ligne de temporisation en commentaire.

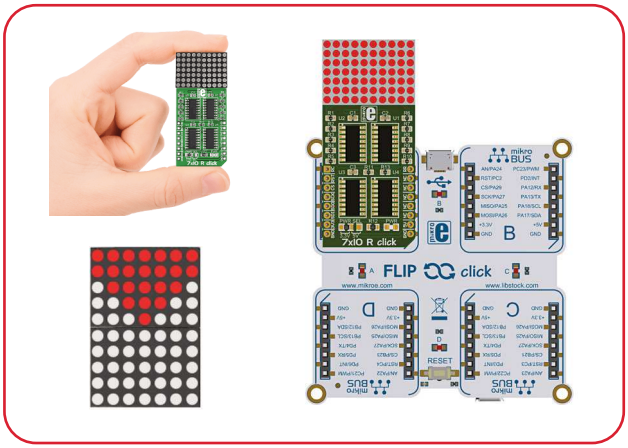
// delay(50);

Amusez-vous à la rendre active en enlevant les 2 caractères // en début de ligne et vous verrez alors l'affichage de la matrice s'effectuer de façon décomposée (l'effet de persistance rétinienne étant alors inopérant).

Vous avez maintenant toutes les explications nécessaires pour modifier le petit dessin que nous avons choisi dans notre exemple afin de pouvoir afficher d'autres motifs, ou des lettres ou des chiffres.

Application N° 019 Sablier animé avec module "7 x 10 R Click"

Cette application va vous permettre de realiser un petit sablier animé à l'aide de la matrice à led "7x10 R Click". Celui-ci va « de facon imagée » déverser son sable de haut en bas de la matrice (comme le ferait un vrai sablier).



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7x10 R click » sur le support **A** (attention au sens) et déconnecter tous les autres composants et fils de liaisons potentiellement existants sur le platine. Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-contre et téléversez-le dans la platine.

Le fonctionnement de cette application est relativement simple à comprendre et repose en très grande partie sur les explications détaillées que nous vous avons déja fournies pour l'application précédente.

```
byte Matrice[2][7] ={{0b00000000,0b00000000,0b00000000,
0b00000000,0b00000000,0b00000000,0b00000000},
{0b00000011,0b00000111,0b00001111,0b00011111,0b00001111,0b0
00000111,0b00000011}};

void setup()  {
  pinMode(CS0, OUTPUT);          // La broche CS0 est une sortie
  pinMode(RCLK, OUTPUT);         // La broche RCLK est une sortie
  pinMode(MR, OUTPUT);           // La broche MR est une sortie
  pinMode(RST, OUTPUT);          // La broche RST est une sortie
  digitalWrite(MR, HIGH);        // Force broche MR au niveau haut
  digitalWrite(RCLK, LOW);       // Niveau logique bas horloge 4017
  digitalWrite(RST, LOW);        // Niveau logique bas RESET 4017
  SPI.begin();                   // Initialiation du port SPI
}

void loop()  {
    for (byte sablier = 0 ; sablier < 5 ; sablier++){    // Boucle du sablier
      for (int Tempo = 0 ; Tempo < 100 ; Tempo++) {Rmatrice();}
      for (byte i = 0 ; i < 7; i++) {bitWrite(Matrice[0][i],4- sablier,
      bitRead(Matrice[1][i],sablier));}
      for (byte i = 0 ; i < 7; i++) {bitWrite(Matrice[1][i],sablier,0);}
    }
    while(1)    // Fin durée d'écoulement -> entre dans boucle sans fin
    {Rmatrice();}    // Dans laquelle on rafraîchit constamment la matrice
}

void Rmatrice() {      // Fonction permettant de rafraichir la matrice
   digitalWrite(RST, HIGH);   // Reset du 4017 du module 7x10 R click
   digitalWrite(RST, LOW);
   for (byte i = 0 ; i < 7; i++){        // Boucle générant 7 coups d'horloge
     digitalWrite(CS0, LOW);             // Sélection broche CSO
     SPI.transfer(Matrice[0][i]);        // Envoi données première matrice
     SPI.transfer(Matrice[1][i]);        // Envoi données deuxième matrice
     digitalWrite(CS0, HIGH);            // Désélection de la broche CS0
     delay(2);
     digitalWrite(MR, LOW);        // Vide le registre à décalage 74HC595
     digitalWrite(MR, HIGH);
     digitalWrite(RCLK, HIGH);         // Coup d'horloge sur le 4017
     digitalWrite(RCLK, LOW);}
     digitalWrite(CS0, LOW);             // Sélection broche CSO
     SPI.transfer(Matrice[0][0]);        // Envoi données première matrice
     SPI.transfer(Matrice[1][0]);        // Envoi données deuxième matrice
     digitalWrite(CS0, HIGH);            // Désélection de la broche CS0
}
```

Description et explications

Ce programme consiste à réaliser une « petite animation » sur la matrice à led (sous la forme d'un sablier qui s'écoule). Ceci revient en fait à afficher une suite de motifs qui se succèdent sur la matrice.

Ce programme sera très similaire à celui de l'application précédente et nous ne reviendrons donc pas sur toute la première partie liée aux initialisations... mis à part sur les données mémorisées dans la variable de type matrice qui cette fois-ci ne représenteront plus un petit bateau à voile comme sur l'application précédente.. mais un « tas de sable » en forme de triangle (censé représenter le sablier).

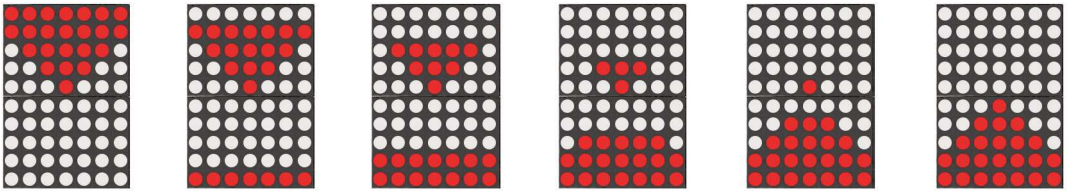
La représentation de ce sablier se fera non pas à l'horizontale, mais à la verticale sur la matrice.

La modification importante que nous avons opéré pour ce programme est que nous avons désormais réalisé une fonction qui s'occupe exclusivement du rafraichissement de la matrice `void Rmatrice()`

Des lors, nous pourrons appeler cette fonction à tout moment pour afficher l'état du sablier sur la matrice.

Le fonctionnement du sablier repose essentiellement sur l'étude de la boucle principale du programme. Son écoulement va consister à effacer tour à tour les 5 lignes du haut de la matrice pour les faire apparaître au bas de la matrice et ainsi simuler l'écoulement du sable.

Il en résulte que le programme utilise une boucle de type **`for (byte sablier = 0 ; sablier < 5 ; sablier++)`** pour gérer l'écoulement des 5 lignes.



La fonction première d'un sablier étant de « mesurer » un temps, le programme à l'intérieur de la boucle commence par une temporisation qui servira de base de temps entre chaque écoulement.

Le reflexe de base serait d'appeler la fonction de rafraichissement de la matrice (pour afficher l'état initial de la matrice).. puis d'ajouter une instruction **`delay (xx)`** pour générer une temporisation. Toutefois, comme indiqué lors que l'application précédente, la gestion de la matrice nécessite que cette dernière soit rafraîchie en permanence afin qu'elle reste allumée (via l'effet de persistance rétinienne). Dans le cas présent, l'utilisation de l'instruction **`delay (xx)`** aurait eu pour effet de voir la matrice s'éteindre durant cette temporisation !

Description et explications (Suite)

Il nous a donc fallu réaliser une temporisation au moyen d'une seconde boucle **for ()** dans laquelle on incrémente une variable jusqu'à 99 et à chaque incrémentation, le programme appelle la fonction de rafraîchissement de la matrice afin quelle reste bien allumée durant toute la temporisation.

for (int Tempo = 0 ; Tempo < 100 ; Tempo++) {Rmatrice();}

La ligne qui suit dans le programme permet de transférer les données liées aux lignes des leds du haut de la matrice vers le bas de la matrice en « jouant » sur la valeur des index de la variable tableau **Matrice[ ][ ]** et via l'instruction **bitWrite( )** , laquelle permet de modifier un bit particulier d'une variable.

for (byte i = 0 ; i < 7; i++) {bitWrite(Matrice[0][i],4-sablier,bitRead(Matrice[1][i],sablier));

L'instruction **bitWrite( )** nécessite 3 paramètres pour pouvoir être utilisée.

Le premier paramètre concerne la variable pour laquelle on désire modifier l'état d'un bit (ici **Matrice[ ][ ]**), Le second paramètre concerne l'emplacement du bit : ici 0 à 6 pour sélectionner les 7 lignes. Le dernier paramètre concerne l'état que doit prendre le bit 0 ou 1. Dans notre cas, on recopie l'état de chaque led de la matrice du haut.

L'utilisation d'une mise à jour avec l'instruction **bitWrite( )** s'impose car nous avons utilisé la matrice dans le sens vertical et nous avons été obligés de remettre à jour l'état des leds de chaque colonne de la matrice et non pas l'état d'une ligne complète de la matrice.

L'instruction qui suit dans le programme **for (byte i = 0 ; i < 7; i++) {bitWrite(Matrice[1][i],sablier,0);}**
reprend le même pricipe pour éteindre successivement les leds de la matrice du haut en forçant le dernier paramètre de l'instruction **bitWrite( )** à 0.

Le déroulement de la boucle **for (byte sablier = 0 ; sablier < 5 ; sablier++)** va donc réaliser l'écoulement du sable contenu dans la matrice du haut vers la matrice du bas.

Au terme de cette boucle (qui indique la fin d'écoulement du sablier), le programme entame une boucle sans fin

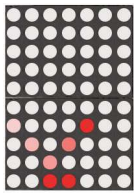
while(1) {Rmatrice();}

dans laquelle la platine « Flip & Click » ne fera que rafraîchir la matrice en permanence (afin qu'elle ne s'eteigne pas).

Si vous désirez « relancer » le sablier, il vous suffit simplement de solliciter le bouton-poussoir Reset de la platine « Flip & Click ».

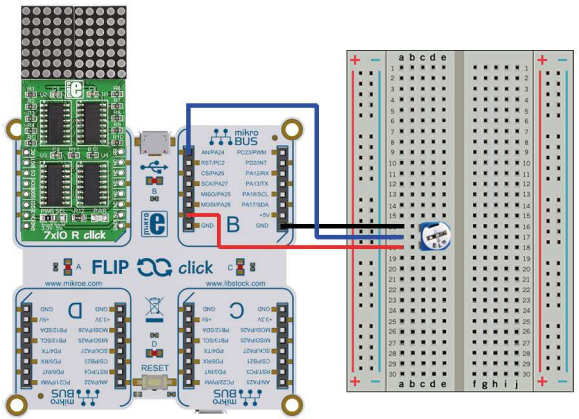
Si vous voulez aller plus loin :
Vous pouvez modifier la valeur de la temporisation du sablier pour parvenir à une durée de 3 mn et ajouter un signal sonore à l'aide d'un buzzer en fin de cycle afin de pouvoir réaliser une minuterie utilisable pour la cuisson de vos œufs à la coque !

Application N° 020 Jeu vidéo Tennis (1 joueur)



Cette application va vous permettre de réaliser un petit jeu vidéo (type tennis 1 joueur) à l'aide du module «7x10 R click ». Pour l'occasion, la matrice à led du module sera exploitée dans le sens vertical. Vous disposez d'une petite raquette au bas de la matrice (matérialisée par 2 leds côte à côte) que vous pourrez déplacer de droite à gauche et de gauche à droite à l'aide d'une résistance ajustable.

Dans le même temps une petite balle (matérialisée par une led de la matrice) se déplace vers le bas de l'aire de jeu. Votre but sera de renvoyer cette balle aussi longtemps que possible afin qu'elle ne sorte pas de la matrice. A chaque fois que la balle touchera votre raquette, celle-ci remontera vers le haut de la matrice en rebondissant sur des murs virtuels et en retombant systématiquement vers le bas de la matrice. Au moment où vous raterez la balle et qu'elle descendra au delà de votre raquette, une petite animation visuelle vous signalera la fin de la partie. En sollicitant le bouton-poussoir de Reset de la platine « Flip & Click », vous relancerez une nouvelle la partie.



Réalisation du câblage - Description & Explications

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7x10 R click » sur le support **A** (attention au sens). Puis procédez au câblage de la résistance ajustable. Son curseur sera relié sur l'entrée de conversion « analogique/numérique » AN/PA24 (port 55) de la platine « Flip & Click » tandis que les 2 contacts extrêmes de la résistance ajustable sont reliés à la masse pour l'un et au 3,3 V pour l'autre. ATTENTION, utilisez bien la broche 3,3 V (PAS la broche 5 V... sous peine de destruction de la platine « Flip & Click »). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme de le page suivante et téléversez-le dans la platine.

Cette application met en œuvre beaucoup de compétences déjà acquises au cours des montages précédents (gestion de variables de type matrice, conversion « analogique/numérique », pilotage d'une matrice à leds « 7x 10 R Click », génération de nombres aléatoires, etc...).

Le programme associé est toutefois assez conséquent. Aussi, afin de vous éviter une lecture trop fastidieuse, nous n'allons pas détailler les instructions de ce dernier les unes après les autres selon le déroulement du listing, mais plutôt vous exposer les différents concepts et astuces utilisés pour parvenir à la réalisation de cette application.

```
#include <SPI.h>                                                  // Intégration de la librairie SPI
#define CS0 77                                                   // Attribution du signal CS0 au port 77
#define RCLK 54                                                  // Attribution du signal RCLK au port 54
#define MR 33                                                    // Attribution du signal MR au port 33
#define RST 6                                                    // Attribution du signal RST au port 6
#define Rajustable 55                                            // N° du port sur lequel est relié la résistance ajustable

byte Matrice[2][7];                                              // Définition de la matrice (aire de jeu)
byte PosRaquette;                                                // Définition de la position initiale de la raquette
byte Decompte,Vitesse=20;                                        // Variables gérant la vitesse de déplacement de la balle
byte MBall=0, xBall, yBall=3, DxBall=1, DyBall=1;                // Variables pour la gestion de la balle


void setup() {
  pinMode(CS0, OUTPUT);pinMode(RCLK, OUTPUT);                    // Les broches CS0 et RCLK sont des sorties
  pinMode(MR, OUTPUT);pinMode(RST, OUTPUT);                      // Les broches MR et RST sont des sorties
  digitalWrite(MR, HIGH);                                        // Force la broche MR au niveau logique haut
  digitalWrite(RCLK, LOW);                                       // Niveau logique bas sur l'horloge du 4017 du module 7x10 R click
  digitalWrite(RST, LOW);                                        // Niveau logique bas sur l'entrée de RESET du 4017 du module 7x10 R click
  SPI.begin();                                                   // Initialiation du port SPI
  for (byte i = 0 ; i < 7; i++)                                  // Efface toutes les leds de la matrice
  {Matrice[0][i]=0;Matrice[1][i]=0;}
  randomSeed(analogRead(56));                                    // Réinitialise la séquence aléatoire
  xBall=random(1,6);                                             // Positionnement aléatoire la ba balle
  bitSet(Matrice[MBall][xBall],yBall);                           // Affiche position initiale de la balle
  Decompte=Vitesse;                                              // Initialise la vitesse de déplacement de la balle
}

void loop()
 {

   //---------------------------------- Gestion du déplacement de la raquette ----------------------------

   PosRaquette= analogRead(Rajustable)/100;
   if (PosRaquette < 1) PosRaquette=1; if (PosRaquette > 6) PosRaquette=6;        // Limite le déplacement de la raquette
   bitSet(Matrice[0][PosRaquette],4);bitSet(Matrice[0][PosRaquette-1],4);         // Configure les bits de la raquette à "1"
   bitClear(Matrice[0][PosRaquette+1],4);bitClear(Matrice[0][PosRaquette-2],4);   // Configure les bits autour de la raquette à "0"

   //---------------------------------- Gestion du déplacement de la balle ----------------------------

   Decompte--;                                                   // Décompte temporisation pour déterminer la vitesse de déplacement de la balle
   if (Decompte == 0)                                            // Regarde si on déplace la balle
    {Decompte=Vitesse; bitClear(Matrice[MBall][xBall],yBall);    // Réinitialise le compteur et efface l'ancienne position de la balle
    if (DxBall==1)                                               // Teste collision balle avec les murs de droite et gauche
      {xBall--;if(xBall==255){xBall=1;DxBall=0;}}                // Collision mur de droite -> Inverse sens de déplacement
    else
      {xBall++;if(xBall==7)  {xBall=5;DxBall=1;}}                // Collision mur de droite -> Inverse sens de déplacement
    if (DyBall==1)                                               // Teste collision balle avec le mur du haut et déplacement entre les matrices
      {yBall--; if (yBall==255 & MBall==0) {MBall=1;yBall=4;}    // Teste passage matrice basse vers matrice haute
      if (yBall==255 & MBall==1) {yBall=1;DyBall=0;}}            // Teste collision avec mur du haut
    else
     {yBall++; if (yBall==5 & MBall==1) {MBall=0;yBall=0;}       // Teste passage matrice du haut vers matrice du bas
      if (yBall==4 & MBall==0 & bitRead(Matrice[0][xBall],4)==1) // Teste rebond avec la raquette
        {Vitesse--;if (Vitesse<5) Vitesse=5;DyBall=1;yBall=2;}   // Augmente la vitesse de la balle
      else
```

```
      if (yBall==4 & MBall==0 & bitRead(Matrice[0][xBall],4)==0)                    // Teste si la balle est perdue
      {for (byte i = 0 ; i < 5; i++){                                               // Clignotement de la balle
      bitSet(Matrice[MBall][xBall],yBall);for (byte i = 0 ; i < 9; i++){Rmatrice();}
      bitClear(Matrice[MBall][xBall],yBall);for (byte i = 0 ; i < 9; i++){Rmatrice();}}
      for (byte i = 0 ; i < 100; i++)                                               // Animation fin de partie
      {bitSet(Matrice[random(0,2)][random(0,7)],random(0,5));bitSet(Matrice[random(0,2)][random(0,7)],
      random(0,5));bitSet(Matrice[random(0,2)][random(0,7)],random(0,5));Rmatrice();}
      for (byte i = 0 ; i < 150; i++) {bitClear(Matrice[random(0,2)][random(0,7)],random(0,5));bitClear(Matrice[random(0,2)][random(0,7)],random(0,5));
      bitClear(Matrice[random(0,2)][random(0,7)],random(0,5));bitClear(Matrice[random(0,2)][random(0,7)],random(0,5));Rmatrice();}
      while(1)  {Rmatrice();}}}}
  bitSet(Matrice[MBall][xBall],yBall);                                              // Mémorise nouvelle position de la balle
  Rmatrice();}                                                                      // Rafraîchissement de la matrice

void Rmatrice() {// Fonction permettant de rafraichir la matrice
    digitalWrite(RST, HIGH);digitalWrite(RST, LOW);                                 // Reset du 4017 du module 7x10 R click
    for (byte i = 0 ; i < 7; i++){                                                  // Boucle générant 7 coups d'horloge sur le 4017
    digitalWrite(CS0, LOW);                                                         // Sélection broche CSO
    SPI.transfer(Matrice[0][i]); SPI.transfer(Matrice[1][i]);                       // Envoie les données aux matrices
    digitalWrite(CS0, HIGH);delay(3);                                               // Désélection de la broche CS0
    digitalWrite(MR, LOW);digitalWrite(MR, HIGH);                                   // Vide le registre à décalage des 74HC595
    digitalWrite(RCLK, HIGH);digitalWrite(RCLK, LOW);}                              // Coup d'horloge sur le 4017 du module 7x10 R click
    digitalWrite(CS0, LOW);SPI.transfer(Matrice[0][0]); SPI.transfer(Matrice[1][0]);digitalWrite(CS0, HIGH);}
```

Description et explications (Suite)

L'ensemble du programme permettant la création de ce petit jeu vidéo est présenté ci-contre. Vous pourrez toutefois constater que ce dernier est plus "condensé" qu'à l'habitude. En effet afin de gagner de la place et pour réussir à le faire "rentrer" dans ces 2 pages, nous avons écrit plusieurs instructions les unes à la suite des autres sur une même ligne sans sauter systématiquement une ligne après chaque instruction. D'un point de vue fonctionnel, ceci ne posera aucun problème à l'environnement de développement de la platine "Flip & Click". Par contre la "lecture" du code par un utilisateur est dès lors un peu moins "digeste"... mais nous n'avions pas le choix et avons dû trouver un compromis. Ce programme débute par les habituelles déclarations et initialisations (y comprises celles liées à la gestion du module « 7x10 R click »). Pour les besoins de la gestion des éléments du jeu, nous avons également été amené à déclarer plusieurs variables :

PosRaquette qui servira à mémoriser la position horizontale de la raquette sur la matrice à led.

Decompte et **Vitesse** qui serviront à définir la vitesse de déplacement initiale de déplacement de la balle (avec un phénomène d'accélération au cours de la partie),

Xball et **Yball** qui gèrent la position (X/Y) de la balle sur la matrice.

DxBall et **DyBall** qui gèrent la direction de déplacement de la balle sur la matrice (si DxBall = 1 alors la balle se déplace vers la droite. Si DXBall = 0 alors la balle va dans l'autre sens. De même, si DyBall = 1 alors la balle va vers le haut de la matrice et si DyBall = 0, la balle va vers le bas de la matrice),

Matrice[2][7] qui gère le « plan » mémoire de la matrice.

Mball qui est une variable permettant de savoir sur quelle matrice se trouve la balle.

Description et explications (suite)

Pour rappel, la matrice utilisée sur le module « 7x 10 R click » est en fait composée de 2 mini-matrices de 5 x 7 leds placées l'une à côté de l'autre (raison pour laquelle la variable destinée à définir le plan mémoire est sur 2 dimensions : **Matrice[2][7]**).

La difficulté majeure liée à cette application est qu'il faut afficher et gérer la position de la balle en fonction de sa position sur la matrice haute ou sur la matrice basse. La variable **Mball** est destinée à mémoriser cette information (Mball = 0 pour la matrice du bas et Mball = 1 pour la matrice du bas).

En début de programme, le plan mémoire de la matrice est complètement effacé, puis on sélectionne d'office un déplacement de la balle vers le haut à droite (**DxBall = 1** et **DyBall = 1**) avec une position initiale verticale fixe de la balle et une position horizontale aléatoire afin qu'elle ne parte jamais du même endroit au fil des différentes parties.

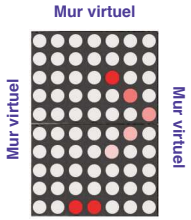
Le déplacement de la raquette est obtenu assez simplement en récupérant la valeur de la conversion analogique/numérique de la tension présente sur le curseur de la résistance ajustable (qui nous donne un chiffre compris entre 0 et 1023) et en divisant cette valeur par 100, on obtiendra un nombre compris au final entre 0 et 10.

Nous utiliserons ce nombre pour déterminer la position de la raquette en réalisant des comparaisons avec les valeurs droite et gauche extrêmes pour que son déplacement soit limité aux dimensions de la matrice. Une fois la position de la raquette connue, on configure à 1 les bits liés à l'emplacement de la raquette dans la matrice (pour allumer les leds représentant cette raquette) et on met les bits présents aux extrémités droite et gauche de la raquette à 0 afin de gérer l'effacement automatique de celle-ci lorsqu'on la déplace horizontalement.

L'utilisation de cette méthode est donc relativement simple à mettre en oeuvre avec par contre une limitation si on tourne le curseur de la resistance trop rapidement. Dans ce cas, la progression du résultat suite à la lecture de la valeur de la tension sur le curseur de la résistance ajustable sera également trop rapide et risque de faire déplacer la raquette sur plus d'une position horizontale à la fois. Dès lors, notre méthode qui consiste à effacer une led de part et d'autre de la raquette ne sera plus suffisante et des « bouts » de raquettes risquent de rester à l'affichage dans ce cas de figure. Toutefois, il s'agit d'un cas extrême et en condition normale de jeu, cette limitation n'est pas un problème. Voir également la note au bas de la page 73.

Le programme principal de notre application doit donc consister à :

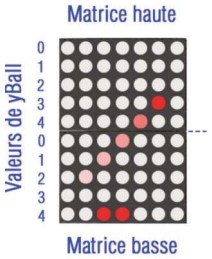
- Gérer le déplacement de la raquette sur la matrice
- Gérer le déplacement de la balle entre les 2 matrices
- Vérifier si la balle atteint les murs virtuels droit, gauche et haut (auquel cas, la balle devra rebondir)
- Vérifier si la balle frappe la raquette (auquel cas la balle devra rebondir)
- Vérifier si la balle passe au delà de la limite de la position horizontale de la raquette (auquel cas la balle devra se mettre à clignoter et une petite animation devra annoncer la fin de partie)
- Gérer une temporisation en permamence pour déterminer la vitesse de déplacement de la balle (avec un phénomène d'accélération progressive au cours de la partie)



Description et explications (Suite)

Il nous faudra également prévoir un rafraîchissement permanent de la matrice durant toutes ces opérations afin d'obtenir une jouabilité parfaite et maintenir l'effet de persistance rétinienne sur la matrice. En détaillant les choses, le programme va donc devoir surveiller (à chaque déplacement de la balle) les valeurs de ses coordonnées X et Y (avec les variables **xBall** et **yBall**).

La gestion de la coordonnée **yBall** de la balle s'effectue ainsi. yBall = 4 lorsque la balle est sur la matrice du bas juste au niveau de la raquette. YBall = 3, si la balle est juste une led au dessus de la raquette... ainsi que suite... jusqu'à yBall = 0 si la balle est sur la led la plus haute de la matrice du bas. Un des tests du programme consiste à vérifier si la balle est sur la matrice du bas et si (après la mise à jour de la position des coordonnées de la balle), yBall est inférieur à 0 (dans notre cas, nous testons si yBall = 255 car une variable sur 8 bits dont la valeur est égale à 0 passera à la valeur 255 si on la décompte). Si tel est le cas, le programme sera en mesure de déterminer que la balle est passée sur la matrice du haut.



Lorsque la balle passe sur la matrice du haut, la mémorisation en yBall reprend la valeur 4, puis 3 si elle est sur la 2ème ligne en partant du bas de la matrice haute.. et ainsi de suite jusqu'à la valeur 0 si la balle est sur la dernière ligne en haut de la matrice du haut.

La gestion de la collision avec le mur virtuel du haut de la matrice s'effectue donc en vérifiant si on est sur la matrice du haut et si yBall = 255 (si on est inférieur à la positon 0). Dans ce cas, le programme va modifier le sens de déplacement **DyBall** de la balle afin de la faire rebondir vers le bas de la matrice.

Un troisième test consiste à détecter le passage de la matrice du haut vers la matrice du bas en remettant à jour la valeur de yBall.

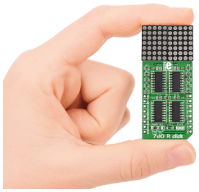
Dans le même ordre d'esprit, le programme va vérifier la position xBall de la balle afin de vérifier si on a atteint les limites des murs virtuels droit et gauche (si Xball < 0 (soit Xball = 255) ou si Xball = 7). Si tel est le cas, le programme inverse le sens de déplacement horizontal de la balle pour gérer le rebond de celle-ci.

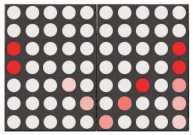
Le dernier test concerne la gestion de la collision avec la raquette. Pour ce faire, le programme va simplement vérifier à chaque déplacement si la balle est sur la matrice du bas et si sa prochaine position pour DyBall = 4 indique la présence d'une led allumée (celle de la raquette). Si tel est le cas, ceci signifie que la raquette est bien en dessous de la balle et qu'il faut inverser le sens de déplacement vertical de la balle (pour gérer son rebond) en mettant la variable DyBall à 1.

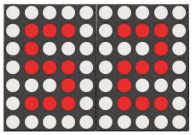
Si par contre le programme ne détecte pas de led allumée sur la dernière ligne lors du déplacement de la balle, celui-ci en déduit que la raquette n'est pas présente et qu'il faut faire clignoter la balle rapidement plusieurs fois de suite (afin de monter l'endroit ou elle a atterri). S'en suit alors une petite animation qui consiste à allumer successivement les différentes leds de la matrice de façon aléatoire avant que celles-ci ne s'effacent (également de façon aléatoire).

A ce stade, le programme entre dans une boucle sans fin et la partie est définitivement terminée. Il ne vous reste plus qu'à appuyer sur le bouton-poussoir Reset de la platine pour jouer à nouveau. A noter que durant toute la gestion des déplacements et des collisions, le programme utilise la variable **decompte** pour réaliser une temporisation entre chaque déplacement ainsi que la variable **Vitesse** (qui est décrémentée régulièrement) pour accélérer le déplacement de la balle au fur et à mesure que la partie se déroule (avec une valeur limite fixée à 5 afin que la partie de tennis ne devienne pas injouable).

Application N° 021 Jeu vidéo Tennis (2 joueurs)







Cette application va vous permettre de réaliser un petit jeu vidéo (type tennis 2 joueurs avec gestion et affichage du score) à l'aide du module «7x10 R click ». Pour l'occasion la matrice à led du module sera exploitée à l'horizontale.

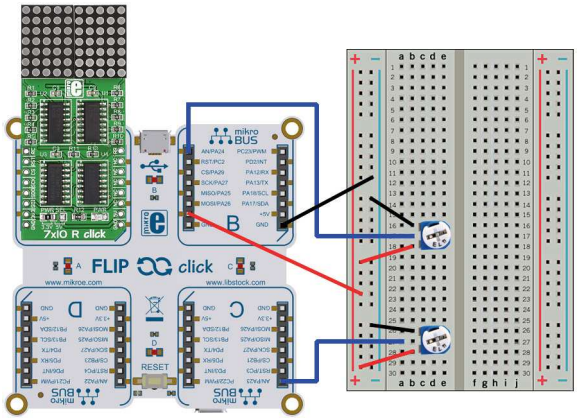
Chaque joueur dispose d'une petite raquette de part et d'autre de la matrice (matérialisée par 2 leds côte à côte) qu'il pourra déplacer de bas en haut et de haut en bas à l'aide d'une résistance ajustable.

Au démarrage du jeu, le score de la partie s'affichera (0 – 0). Puis une petite balle (matérialisée par une led de la matrice) s'élancera dans l'aire de jeu. Le challenge pour chaque joueur consistera à renvoyer la balle vers le joueur adverse à l'aide de sa raquette.

Dès qu'un joueur rate la balle, celle-ci se met à clignoter à l'endroit où a été commise la faute et un point sera attribué au joueur d'en face. La platine « Flip & Click » en profitera pour afficher temporairement le score avant de lancer à nouveau la balle dans l'aire de jeu pour que la partie se poursuive.

Le joueur qui atteint les 5 points en premier a gagné la partie (ce qui se matérialise par un clignotement continu du score final sur la matrice). Pour jouer à nouveau, il vous suffira de solliciter le bouton-poussoir Reset de la platine.

Le programme de cette application étant relativement grand et du fait de sa très grande similitude avec le programme précédent, nous avons choisi de ne pas l'intégrer à l'ouvrage. Vous pourrez bien sur le télécharger comme indiqué en page 17 de ce manuel.



Réalisation du câblage

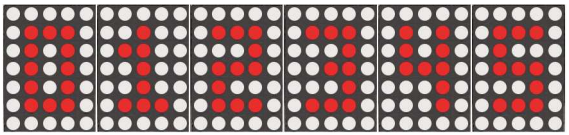
Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7x10 R click » sur le support **A** (attention au sens). Puis procédez au câblage des 2 résistances ajustables. Leurs curseurs seront reliés sur les entrées de conversion « analogique/numérique » AN/PA24 et AN/PA23 (ports 55 et 56) de la platine « Flip & Click » tandis que les 2 contacts extrêmes des résistances ajustables seront reliés à la masse pour l'un et au 3,3 V pour l'autre. ATTENTION, utilisez bien la broche 3,3 V (PAS la broche 5 V... sous peine de destruction de la platine « Flip & Click »).

Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis récupérez le programme de cette application et téléversez-le dans la platine.

Description et explications

Cette nouvelle application est une version améliorée et plus complète du jeu de tennis 1 joueur présenté ci-avant. Sa description sera donc assez limitée car la plupart de son fonctionnement a déja été vu auparavant. Celui-ci reprend en effet toutes les bases précédentes (liées au déplacement de la balle) avec des tests complémentaires pour gérer cette fois-ci la collision de la balle avec les 2 raquettes ainsi que la perte de la balle dans les 2 camps.

Des variables supplémentaires **ScoreJ1** et **ScoreJ2** sont également utilisées pour la gestion des scores de chaque joueur.
L'affichage des scores a aussi nécessité d'avoir recours à de nouvelles variables dont une variable de type tableau **Mscore[ ][ ]** dans laquelle nous avons mémorisé en début de programme la position de chaque led de la matrice nécessaire pour « dessiner » les chiffres 0 à 5.



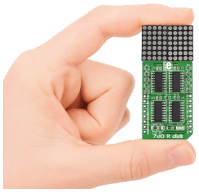
Chaque chiffre a ainsi été "modélisé" avec 3 leds de large et 5 leds de haut (ce qui permet de positionner le chiffre en plein milieu de chaque mini-matrice et d'afficher le score des 2 joueurs sur l'ensemble de la matrice.

L'affichage du score consiste à utiliser une fonction dans laquelle on transfère les données de la variable **Mscore[ ][ ]** vers la variable principale de la matrice (afin de substituer l'affichage de la balle et des raquettes par celui du score). Les index de la variable **Mscore[ ][ ]** étant modifiés en fonction des chiffres à afficher et du joueur. A chaque fois qu'un joueur rate la balle et qu'un point est attribué au joueur adverse, le programme teste si on a pas atteint la limite des 5 points afin de déclarer la fin du match (matérialisé par le clignotement continu du score final). Ce clignotement est réalisé par l'intermédiaire d'une seconde fonction au cours de laquelle la variable principale de la matrice est remise à jour avec les données de la variable **Mscore[ ][ ]** puis après un petit délai d'affichage, elle est effacée (pour que le score disparaisse également un moment) et ainsi de suite dans une boucle sans fin afin de pouvoir créer l'effet de clignotement.

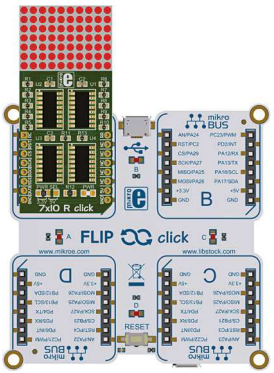
A noter que nous avons utilisé une petite subtilité à chaque engagement de la balle, qui consiste a choisir une position et un sens de déplacement aléatoire de la balle afin de ne privilégier aucun des 2 joueurs et d'éviter que la partie ne débute toujours de la même façon à chaque engagement de la balle. Enfin comme pour l'application précédente, la vitesse de déplacement de la balle augmentera au fur et à mesure que les échanges entre les 2 joueurs se prolongeront jusqu'à arriver à une vitesse maximale (restant dans la limite de la jouabilité). Lorsqu'une balle est perdue, la vitesse reprend à nouveau à sa valeur initiale avant d'augmenter une fois encore au fil des échanges.

Notes concernant les applications Tennis 1 et 2 joueurs: Lors du déplacement des raquettes à l'aide des résistances ajustables, vous pourrez quelques fois remarquer que la raquette pourra avoir tendance à "frétiller". Ceci est en partie normal et dû au fait que la tension présente sur le curseur de la résistance ajustable soit entre 2 positions de raquette (voir explication en page 70) et que de part le câblage utilisé (straps de grande longueur et plaque de développement sans soudure apportant des capacités "parasites"), il y ait une légère oscillation de la tension mesurée provoquant ce phénomène (passage d'une position à l'autre). Ce même phénomène peut également intervenir lors du déplacement d'une raquette pouvant entrainer dans certains cas un frétillement de l'autre raquette (toujours dû à des phénomènes d'oscillations parasites sur les tensions mesurées). Gardez à l'esprit que nous avons utilisé une gestion de la position des raquettes des plus simples entrainant ces petites limitations.

Application N° 022 Journal lumineux défilant



Cette application va vous permettre de réaliser un mini-journal lumineux défilant à l'aide du module "7x10 R Click". Une variable à configurer en début de programme va vous permettre de choisir le texte du message (pouvant comporter des lettres et des chiffres) qui défilera de la droite vers la gauche de la matrice. La vitesse de défilement du message pourra également être ajustée si vous le désirez.



Réalisation du câblage

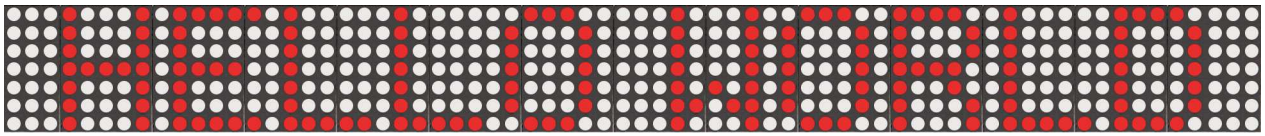
Déconnectez la platine « Flip & Click » de votre ordinateur, puis insérez le module «7x10 R click » sur le support **A** (attention au sens) et deconnectez tous les autres composants potentiellement raccordés à la platine. Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis téléversez le programme dans la platine.

Description et explications

Cette nouvelle application met en œuvre la plupart des choses déjà vues au cours de l'application de gestion du module '7x10 R click ». La principale nouveauté réside dans l'utilisation d'une variable de type tableau à une seule dimension **Message[ ]** dans laquelle vous allez pouvoir écrire votre message en début de programme. Ce message peut être composé de lettres (en majuscules, de chiffres et d'espaces). Pour des raisons usuelles, nous avons choisi d'utiliser le caractère [pour signifier la présence d'un espace dans la phrase.

Le caractère * permettra de signifier au programme la fin du message. Par exemple pour afficher le message HELLO WORLD, il vous faudra initialiser la variable **Message[ ]** avec **const char Message[] = "HELLO[WORLD[[[*";**

Nous avons eu recours à plusieurs caractères **[** en fin de message qui seront considérés par le programme comme autant d'espaces en fin de message.



```
#include <SPI.h>                          // Intégration de la librairie SPI
#define CS0 77                           // Attribution du signal CS0 au port 77
#define RCLK 54                          // Attribution du signal RCLK au port 54
#define MR 33                            // Attribution du signal MR au port 33
#define RST 6                            // Attribution du signal RST au port 6

byte Matrice[2][7];                      // Définition de la matrice
byte Pos;                                // Variable gestion de la position dans le message
byte Decal;                              // Variable calcule position dans la matrice
byte Espace;                             // Variable pour gérer un décalage moindre en cas d'espace
const byte Vitesse = 3;                  // Vitesse de déplacement du scrolling

const char Message[] = "STIMULEZ[VOTRE[CREATIVITE[AVEC[LES[MODULES[CLICK[BOARDS[[[*";  // Votre mesage

const byte MLettres[37][7] ={{0b01110000,0b10001000,0b10001000,0b11111000,0b10001000,0b10001000,0b10001000},   // Lettre A
                             {0b11110000,0b10001000,0b10001000,0b11110000,0b10001000,0b10001000,0b11110000},    // Lettre B
                             {0b01111000,0b10000000,0b10000000,0b10000000,0b10000000,0b10000000,0b01111000},    // Lettre C
                             {0b11110000,0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b11110000},    // Lettre D
                             {0b11111000,0b10000000,0b10000000,0b11110000,0b10000000,0b10000000,0b11111000},    // Lettre E
                             {0b11111000,0b10000000,0b10000000,0b11110000,0b10000000,0b10000000,0b10000000},    // Lettre F
                             {0b11111000,0b10000000,0b10000000,0b10111000,0b10001000,0b10001000,0b11111000},    // Lettre G
                             {0b10001000,0b10001000,0b10001000,0b11111000,0b10001000,0b10001000,0b10001000},    // Lettre H
                             {0b11111000,0b00100000,0b00100000,0b00100000,0b00100000,0b00100000,0b11111000},    // Lettre I
                             {0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000},    // Lettre H
                             {0b11111000,0b00100000,0b00100000,0b00100000,0b00100000,0b00100000,0b11000000},    // Lettre J
                             {0b10001000,0b10010000,0b10100000,0b11000000,0b10100000,0b10010000,0b10001000},    // Lettre K
                             {0b10000000,0b10000000,0b10000000,0b10000000,0b10000000,0b10000000,0b11111000},    // Lettre L
                             {0b10001000,0b11011000,0b10101000,0b10001000,0b10001000,0b10001000,0b10001000},    // Lettre M
                             {0b10001000,0b10001000,0b11001000,0b10101000,0b10011000,0b10001000,0b10001000},    // Lettre N
                             {0b01110000,0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b01110000},    // Lettre O
                             {0b11110000,0b10001000,0b10001000,0b11110000,0b10000000,0b10000000,0b10000000},    // Lettre P
                             {0b01110000,0b10001000,0b10001000,0b10001000,0b10101000,0b10010000,0b01101000},    // Lettre Q
                             {0b11110000,0b10001000,0b10001000,0b11110000,0b10010000,0b10001000,0b10001000},    // Lettre R
                             {0b01110000,0b10001000,0b10000000,0b01110000,0b00001000,0b10001000,0b01110000},    // Lettre S
                             {0b11111000,0b00100000,0b00100000,0b00100000,0b00100000,0b00100000,0b00100000},    // Lettre T
                             {0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b01110000},    // Lettre U
                             {0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b01010000,0b00100000},    // Lettre V
                             {0b10001000,0b10001000,0b10101000,0b10101000,0b11011000,0b10001000,0b10001000},    // Lettre W
                             {0b10001000,0b10001000,0b01010000,0b00100000,0b01010000,0b10001000,0b10001000},    // Lettre X
                             {0b10001000,0b10001000,0b01010000,0b00100000,0b00100000,0b00100000,0b00100000},    // Lettre Y
                             {0b11111000,0b00001000,0b00010000,0b00100000,0b01000000,0b10000000,0b11111000},    // Lettre Z
                             {0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000,0b00000000},    // ESPACE
                             {0b11111000,0b10001000,0b10001000,0b10001000,0b10001000,0b10001000,0b11111000},    // Chiffre 0
                             {0b00100000,0b01100000,0b10100000,0b00100000,0b00100000,0b00100000,0b11111000},    // Chiffre 1
                             {0b11111000,0b00001000,0b00001000,0b11111000,0b10000000,0b10000000,0b11111000},    // Chiffre 2
                             {0b11111000,0b00001000,0b00001000,0b01111000,0b00001000,0b00001000,0b11111000},    // Chiffre 3
```

```
                {0b00010000,0b00110000,0b01010000,0b11111000,0b00010000,0b00010000,0b00010000},   // Chiffre 4
                {0b11111000,0b10000000,0b10000000,0b11111000,0b00001000,0b00001000,0b11111000},   // Chiffre 5
                {0b11111000,0b10000000,0b10000000,0b11111000,0b10001000,0b10001000,0b11111000},   // Chiffre 6
                {0b11111000,0b00001000,0b00010000,0b00100000,0b01000000,0b01000000,0b01000000},   // Chiffre 7
                {0b11111000,0b10001000,0b10001000,0b11111000,0b10001000,0b10001000,0b11111000},   // Chiffre 8
                {0b11111000,0b10001000,0b10001000,0b11111000,0b00001000,0b00001000,0b11111000}};  // Chiffre 9


void setup() {
  randomSeed(analogRead(57));                          // Réinitialise la séquence aléatoire
  pinMode(CS0, OUTPUT);pinMode(RCLK, OUTPUT);          // Les broches CS0 et RCLK sont des sorties
  pinMode(MR, OUTPUT);pinMode(RST, OUTPUT);            // Les broches MR et RST sont des sorties
  digitalWrite(MR, HIGH);                              // Force la broche MR au niveau logique haut
  digitalWrite(RCLK, LOW);                             // Niveau logique bas sur l'horloge du 4017 du module 7x10 R click
  digitalWrite(RST, LOW);                              // Niveau logique bas sur l'entrée de RESET du 4017 du module 7x10 R click
  SPI.begin();                                         // Initialiation du port SPI
}

void loop(){
  Pos = 0;                                             // On se place sur le premier caractère du message
  do
  {Scrolling ();Pos++;
  } while (Message[Pos]!=42);}                         // Tant que l'on est pas arrivé à la fin du message

void Scrolling (){                                     // Fonction permettant de réaliser le scrolling
    if (Message[Pos]>57) {Decal=65;}else{Decal=21;}    // Repositionnement dans la matrice chiffre/lettre
    Espace = 6;if (Message[Pos]==91) Espace = 3;
    for (byte j = 0 ; j < Espace; j++)
      {for (byte k = 0 ; k < Vitesse; k++) {Rmatrice();}          // Temporisation gérant la vitesse du scrolling
       for (byte i = 0 ; i < 7; i++){Matrice[1][i]=Matrice[1][i]>>1;
       bitWrite(Matrice[1][i],4,bitRead(Matrice[0][i],0));
       Matrice[0][i]=Matrice[0][i]>>1;   bitWrite(Matrice[0][i],4,bitRead(MLettres[Message[Pos]-Decal][i],7-j));}}}

void Rmatrice() {                                      // Fonction permettant de rafraichir la matrice
     digitalWrite(RST, HIGH);digitalWrite(RST, LOW);   // Reset du 4017 du module 7x10 R click
     for (byte i = 0 ; i < 7; i++){                    // Boucle générant 7 coups d'horloge sur le 4017
     digitalWrite(CS0, LOW);                           // Sélection broche CSO
     SPI.transfer(Matrice[0][i]); SPI.transfer(Matrice[1][i]);   // Envoie les données aux matrices
     digitalWrite(CS0, HIGH);delay(3);                 // Désélection de la broche CS0
     digitalWrite(MR, LOW);digitalWrite(MR, HIGH);     // Vide le registre à décalage des 74HC595
     digitalWrite(RCLK, HIGH);digitalWrite(RCLK, LOW);}  // Coup d'horloge sur le 4017 du module 7x10 R click
     digitalWrite(CS0, LOW);
     SPI.transfer(Matrice[0][0]);
     SPI.transfer(Matrice[1][0]);
    digitalWrite(CS0, HIGH);
 }
```

Description et explications (Suite)

Dans le cadre de notre application et du programme proposé, nous avons choisi un message plus long vantant les possibilités des modules Click Boards de Mikroelektronika ! STIMULEZ VOTRE CREATIVITE AVEC LES MODULES CLICK BOARD

Une seconde variable de type tableau (à deux dimensions) **Mlettres[ ][ ]** a été utilisée afin que nous puissions y stocker la représentation de chaque lettre et de chaque chiffre sur la matrice à led selon le même principe que celui que nous avions utilisé lors du tout premier programme de pilotage du module « 7x10 R Click ».

Chaque lettre et chaque chiffre est donc dessiné sur une matrice de 5 leds de large et de 7 leds de haut.

La principale différence par rapport au premier programme de gestion du module « 7x 10 R Click » est qu'il nous est possible désormais de pouvoir choisir plusieurs caractères à afficher au lieu d'un seul.

La variable **Vitesse** a aussi été utilisée pour déterminer la vitesse de déplacement du journal lumineux.

La boucle principale du programme va consister à récupérer les caractères du message un à un via la variable **Message[ ]**.

Lors de la récupération des données on effectue un test pour vérifier si le code ascii du caractère ainsi récupéré n'est pas égal à la valeur 42 (ce qui correspondant à la valeur ascii du caractère ***** à qui on a attribué le rôle de définir la fin du message). Si tel est le cas, le défilement du message reprendra depuis le début de la variable **Message[ ]**.

Le programme va ensuite se servir de la valeur ascii du caractère récupéré pour déterminer et sélectionner l'emplacement du caractère à afficher depuis la variable tableau **Message[ ]**.

On notera que les codes ascii des chiffres et des lettres ne se suivent pas :

Les chiffres 0 à 9 ont pour codes ascii des valeurs allant de 48 à 57 alors que les lettres A à Z ont pour codes ascii des valeurs allant de 65 à 90 … et même 91 pour le caractère [- raison pour laquelle nous l'avons choisi pour déterminer un espace.

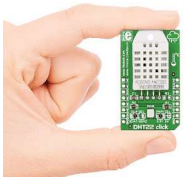
En conséquence il nous a fallu réaliser quelques tests pour ajouter un index dans le calcul et la récupération des caractères à afficher depuis la variable **Message[ ]** en fonction de la valeur du code ascii récupéré.

Le reste du programme consiste à récupérer et à introduire bit à bit et colonne par colonne les leds à allumer de chaque caractère récupéré sur la colonne complètement à droite de la matrice, puis à effectuer un décalage de toute la matrice vers la gauche et ainsi de suite pour afficher complètement le nouveau caractère (nécessitant 5 décalages vers la gauche – ce qui correspond à la largeur de chaque caractère).

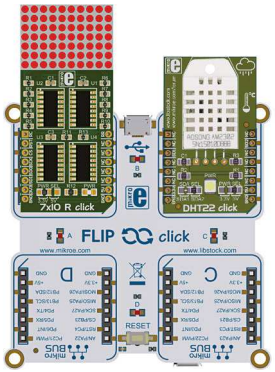
Le programme effectue d'office un décalage supplémentaire vers la gauche de la matrice après chaque caractère (afin d'obtenir un espace automatique entre chaque caractère afin que ces derniers ne soient pas collés les uns aux autres).

Cet espace automatique est plus ou moins grand si le caractère à afficher est déjà un espace afin d'obtenir un affichage uniforme.

Application N° 023 Petite station météorologique



Cette application va vous permettre de réaliser une mini-station météo à l'aide du module "7x10 R Click" et du module « DHT22 Click ». Cette dernière sera capable d'afficher la tendance du temps qu'il fait (ou fera) par le biais de 3 petits pictogrammes animés (soleil rayonnant, nuage défilant ou pluie tombante). La mini-station fera également défiler la valeur de la température ambiante ainsi que le niveau d'humidité de l'air mesuré par le module « DHT22 Click »



Réalisation du câblage

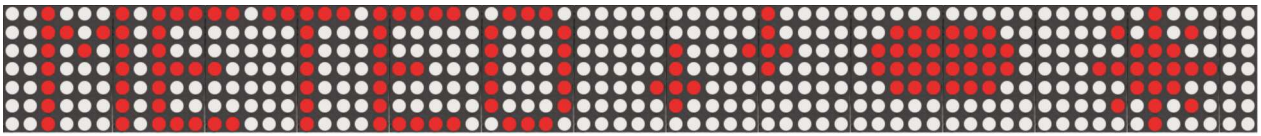
Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7x10 R click » sur le support **A** et le module « DHT22 Click » sur le support **B** (attention au sens). Déconnectez ensuite tous les autres composants et fils de liaisons potentiellement existants sur le platine. Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis téléversez le programme de la mini-station météo dans la platine.

Description et explications

Le programme de cette application étant particulièrement « grand », nous avons choisi de ne pas le présenter dans cet ouvrage afin qu'il ne monopolise pas trop de pages inutilement. Vous pourrez retrouver ce dernier en téléchargement à l'adresse indiquée en page 17.

Ce programme représente une synthèse de plusieurs applications déjà vues au sein de cet ouvrage (gestion du module « DHT 22 Click », gestion du module « 7x10 R click », gestion d'un message défilant, etc...).

Aussi une fois encore, nous n'allons pas détailler les instructions de ce dernier les unes après les autres selon le déroulement du listing, mais plutôt vous exposer les différents concepts et astuces utilisés pour parvenir à la réalisation de cette application.



Description et explications (Suite)

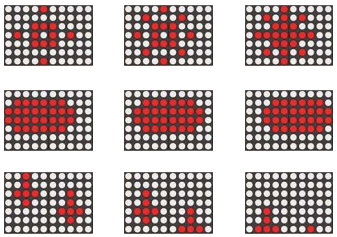
Comme indiqué ci-avant le programme reprend une grande partie des initialisations et fonctions utilisées dans l'application du journal lumineux défilant, lesquels nous serviront à afficher les informations de la station météo.

Vous pourrez toutefois trouver de nouvelles variables de type tableau à 2 éléments **Soleil[ ][ ]**, **Nuage[ ][ ]** et **Pluie[ ][ ]** que nous avons utilisé pour mémoriser les animations des petits pictogrammes de prédiction de la météo.

Ces animations consisteront à faire défiler successivement des petits éléments graphiques sur la matrice. En cas de temps sec, un soleil dont les rayons brilleront successivement sera affiché via ces animations.

Dans le cas d'un temps légèrement humide, un nuage se déplaçant de droite à gauche et de gauche à droite fera son apparition.

En cas de mesure d'un fort taux d'humidité, une petite animation (plus complète que sur les dessins ci-contre) simulera des gouttes de pluie qui tombent du ciel.

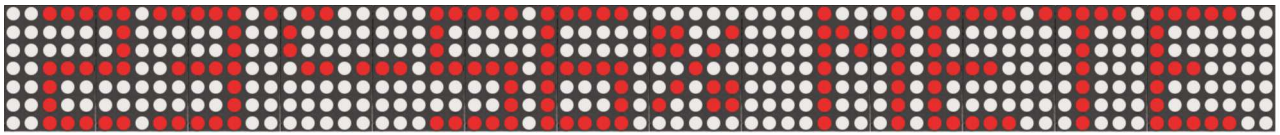


Une autre variable de type tableau à une seule dimension **Message[] = " ";** est également utilisée en début de programme. Celle-ci servira à stocker le message qui défilera à l'écran. La variable est initialement configurée avec un message « vide » dans un premier temps, lequel sera composé au cas par cas par le programme avec les données récupérées grâce au capteur DHT22.

La suite du programme reprend également l'initialisation des caractères A à Z et des chiffres 0 à 9 déjà vues dans le programme du journal défilant avec l'apparition de 3 nouveaux caractères **°C**, **%** et **–** qui seront utilisés lors de la composition du message affiché par la station météo.

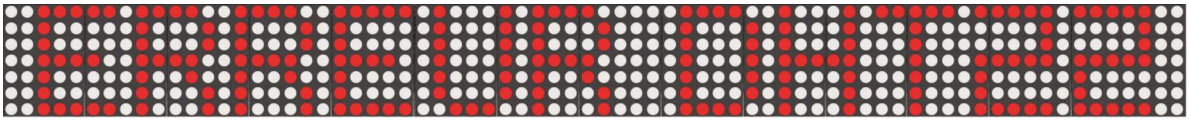
Le programme commencera par effectuer une temporisation de 3 secondes afin de laisser le temps au capteur DHT22 de se stabiliser (dans les faits ce dernier nécessite plusieurs minutes avant de délivrer des valeurs vraiment stables).

On récupère alors la valeur de la température et du niveau d'humidité via le module « Dht22 click » et on vérifie si les 2 ont des valeurs nulles. Si tel est le cas, la platine « Flip & Click » considèrera qu'il y a un problème de communication avec le module « DHT22 click » (ou qu'il est absent) et le programme va configurer le message **ERREUR DHT22** à afficher sur la matrice, lequel défilera en boucle.



Description et explications (suite)

Attention votre premier réflexe va être de retirer le module « DHT22 click » de son support pour tester l'apparition de ce message... **STOP !!!** n'oubliez pas qu'il est impératif de **TOUJOURS** déconnecter l'alimentation de la platine « Flip & Click » lorsque vous insérez ou retirez un module Click Board. Donc n'oubliez pas de vous conformer à ces recommandations si vous voulez tester l'apparition du message en question.



Si en revanche, les valeurs de la température et de l'humidité retournées par le capteur DHT22 ne sont pas nulles, le programme va composer le message à afficher avec le texte **METEO**. Ce message va alors défiler sur la matrice, puis le programme effectuera un test sur le niveau d'humidité mesuré afin de déterminer quel pictogramme animé il devra afficher.

De base nous avons choisi d'afficher l'animation du soleil en cas de taux d'humidité inférieur ou égale à 49 %, et d'afficher l'animation avec le nuage lorsque l'humidité est supérieur à 49 % et inférieur à 59 % et l'animation avec les gouttes de pluie pour un taux d'humidité supérieur à 59 %. Ces valeurs de seuils peuvent bien sur être modifiées à volonté en fonction de l'emplacement de votre platine et des conditions d'utilisation de la platine. L'affichage des pictogrammes est effectué via différentes fonctions qui vont faire défiler les animations à l'écran.

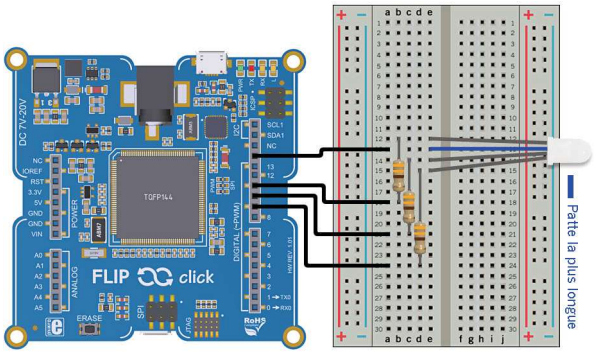
Une fois l'animation du pictogramme terminée, le programme va composer un nouveau message à l'aide des valeurs de la température et de l'humidité (c'est à ce niveau que les nouveaux caractères **°C –** et **%** sont utilisés). Le message ainsi composé va alors défiler à son tour à l'écran de la droite vers la gauche.

Le programme reprend alors une nouvelle mesure et affiche à nouveau le message **METEO** puis le pictogramme.... puis à nouveau la température et le taux d'humidité et ainsi de suite.

Vous pourrez bien sûr également modifier à volonté la vitesse de défilement des messages et des animations des pictogrammes. Rien de vous empêche de modifier carrément les animations des pictogrammes ou de créer plus de seuils et d'ajouter d'autres pictogrammes.

Attention, pour rappel, la platine « Flip & Click » ainsi que les modules Click Boards ne sont pas étanches !! ne les placez pas à l'extérieur .. même sous abris.. Sans protection l'humidité finira par les abîmer. Evitez également de placer celle-ci à proximité de votre cuisine ou de votre salle de bain afin d'éviter que les vapeurs ne faussent les mesures. Cette petite station météo est uniquement prévue pour une utilisation en intérieur. La tendance météo est bien sur donnée à titre indicatif. Celle-ci est basée exclusivement sur la mesure du taux d'humidité de l'air. L'interprétation de la météo sera donc donnée avec une certaine inertie. Si par exemple, le temps a été à la pluie depuis plusieurs jours et si le soleil finit par pointer le « bout de son nez», ne vous attendez pas à ce que la station météo affiche le pictogramme du soleil immédiatement. En effet, en attendant que l'humidité de l'air diminue, celle-ci continuera à afficher le pictogramme du nuage ou de la pluie pendant quelques heures ou quelques jours suivant les cas.

Application N° 024 Pilotage d'une led RGB




```
const byte LedRouge = 11;      // Attribution de la led rouge au port 11
const byte LedVerte = 10;      // Attribution de la led verte au port 10
const byte LedBleue = 9;       // Attribution de la led bleue au port 9

void setup()  {
  pinMode(LedRouge, OUTPUT);   // Port led Rouge en sortie
  pinMode(LedVerte, OUTPUT);   // Port led Verte en sortie
  pinMode(LedBleue, OUTPUT);   // Port led Bleue en sortie
  randomSeed(analogRead(0)); } // Réinitialise séquence aléatoire

void loop()   {
  analogWrite(LedRouge, 255);      // PWM pour allumer led Rouge
  analogWrite(LedVerte, 0);        // PWM pour éteindre la led Verte
  analogWrite(LedBleue, 0);        //  PWM pour éteindre la led Bleue
  delay(1000);                     // Temporisation 1 seconde
  analogWrite(LedRouge, 0);        // PWM pour éteindre led Rouge
  analogWrite(LedVerte, 255);      // PWM pour allumer led Verte
  analogWrite(LedBleue, 0);        // PWM pour éteindre la led Bleue
  delay(1000);                     // Temporisation 1 seconde
  analogWrite(LedRouge, 0);        // PWM pour éteindre led Rouge
  analogWrite(LedVerte, 0);        // PWM pour éteindre la led Verte
  analogWrite(LedBleue, 255);      // PWM pour allumer led Bleue
  delay(1000);                     // Temporisation 1 seconde
  analogWrite(LedRouge, random(0,255));  // PWM aléatoire led Rouge
  analogWrite(LedVerte, random(0,255));  // PWM aléatoire led Verte
  analogWrite(LedBleue, random(0,255));  // PWM aléatoire led Bleue
  delay(2000); }                   // Temporisation 2 secondes
```

Cette application va vous permettre de modifier la couleur d'une led RVB afin d'obtenir son allumage en rouge, puis en bleu, puis en vert, puis avec une couleur aléatoire.

Réalisation - Description et explications

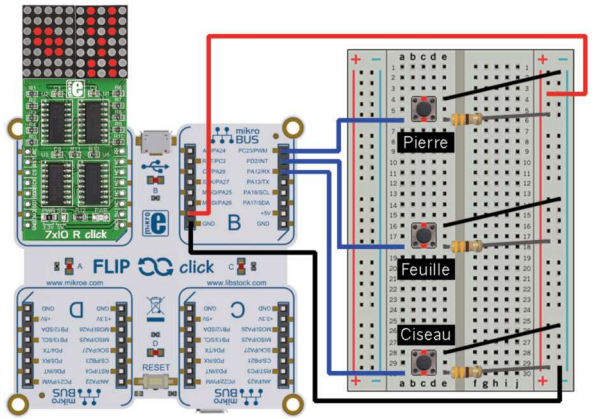
Déconnectez la platine « Flip & Click » de votre ordinateur et réalisez le montage ci-contre. Les ports 9 à 11 de la platine pilotent les 3 leds (avec les habituelles résistances en série). Saisissez alors le programme, raccordez la platine à votre ordinateur et téléversez le programme pour tester ce dernier.

Une led RVB est en fait composée (intérieurement) de 3 leds de couleurs différentes. Une led rouge, une led verte et une led bleue. Dans la cadre du modèle livré avec votre starter-kit, ces 3 leds ont une cathode commune (le raccordement à la masse des 3 leds est donc commun et se fait avec une seule broche – la broche la plus longue). Les 3 autres broches correspondent à l'anode de chacune des leds rouge, verte et bleue.

Le programme qui pilote cette led est très simple à comprendre. Après une phase d'initialisation et de déclaration des 3 ports en tant que sortie (sur lesquels sont reliés les leds internes à la led RVB), on initialise la séquence de génération des nombres aléatoires (déjà vu au cours des montages précédents).

Dans la boucle principale du programme, on va tour à tour générer un signal PWM de rapport cyclique max. sur une des leds (et de rapport cyclique min. sur les autres) afin d'allumer une seule led interne à la fois (et donc une seule couleur à la fois). On réalise une petite temporisation d'une seconde entre chaque modification de couleur. Lors de la dernière étape, cette fois-ci on génère un signal PWM de valeur aléatoire pour chacune des leds afin d'obtenir au final une couleur aléatoire (par mélange des couleurs). On réalise une petite temporisation plus longue de 2 secondes pour se rendre mieux compte de la couleur ainsi générée.

Application N° 025 Pierre - Feuille - Ciseau avec module "7 x 10 R Click"



Réalisation du câblage

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7x10 R click » sur le support **A** (attention au sens) et réalisez le montage ci-dessus. Connectez ensuite la platine « Flip & Click » à votre ordinateur.

Le programme de cette application étant relativement grand et du fait qu'il exploite de nombreuses parties des programmes précédents, nous avons choisi de ne pas l'intégrer à l'ouvrage. Vous pourrez bien sûr le télécharger comme indiqué en page 17 de ce manuel et le téléversez dans la platine.

De même, afin de vous éviter une lecture trop fastidieuse, nous n'allons pas détailler les instructions de ce dernier les unes après les autres selon le déroulement du listing, mais plutôt vous exposer les différents concepts et astuces utilisés pour parvenir à la réalisation de cette application.

Cette application va vous permettre de défier la platine "Flip & Click" dans un jeu de "**Pierre - Feuille - Ciseau**". Dans ce jeu de hasard, vous serez amené à choisir une figure parmi 3 au choix (Pierre - Feuille - Ciseau) à l'aide de 3 boutons-poussoirs.

Notre version du jeu se gagne sur une manche en 5 points. Au début de la partie, les 2 joueurs disposent d'un score nul et la matrice à led "7x10 R Click" affiche **0 - 0**. Ensuite la platine "Flip & Click" vous invite (via le message **GO !** qui s'affiche sur la matrice à led "7x10 R Click") à choisir une figure (à l'aide des boutons-poussoirs).

Au moment où vous solliciterez le bouton-poussoir correspondant à la figure de votre choix, la platine "Flip & Click" choisira également une de ces 3 figures (de façon aléatoire). Les 2 figures (la vôtre et celle de la platine "Flip & Click" seront alors affichées côte à côte sur la matrice à led "7x10 R Click" (exploitée pour l'occasion horizontalement). Votre figure s'affichera sur la partie droite de la matrice et la figure de la platine "Flip & Click" s'affichera sur la partie gauche de la matrice.

A ce moment les 2 figures s'affronteront selon les règles:

- La pierre bat le ciseau (en le cassant ou en l'émoussant)
- La feuille bat la pierre (en l'enveloppant)
- Le ciseau bat la feuille (en la coupant)

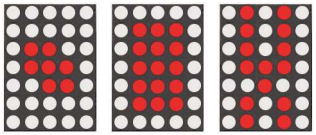
La figure gagnante fera remporter 1 point à son propriétaire avec en prime une petite animation matérialisée par une ligne de leds qui balaie la matrice de haut en bas en effaçant le côté de la matrice dans lequel se trouve la figure qui a perdu (en laissant ainsi seulement sur la matrice la figure gagnante).

A ce stade, la platine "Flip & Click" indiquera l'état du score pendant quelques instants avant d'afficher le message **GO !** pour vous inviter à choisir de nouveau une figure.

A noter que si vous avez choisi la même figure que celle de la platine "Flip & Click", ces figures s'annulent (aucun point n'est gagné) et les 2 figures seront effacées via l'animation décrite ci-dessus.

Description et explications

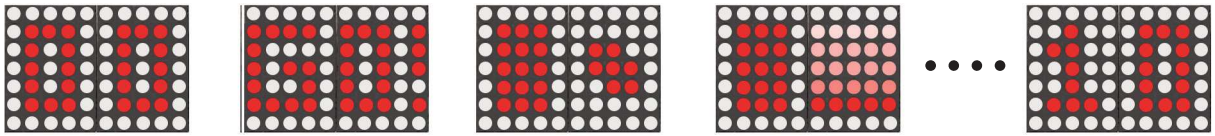
Comme indiqué ci-avant le programme reprend une grande partie des initialisations et fonctions utilisées dans les applications du sablier animé et du jeu de tennis 2 joueurs. Celles-ci concernent la déclaration des broches attribuées à la gestion de la matrice ainsi qu'aux 3 broches sur lesquelles sont raccordés les boutons-poussoirs. Une variable de type tableau à 2 dimensions **MScore[11][7]** est utilisée pour définir la représentation des chiffres **0** à **5** (lesquels serviront à indiquer les scores) ainsi que les lettres **GO !** et les 3 figures **Pierre** / **Feuille** / **Ciseau**.



Pierre **Feuille** **Ciseau**

Une autre variable de type tableau à 2 dimensions **Matrice[2][7]** est utilisée pour le stockage des leds à afficher sur la matrice, tandis que les variables **ScoreJ1**, **ScoreJ2** et **MainJ1**, **MainJ2** permettront de mémoriser votre score et celui de la platine "Flip & Click" ainsi que la sélection du signe que vous (et la platine "Flip & Click") aurez choisi.

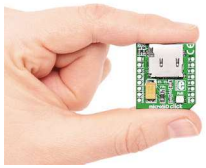
Après l'initialisation du fonctionnement des différents ports, le programme commence par afficher le score de début de partie. Pour ce faire, nous avons créé une fonction: **AffScore(ScoreJ1,ScoreJ2);** Comme vous pouvez le constater, les variables **ScoreJ1** et **ScoreJ2** sont présentes à l'intérieur des parenthèses lorsqu'on appelle la fonction. Si nous examinons la structure de la fonction en question: **void AffScore(byte MatriceD, byte MatriceG)** vous pouvez également constater la présence de 2 autres variables **MatriceD** et **MatriceG**. Ces dernières vont automatiquement recevoir la valeur des variables **ScoreJ1** et **ScoreJ2** lorsque vous appelerez la fonction afin d'afficher le score sur la matrice.



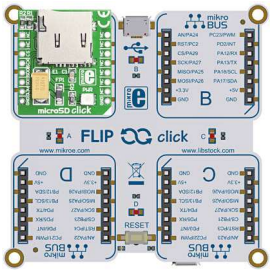
Affichage du score de début de partie | **Choisissez votre signe** | **Affichage des signes** | **Effacement du signe qui a perdu** | **Mise à jour et affichage du score**

Le programme continue ensuite en affichant le mot **GO !** sur la matrice et en attendant qu'une des 3 touches soit sollicitée. Si tel est le cas, la variable **MainJ2** sera initialisée avec la valeur 6, 7 ou 8 en fonction du signe choisi. Puis la variable **MainJ1** recevra un nombre aléatoire également compris entre 6 et 8. Ces valeurs ont été choisies car elles correspondent aux emplacements des signes dans la variable **Matrice[2][7]** . A ce stade, le programme appelle à nouveau la fonction **AffScore(MainJ1,MainJ2);** mais en passant cette fois ci en paramètre d'entrée les variables **MainJ1** et **MainJ2** afin d'afficher non plus le score... mais les signes que vous et la platine "Flip & Click" avez choisi. Maintenant le programme va tester si vous et la platine "Flip & Click" avez choisi le même signe ou (si tel n'est pas le cas), quel est le signe qui prend le dessus sur l'autre (selon les règles indiquées sur la page de gauche). En fonction du résultat, on appelle une nouvelle fonction **EffMatrice(MainJ1,MainJ2,1,1);** avec un passage de paramètres permettant de sélectionner les signes à afficher ainsi que les signes à effacer (celui de gauche ou de droite... ou les deux... via l'animation décrite dans la page de gauche). Suivant les cas, votre score ou celui de la platine "Flip & Click" sera incrémenté et affiché. Le programme entre dans une boucle sans fin faisant clignoter le score dès qu'un de vous arrive à 5 points.

Application N° 026 Utilisation du module microSD Click MIKROE-924



Le module « microSD Click » permettra d'ajouter une capacité de stockage mémoire à votre platine « Flip & Click » par le biais d'un petit connecteur sur lequel vous pourrez lire et écrire sur une carte mémoire microSD™ (non livrée). La carte mémoire microSD™ et le module « microSD Click » devront impérativement être enfichés ou retirés exclusivement lorsque la platine « Flip & Click » est hors tension.



Le module microSD Click n'est pas livré avec votre starter-kit

Déconnectez donc la platine « Flip & Click » de votre ordinateur, insérez le module «microSD click » sur le support **A** (attention au sens), puis insérez une carte mémoire microSD™ dans le connecteur prévu à cet effet. Connectez ensuite la platine « Flip & Click » à votre ordinateur.

Vous disposez de plusieurs exemples prêts à l'emploi pour la gestion de ce type de carte mémoire dans l'environnement IDE.

Ainsi, chargez le programme depuis Fichiers → Exemples → SD → CardInfo

Pour indiquer au programme le « bon » numéro de broche CS que vous utilisez, modifiez la ligne
const int chipSelect = 4; par **const int chipSelect = 77;**

Téléversez ensuite le programme et lancez le moniteur série dans l'éditeur IDE (bouton loupe en haut à droite). A ce stade (si votre carte microSD™ a bien été préalablement formatée), la fenêtre doit vous afficher les informations présentes sur la carte mémoire, comme dans l'exemple ci-dessous) :

Initializing SD card...Wiring is correct and a card is present.

Card type: SDHC

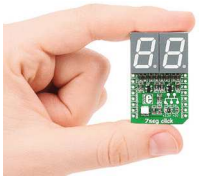
Volume type is FAT32

Volume size (bytes): 3900702720
Volume size (Kbytes): 3809280
Volume size (Mbytes): 3720

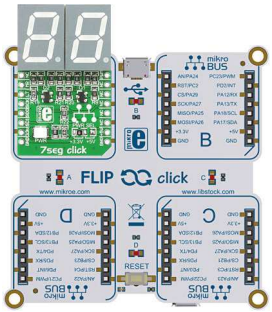
Files found on the card (name, date and size in bytes):

Dans notre exemple, la carte est vierge. Vous trouverez d'autres exemples vous permettant de lire et d'écrire sur vos cartes microSD™ au sein de l'IDE accessibles depuis Fichiers → Exemples → SD (pensez à modifier le port de CS comme indiqué ci-dessus).

Application N° 027 Utilisation du module 7Seg Click MIKROE-1201



Le module « 7 seg Click » intègre 2 afficheurs 7 segments à leds associés à 2 circuits intégrés de type 74HC595 (registres à décalage série). Les sorties de ces circuits intégrés alimentent les différents segments des afficheurs en fonction des données qui leurs sont transmises (sous la forme série via leur port SPI) afin de pouvoir (selon votre programmation) afficher des chiffres et mêmes quelques lettres de l'alphabet.



Le module 7Seg Click n'est pas livré avec votre starter-kit

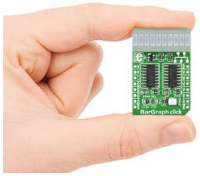
L'avantage apporté par l'utilisation des circuits intégrés 74HC595 est que par rapport à l'application N° 15 de cet ouvrage, nous n'utilisons que 3 broches de la platine « Flip & Click » pour piloter 2 afficheurs !

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «7 seg click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-dessous (lequel va vous permettre de réaliser un compteur de 00 à 99) et téléversez-le dans la platine.

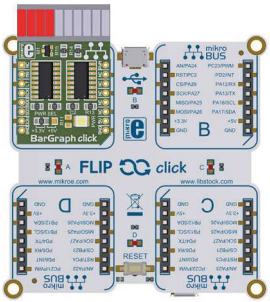
```
#include <SPI.h>                  // Intégration de la librairie SPI
#define CS0 77                    // Attribution du signal CS0 au port 77
byte Digit1=0, Digit2=0;          // Variables compteur des 2 digits
const byte Digit[10]
={0b01111110,0b00001010,0b10110110,0b10011110,0b11001010,0b
11011100,0b11111100,0b00001110,0b11111110,0b11011110};

void setup() {
  pinMode(CS0, OUTPUT);      // La broche CS0 est une sortie
  SPI.begin();               // Initialiation du port SPI
}

void loop() {
   digitalWrite(CS0, LOW);      // Sélection broche CSO
   SPI.transfer(Digit[Digit2]); // Envoie les données du 2ème digit
   SPI.transfer(Digit[Digit1]); // Envoie les données du 1er digit
   digitalWrite(CS0, HIGH);     // Désélection broche CS0
   Digit2++;                    // Incrémente le compteur (sur digit 2)
   if (Digit2==10){             // Regarde si unité > 9 ?
     Digit2 = 0;                // Remet les unités à zéro
     Digit1++;                  // Incrémente les dizaines
     if (Digit1==10) Digit1=0; } // Regarde si dizaine > 9 -> Dizaine = 0
   delay(1000);                 // Temporisation 1 seconde
}
```

Ce programme est relativement simple à comprendre. Ce dernier fait appel à la librairie SPI, laquelle va prendre en charge le dialogue entre la platine « Flip&Click » et les circuits intégrés 74HC595. Dans la partie désignation et initialisation, on indique le numéro du port attribué à la broche CS des 74HC595 ainsi que les différentes valeurs à transmettre pour afficher les chiffres 0 à 9 (via le tableau de la variable **Digit [ ]**). 2 variables **Digit1** et **Digit2** permettront de gérer le compteur et l'affichage des 2 digits de l'afficheur. Dans la boucle principale, on va commencer par placer la broche CS au niveau logique bas, puis par transmettre successivement la valeur de la variable **Digit2**, puis **Digit1** aux 74HC595 (qui sont « montés en cascade »). En appliquant un niveau logique haut sur CS, les données ainsi transmises aux 74HC595 seront alors appliquées sur leurs sorties afin de pouvoir allumer les segments lumineux des afficheurs. Le reste du programme consiste à gérer le compteur en testant l'incrémentation des unités de la variable **Digit2** et en incrémentant les dizaines quand c'est nécessaire. Une temporisation d'une seconde est ajoutée entre chaque rafraîchissement du compteur. En modifiant les données mémorisées dans le tableau **Digit [ ]**, il serait possible d'afficher les lettres A, C, E, F ainsi que le point de séparation.

Application N° 028 Utilisation du module BarGraph Click MIKROE-1423



Le module « BarGraph Click » intègre un bargraph (composé de 10 leds associées à 2 circuits intégrés de type 74HC595 registres à décalage série). Les sorties de ces circuits intégrés alimentent les différents segments du bargraph en fonction des données qui leur sont transmises (sous la forme série via leur port SPI).



Le module BarGraph Click n'est pas livré avec votre starter-kit

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «BarGraph click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-dessous (qui va vous permettre de réaliser une « animation » du bargraph, lequel va allumer une à une toutes ses leds) et téléversez-le dans la platine.

Ce programme relativement simple à comprendre, fonctionne selon le même procédé que le programme précédemment. Il utilise le même mode de pilotage SPI avec les circuits intégrés 74HC595. On a recours à une variable **Valeur** dont la valeur sera incrémentée de 0 à 9 (permettant de gérer l'allumage successif des segments 1 à 10 du bargraph) ainsi que 2 autres variables **SegmentA** et **SegmentB** dont on positionnera les bits à « 1 » en fonction de la variable **Valeur**.

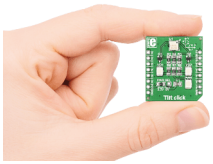
Pour ce faire, on utilise l'instruction **bitwrite( )**.

Le premier paramètre de cette instruction correspond à la variable sur laquelle on désire opérer **SegmentA** ou **SegmentB**, le second paramètre correspond à l'emplacement du bit que l'on désire modifier et le troisième paramètre à la valeur que l'on désire appliquer au bit (ici « 1 »). Dans une première boucle, on va tour à tour positionner les bits de la variable **SegmentA** à « 1 » en fonction de la variable **Valeur**. Ensuite on réalise simplement 2 tests via des instructions **if ( )** pour déterminer si on doit allumer le segment 9 et 10 en fonction de la variable Valeur.

La suite de notre programme envoie les données des variables **SegmentA** et **SegmentB** aux circuits intégrés 74HC595 avec une temporisation et un test pour vérifier si on est arrivé au dernier segment à afficher (auquel cas, on repart de zéro).

```
#include <SPI.h>                    // Intégration de la librairie SPI
#define CS0 77                      // Attribution du signal CS0 au port 77
byte SegmentA, SegmentB;            // Variables gestion segments
byte Valeur=0;                      // Variable à afficher sur le bargraph

void setup() {
  pinMode(CS0, OUTPUT);             // La broche CS0 est une sortie
  SPI.begin();                      // Initialiation du port SPI
}

void loop() {
  SegmentA = 0;                     // Initialisation des Segments A
  SegmentB = 0;                     // Initialisation des Segments B
  for (byte i = 0 ; i < Valeur; i++){  // Sélectionne les Segments
  bitWrite(SegmentA,i,1);}
  if (Valeur > 8) bitWrite(SegmentB,0,1); // Sélectionne les Segments
  digitalWrite(CS0, LOW);           // Sélection broche CSO
  SPI.transfer(SegmentB);           // Envoie les données des Segments B
  SPI.transfer(SegmentA);           // Envoie les données des Segments A
  digitalWrite(CS0, HIGH);          // Désélection broche CS0
  Valeur++;                         // Incrémente la valeur à afficher
  if (Valeur==11) Valeur=0;         // Si Valeur > 10 -> Valeur = 0
  delay(80);                        // Temporisation 80 milli-secondes
}
```

Application N° 029 Utilisation du module Tilt Click MIKROE-1834



Le module « Tilt Click » intègre un capteur d'orientation optique de type RPI-1035. Ce dernier délivre une information sur le positionnement à droite, à gauche, en avant ou en arrière du module par le biais de 2 bits (codage binaire).

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «Tilt click » sur le support **A** (attention su sens).

Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-dessous et téléversez-le dans la platine.

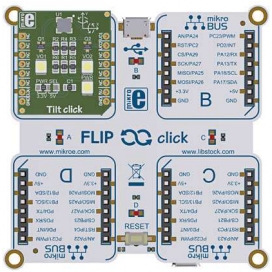
Ce programme très simple permet de récupérer le code binaire ainsi généré par les 2 sorties du capteur afin de pouvoir allumer une des 4 leds A / B / C/ D présentes sur la platine « Flip&Click » (côté face blanche).

Notez que le capteur détecte une inclinaison complète de la platine « flip&click » dans une position. Si vous inclinez la platine sur un côté, la led associée à ce côté va s'allumer.

Si par contre vous ramenez la platine « Flip & Click » à l'horizontale, la led ne s'éteindra pas.

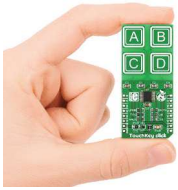
C'est en inclinant la platine sur une autre position extrême que la led s'éteindra au profit de la led représentant la nouvelle position.

```
const byte Vout1 = 6;       // Attribution de la sortie Vout1 du module Click au port 6
const byte Vout2 = 26;      // Attribution de la sortie Vout2 du module Click au port 26

const byte Sled[4] = {38,39,40,37};            // N° des ports affectés aux leds A/B/C/D
byte Orientation;                      // Variable dans laquelle est mémorisée l'orientation
const byte Led[4][4]= {{0,0,1,0},{0,1,0,0},{1,0,0,0},{0,0,0,1}};  // Attribution des leds en
fonction de l'orientation

void setup(){
  pinMode(Vout1, INPUT);          // Configure port sur lequel est relié Vout1 en entrée
  pinMode(Vout2, INPUT);          // Configure port sur lequel est relié Vout2 en entrée
  for (byte i = 0 ; i < 4; i++)          // Initialisation des ports des leds A/B/C/D
  {pinMode(Sled[i],OUTPUT);     // Attribution des ports en sortie
  digitalWrite(i, LOW);}}             // Eteint toutes les leds

void loop(){
  bitWrite(Orientation,0,digitalRead(Vout1)); // Met à jour le bit 0 en fonction de Vout1
  bitWrite(Orientation,1,digitalRead(Vout2)); // Met à jour le bit 1 en fonction de Vout2
  for (byte i = 0 ; i < 4; i++)                          // Initialisation des ports des leds A/B/C/D
    {digitalWrite(Sled[i],Led[Orientation][i]);}
  delay(100);}
```



Le module Tilt Click n'est pas livré avec votre starter-kit

Application N° 030 Utilisation du module TouchKey Click MIKROE-1906

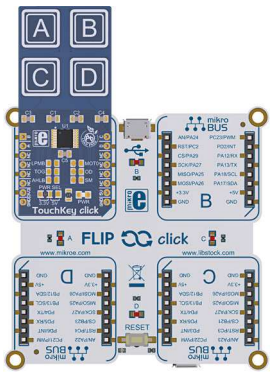


Le module « TouchKey Click » intègre 4 touches capacitives gérées par un capteur TTP224. Ces touches réagissent comme un bouton-poussoir qu'il vous suffit d'effleurer pour être manipulé. A noter que ces touches peuvent également être sollicitées sans contact physique direct (au travers d'une vitre de faible épaisseur ou d'une feuille de papier).

L'application qui suit va consister à réaliser une serrure codée sur laquelle il faudra solliciter 4 touches dans le bon ordre (dans notre cas les touches C/B/D/A) pour activer/désactiver l'allumage d'une led. Une seconde led s'allume à chaque fois qu'une des touches du clavier est sollicitée.

Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «TouchKey Click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-contre et téléversez-le dans la platine.

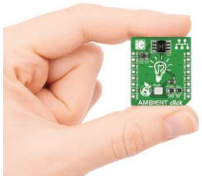
Ce programme utilise une variable de type tableau **code[4]** dans laquelle on va mémoriser l'ordre des touches nécessaire à la prise en compte du code de la serrure. La boucle principale va ensuite vérifier si au moins une des 4 touches est sollicitée (via des conditions OU sur chaque sortie logique des touches). Si tel est le cas, on appelle la fonction **Relache();** qui va allumer une led indiquant qu'on appuie sur une touche (en mémorisant celle-ci) puis qui va attendre que toutes les touches soient relâchées (en éteignant alors la led). A ce stade le programme utilise la variable **Progression** pour aller vérifier tour à tour si les touches sollicitées correspondent à l'ordre qui a été configuré dans la variable **code[4]** . Si tel est le cas, on inverse l'état de la seconde led (pour montrer la bonne prise en compte du code). Si l'ordre n'est pas respecté, on réinitalise la variable **Progression** pour attendre à nouveau la bonne séquence.



Le module Touch Click n'est pas livré avec votre starter-kit

```
const byte BPA = 33;        // Attribution du BP A au port 33
const byte BPB = A0;        // Attribution du BP B au port A0
const byte BPC = 6;         // Attribution du BP C au port 6
const byte BPD = 26;        // Attribution du BP D au port 26
const byte LedA = 38;       // Attribution de la led A au port 38
const byte LedB = 37;       // Attribution de la led B au port 37
byte Etat = 0;          // Variable état de la sortie de la serrure
byte Progression = 0;     // Variable progression saisie code
byte Touche;              // Variable touche sollicitée
const byte code[4] = {3,2,4,1};
// Séquence du code attendue ( 1=A / 2=B / 3=C / 4=D )

void setup()
{
  pinMode(BPA, INPUT);      // Configure port BP A en entrée
  pinMode(BPB, INPUT);      // Configure port BP B en entrée
  pinMode(BPC, INPUT);      // Configure port BP C en entrée
  pinMode(BPD, INPUT);      // Configure port BP D en entrée
  pinMode(LedA, OUTPUT);    // Port led A en sortie
  digitalWrite(LedA, Etat);       // Eteint la led A
  pinMode(LedB, OUTPUT);    // Port  led B en sortie
  digitalWrite(LedB, LOW);        // Eteint la led B
}

void loop()
{
if (digitalRead(BPA)==HIGHIdigitalRead(BPB)==HIGHI
digitalRead(BPC)==HIGHIdigitalRead(BPD)==HIGH){
   Relache();   // Attend relachement de toutes les touches
   if (code[Progression]==Touche)  // Touche attendue ?
     {Progression++;}                    // Oui avance progression
   else
     {Progression=0;}                    // Non RAZ progression
   if (Progression==4){
      Etat = !Etat;              // Inverse l'état de la variable Etat
      digitalWrite(LedA, Etat);     // Met à jour l'état de la led A
      Progression=0;}}}              // RAZ progression du code

    void Relache(){
    digitalWrite(LedB, HIGH);       // Allume la led B
    delay (20);                      // Tempo anti-rebond
    if (digitalRead(BPA)==HIGH) Touche=1;     // Touche A ?
    if (digitalRead(BPB)==HIGH) Touche=2;     // Touche B ?
    if (digitalRead(BPC)==HIGH) Touche=3;     // Touche C ?
    if (digitalRead(BPD)==HIGH) Touche=4;     // Touche D ?
    while (digitalRead(BPA)==HIGHIdigital
    Read(BPB)==HIGHIdigital
    Read(BPC)==HIGHIdigitalRead(BPD)==HIGH){}
    digitalWrite(LedB, LOW);        // Eteint la led B
    delay (20);                     // Tempo anti-rebond
}
```

Application N° 031 Utilisation du module Ambiant Click MIKROE-1890



Le module « Ambiant Click » intègre un capteur de luminosité.

Un peu à l'image d'une LDR, ce dernier capte la lumière ambiante et vous délivre une tension proportionnelle à celle-ci (de façon plus précise et plus linéaire qu'une LDR).

C'est ce type de capteur qui est généralement utilisé pour modifier automatiquement le rétro-éclairage de votre smartphone en fonction des conditions de luminosité ambiante afin que vous disposiez d'une lisibilité optimale sur votre écran.

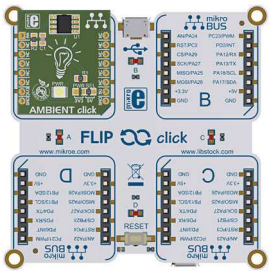
Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «Ambiante click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-dessous et téléversez-le dans la platine.

Ce programme (on ne peut plus simple) consiste à récupérer la tension en sortie du capteur afin de l'utiliser pour modifier le rapport cyclique d'un signal appliqué sur une des leds de la platine.

Plus la pièce dans laquelle sera utilisée la platine « Flip&Click » sera sombre et plus la led s'éclairera avec une forte intensité et inversement.

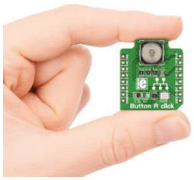
Le résultat de la conversion « Analogique / Numérique » issue du capteur est également envoyé au moniteur de l'environnement de développement via le port série de la platine « Flip & Cick » afin que vous puissiez visualiser les valeurs qu'il retourne sur l'écran de votre ordinateur.

```
const byte Led = 13;              // Attribution de la led B au port N° 13
int MIKROE1890 = A0;              // N° du port sur lequel est relié le capteur de luminosité
int Rcycle = 0;                   // Variable dédiée au stockage du rapport cyclique

void setup()
{
  Serial.begin(9600);             // initialise le port de communication série à 9600 bds
  pinMode(Led, OUTPUT);           // Configure le port sur lequel est relié la led B en sortie
}

void loop()
{
  Rcycle = analogRead(MIKROE1890);    // Lecture de la tension en sortie du capteur
  Serial.println(Rcycle);    // Envoi la valeur mesurée sur le terminal série (viaport série)
  delay (10);                         // Temporisation
  analogWrite(Led, (1023-Rcycle)/4);      // Génère le signal PWM sur la led 13
}
```



Le module Ambiant Click n'est pas livré avec votre starter-kit

Application N° 032 Utilisation du module Button R Click MIKROE-1901



Le module « Button R Click » intègre un bouton-poussoir avec un capuchon transparent dans lequel se trouve une Led.

L'application décrite ci-dessous va vous permettre d'apprendre à utiliser une interruption. En apparence, un programme très simple.. quoi que...

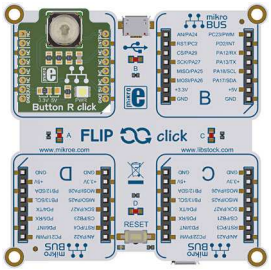
Déconnectez la platine « Flip & Click » de votre ordinateur, insérez le module «Button R click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur. Puis saisissez le programme ci-contre et téléversez-le dans la platine.

On retrouve les habituelles phases d'initialisation et de déclaration dans lesquelles on va définir avec quels ports de la platine on va piloter les leds ou lire l'état du bouton-poussoir. Si on reprend la description de la fonctionnalité de notre programme, celui-ci va consister à réaliser une boucle dans laquelle on va allumer tour à tour une des 4 leds de la platine (en éteignant les autres) avec une temporisation entre chaque transition. Le problème est que nous avons également décidé de modifier l'état de la led interne du bouton-poussoir à chaque fois qu'on le sollicite.

A bien y réfléchir, le programme supposé initialement simple va devoir se compliquer puisqu'il faudrait « parsemer » le programme avec de nombreuses instructions **if (digitalRead( ))** pour tester à plusieurs reprises l'état du bouton-poussoir (y compris pendant la phase de temporisation entre les transitions des leds). Pas de panique, nous allons faire appel aux interruptions ! Les interruptions permettent à votre platine « Flip & Click » de surveiller « automatiquement » (en tâche de fond) l'état de certaines de ses entrées, alors même que votre programme est en train de se dérouler. Pour ce faire, nous allons utiliser l'instruction **attachInterrupt( );** En cas de sollicitation de cette entrée, le programme va s'interrompre (depuis n'importe quel endroit où il se trouve) pour aller réaliser une action bien précise que vous aurez programmé dans une fonction.

```
const byte LedBP = 6;      // Association Led au port 6
const byte BP = 26;        // Attribution BP au port 26
volatile byte Etat = 0;    // Etat initial de la variable Etat
const byte Led[4] = {38,37,39,40};  // Gestion affichage leds
byte j = 0;      // Variable utilisée pour le défilement des leds

void setup()
{
 for (byte i = 37 ; i < 41; i++) // Initialisation ports en sortie
 {
 pinMode(i, OUTPUT);        // Attribution ports en sortie
 digitalWrite(i, HIGH);     // Eteint les 4 leds A / B / C / D
 }

 pinMode(LedBP, OUTPUT);    // Port led en sortie
 digitalWrite(LedBP, Etat);      // Eteint la led du BP
 pinMode(BP, INPUT);        // Configure port BP en entrée
 attachInterrupt(digitalPinToInterrupt(BP), Inverse,
 RISING); }
// Configure l'entrée du BP en interruption réagissant sur
   un front montant et générant l'appel de la fonction
   "Inverse"

void loop()
{
 for (byte i = 0 ; i < 4; i++) // Affectation ordre leds à allumer
 {
   if (j == i)    // Sélectionne  les leds à allumer et à éteindre
     {
     digitalWrite(Led[i], HIGH);        // Allume la led
     }
   else
     {
     digitalWrite(Led[i], LOW);         // Eteint la led
     }
 }
 digitalWrite(LedBP, Etat);    // Allume ou éteint la led
 j++;              // Sélectionne la prochaine led à allumer
 if (j > 3) j=0;  // Si arrivé dernière led -> Retourne à la 1ère
 delay(250);                    // Temporisation de 1 seconde
}

void Inverse() {Etat = !Etat; }    // Inverse variable Etat
```

Le premier paramètre de cette instruction doit définir la broche que l'on va utiliser en interruption (ici il s'agira de celle reliée au bouton-poussoir). Le second paramètre doit correspondre au nom de la fonction qui devra être appelée lors de la détection de l'interruption sur cette touche (il s'agira ici du petit bout de programme présent dans notre fonction **Inverse** qui consiste simplement à inverser l'état d'une variable). Le dernier paramètre consiste à définir comment doit se comporter la détection sur l'entrée d'interruption.



Le module Button R Click n'est pas livré avec votre starter-kit

Il est possible de déclencher l'interruption sur la détection:

- D'un niveau bas.
- D'un changement d'état.
- De l'apparition d'un front descendant du signal.
- De l'apparition d'un front montant du signal.

Dans notre cas, l'interruption sera générée suite à la détection d'un front montant sur l'entrée de la platine sur laquelle est reliée le bouton-poussoir du module « Button R Click ».

Note importante:
La fonction liée à une interruption doit être la plus courte possible et ne pas comporter d'instructions complexes, ni d'appel à d'autres fonctions. Il faut en effet garder à l'esprit que lorsqu'une interruption est générée, le déroulement de votre programme principal est « stoppé » pour exécuter la fonction de l'interruption. Celle-ci ne devra donc pas perturber le fonctionnement de votre application.

Revenons à la description du programme principal. Celui-ci va consister à réaliser une boucle dans laquelle on va éteindre ou allumer les différentes leds grâce à une variable de type tableau et à la variable **J** que l'on incrémente après chaque boucle pour pouvoir déterminer quelle sera l'unique led à allumer.

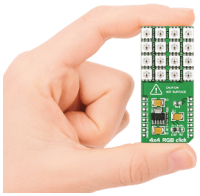
Au terme de cette boucle on « envoie » la valeur de la variable **Etat** sur la led du bouton-poussoir puis on réalise une légère temporisation avant de recommencer à nouveau le programme principal depuis le début.

Comme vous pouvez le constater, à aucun autre endroit du programme on vient tester l'état du bouton-poussoir. Pourtant sa sollicitation sera bien prise en compte à tout moment.

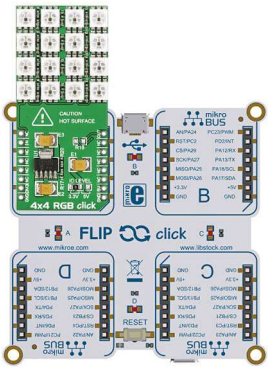
Notes

(Emplacement réservé pour indiquer vos annotaions personnelles)

Application N° 033 Utilisation du module 4x4 RGB Click MIKROE-1881



Le module « 4x4 RGB Click » est une matrice composée de 4 x 4 leds RGB (lesquelles intègrent un contrôleur WS2812). Les leds peuvent être pilotées indépendamment en couleur et en intensité avec un seul port de votre platine »Flip & Click ». La communication avec les leds nécessite un dialogue série au timing très rigoureux. Afin de vous simplifier la « vie », il existe bien sûr une excellente librairie qui s'occupera de toute la partie communication.



Le module 4x4 RGB Click n'est pas livré avec votre starter-kit

Pour utiliser cette librairie, il vous faut quitter complètement l'environnement IDE, puis télécharger la librairie à partir de cette adresse :

https://github.com/adafruit/Adafruit_NeoPixel
(via le bouton « Download ZIP)

Décompressez alors le fichier (qui se présente sous la forme d'un dossier) dans le répertoire C:\Program Files (x86)\Arduino\libraries (si bien sûr vous avez installé l'environnement IDE dans le dossier C:\Program Files (x86).

A ce stade, renommez le nom du dossier en : **Adafruit_NeoPixel** Déconnectez la platine « Flip & Click » de votre ordinateur et insérez le module «4x4 RGB click » sur le support **A** (attention au sens). Puis lancez à nouveau l'éditeur IDE.

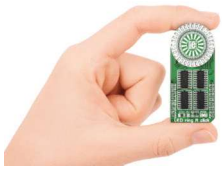
Maintenant, chargez un exemple pour ce module à partir du menu : Fichier → Exemples → Adafruit Neopixel → Simple

Modifiez ensuite la ligne **#define PIN 6** par **#define PIN 33**
(afin d'indiquer au programme la broche utilisée par la platine « Flip & Click » pour piloter les leds.

Téléversez le programme et vérifiez que les leds s'allument avec la couleur verte. Il vous sera facilement possible de modifier les couleurs en modifiant les valeurs comprises dans l'instruction **pixels.setPixelColor ( )**.

Notes importantes :
Bien que d'usage très ludique, les personnes utilisant le module « 4x4 Click Board » doivent être expérimentées et capables de maîtriser complètement leur programme. Les leds présentes sur le module «4x4 Click Board » peuvent dégager beaucoup de chaleur en fonction des couleurs que vous allez sélectionner. Ceci peut rendre la surface du module très chaude. Ne touchez JAMAIS celle-ci avec les doigts. Attendez également plusieurs minutes après avoir déconnecté la platine de son alimentation avant de toucher celle-ci. Plus les couleurs choisies sont claires et vives, plus les leds vont consommer et chauffer (dans le pire des cas, les leds peuvent amener à se détruire si vous maintenez des couleurs vives et claires trop longtemps à l'affichage). Ceci peut également créer une sur-consommation et endommager le port USB de votre ordinateur. Il est également important d'indiquer que certaines couleurs générées par les leds peuvent être très vives. **Il est donc IMPERATIF de ne JAMAIS regarder les leds de près, ni de face et de tenir le module éloigné le plus possible de toutes personnes (enfants et adultes) ou animal.**

Application N° 034 Utilisation du module LED ring R click MIKROE-2153



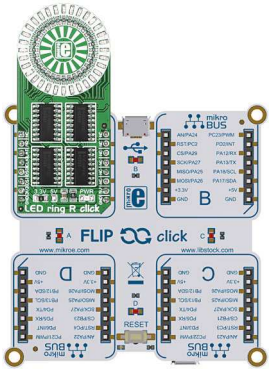
Le module « LED Ring R Click » intègre 32 leds rouges montées sous la forme d'un cercle.

Ces leds sont pilotées par l'intermédiaire de 4 circuits intégrés de type 74HC595.

C'est le même type de circuit intégré qui équipe plusieurs modules déjà étudiés dans cet ouvrage (module 7X10 R Click, module 7seg Click, module Bargraph Click).

Déconnectez la platine « Flip & Click » de votre ordinateur, puis insérez le module «LED ring R click » sur le support **A** (attention au sens). Connectez ensuite la platine « Flip & Click » à votre ordinateur.

Saisissez enfin le programme ci-contre.



Le module LED Ring R Click n'est pas livré avec votre starter-kit

```
#include <SPI.h>                        // Intégration de la librairie SPI
#define CS0 77                          // Attribution du signal CS0 au port 77
byte Segment1=128, Segment2=0;          // Variables pour la gestion des segments 1 et 2
byte Segment3=0, Segment4=0;            // Variables pour la gestion des segments 3 et 4
int Rotation,Compteur=0;                // Variables gérant la rotation de la roulette
int Temporisation = 15;                 // Variable gérant la vitesse de la roulette

void setup()
{
  pinMode(CS0, OUTPUT);                 // La broche CS0 est une sortie
  SPI.begin();                          // Initialisation du port SPI
  randomSeed(analogRead(0));            // Réinitialise la séquence aléatoire
}

void loop()
{
Rotation=random (160,193);      // Choisit position aléatoire où la roulette doit stopper
do
{
   digitalWrite(CS0, LOW);                  // Sélectionne broche CSO
   SPI.transfer(Segment1);                  // Envoie les données du 1 er segment
   SPI.transfer(Segment2);                  // Envoie les données du 2 ème segment
   SPI.transfer(Segment3);                  // Envoie les données du 3 ème segment
   SPI.transfer(Segment4);                  // Envoie les données du 4 ème segment
   digitalWrite(CS0, HIGH);                 // Désélectionne la broche CS0
   Segment1=Segment1>>1;                    // Décalage des bits du segment 1
   Segment2=Segment2>>1;                    // Décalage des bits du segment 2
   Segment3=Segment3>>1;                    // Décalage des bits du segment 3
   Segment4=Segment4>>1;                    // Décalage des bits du segment 4
   delay(Temporisation);     // Temporisation pour gestion de la vitesse de la roulette
   Temporisation++;          // Augmente durée de la temporisation (la roulette ralentie)
   Compteur++; Rotation--;                  // Incrémente nombre de pas de la roulette
   if (Compteur==8)  {Segment4=128;}        // Teste passage fin du 1er segment
   if (Compteur==16) {Segment3=128;}        // Teste passage fin du 2eme segment
   if (Compteur==24) {Segment2=128;}        // Teste passage fin du 3ème segment
   if (Compteur==32) {Segment1=128;Compteur=0;}  // Teste fin 4ème segment
  }
  while (Rotation > 0);
  while(1){}                                // La roulette s'est arrêtée -> Boucle sans fin
}
```

Description et explications

Cette application vous permettra de réaliser une superbe petite roulette électronique (similaire à celle de l'application N° 5). A l'exécution du programme, la roulette va s'élancer rapidement (ceci est matérialisé par l'allumage successif des différentes leds avec une rotation dans le sens des aiguilles d'une montre), puis ralentir petit à petit avant de stopper aléatoirement sur une des 32 leds. En sollicitant le bouton-poussoir Reset, la roulette sera de nouveau lancée. Une fois encore, cette application fait appel à beaucoup de notions déja vues et normalement déja assimilées au cours des différents autres montages.

Dans la partie déclaration du programme, on fait le nécessaire pour utiliser un port SPI dans le but de pouvoir piloter les 4 circuits intégrés 74HC595. On a également recours à différentes variables **Segment1 Segment2 Segment3 Segment4** lesquelles nous permettront de stocker l'état de chacune des leds de la roulette. En fait chacun des 4 circuits intégrés 74HC595 pilote un arc de cercle de leds (les 4 arcs de cercle formant ainsi le cercle complet). Au passage on initialise toutes les variables à zéro (pour indiquer que toutes les leds seront effacées... sauf la première led de la variable **Segment1** à qui on attribue la valeur 128... qui en mode binaire signifie que seul le bit de poids fort de la variable sera à 1 ... et donc que seule la première led du premier arc de cercle sera allumée).

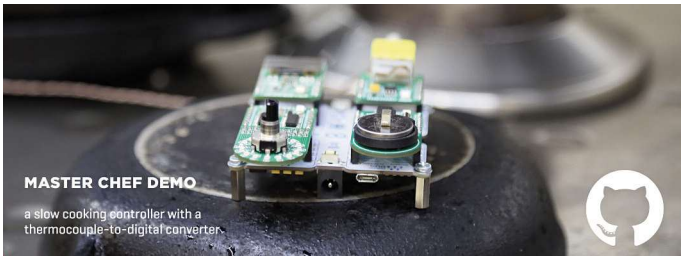
On déclare également les variables **Rotation** et **Compteur** qui nous seront utiles dans la gestion de la rotation des leds de la roulette ainsi que la variable **Temporisation** qui nous permettra de gérer la vitesse de rotation de la roulette.

Dans la partie initialisation, on définit les ports qui seront utilisés pour la communication SPI et on initialise également la séquence aléatoire. Le programme principal commence par choisir un nombre aléatoire compris entre 160 et 192, lequel sera stocké dans la variable **Rotation**. La suite du programme entre dans une boucle **do { } .. while** conditionnée par l'état de la variable **Rotation** (la boucle sera effectuée tant que la variable **Rotation** sera supérieure à zéro). Dans cette boucle, le programme sélectionne les circuits intégrés 74HC595, puis leur envoi successivement les données des 4 segments, puis désélectionne les 74HC595 afin d'afficher l'état des leds de la roulette.

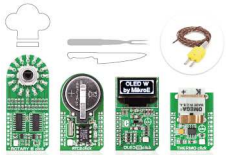
A ce stade, on effectue une rotation des bits de chacune des variables **Segment1 Segment2 Segment3 Segment4** afin d'obtenir le déplacement des leds sur le cercle. Le programme effectue alors une pause (conditionnée par la valeur de la variable **Temporisation**). Cette variable est alors incrémentée afin que la durée de la pause soit de plus en plus longue (et ainsi obtenir l'effet de ralentissement progressif de la roulette). La variable **Compteur** est elle aussi incrémentée (afin qu'on puisse savoir combien il y a eu de décalages et donc savoir où se trouve la led). La variable **Rotation** sera décrémentée afin de déterminer le moment où la roulette devra stopper. Pour rappel, la variable **Rotation** a été initialisée avec une valeur aléatoire comprise entre 160 et 192, ce qui veut dire que la roulette effectuera dans tous les cas une rotation de 5 tours au minimum et que la led de la roulette stoppera alors aléatoirement sur une des 32 positions après le 5 ème tour. Par exemple, si la variable **Rotation** avait pour valeur initiale 180, la roulette fera 5 tours puis elle continuera à rouler jusqu'à la 20ème led (180 - 160 = 20). La valeur 160 est déduite du calcul (5 tours x 32 leds = 160).

S'en suit alors 4 tests **if ()** permettant de détecter le passage des leds d'un segments à l'autre. Ainsi lorsqu'on aura effectué 8 décalages, on considèrera que la dernière led du premier segment ne sera plus allumée et qu'il faudra alors allumer la première led du segment suivant. C'est ce qui est fait via l'instruction **if (Compteur==8) {Segment4=128;}** (notez par contre que les segments ne sont pas rangés dans l'ordre chronologique pour représenter le cercle. En fait la rotation s'effectue du segment 1, vers le segment 4, puis 3 et enfin le 2. Dans le dernier test, on remet la variable **Compteur** à zéro une fois qu'on a détecté que la roulette a fait un tour complet (une fois qu'il y a eu 32 décalages). Lorsque la variable **Rotation** arrive à zéro, le programme entre alors dans une boucle sans fin **while { }** vous obligeant à utiliser le bouton-poussoir de Reset pour relancer la roulette.

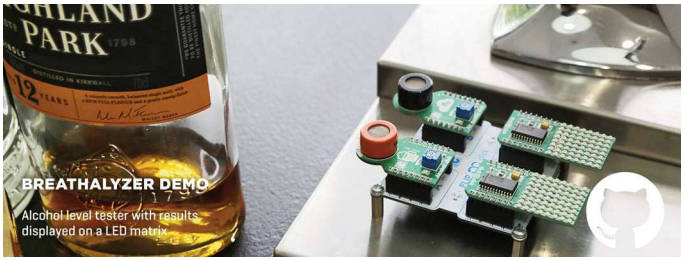
Applications N° 035 / 036 / 037



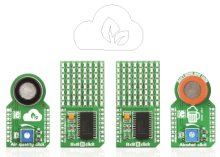
Master Chef Démo



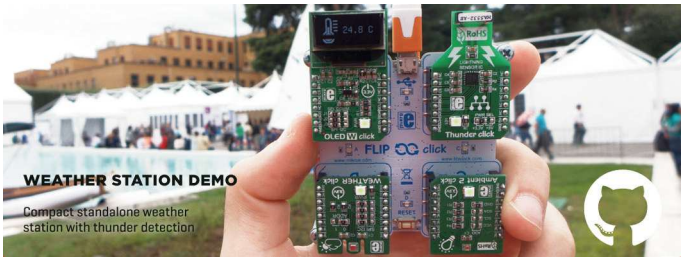
Cette application met en oeuvre les modules « THERMO click », « OLED W click », « RTC2 click » et « Rotary R Click ». Elle vous permettra de réaliser un système d'affichage de température et de temporisation avec réglage par encodeur rotatif et visualisation sur un écran Oled graphique. Avec cette application, vous ne pourrez plus rater la cuisson de vos petits plats !



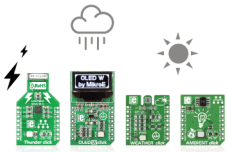
Breathalyzer Démo



Cette application met en oeuvre 2 modules « 8 x 8 clicks » ainsi que les modules « Alcohol click» et « Air Quality click ». Elle vous permettra de réaliser un testeur **expérimental** permettant de donner une information sur votre niveau d'alcool avec une visualisation sur des matrices à leds. Cette application est bien évidemment uniquement destinée pour un usage ludique et expérimental.



Weather Station Démo

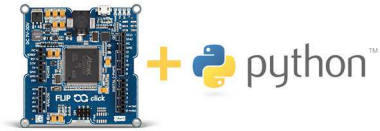


Cette application met en oeuvre les modules « Ambient light click », « Thunder click », « OLED W click », et « Weather click ». Elle vous permettra de réaliser une petite station météorologique avec affichage sur écran graphique Oled. Nec plus ultra, cette dernière intègre également un dispositif capable de détecter les éclairs (via le module « Thunder click »).

Les codes sources de ces démos sont disponibles ici: **https://github.com/MikroElektronika/Flip_n_Click_Examples**

Les modules présentés dans ces applications ne sont pas livrés de base avec votre starter-kit. Il vous est toutefois possible de les acheter au détail sur notre site Internet: **www.lextronic.fr**

Utilisation de la platine Flip & Click avec Python™



Cette page s'adresse aux utilisateurs expérimentés ayant déjà des notions techniques avancées et de solides compétences en programmation.

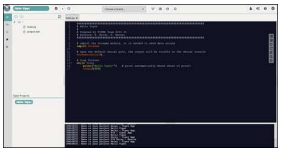
Lorsque vous développez sur un microcontrôleur avec des ressources limitées, le recours à un système d'exploitation est si loin de la réalité qu'il n'est quasiment jamais envisagé. Dès lors, en schématisant quelque peu, votre application se résume généralement en une boucle principale dans laquelle s'effectue une seule tâche.

Avec la platine « Flip & Click » ce n'est pas la puissance qu'il vous manque. Aussi, l'utilisation de son microcontrôleur cadencé à 80 MHz, de ses 512 Ko de mémoire flash et de ses 100 Ko de mémoire SRAM pour s'amuser à faire clignoter quelques leds peuvent s'apparenter quelquefois à du « gâchis ».

Il est donc légitime de vouloir envisager de l'exploiter pour des applications beaucoup plus complexes et d'avoir recours à un système d'exploitation ou à une application lui permettant de pouvoir gérer en symbiose les cycles de son MCU, sa mémoire et les différents traitements ... ou plus simplement de pouvoir la programmer en Python !

Pourquoi choisir le langage Python ?

- Python est un langage de programmation objet de type multi-thread (si vous souhaitez que plusieurs tâches s'exécutent en même temps en lagage C, vous aurez besoin d'un RTOS)
- Python est très facile à apprendre
- Python dispose d'une gestion de la mémoire intégrée
- Python peut accéder aux fichiers C



Comment utiliser la platine « Flip & Click » avec Python™ ?

1) Procurez-vous et exécutez « Zerynth Studio »
2) Connectez la platine « Flip & Click » à votre ordinateur
3) Sélectionnez la platine « Flip & Click » de MikroElektronika parmi les cartes proposées
4) Cliquez sur « install virtual machine »
5) Vous pouvez commencer à écrire des scripts en Python !

Consultez ces liens pour plus d'informations et pour tester une application multi-tâche à l'aide de la platine « Flip & Click »:

http://learn.mikroe.com/running-python-on-a-flip-n-click/

http://learn.mikroe.com/multi-threading-flip-n-click/

Informations et recommandations

Ce symbole présent sur les modules de votre starter-kit indique que vous ne pouvez pas vous débarrasser de ces derniers de la même façon que vos déchets courants. Au contraire, vous êtes responsable de l'évacuation de ces derniers lorsqu'ils arrivent en fin de vie et à cet effet, vous êtes tenus de les remettre à un point de collecte agréé pour le recyclage des équipements électriques et électroniques usagés. Le tri, l'évacuation et le recyclage séparés de vos équipements usagés permettent de préserver les ressources naturelles et de s'assurer que ces équipements sont recyclés dans le respect de la santé humaine et de l'environnement. Pour plus d'informations sur les lieux de collecte des équipements électroniques usagés, contactez votre mairie ou votre service local de traitement des déchets.

Ce starter-kit ne peut être utilisé que par des personnes ayant des compétences en électronique et en programmation informatique. Si vous ne disposez pas de l'ensemble de ces compétences et que vous ne pouvez pas vous faire assister par une personne maîtrisant toutes ces compétences, n'utilisez pas ce starter-kit et retournez nous celui-ci afin qu'il vous soit remboursé. Ce starter-kit est exclusivement destiné aux personnes de plus de 14 ans. Veuillez le tenir éloigné des enfants en bas âges qui pourraient inhaler ou avaler les petits objets qui le compose.

Ce manuel a été conçu avec la plus grande attention. Tous les efforts ont été mis en œuvre pour éviter les anomalies. Toutefois, nous ne pouvons garantir que ce dernier soit à 100% exempt de toute erreur.

Les listings des notes d'applications de ce manuel ainsi que les fichiers des listings disponibles en téléchargement sont proposés exclusivement à titre indicatif (nous ne pouvons garantir leur entière fonctionnalité - considérez ces listings comme des versions bêta, susceptibles d'évolutions et d'améliorations). Certaines librairies utilisées dans les listings sont sous licence publique générale GNU (merci de consulter leurs conditions d'utilisations).

Toutes les applications décrites dans ce manuel et tous les produits livrés dans ce starter-kit sont exclusivement destinés à un usage ludique et pédagogique dans le cadre d'une aide à l'apprentissage à la programmation. Les applications décrites dans ce manuel ainsi que ses exemples de programmes et tous les produits livrés dans ce starter-kit ne sont en aucun cas prévus, ni conçus pour être utilisés au sein d'applications réelles.

Les éléments de ce starter-kit ainsi que les exemples de programmes et les exemples d'applications qu'il renferme ne sont pas conçus, ni prévus, ni autorisés pour être exploités au sein d'applications médicales, ni d'applications militaires, ni d'applications embarquées dans des véhicules (de quelques natures soient-ils: voitures, camions, trains, avions, bateaux, modèles réduits, etc...), ni d'applications dans les aéroports, ni d'applications d'alarme anti-intrusion, ni d'alerte incendie, ni d'applications sur ascenseur, ni d'applications sur des sites nucléaires, ni d'applications en milieux explosifs ou inflammables, ni d'applications visant à automatiser le déplacement d'engins mobiles, ni d'applications dans lesquelles une défaillance des éléments de ce starter-kit pourrait créer une situation dangereuse susceptible d'entraîner des dégâts matériels, une perte financière, des blessures ou la mort de personnes ou d'animaux (LEXTRONIC décline toute responsabilité en cas de non respect de ces conditions d'utilisation).

Toutes les marques et appelations commerciales citées dans ce manuel sont la propriété de leurs détenteurs respectifs.

Lisez ce manuel avec attention avant toute utilisation du starter-kit et mettez ce dernier à disposition des utilisateurs désirant exploiter le starter-kit. Ne jetez pas ce manuel sur la voie publique.