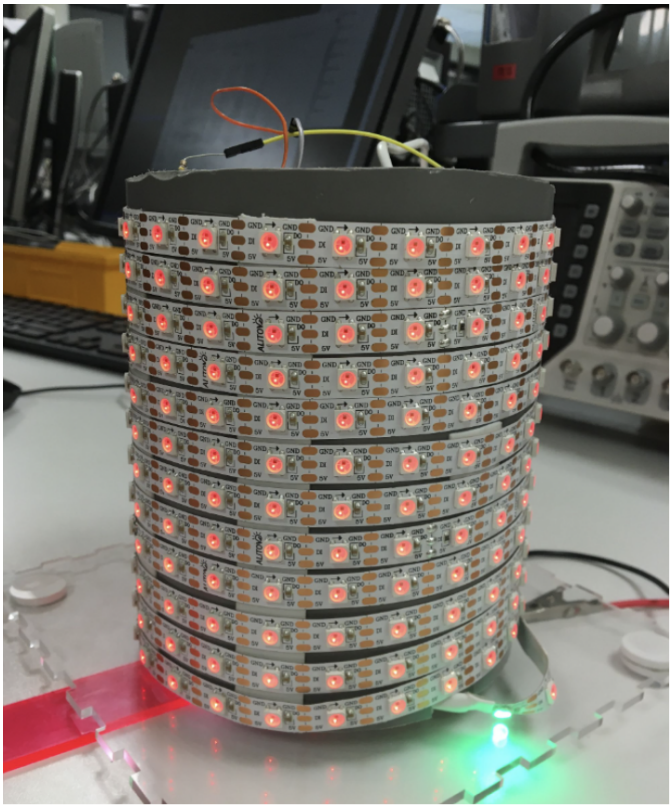
Rapport Technique: Projet Caméléon



AMARI Tanouir COUGET Mathis PAGES Marie

INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental

Contents

1 Introduction

L'objectif de ce projet est de produire une source de lumière qui s'éclaire de la couleur de son support telle un caméléon. L'idée de départ était ainsi de pouvoir éclairer une partie d'une pièce sombre de n'importe quelle couleur en déposant la lampe portative sur des supports variés ou simplement en plaçant dessous des échantillons de couleurs. Cette partie introductive a pour objectif de montrer l'organisation du projet de l'imagination de ses caractéristiques jusqu'à l'identification des différents jalons à atteindre.

1.1 Cahier des charges

Remarque: Un premier cahier des charges détaillait nos attentes concernant une boîte portative miniaturisée d'environ 20 cm de côté, que nous voulions totalement autonome grâce à une batterie. Cependant pour un nombre d'environ 300 leds de sorte à avoir un éclairement raisonnable, nous avons rapidement réalisé que pour un éclairage blanc (à savoir l'intensité maximale pour chacune des trois leds rouge, vert, bleu constitutives d'une led du bandeau) il faut avoir une alimentation qui dépasse ce que les batteries disponibles pouvaient offrir. Ainsi, nous nous sommes rabattus sur une lampe branchée sur alimentation externe, dont le cahier des charges est le suivant :

Objectifs	Contraintes
Obtenir une couleur d'éclairement identique à celle du support	Evaluation qualitative par l'oeil
Interfacer facilement avec l'utilisateur	Interrupteur
Eclairage intense	La diffusion doit permettre d'être ressentie dans une pièce telle qu'une petite chambre
Miniaturiser le système, et le rendre ergonomique	Lampe cubique d'environ 20 cm de côté, le poids n'excède pas 2kg, certaine esthétique
Fonctionner dans le noir	Mise en place d'un éclairage de la surface (nécessaire à la détection) intégré à la lampe

Table 1: Cahier des charges de la lampe Caméléon

1.2 Description fonctionnelle

Le composant clé de notre projet est un capteur de couleur (Color 10 Click). Ainsi l'enjeu est de pouvoir communiquer avec ce composant pour récupérer d'une part la couleur qu'il lit, et d'autre part de la fournir au bandeau de leds afin qu'il s'éclaire de la bonne couleur. Pour permettre ce dialogue, on utilise le protocole de communication I2C (dont nous expliquerons le principe) entre la nucléo maître et le bandeau de leds esclave. La description fonctionnelle est détaillée en Figure 1.

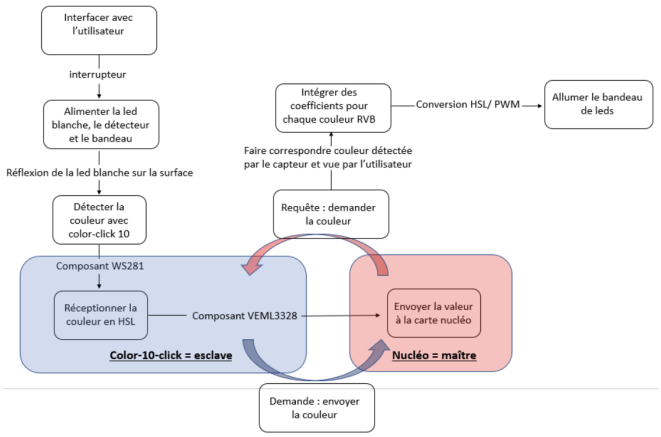


Figure 1: Description fonctionnelle de la lampe Caméléon

Ainsi cette description fonctionnelle laisse apparaître les grandes parties du projet :

- La communication avec l'utilisateur via un interrupteur.
- L'apport de l'information au capteur via une led blanche dont la lumière se réfléchit sur la surface considérée et atteint le capteur avec la couleur sondée.
- La communication avec le capteur via une fonction qui lit la couleur détectée.
- La communication avec le bandeau via une fonction qui contrôle la couleur de toutes les leds du bandeau.
- La communication entre le capteur et le bandeau via une conversion des langages de couleur de chacun.
- La conception technique de l'objet final avec découpe laser, soudure...

2 Elements du projet

2.1 Capteur: Color_10_Click

Le Color_10_Click est équipé d'un capteur de lumière RGB et IR appelé VEML3328. Les fonctionnalitées de ce capteur sont facilement exploitable grâce au protocole I2C. Celui ci à une plage de tension de fonctionnement de 2,6V à 3,6V (adapté à notre Nucléo de 3,3V).

Figure 2: Color_10_Click

Ce capteur utilise un filtre de couleur RGB (rouge, vert, bleu) pour détecter les composantes de couleur dans l'environnement. Il peut mesurer la quantité de lumière émise par chaque composante de couleur et fournir des données précises sur les niveaux de couleur rouge, vert et bleu présents.

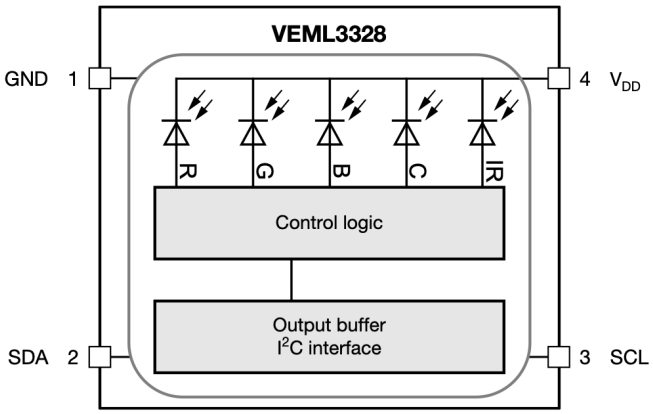


Figure 3: Diagramme de bloc du capteur VEML3328

Nous joignons le tableau récapitulatifs des différents pin qui seront plus détaillées dans la section dédié au protocole I2C.

Pin	Symbole	Type	Fonction
1	GND	-	Masse de l'alimentation électrique ; toutes les tensions sont référencées aux GND
2	SDA	I/O	Entrée/sortie de données du bus numérique I2C
3	SCL	I	Entrée de l'horloge du bus numérique I2C
4	VDD	-	Tension d'alimentation

Table 2: Attributions des pins

Ensuite, à travers les caractéristique ci-dessous, nous pouvons voir que le capteur est plus sensible au rouge et au vert qu'au bleu, cela à un impact sur notre rendu final de couleur, nous verrons dans lors de la présentation des fonctions comment nous avons essayer de limiter cet effet. De plus, nous avons fait en sorte que l'objet éclairé par la led blanche soit à 0° par rapport au capteur afin d'avoir une réponse la plus optimale à température ambiante.

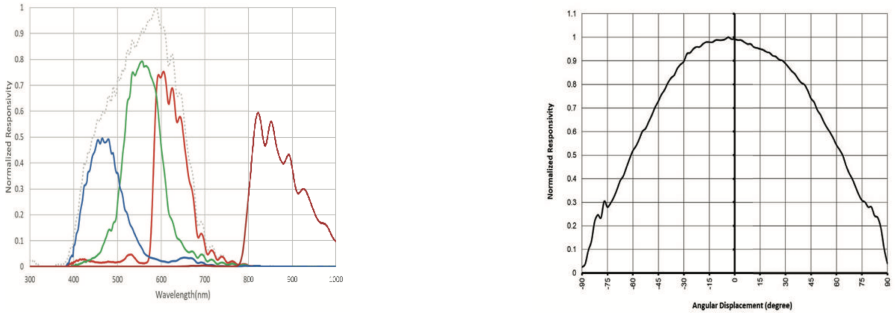


Figure 4: Réponse Normaliser en fonction de la longueur d'onde et du déplacement angulaire

2.2 Bandeau de Led: WS2812b

Le bandeau de LED WS2812B est un produit populaire et largement utilisé dans le domaine de l'éclairage LED programmable. Il est constitué de plusieurs LED individuelles intégrées dans un seul boîtier, permettant ainsi de créer des effets d'éclairage dynamiques et colorés.

Figure 5: Bandeau de Led WS2812b

Voici quelques caractéristiques associées à ce bandeau:

- LEDs intégrées : Chaque LED WS2812B est composée de trois éléments de couleur : rouge, vert et bleu (RGB). Ces LED sont regroupées dans un seul boîtier, ce qui permet un contrôle précis de chaque LED individuelle.
- Communication en série : Le contrôle des LEDs WS2812B s'effectue via une communication

en série, où les données sont transmises d'une LED à l'autre. Cela signifie que l'on peut chaîner plusieurs bandes de LED WS2812B pour créer des affichages lumineux étendus.

- Contrôle individuel des LEDs : Une des caractéristiques les plus intéressantes du WS2812B est la possibilité de contrôler chaque LED individuellement. Cela signifie que l'on peut ajuster la couleur, l'intensité lumineuse et les effets d'éclairage de chaque LED sur le bandeau.
- Programmable : Les LEDs WS2812B sont programmables, ce qui signifie que l'on peut créer des animations et des effets d'éclairage personnalisés en utilisant notre carte Nucléo.

2.3 Nucléo L476RG

La Nucleo L476RG est une carte de développement basée sur le microcontrôleur STM32L476RG de STMicroelectronics. Le STM32L476RG est un microcontrôleur puissant de la famille STM32 basée sur l'architecture ARM Cortex-M4.

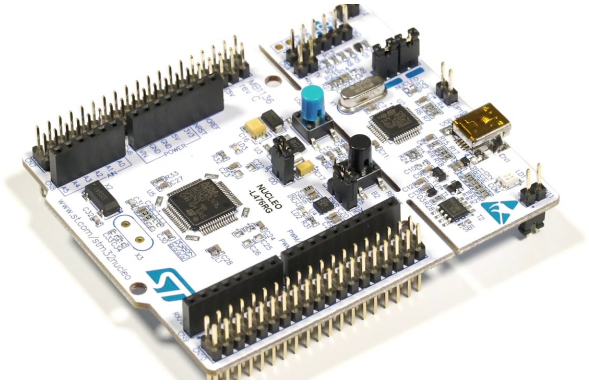


Figure 6: Nucléo L476RG

Dans le protocole I2C, en tant que maître, la carte Nucleo L476RG initie et contrôle la communication I2C. Elle génère les signaux de synchronisation et d'horloge nécessaires à la transmission des données. La carte Nucleo L476RG envoie des commandes et des requêtes à l'esclave (capteur COLOR_10_CLICK) et reçoit les réponses ou les données demandées. (Tout ceci est détaillé plus tard)

3 Communication entre l'utilisateur et le bandeau de led WS2812

3.1 Protocole I2C

Pour faire communiquer la Nucléo avec le bandeau de Led WS2812, nous avons utilisé le protocole I2C.

Le protocole I2C (Inter-Integrated Circuit) est un protocole de communication série largement utilisé pour interconnecter des composants électroniques sur une carte de circuit imprimé ou entre plusieurs appareils. Le capteur Color_10_Click peut également utiliser ce protocole pour communiquer avec d'autres dispositifs.

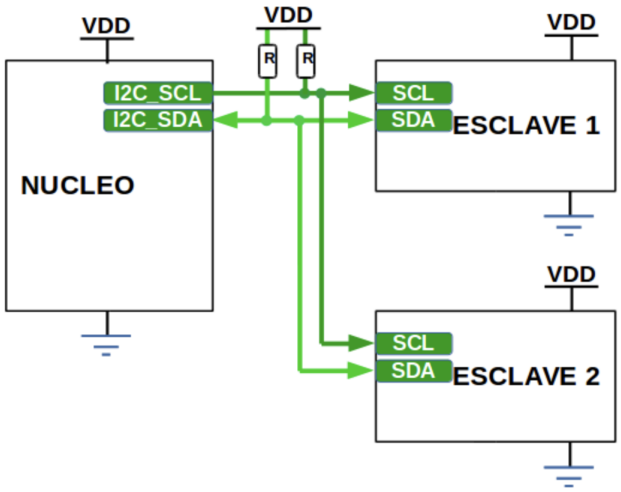


Figure 7: Liaison maître-escalve

Le protocole I2C est basé sur une architecture maître-esclave, où un seul dispositif maître contrôle la communication avec plusieurs périphériques esclaves. Dans le cas du Color_10_Click, il agirait généralement en tant que périphérique esclave, tandis que la plateforme de développement, Nucléo ici jouerait le rôle de maître.

Voici comment celui-ci fonctionne:

- Ligne de données (SDA) : Cette ligne est utilisée pour transférer les données entre les périphériques maître et esclave. Le maître envoie les commandes et les données au périphérique esclave via cette ligne.
- Ligne d'horloge (SCL) : Cette ligne est utilisée pour synchroniser les transferts de données entre le maître et l'esclave. Le signal d'horloge est généré par le maître et est utilisé pour contrôler le moment où les données sont lues ou écrites.
- Adresse du périphérique : Chaque périphérique I2C a une adresse unique qui lui est attribuée. Le maître utilise cette adresse pour sélectionner le périphérique avec lequel il souhaite communiquer.

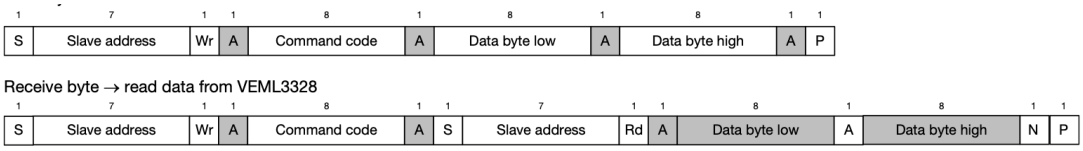


Figure 8: Format du protocole de commande

Légende:

S: Début; P: Fin; A: Message reçu; N: Message non reçu

4 Présentation des fonctions

4.1 Fonction du capteur

Nous avons utilisé 5 fonctions:

powerUp() : allume le capteur et l'initialise en mode automatique

```
1       void Color_10_Click::powerUp(void){
2           cmd[0] = COLOR_10_CLICK_COMMAND;
3           cmd[1] = 0;
4           cmd[2] = 0;
5           ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 3);
6           if(DEBUG_MODE) printf("Init Acq = %d\r\n", ack1);
7           wait_us(1000);
8   }
```

Cette fonction allume le capteur et l'initialise en mode automatique.

getPartID() et getCommandValue() :

```
1       int Color_10_Click::getPartID(void){
2           cmd[0] = COLOR_10_CLICK_PART_ID;
3           ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
4           ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
5           if(DEBUG_MODE)   printf("Part ID Acq (W) = %d\r\n", ack1);
6           if(DEBUG_MODE)   printf("Part ID Acq (R) = %d\r\n", ack2);
7           return data[0];
8   }
```

```
1       int Color_10_Click::getCommandValue(void){
2           cmd[0] = COLOR_10_CLICK_COMMAND;
3           ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
4           ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
5           if(DEBUG_MODE)   printf("Command Value Acq (W) = %d\r\n", ack1);
6           if(DEBUG_MODE)   printf("Command Value Acq (R) = %d\r\n", ack2);
7           return (data[1] << 8) + data[0];
8   }
```

Ces fonctions servent a collecter les informations lues par le capteur à l'aide du protocole I2C. La première renvoi l'ID de l'esclave (elle vaut par défaut $0x28$ en hexadécimal) la deuxième sert à ordonner l'esclave de lire les valeurs de couleurs RVB, IR et l'intensité lumineuse de la source qui l'éclaire à l'aide de la variable locale cmd qui vaut ici $0x10$.

setGainRGB() :

```
1       void Color_10_Click::setGainRGB(int val){
2           int command_value = getCommandValue();
3           cmd[0] = COLOR_10_CLICK_COMMAND;
4           cmd[1] = (command_value & 0xFF);
5           cmd[2] = ((command_value >> 8) & 0b11110011) | (val << 2);
6           ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 3);
```

```
7           if(DEBUG_MODE)   printf("Gain Acq (W) = %d\r\n", ack1);
8   }
```

Cette fonction sert à appliquer un gain qui est de base 3x à 2x (ici un gain 0b01 ce qui correspond à 1 en décimal)

readRGBCIRValue() :

```
1       void Color_10_Click::readRGBCIRValue(int rgbcIR[]){
2           rgbcIR[0] = readRedValue();
3           rgbcIR[1] = readGreenValue();
4           rgbcIR[2] = readBlueValue();
5           rgbcIR[3] = readClearValue();
6           rgbcIR[4] = readIRValue();
7   }
```

Cette fonction stock les valeurs de rouge, vert, bleu, infrarouge et d'intensité lumineuse lue par le capteur dans un tableau. Ces valeurs sont commandées via le processus I2C par les différentes fonctions read.

Ces fonctions read vont ordonner au capteur de leur renvoyer la valeur de la variable voulu pour ensuite renvoyer un entiers correspondant à cette valeur lue par le capteur.

```
1       int Color_10_Click::readGreenValue(void){
2           cmd[0] = COLOR_10_CLICK_GREEN_CHAN;
3           ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
4           ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
5           if(DEBUG_MODE)   printf("Green Chan Acq (W) = %d\r\n", ack1);
6           if(DEBUG_MODE)   printf("Green Chan Acq (R) = %d\r\n", ack2);
7           Green_color = (data[1] << 8) + data[0];
8           return Green_color;
9   }
```

4.2 Communication

Avant de présenter les fonctions utiliser pour piloter le bandeau nous allons présenter les lignes de codes permettant d'assurer la communication entre le capteur et le bandeau.

```
1           unsigned char R = (RGBCir[0]*256)/65536;
2           unsigned char G = 0.5*(RGBCir[1]*256)/65536;
3           unsigned char B = (RGBCir[2]*256)/65536;
```

Ici on convertie les valeurs de RVB renvoyées par le capteur en 16 bits en valeur unsigned char de 8 bits. de cette façon on pourra utiliser ces valeurs dans les fonctions du WS2812b. Le facteur 0,5 sert à compenser la valeur du vert qui avait un gain pour pouvoir imiter la caractéristique d'absorption de l'oeil humain (cf figure 4). Or cet augmentation de la valeur du vert ne nous arrange pas car elle va modifier la couleur afficher par le bandeau.

```
1           unsigned int color = ((int)(R) << 16) | ((int)(G) << 8) | (int)(B);
```

Cette ligne de code sert à convertir les valeurs de 0-1 à 0-255 et les mettre dans une variable unsigned int, cette étape est également nécessaire pour pouvoir assurer la communication entre le capteur et le bandeau de LED. Le "sens de conversion" est déterminé par l'ordre que l'on a soumis au capteur en I2C.

4.3 Fonction du bandeau de LED

Nous avons eu recours à seulement 2 fonctions pour piloter le bandeau de LED avec les informations issues du capteur.

Set() :

```
1       void PixelArray::Set(int i, unsigned int value){
2           if ((i >= 0) && (i < pbufsize)) {
3               __set_pixel(i,value);
4           }
5       }
```

```
1       void PixelArray::__set_pixel(int index, int value){
2           pbuf[index] = value;
3       }
```

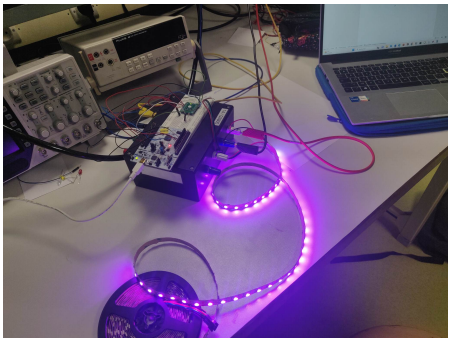
Cette fonction va faire appel à une autre fonction, qui allumera la i-ème LED à la couleur de value. Cela est possible notamment via l'utilisation d'une matrice (pbuf) dont chaque élément correspond à une LED, on fait correspondre chaque valeurs des éléments composants de la matrice à une couleur définie par value. Value et color possède le même type, la communication entre le bandeau de LED et le capteur est assuré.

5 Tests de validations

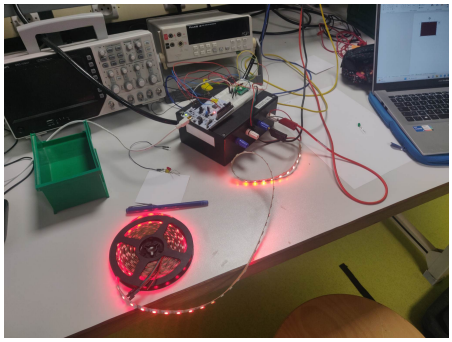
Après avoir fini le code permettant de piloter le bandeau de LED à partir de la couleur détecter par le capteur nous avons rencontrer plusieurs échecs lors de la mise en place du dispositif. Pour localiser la source du problème (mauvais câblage, problème de branchement des composants, pilotage du bandeau/ capteur défaillant) nous avons mis en place plusieurs tests.

5.1 Test du bandeau

Pour vérifier la fiabilité de notre bandeau de LED nous avons utilisés les fonctions SetG, SetR et SetB. C'est trois fonction prennent en argument un entier (qui correspond à la i-ème led du bandeau) et un unsigned char (compris entre 0 et 255, il permet de régler l'intensité de couleur du rouge du vert ou du bleu en fonction des fonctions Set), ces fonctions servent à allumer la i-ème LED du bandeau à une couleur spécifique codé en RVB. Pour vérifier que notre bandeau marche il suffit de rentrer ces trois fonction dans une boucle sur i pour allumer toute les LEDs à une couleurs souhaité.



(a) couleur en rentrant les valeurs R=238, G=130, B=238



(b) couleur en rentrant les valeurs R=255, G=0, B=0

5.2 Test du capteur

Les valeurs lue par le capteur sont les valeur de rouge, de bleu, de vert, d'infrarouge et d'intensité de la source lumineuse. elles sont stockées dans un tableau (RGBCir). Pour savoir si notre capteur était actif ou non nous avons avons simplement eu besoin d'afficher les valeurs stockées sur ce tableau dans TeraTerm à l'aide du code suivant :

```
1  printf("[%4d] R=%d / G=%d / B=%d / C=%d / IR = %d \r\n\n ", k, R, G, B, RGBCir[3],
       RGBCir[4]);
```

Le capteur lisant les couleurs toutes les 0,5s il est alors facile de savoir si il marche en l'exposant à un flash de téléphone (les valeurs R, G, B et C vont considérablement augmenter).

5.3 Test de la communication capteur/bandeau

Pour savoir si la communication marche, il suffit d'exposer une surface de couleur au capteur, si le bandeau de LED s'allume de la même couleur c'est que la communication fonctionne.

6 Résultats

Nous avons réussi à accomplir la communication entre le capteur de couleur et le bandeau de LED. Nous n'avons malheureusement pas eu le temps de réaliser une étude de colorimétrie pour savoir si la couleur émise est fidèle a la couleur lue. Étant donné les conversions appliqués a la valeur lue par le capteur au cours du code pour assurer la communication vers le bandeau (conversion HSL/RVB, conversion de 16 bits (capteur) à 8 Bits (bandeau)). On aurait pu réaliser cette étude en utilisant un nouveau capteur qui lirait la couleur du bandeau de LED et comparer les deux valeurs renvoyer par les deux capteur sur TeraTerm par exemple.

Figure 10: Vidéo Fonctionnement du Caméléon

Néanmoins si il faudrait apporter des modifications au codes pour que l'intensité lumineuse des LEDs soient plus grande.

7 Bibliographie

unsigned char - En langage C, le type de données "unsigned char" est utilisé pour représenter un octet non signé, c'est-à-dire une valeur entière comprise entre 0 et 255, sans prendre en compte le signe. L'octet (8 bits) est la plus petite unité de stockage de données dans la mémoire de l'ordinateur.

codes utilisés - main :

```
1
2  #include "mbed.h"
3  #include "COLOR_10_CLICK.h"
4  #include "PixelArray.h"
5  #include "WS2812.h"
6
7
8  #define      WS2812_BUF  300      \\allume les 300 LEDs
9  PixelArray   px(WS2812_BUF);
10 // For Nucleo F476 : 3, 12, 9, 12
11 WS2812 ws(D7, WS2812_BUF, 3, 12, 9, 12);
12
13
14 #define WAIT_TIME_MS 500
15 DigitalOut led1(LED1);
16
```

```
17 /// Create an I2C interface with specific pins : SDA / SCL
18 I2C                 my_color_i2c(D14, D15); // SDA / SCL
19
20 /// Create a digital output for WS2812 RGB Led
21 DigitalOut          my_led(D9);
22 /// Create a Color_10_Click module connection with an I2C interface and a Digital
       Out for the led
23 Color_10_Click      my_sensor_(&my_color_i2c, &my_led);
24
25 //declaration de la led qui éclaire la surface
26 DigitalOut petite_led(D8,1);
27
28 /// Create a 5 integers array to collect R, G, B, clear and IR data from the
       sensor
29 int             RGBCir[5] = {0};
30
31
32 int main(){
33         //allume la petite led qui sert à éclairer la surface
34         petite_led = 1;
35
36         int k = 0;
37         printf("Test.\n");
38         /// PowerUp and initialize the module in automode
39         my_sensor_.powerUp();
40         /// Collect informations about the sensor
41         printf("Part ID = %d\r\n", my_sensor_.getPartID());
42         printf("Command = %d\r\n", my_sensor_.getCommandValue());
43
44         /// Modify the analog gain of the sensor - 3x by default
45         my_sensor_.setGainRGB(COLOR_10_CLICK_GAIN_2x);
46
47
48         while (true){
49             k++;
50             led1 = !led1;
51             /// Collect R, G, B, Clear and IR data from the sensor
52             my_sensor_.readRGBCIRValue(RGBCir);
53
54             unsigned char R = (RGBCir[0]*256)/65536;       //convertion de 16 bits
       (capteur) à 8 Bits (bandeau)
55             unsigned char G = 0.5*(RGBCir[1]*256)/65536;     //0.5 = facteur d'
       attenuation du vert car on pense que le constructeur à mis un gain pour que le
        capteur soit comparable à l'oeil humain
56             unsigned char B = (RGBCir[2]*256)/65536;
57
58
59             printf("[%4d] R=%d / G=%d / B=%d / C=%d / IR = %d \r\n\n ", k, R, G, B
       , RGBCir[3], RGBCir[4]);       //pour TeraTerm
60             // Convertir les valeurs de 0-1 à 0-255 et les mettre dans une
       variable unsigned int
61             unsigned int color = ((int)(R) << 16) | ((int)(G) << 8) | (int)(B);
62
63             for (int i = 1 ; i < WS2812_BUF; i++) {
64             px.Set(i,color); // FF : 255 en hexadecimal et ordre RGB
65
66             }
67         ws.write(px.getBuf());
68         thread_sleep_for(WAIT_TIME_MS);
69     }
70 }
71
```

```
72
73
74  /*
75  Aide pour unsigned char : https://stackoverflow.com/questions/25762871/equivalent-
        for-nop-in-c-for-embedded
76  doc du bandeau : https://www.digikey.fr/en/datasheets/parallaxinc/parallax-inc
        -28085-ws2812b-rgb-led-datasheet
77  */
```

codes utilisés - bibliothèque fonction du bandeau :

```
1   #include "WS2812.h"
2
3   int FRAME_SIZE, TYPE_BANDEAU;
4
5   WS2812::WS2812(PinName pin, int size, int zeroHigh, int zeroLow, int oneHigh, int
        oneLow, int typeBandeau) : __gpo(pin)
6   {
7       __size = size;
8       __transmitBuf = new bool[size * FRAME_SIZE];
9       __use_II = OFF;
10      __II = 0xFF; // set global intensity to full
11      __outPin = pin;
12      if(typeBandeau == 0){
13          TYPE_BANDEAU = 3;
14          FRAME_SIZE = 3 * FRAME_SIZE_LED;
15      }
16      else{
17          TYPE_BANDEAU = 4;
18          FRAME_SIZE = 4 * FRAME_SIZE_LED;
19      }
20
21      // Default values designed for K64f. Assumes GPIO toggle takes ~0.4us
22      setDelays(zeroHigh, zeroLow, oneHigh, oneLow);
23  }
24
25
26  WS2812::~WS2812()
27  {
28      delete[] __transmitBuf;
29  }
30
31  void WS2812::setDelays(int zeroHigh, int zeroLow, int oneHigh, int oneLow) {
32      __zeroHigh = zeroHigh;
33      __zeroLow = zeroLow;
34      __oneHigh = oneHigh;
35      __oneLow = oneLow;
36  }
37
38  void WS2812::__loadBuf(int buf[],int r_offset, int g_offset, int b_offset) {
39      for (int i = 0; i < __size; i++) {
40          int color = 0;
41
42          color |= ((buf[(i+g_offset)%__size] & 0x0000FF00));
43          color |= ((buf[(i+r_offset)%__size] & 0x00FF0000));
44          color |=  (buf[(i+b_offset)%__size] & 0x000000FF);
45          color |= (buf[i] & 0xFF000000);
46
47          // Outut format : GGRRBB
```

```
48          // Inout format : IIRRGGBB
49          unsigned char agrb[4] = {0x0, 0x0, 0x0, 0x0};
50
51          unsigned char sf; // scaling factor for  II
52
53          // extract colour fields from incoming
54          // 0 = green, 1 = red, 2 = blue, 3 = brightness
55          agrb[0] = (color & 0x0000FF00) >> 8;
56          agrb[1] = (color & 0x00FF0000) >> 16;
57          agrb[2] = color  & 0x000000FF;
58          agrb[3] = (color & 0xFF000000) >> 24;
59
60          // set the intensity scaling factor (global, per pixel, none)
61          if (__use_II == GLOBAL) {
62              sf = __II;
63          } else if (__use_II == PER_PIXEL) {
64              sf = agrb[3];
65          } else {
66              sf = 0xFF;
67          }
68
69          // Apply the scaling factor to each othe colour components
70          for (int clr = 0; clr < TYPE_BANDEAU; clr++) {
71              agrb[clr] = ((agrb[clr] * sf) >> 8);
72
73              for (int j = 0; j < 8; j++) {
74                  if (((agrb[clr] << j) & 0x80) == 0x80) {
75                      // Bit is set (checks MSB fist)
76                      __transmitBuf[(i * FRAME_SIZE) + (clr * 8) + j] = 1;
77                  } else {
78                      // Bit is clear
79                      __transmitBuf[(i * FRAME_SIZE) + (clr * 8) + j] = 0;
80                  }
81              }
82          }
83      }
84  }
85
86  void WS2812::write(int buf[]) {
87      write_offsets(buf, 0, 0, 0);
88  }
89
90  void WS2812::write_offsets (int buf[],int r_offset, int g_offset, int b_offset) {
91      int i, j;
92
93      // Load the transmit buffer
94      __loadBuf(buf, r_offset, g_offset, b_offset);
95
96      // Entering timing critical section, so disabling interrupts
97      __disable_irq();
98
99      // Begin bit-banging
100     for (i = 0; i < FRAME_SIZE * __size; i++) {
101         j = 0;
102         if (__transmitBuf[i]){
103             __gpo = 1;
104             for (; j < __oneHigh; j++) {
105                 __NOP();
106             }
107             __gpo = 0;
108             for (; j < __oneLow; j++) {
109                 __NOP();
```

```
110             }
111         } else {
112             __gpo = 1;
113             for (; j < __zeroHigh; j++) {
114                 __NOP();
115             }
116             __gpo = 0;
117             for (; j < __zeroLow; j++) {
118                 __NOP();
119             }
120         }
121     }
122 
123     // Exiting timing critical section, so enabling interrutps
124     __enable_irq();
125 }
126 
127 
128 void WS2812::useII(BrightnessControl bc)
129 {
130     if (bc > OFF) {
131         __use_II = bc;
132     } else {
133         __use_II = OFF;
134     }
135 }
136 
137 void WS2812::setII(unsigned char II)
138 {
139     __II = II;
140 }
```

```
1  #ifndef WS2812_H
2  #define WS2812_H
3  
4  #include "mbed.h"
5  
6  #define TYPE_BANDEAU_NEO 4
7  #define TYPE_BANDEAU_NONO 3
8  #define FRAME_SIZE_LED  8
9  
10 //!Library for the WS2812 RGB LED with integrated controller
11 /*!
12 The WS2812 is controller that is built into a range of LEDs
13 */
14 class WS2812
15 {
16 public:
17     enum BrightnessControl { OFF, GLOBAL, PER_PIXEL };
18 
19     /**
20     *   Constructor
21     *
22     * @param pin Output pin. Connect to "Din" on the first WS2812 in the strip
23     * @param size Number of LEDs in your strip
24     * @param zeroHigh How many NOPs to insert to ensure TOH is properly generated.
       See library description for more information.
25     * @param zeroLow How many NOPs to insert to ensure TOL is properly generated.
       See library description for more information.
```

```
26      * @param oneHigh How many NOPs to insert to ensure T1H is properly generated.
        See library description for more information.
27      * @param oneLow How many NOPs to insert to ensure T1L is properly generated.
        See library description for more information.
28      *
29      */
30      WS2812(PinName pin, int size, int zeroHigh, int zeroLow, int oneHigh, int
        oneLow, int typeBandeau = 0);
31
32      /*!
33      Destroys instance.
34      */
35      ~WS2812();
36
37      /**
38      *   Sets the timing parameters for the bit-banged signal
39      *
40      * @param zeroHigh How many NOPs to insert to ensure T0H is properly generated.
         See library description for more information.
41      * @param zeroLow How many NOPs to insert to ensure T0L is properly generated.
        See library description for more information.
42      * @param oneHigh How many NOPs to insert to ensure T1H is properly generated.
        See library description for more information.
43      * @param oneLow How many NOPs to insert to ensure T1L is properly generated.
        See library description for more information.
44      *
45      */
46      void setDelays(int zeroHigh, int zeroLow, int oneHigh, int oneLow);
47
48      /**
49      *   Writes the given buffer to the LED strip with the given offsets.
50      *   NOTE: This function is timing critical, therefore interrupts are disabled
        during the transmission section.
51      *
52      * @param buf Pointer to the PixelArray buffer
53      * @param r_offset The offset where each each pixel pulls its red component.
        Wraps to beginning if end is reached.
54      * @param g_offset The offset where each each pixel pulls its green component.
        Wraps to beginning if end is reached.
55      * @param b_offset The offset where each each pixel pulls its blue component.
        Wraps to beginning if end is reached.
56      *
57      */
58      void write_offsets(int buf[], int r_offset = 0, int g_offset = 0, int b_offset
         = 0);
59
60
61      /**
62      *   Writes the given buffer to the LED strip
63      *   NOTE: This function is timing critical, therefore interrupts are disabled
        during the transmission section.
64      *
65      * @param buf Pointer to the PixelArray buffer
66      *
67      */
68      void write(int buf[]);
69
70      /**
71      *   Sets the brightness mode
72      *
73      * @param bc The brightness control. Defaults to OFF. Possible values include
        OFF, GLOBAL, and PER_PIXEL
```

```
74      *
75      */
76      void useII(BrightnessControl bc);
77
78      /**
79      *   Sets the global brightness level.
80      *
81      * @param II The brightness level. Possible values include 0 - 255 (0x00 - 0xFF
        ).
82      *
83      */
84      void setII(unsigned char II);
85
86  private:
87
88      int __size;
89      int __zeroHigh, __zeroLow, __oneHigh, __oneLow;
90      unsigned char __II;
91      BrightnessControl __use_II;
92      bool *__transmitBuf;
93      void __loadBuf(int buf[],int r_offset=0, int g_offset=0, int b_offset=0);
94      PinName __outPin;
95      DigitalOut __gpo;
96  };
97  #endif
```

codes utilisés - bibliothèque fonction du capteur :

```
1   #include <mbed.h>
2    #include "COLOR_10_CLICK.h"
3
4   Color_10_Click::Color_10_Click(I2C *_i2c, DigitalOut *_led_data){
5       /* Initialisation of interrupt input */
6       if (_led_data){ delete __led_data; }
7       __led_data=_led_data;
8       /* Initialisation of i2c module */
9       if (_i2c){ delete __i2c; }
10      __i2c=_i2c;
11      __i2c->frequency(400000);    // Frequency of 400kHz
12      thread_sleep_for(10);       // 10 ms
13  }
14
15  void Color_10_Click::powerUp(void){
16      cmd[0] = COLOR_10_CLICK_COMMAND;
17      cmd[1] = 0;
18      cmd[2] = 0;
19      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 3);
20      if(DEBUG_MODE) printf("Init Acq = %d\r\n", ack1);
21      wait_us(1000);
22  }
23
24  int Color_10_Click::getPartID(void){
25      // Part ID Status
26      cmd[0] = COLOR_10_CLICK_PART_ID;
27      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
28      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
29      if(DEBUG_MODE)   printf("Part ID Acq (W) = %d\r\n", ack1);
30      if(DEBUG_MODE)   printf("Part ID Acq (R) = %d\r\n", ack2);
31      return data[0];
```

```
32  }
33
34
35  int Color_10_Click::getCommandValue(void){
36      cmd[0] = COLOR_10_CLICK_COMMAND;
37      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
38      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
39      if(DEBUG_MODE)   printf("Command Value Acq (W) = %d\r\n", ack1);
40      if(DEBUG_MODE)   printf("Command Value Acq (R) = %d\r\n", ack2);
41      return (data[1] << 8) + data[0];
42  }
43
44  void Color_10_Click::setGainRGB(int val){
45      int command_value = getCommandValue();
46      cmd[0] = COLOR_10_CLICK_COMMAND;
47      cmd[1] = (command_value & 0xFF);
48      cmd[2] = ((command_value >> 8) & 0b11110011) | (val << 2);
49      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 3);
50      if(DEBUG_MODE)   printf("Gain Acq (W) = %d\r\n", ack1);
51  }
52
53  int Color_10_Click::readRedValue(void){
54      cmd[0] = COLOR_10_CLICK_RED_CHAN;
55      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
56      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
57      if(DEBUG_MODE)   printf("Red Chan Acq (W) = %d\r\n", ack1);
58      if(DEBUG_MODE)   printf("Red Chan Acq (R) = %d\r\n", ack2);
59      Red_color = (data[1] << 8) + data[0];
60      return Red_color;
61  }
62
63  int Color_10_Click::readGreenValue(void){
64      cmd[0] = COLOR_10_CLICK_GREEN_CHAN;
65      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
66      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
67      if(DEBUG_MODE)   printf("Green Chan Acq (W) = %d\r\n", ack1);
68      if(DEBUG_MODE)   printf("Green Chan Acq (R) = %d\r\n", ack2);
69      Green_color = (data[1] << 8) + data[0];
70      return Green_color;
71  }
72
73  int Color_10_Click::readBlueValue(void){
74      cmd[0] = COLOR_10_CLICK_BLUE_CHAN;
75      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
76      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
77      if(DEBUG_MODE)   printf("Blue Chan Acq (W) = %d\r\n", ack1);
78      if(DEBUG_MODE)   printf("Blue Chan Acq (R) = %d\r\n", ack2);
79      Blue_color = (data[1] << 8) + data[0];
80      return Blue_color;
81  }
82
83  int Color_10_Click::readIRValue(void){
84      cmd[0] = COLOR_10_CLICK_IR_CHAN;
85      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
86      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
87      if(DEBUG_MODE)   printf("IR Chan Acq (W) = %d\r\n", ack1);
88      if(DEBUG_MODE)   printf("IR Chan Acq (R) = %d\r\n", ack2);
89      IR_color = (data[1] << 8) + data[0];
90      return IR_color;
91  }
92
93  int Color_10_Click::readClearValue(void){
```

```
94      cmd[0] = COLOR_10_CLICK_CLEAR_CHAN;
95      ack1 = __i2c->write(COLOR_10_CLICK_ADD << 1, cmd, 1, true);
96      ack2 = __i2c->read(COLOR_10_CLICK_ADD << 1, data, 2);
97      if(DEBUG_MODE)   printf("Clear Chan Acq (W) = %d\r\n", ack1);
98      if(DEBUG_MODE)   printf("Clear Chan Acq (R) = %d\r\n", ack2);
99      Clear_color = (data[1] << 8) + data[0];
100     return Clear_color;
101 }
102
103 void Color_10_Click::readRGBCIRValue(int rgbcIR[]){
104     rgbcIR[0] = readRedValue();
105     rgbcIR[1] = readGreenValue();
106     rgbcIR[2] = readBlueValue();
107     rgbcIR[3] = readClearValue();
108     rgbcIR[4] = readIRValue();
109 }
```

```
1  #ifndef __COLOR_10_CLICK_HEADER_H__
2  #define __COLOR_10_CLICK_HEADER_H__
3
4  #include <mbed.h>
5
6  /** Constant definition */
7  #define     DEBUG_MODE                  0
8
9  #define     COLOR_10_CLICK_ADD          0x10
10 #define     COLOR_10_CLICK_COMMAND      0x00
11 #define     COLOR_10_CLICK_PART_ID      0x0C
12 #define     COLOR_10_CLICK_CLEAR_CHAN   0x04
13 #define     COLOR_10_CLICK_RED_CHAN     0x05
14 #define     COLOR_10_CLICK_GREEN_CHAN   0x06
15 #define     COLOR_10_CLICK_BLUE_CHAN    0x07
16 #define     COLOR_10_CLICK_IR_CHAN      0x08
17
18 #define     COLOR_10_CLICK_GAIN_1_2x    0b11
19 #define     COLOR_10_CLICK_GAIN_1x      0b00        // default value
20 #define     COLOR_10_CLICK_GAIN_2x      0b01
21 #define     COLOR_10_CLICK_GAIN_4x      0b10
22
23
24 /**
25  * @class Color_10_Click
26  * @brief Access to the Color10Click module from MikroE
27  * @details     Color10Click module allows to measure RGB light intensities
28  *  with a VEML-3328 sensor from Vishay.
29  */
30 class Color_10_Click{
31     private:
32         /// Intensity of the Red component
33         int     Red_color;
34         /// Intensity of the Green component
35         int     Green_color;
36         /// Intensity of the Blue component
37         int     Blue_color;
38         /// Global intensity
39         int     Clear_color;
40         /// Intensity of the InfraRed component
41         int     IR_color;
42         /// Command to send
```

```
43          char     cmd[3];
44          /// Received Data
45          char     data[2];
46          /// Acknowledgement variables
47          char     ack1, ack2;
48
49          /// I2C interface pins
50          I2C              *__i2c = NULL;
51          /// digital output to control the WS2812 RGB Led
52          DigitalOut       *__led_data = NULL;
53
54      public:
55          /**
56          * @brief Simple constructor of the Color_10_Click class.
57          * @details Create a Color_10_Click object with
58          *    an I2C interface
59          *    I2C communication will be initialized at 400kHz
60          * @param _i2c SPI interface not initialized
61          * @param _led_data digital output to control the WS2812 RGB Led
62          */
63          Color_10_Click(I2C *_i2c, DigitalOut *_led_data);
64
65          /**
66          * @brief Initiatlization of the sensor
67          * @details Initialize the sensor
68          */
69          void powerUp(void);
70
71          /**
72          * @brief Return the ID of the module
73          * @return the part ID of the module - default 0x28 = 40d
74          */
75          int getPartID(void);
76
77          /**
78          * @brief Return the value of the Command register
79          * @return the value of the Command register
80          */
81          int getCommandValue(void);
82
83          /**
84          * @brief Set the analog gain of the sensor
85          * @details Gain in range : 1/2x, 1x (default), 2x, 4x
86          * @param val gain value - COLOR_10_CLICK_GAIN_1X
87          */
88          void setGainRGB(int val);
89
90          /**
91          * @brief Read the red data from the Color_10_Click module
92          * @details Read the red data from the Color_10_Click module
93          *   and update the member value of the object -
94          *
95          * @return the Red component of the light.
96          */
97          int readRedValue(void);
98
99          /**
100         * @brief Read the green data from the Color_10_Click module
101         * @details Read the green data from the Color_10_Click module
102         *   and update the member value of the object -
103         *
104         * @return the Green component of the light.
```

```
105         */
106         int readGreenValue(void);
107
108         /**
109         * @brief Read the blue data from the Color_10_Click module
110         * @details Read the blue data from the Color_10_Click module
111         *   and update the member value of the object -
112         *
113         * @return the Blue component of the light.
114         */
115         int readBlueValue(void);
116
117         /**
118         * @brief Read the InfraRed data from the Color_10_Click module
119         * @details Read the infrared data from the Color_10_Click module
120         *   and update the member value of the object -
121         *
122         * @return the IR component of the light.
123         */
124         int readIRValue(void);
125
126         /**
127         * @brief Read the Clear data from the Color_10_Click module
128         * @details Read the clear data from the Color_10_Click module
129         *   and update the member value of the object -
130         *
131         * @return the clear component of the light.
132         */
133         int readClearValue(void);
134
135         /**
136         * @brief Collect all the RGBIR data from the Color_10_Click module
137         * @details Read all the RGBIR data from the Color_10_Click module
138         *   and update the members value of the object -
139         *
140         * @param rgbcIR first cell of a 5 int arrays
141         * @return R G B C IR value in a 5 int arrays
142         */
143         void readRGBCIRValue(int rgbcIR[]);
144 };
145
146 #endif
```

codes utilisés - bibliothèque fonction pour piloter la matrice de LED :

```
1  #include "PixelArray.h"
2
3  PixelArray::PixelArray(int size)
4  {
5      pbufsize = size;
6      pbuf = new int[pbufsize];
7      SetAll(0x0); // initialise memory to zeros
8  }
9
10 PixelArray::~PixelArray()
11 {
12     delete[] pbuf;
13 }
14
```

```
15  void PixelArray::SetAll(unsigned int value)
16  {
17      // for each pixel
18      for (int i=0 ; i < pbufsize; i++) {
19          __set_pixel(i,value);
20      }
21  }
22
23  void PixelArray::SetAllI(unsigned char value)
24  {
25      // for each pixel
26      for (int i=0 ; i < pbufsize; i++) {
27          __set_pixel_component(i,3,value);
28      }
29  }
30
31  void PixelArray::SetAllR(unsigned char value)
32  {
33      // for each pixel
34      for (int i=0 ; i < pbufsize; i++) {
35          __set_pixel_component(i,2,value);
36      }
37  }
38
39  void PixelArray::SetAllG(unsigned char value)
40  {
41      // for each pixel
42      for (int i=0 ; i < pbufsize; i++) {
43          __set_pixel_component(i,1,value);
44      }
45  }
46
47  void PixelArray::SetAllB(unsigned char value)
48  {
49      // for each pixel
50      for (int i=0 ; i < pbufsize; i++) {
51          __set_pixel_component(i,0,value);
52      }
53  }
54
55  void PixelArray::Set(int i, unsigned int value)
56  {
57      if ((i >= 0) && (i < pbufsize)) {
58          __set_pixel(i,value);
59      }
60  }
61
62  void PixelArray::SetI(int i, unsigned char value)
63  {
64      if ((i >= 0) && (i < pbufsize)) {
65          __set_pixel_component(i,3,value);
66      }
67  }
68
69
70  void PixelArray::SetR(int i, unsigned char value)
71  {
72      if ((i >= 0) && (i < pbufsize)) {
73          __set_pixel_component(i,2,value);
74      }
75  }
76
```

```
77  void PixelArray::SetG(int i, unsigned char value)
78  {
79      if ((i >= 0) && (i < pbufsize)) {
80          __set_pixel_component(i,1,value);
81      }
82  }
83
84  void PixelArray::SetB(int i, unsigned char value)
85  {
86      if ((i >= 0) && (i < pbufsize)) {
87          __set_pixel_component(i,0,value);
88      }
89  }
90
91
92  int* PixelArray::getBuf()
93  {
94      return (pbuf);
95  }
96
97  // set either the I,R,G,B value of specific pixel channel
98  void PixelArray::__set_pixel_component(int index, int channel, int value)
99  {
100
101     // AND with 0x00 shifted to the right location to clear the bits
102     pbuf[index] &= ~(0xFF << (8 * channel));
103
104     // Set the bits with an OR
105     pbuf[index] |= (value << (8 * channel));
106 }
107
108
109 // set either the I,R,G,B value of specific pixel channel
110 void PixelArray::__set_pixel(int index, int value)
111 {
112     // AND with 0x00 shifted to the right location to clear the bits
113     pbuf[index] = value;
114 }
```

```
1   #ifndef PixelArray_H
2   #define PixelArray_H
3
4   #include "mbed.h"
5
6   //!Library for the WS2812 RGB LED with integrated controller
7   /*!
8   PixelArray
9   */
10  class PixelArray
11  {
12  public:
13      //!Creates an instance of the class.
14      /*!
15      Pixel Array
16      */
17      PixelArray(int);
18
19      /*!
20      Destroys instance.
```

```
21      */
22      ~PixelArray();
23
24      int* getBuf();
25
26      void SetAll(unsigned int);
27      void SetAllI(unsigned char);
28      void SetAllR(unsigned char);
29      void SetAllG(unsigned char);
30      void SetAllB(unsigned char);
31
32      // location, value
33      void Set(int, unsigned int);
34      void SetI(int, unsigned char);
35      void SetR(int, unsigned char);
36      void SetG(int, unsigned char);
37      void SetB(int, unsigned char);
38
39  private:
40
41      int *pbuf;
42      int pbufsize;
43
44      void __set_pixel_component(int index, int channel, int value);
45      void __set_pixel(int index, int value);
46
47  };
48  #endif
```