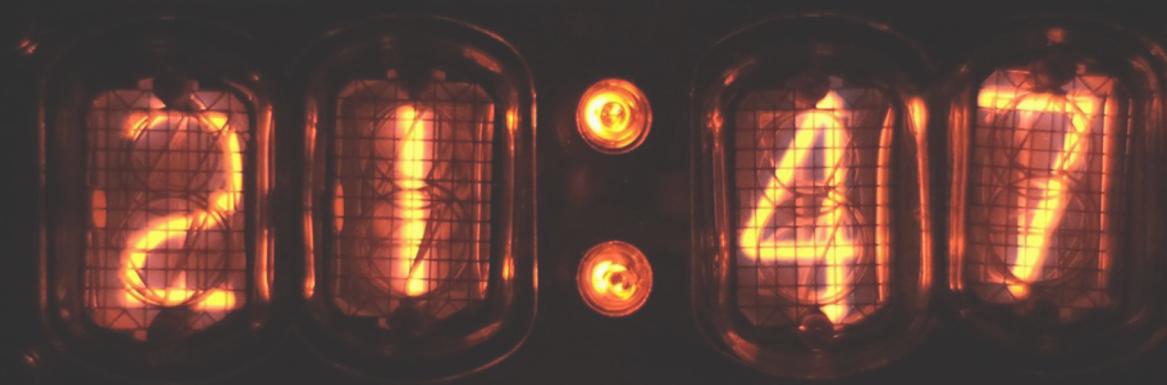


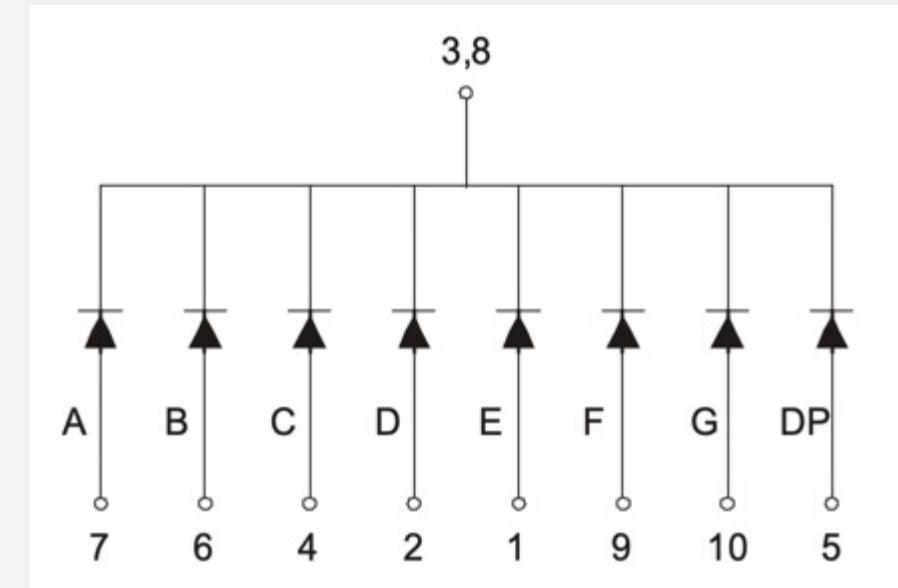
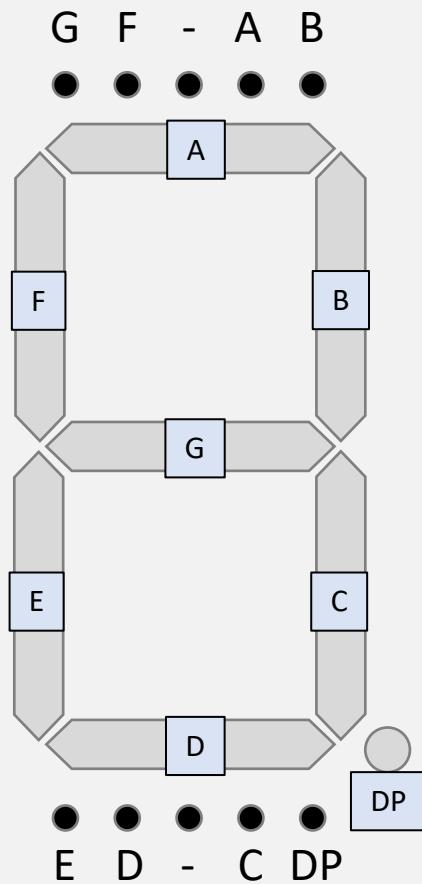
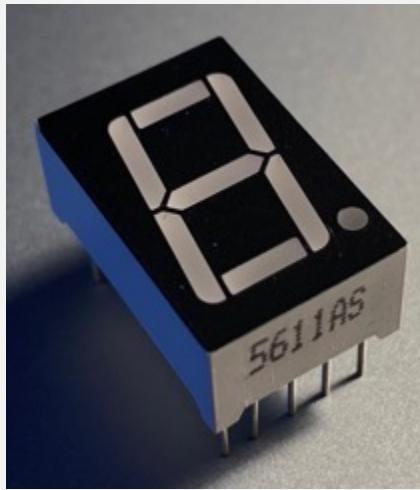
4 digits



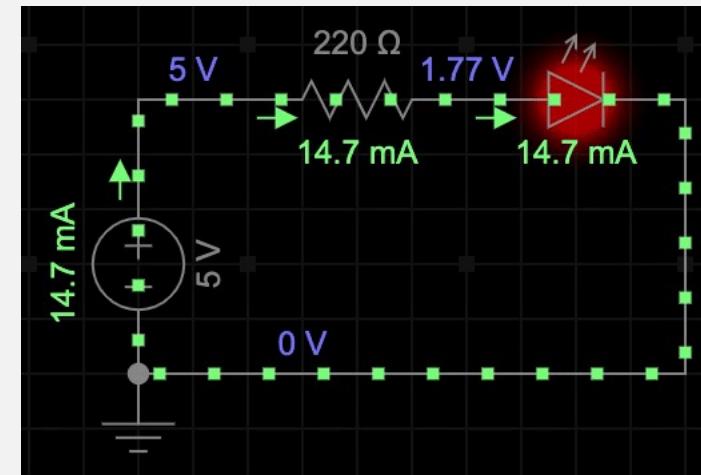
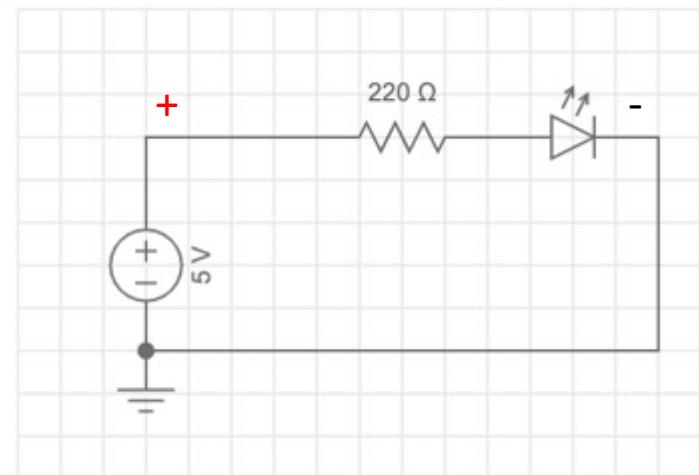
Objectifs

- Affichage 7 segments
- Montage et programme de tests
- Affichage 4 digits
- Montage et programme de tests
- Multiplexage
- Gérer le temps
- TM1637

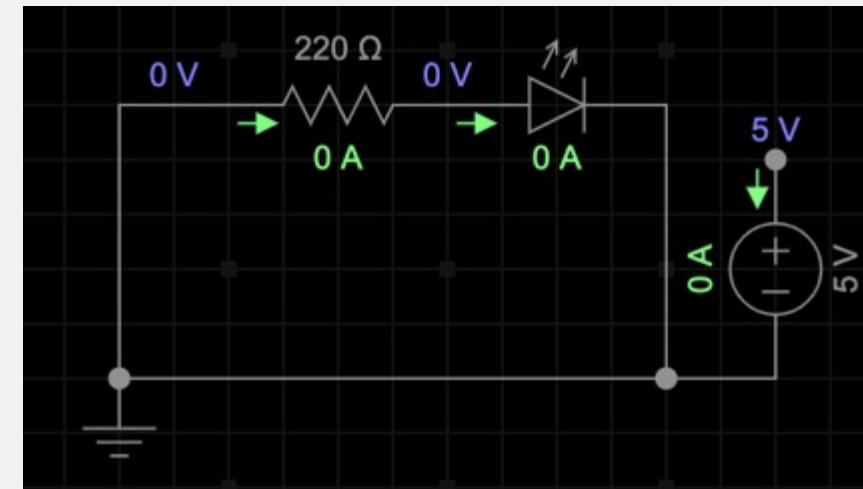
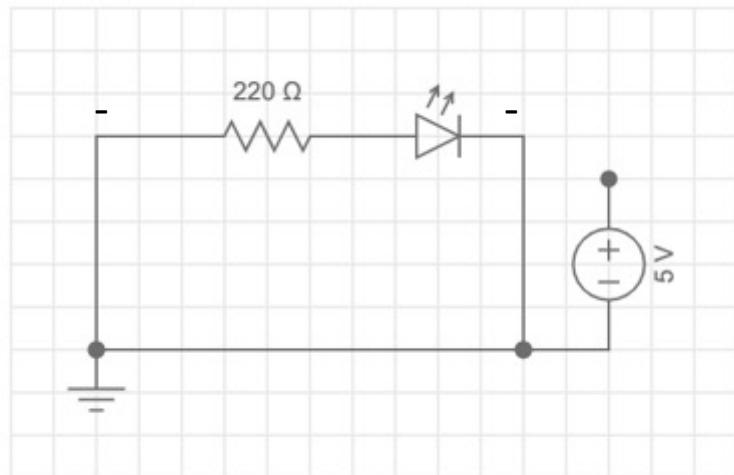
7 segments à cathode commune – 5611AS



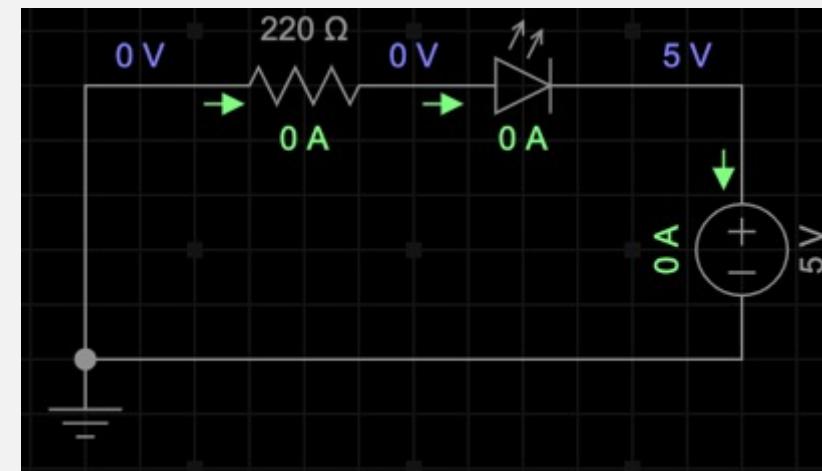
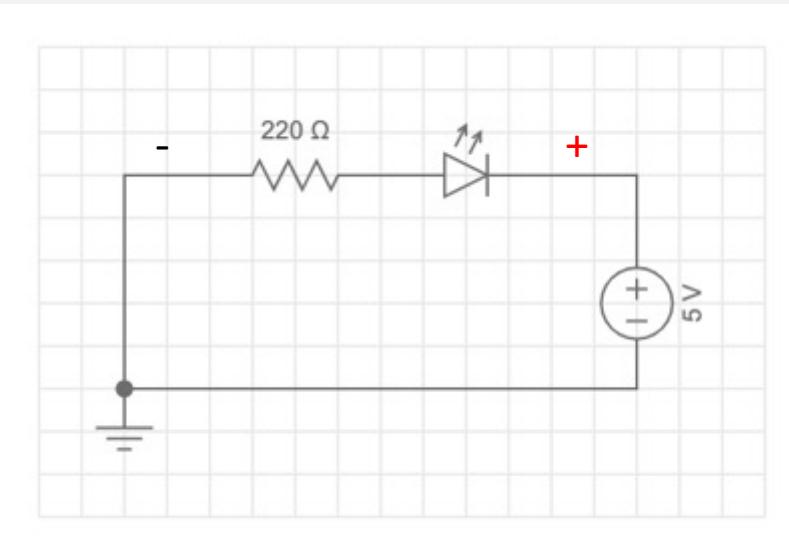
Allumer un segment - Équivalent à une DEL



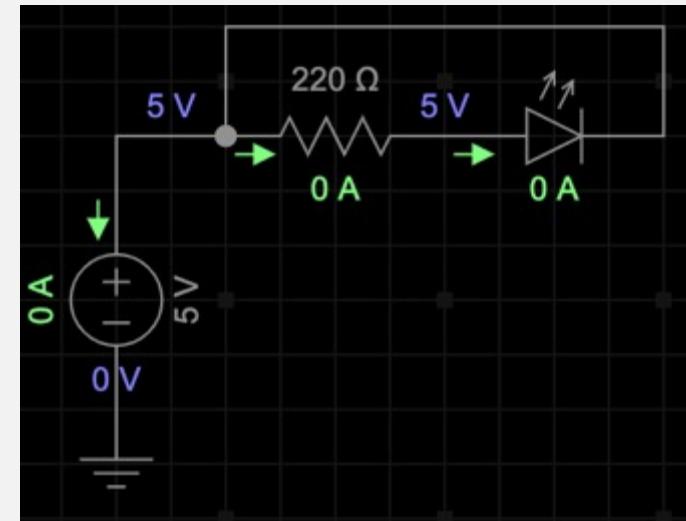
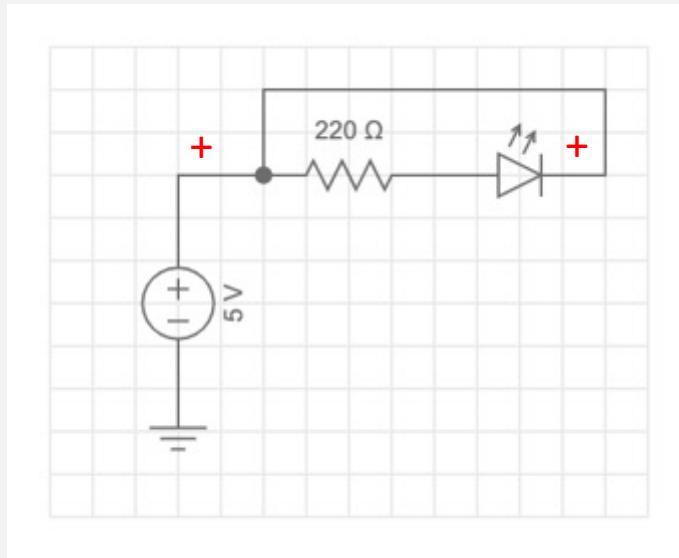
Segment éteint - Équivalent à une DEL



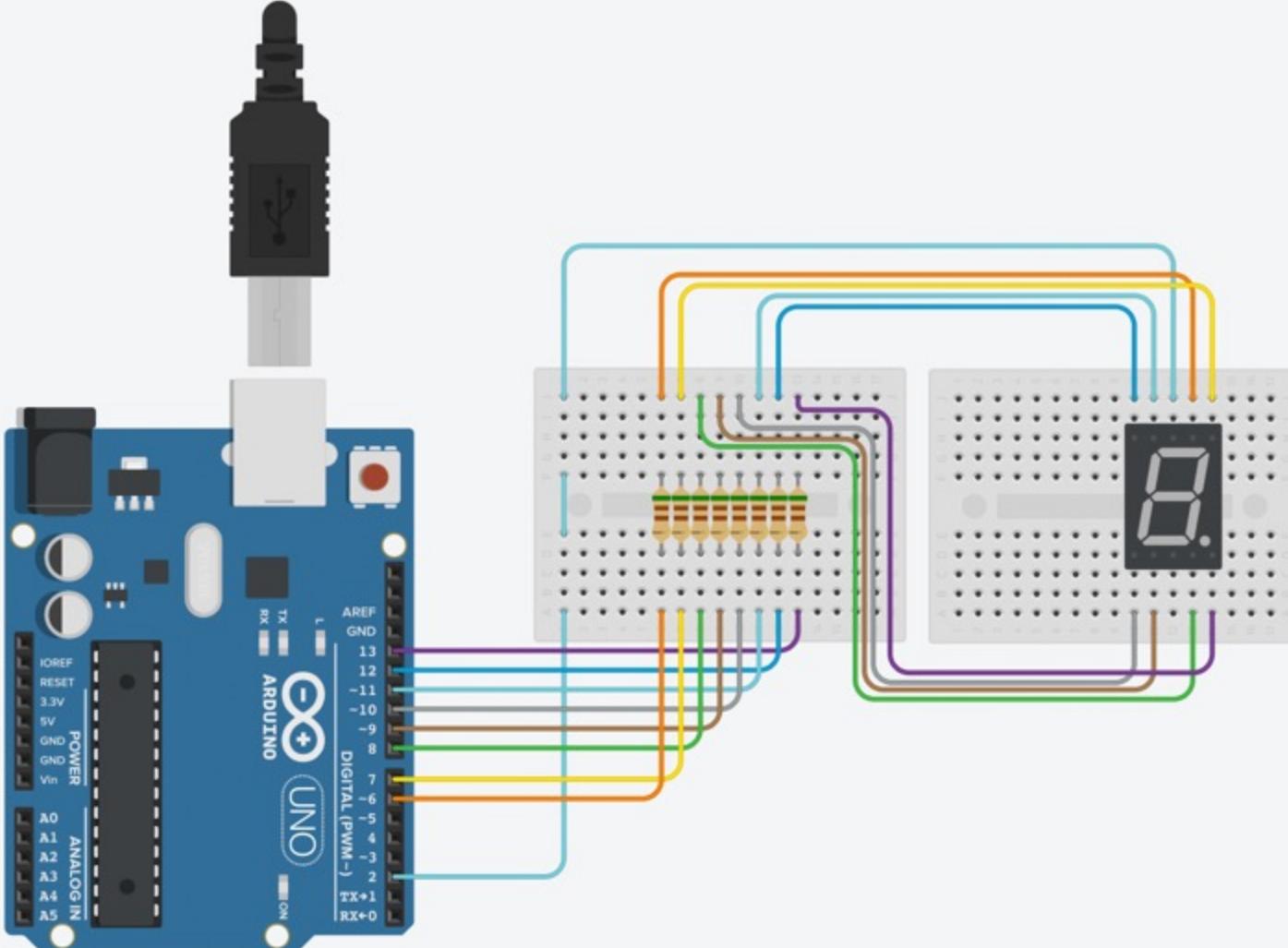
Digit éteint - Segment éteint - Équivalent à une DEL



Digit éteint - Segment éteint - Équivalent à une DEL



Programme de test



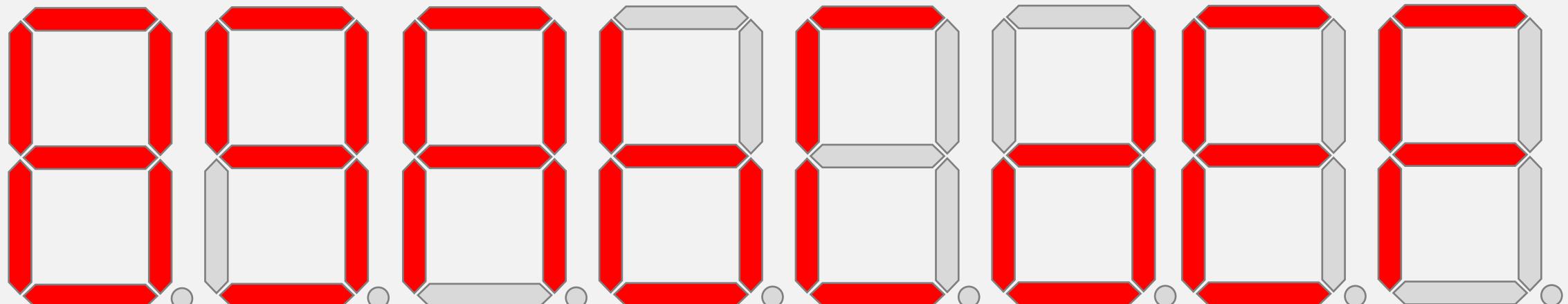
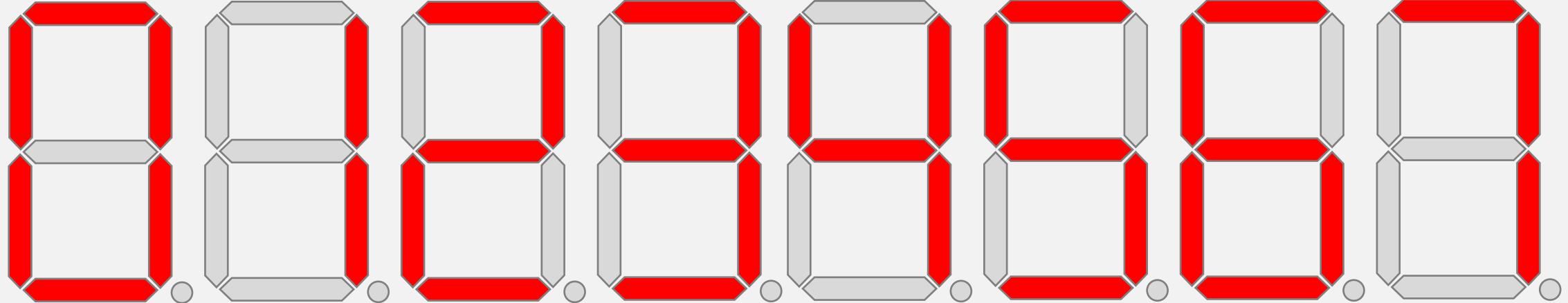
```
void reset();

int segmentCourant = 0;

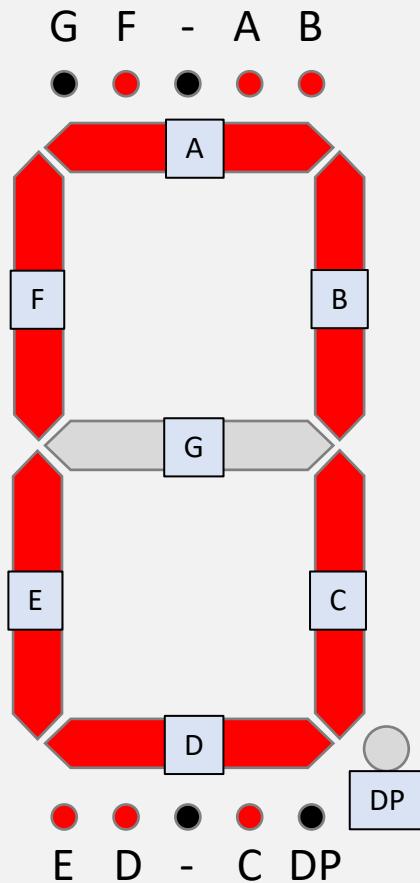
void setup() {
    pinMode(2, OUTPUT);
    for (size_t pin = 6; pin <= 13; ++pin) {
        pinMode(pin, OUTPUT);
    }
    digitalWrite(2, HIGH);
    for (size_t pin = 6; pin <= 13; ++pin) {
        digitalWrite(pin, LOW);
    }
}

void loop() {
    digitalWrite(segmentCourant + 6, HIGH);
    digitalWrite(2, LOW);
    delay(500);
    digitalWrite(segmentCourant + 6, LOW);
    digitalWrite(2, HIGH);
    segmentCourant = (segmentCourant + 1) % 8;
}
```

Affichage des chiffres de 0 à F

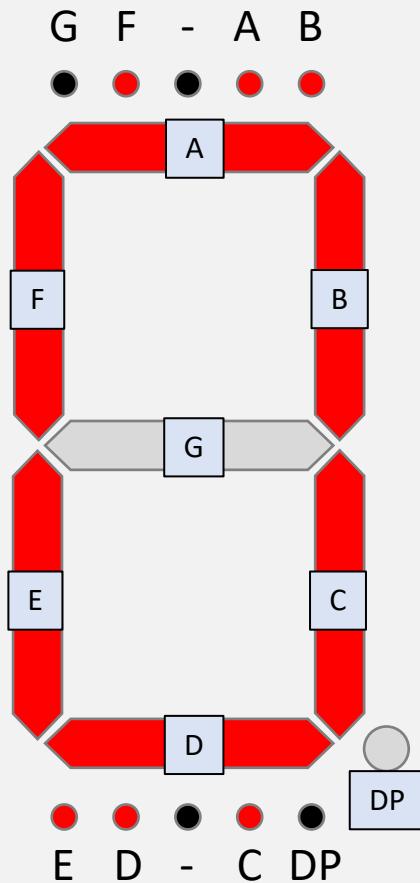


Représentation du 0



| DP | G | F | E | D | C | B | A |
|----|---|---|---|---|---|---|---|
| | | | | | | | |

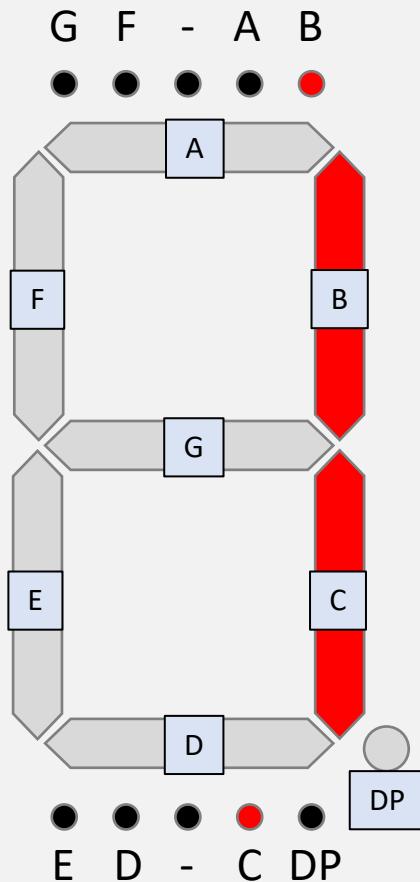
Représentation du 0



| DP | G | F | E | D | C | B | A |
|----|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

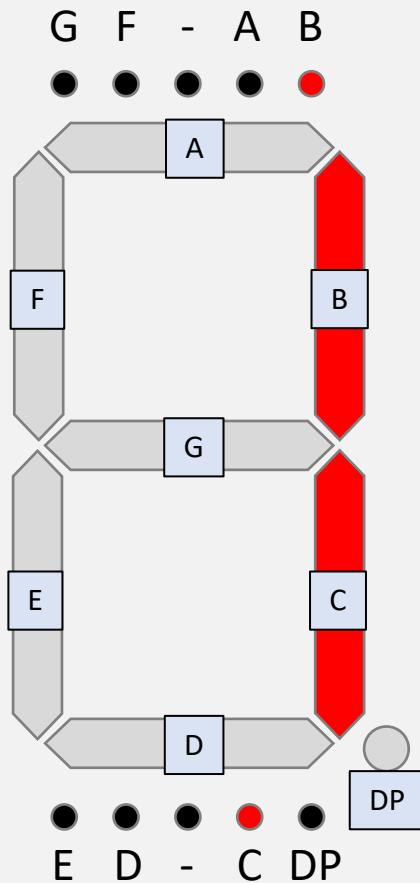
⇒ 0b0011 1111

Représentation du 1



| DP | G | F | E | D | C | B | A |
|----|---|---|---|---|---|---|---|
| | | | | | | | |

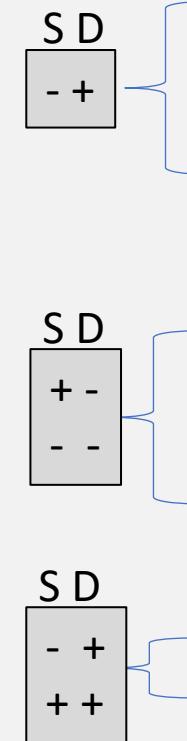
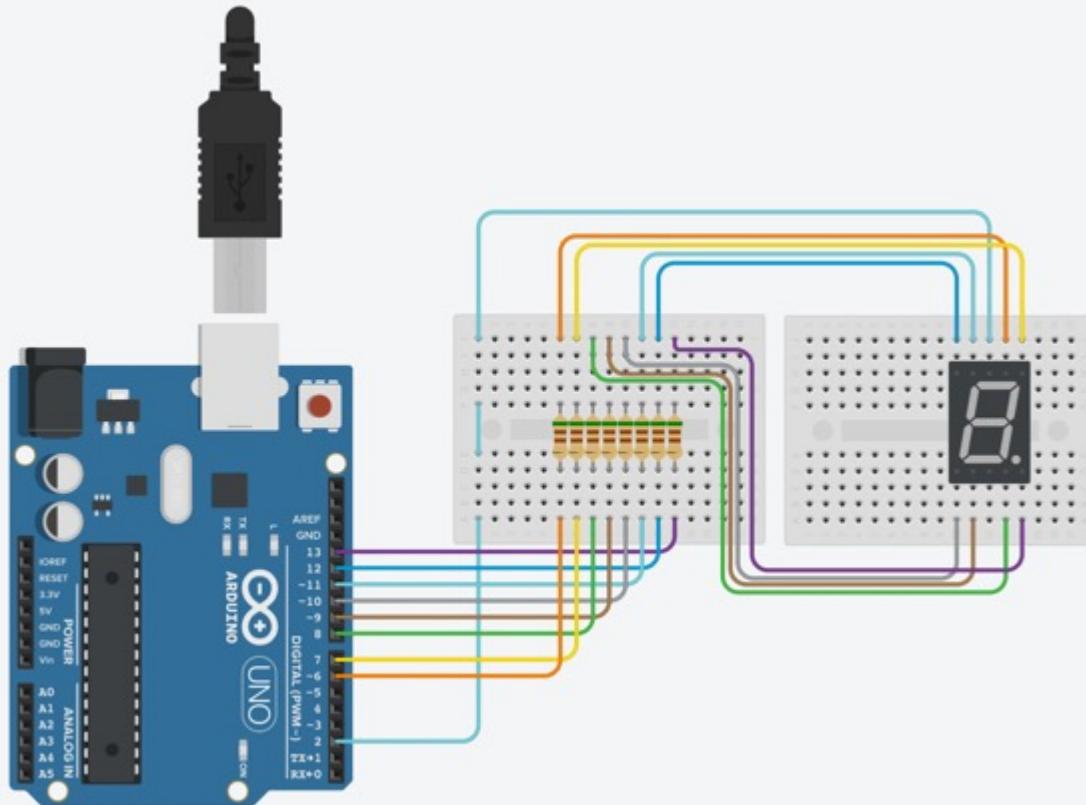
Représentation du 1



| DP | G | F | E | D | C | B | A |
|----|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

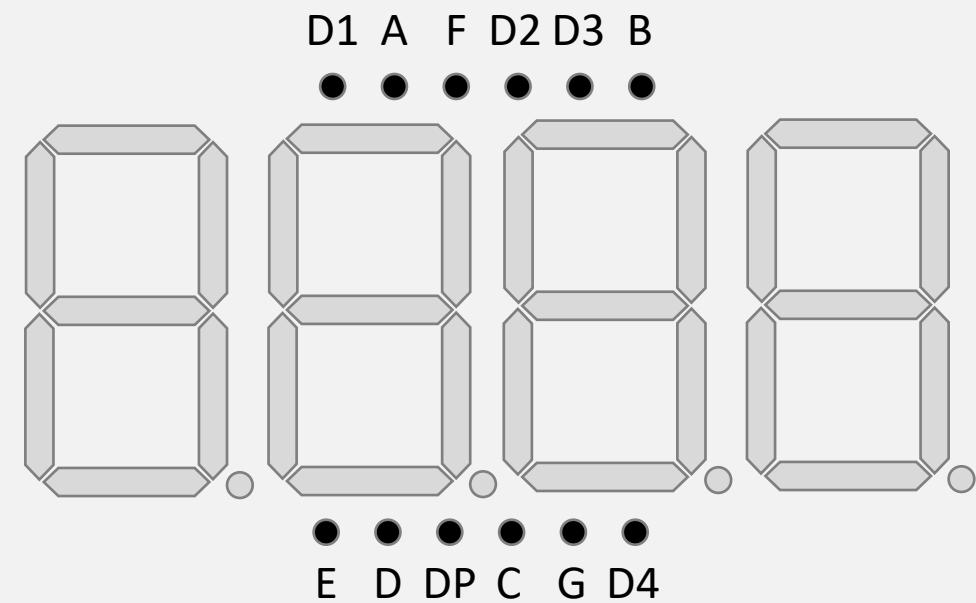
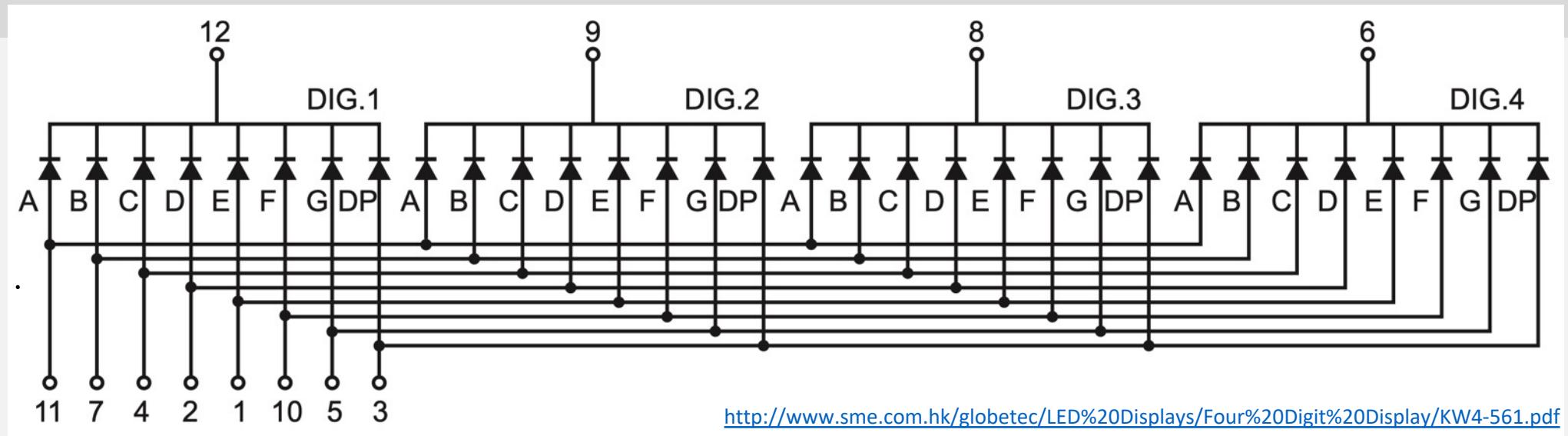
⇒ 0b0110 0000

Affichage des chiffres de 0 à 1

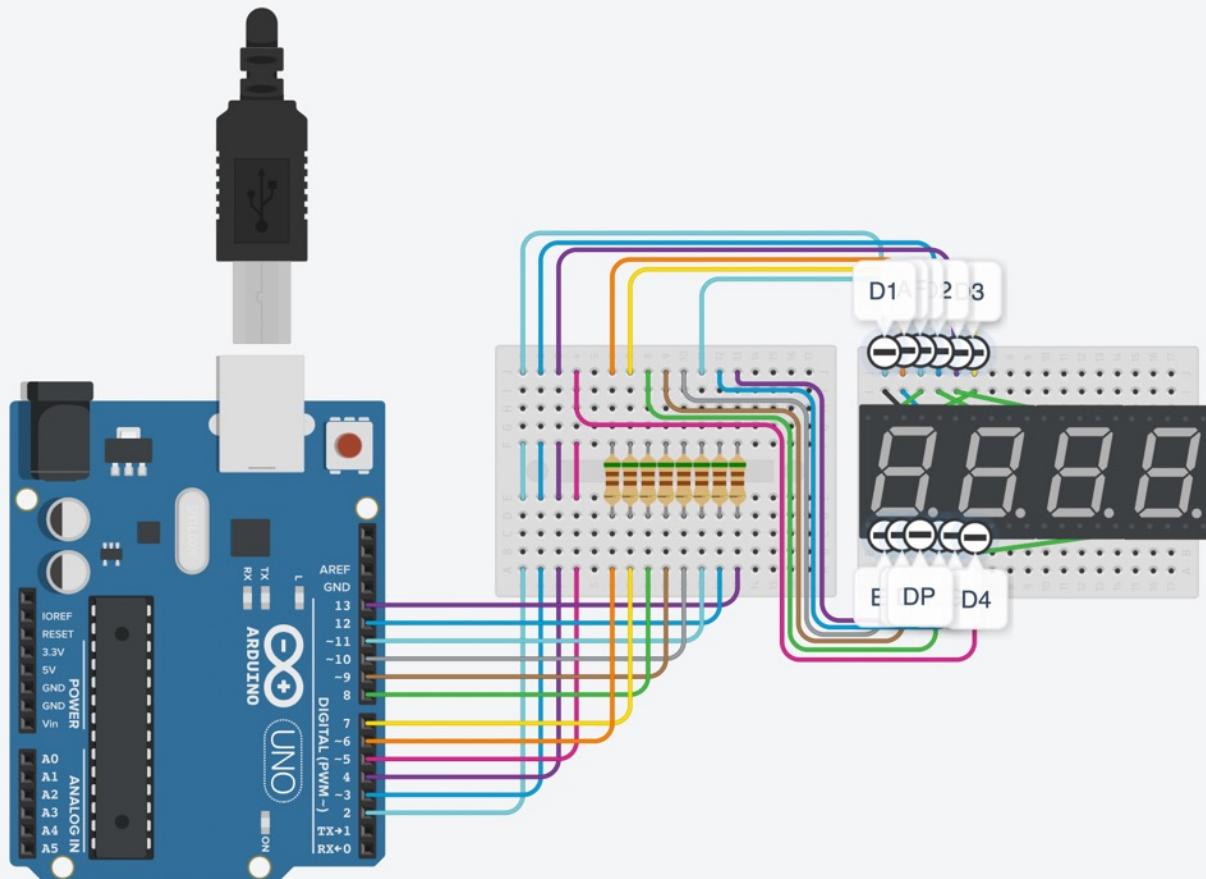


```
byte segments[] = {  
    0b00111111,  
    0b000000110  
};  
  
int valeurCourante = 0;  
void setup() {  
    pinMode(2, OUTPUT);  
    for (size_t pin = 6; pin <= 13; ++pin) {  
        pinMode(pin, OUTPUT);  
    }  
    digitalWrite(2, HIGH);  
    for (size_t pin = 6; pin <= 13; ++pin) {  
        digitalWrite(pin, LOW);  
    }  
}  
  
void loop() {  
    byte valeursSegments = segments[valeurCourante];  
    for (int i = 0; i < 8; ++i) {  
        digitalWrite(i + 6,  
                    (valeursSegments >> i) & 1 ? HIGH : LOW);  
    }  
    digitalWrite(2, LOW);  
  
    delay(500);  
  
    digitalWrite(2, HIGH);  
    valeurCourante = (valeurCourante + 1) % 2;  
}
```

Présentation – 3461AS



Programme de test



```
int segmentCourant = 0;  int digitCourant = 0;

void setup() {
    for (size_t pin = 2; pin <= 13; ++pin) {
        pinMode(pin, OUTPUT);
    }
    reset();
}

void loop() {
    digitalWrite(segmentCourant + 6, HIGH);
    digitalWrite(digitCourant + 2, LOW);

    delay(500);

    digitalWrite(segmentCourant + 6, LOW);
    digitalWrite(digitCourant + 2, HIGH);

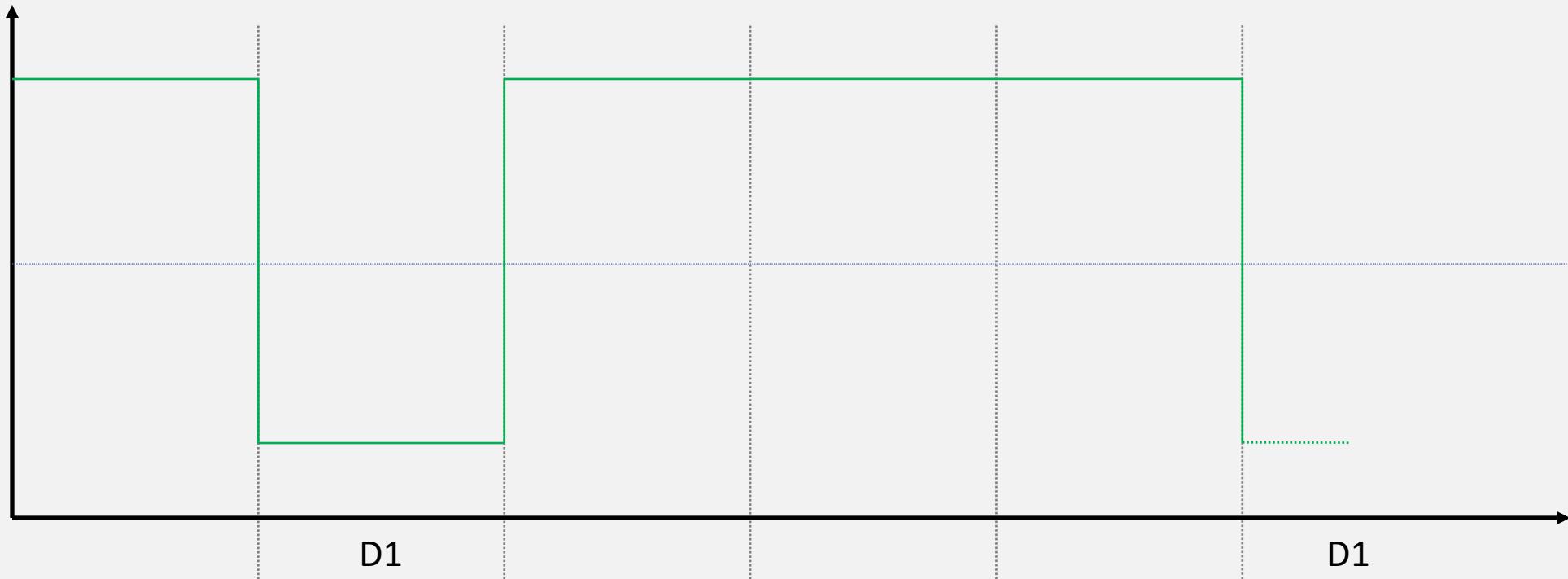
    if (++segmentCourant >= 8) {
        segmentCourant = 0;
        digitCourant = (digitCourant + 1) % 4;
    }
}

void reset() {
    for (size_t pin = 2; pin <= 5; ++pin) { digitalWrite(pin, HIGH); }

    for (size_t pin = 6; pin <= 13; ++pin) { digitalWrite(pin, LOW); }
}
```

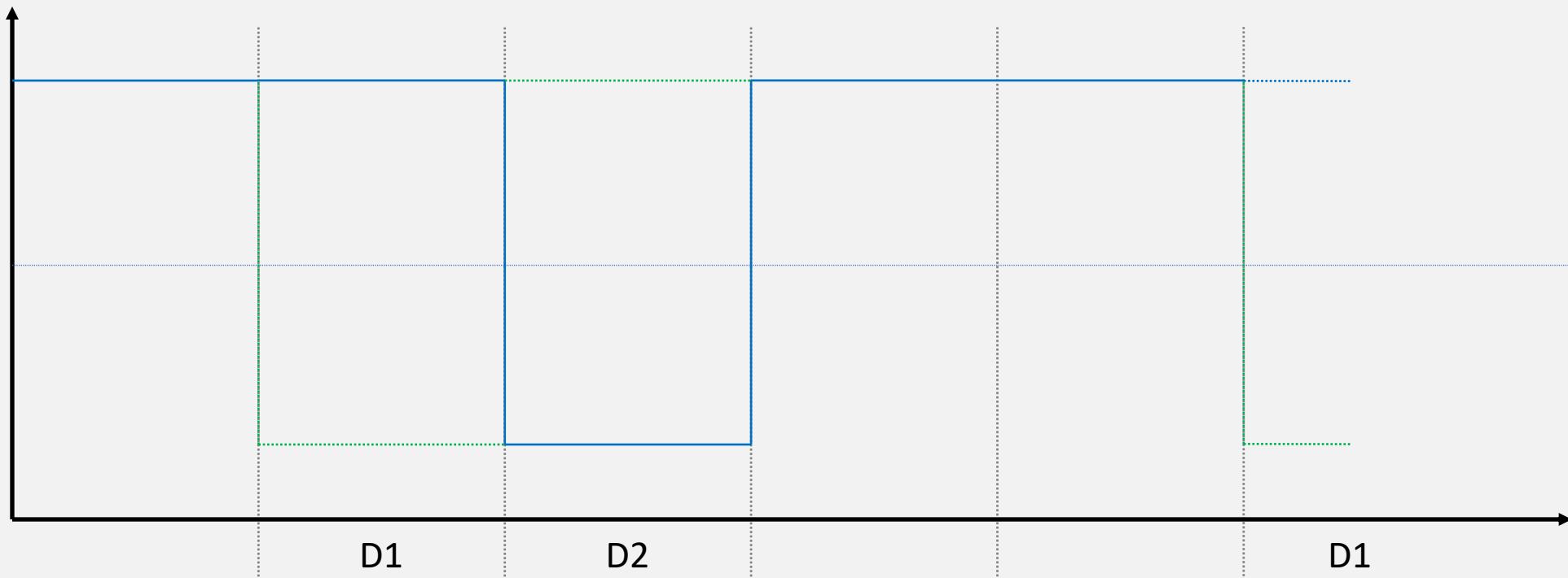
Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



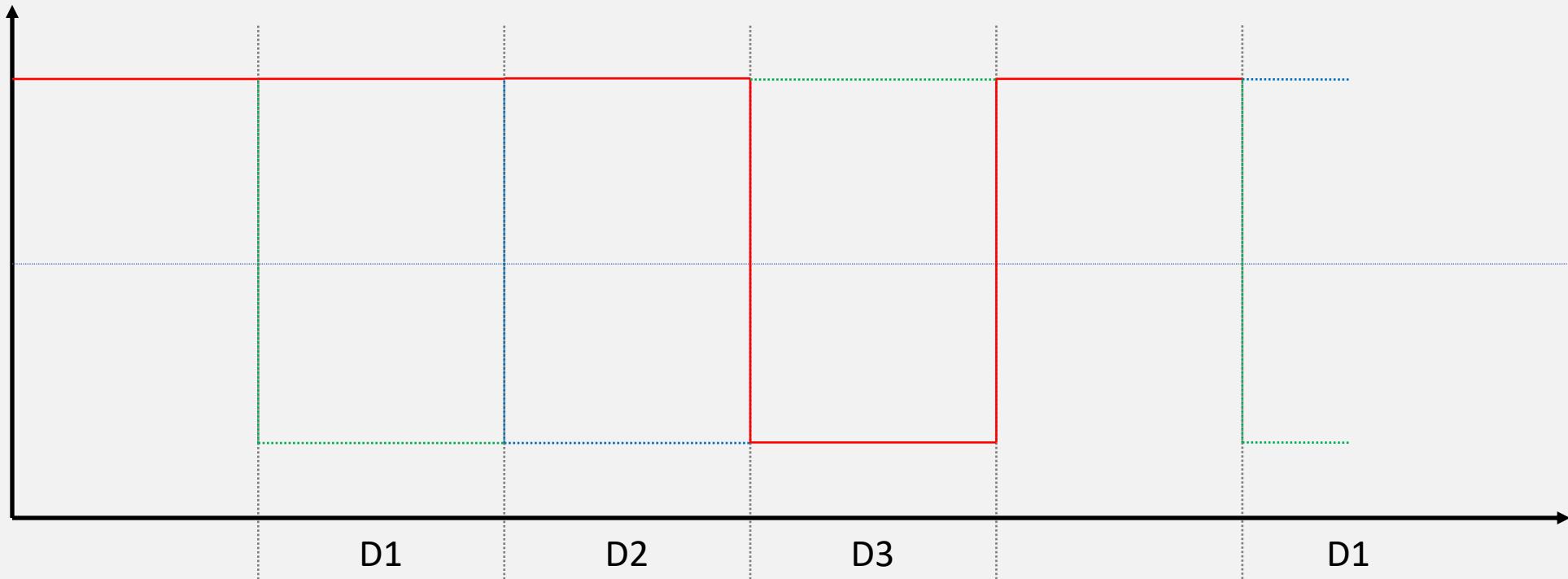
Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte :
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



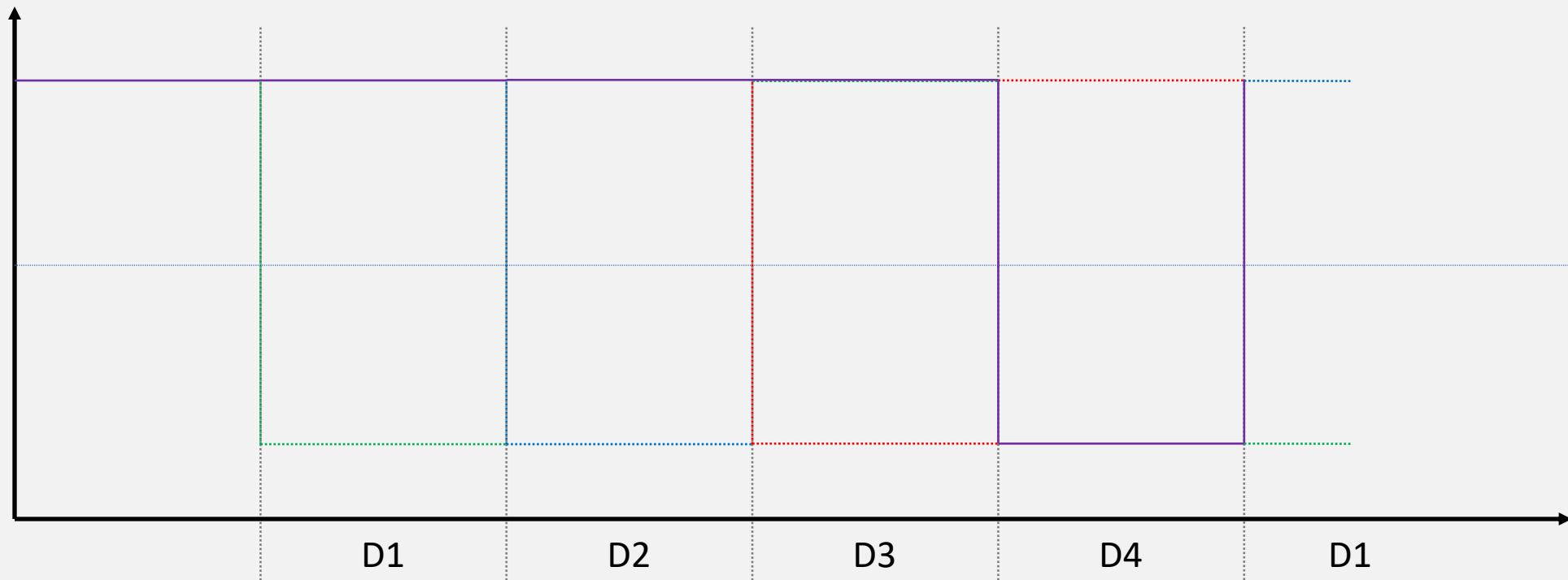
Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte :
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



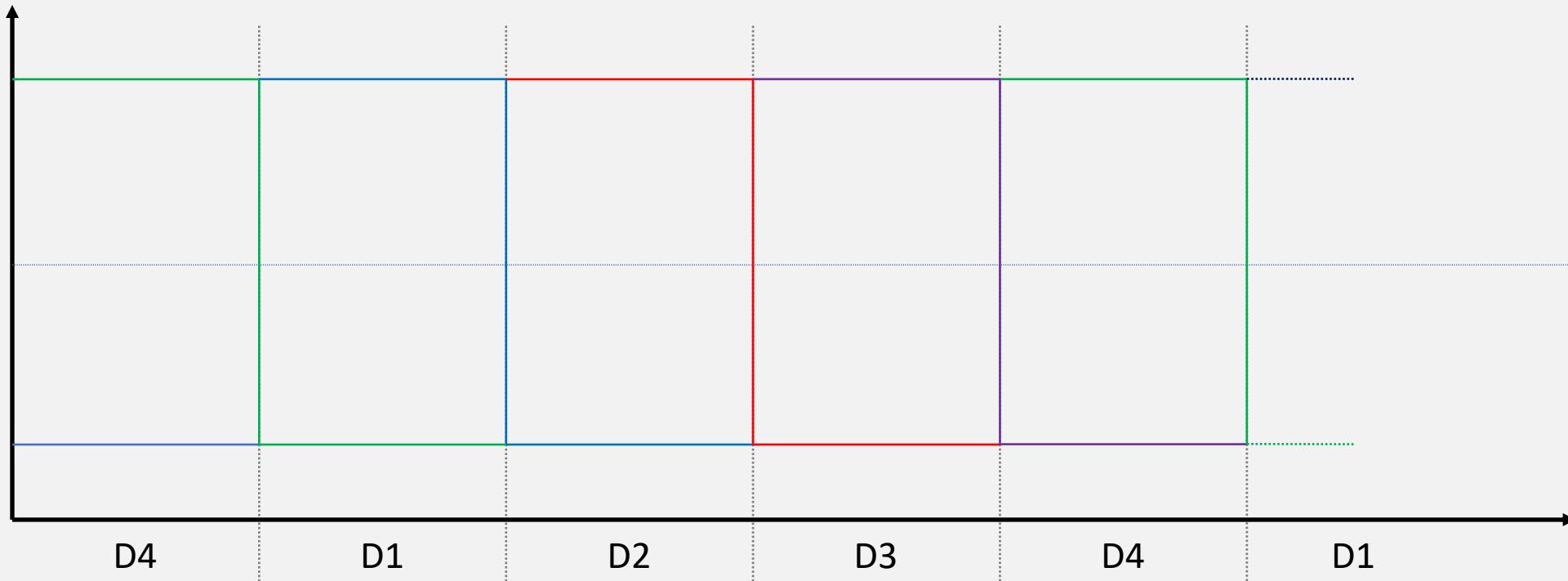
Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte :
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte :
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



Comment afficher les 4 digits - Multiplexage

- Idée :
 - Allumer tous les digits, un digit à la fois pendant une durée très courte :
 - Répéter l'opération plusieurs fois par seconde : la persistance rétinienne va donner l'illusion que le digit est allumé



Comment afficher les 4 digits - Multiplexage

- Plusieurs façons de faire pour contrôler la durée d'affichage de chaque digit :
 1. Boucle principale avec des validations de durée : utilisation de « millis » ou « micros »
 2. Utilisation d'une interruption basée sur le temps

Affichage 4 digits basé sur la boucle principale (loop)

```
class Affichage4DigitsDirect {  
public:  
    Affichage4DigitsDirect(const int& p_pinSegmentA, [...]p_pinSegmentDP,  
                           const int& p_pinD1, [...]p_pinD4,  
                           const unsigned long &p_dureeAffichageDigit);  
  
    void Afficher(const int &p_valeurDigit1, [...]p_valeurDigit4) const;  
  
    void Executer() const;  
  
private:  
    void EnvoyerValeur(const byte p_valeur) const;  
  
private:  
    int m_segmentOff; int m_segmentOn;  
    int m_digitOff; int m_digitOn;  
  
    int m_pinD[4];  
    int m_pinSegments[8];  
  
    volatile mutable byte m_cache[4];  
    volatile mutable int m_digitCourant;  
    volatile mutable unsigned long m_microsDernierChangement;  
  
    const int m_dureeAffichageDigit;  
};
```



```
Affichage4DigitsDirect adr(6, 7, 8, 9,  
                           10, 11, 12, 13,  
                           2, 3, 4, 5);  
  
void setup() {  
    adr.Afficher(1, 2, 3, 4);  
}  
  
void loop() {  
    adr.Executer();  
}
```

Affichage 4 digits basé sur la boucle principale (loop)

```
void Affichage4DigitsDirect::Executer() const {
    unsigned long horloge = micros();

    if (this->m_microsDernierChangement + this->m_dureeAffichageDigit < horloge) {
        this->m_microsDernierChangement = horloge;
        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOff);

        this->m_digitCourant = (this->m_digitCourant + 1) % 4;

        this->EnvoyerValeur(this->m_cache[this->m_digitCourant]);

        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOn);
    }
}
```

Affichage 4 digits basé sur la boucle principale (loop)

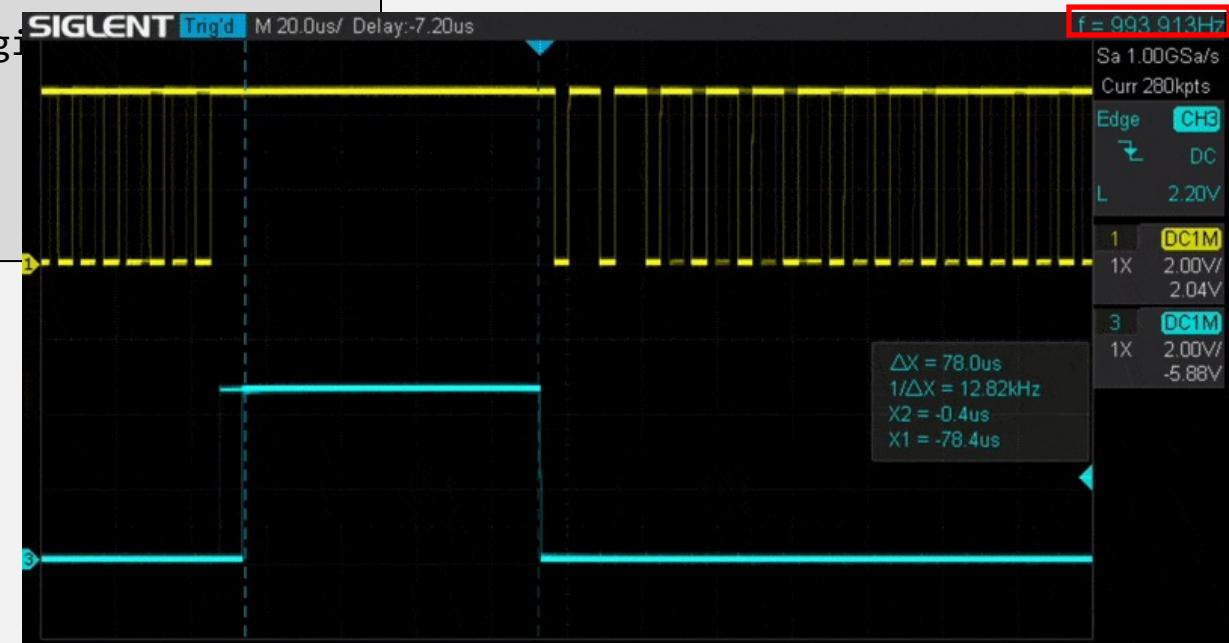
```
void Affichage4DigitsDirect::Executer() const {
    unsigned long horloge = micros();

    digitalWrite(A0, HIGH);
    if (this->m_microsDernierChangement + this->m_dureeAffichageDigit < horloge) {
        digitalWrite(A1, HIGH);
        this->m_microsDernierChangement = horloge;
        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOff);

        this->m_digitCourant = (this->m_digitCourant + 1) % 4;

        this->EnvoyerValeur(this->m_cache[this->m_digitCourant]);

        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOn);
        digitalWrite(A1, LOW);
    }
    digitalWrite(A0, LOW);
}
```



Affichage 4 digits basé sur les interruptions

```
cli();
TCCR1A = 0;
TCCR1B = 0;
TCNT1 = 0;
// doit être < 65536
// Pour 1000Hz (16*10^6) / (64 * 1000) - 1
OCR1A = 249;
// turn on CTC mode
TCCR1B |= (1 << WGM12);
// Déf CS10 et CS11 bits pour 64 prescaler
TCCR1B |= (1 << CS11) | (1 << CS10);
// Activer la comparaison pour l'interruption
TIMSK1 |= (1 << OCIE1A);
sei();
```

```
ISR(TIMER1_COMPA_vect)
{
    adr.Executer();
}
```

Affichage 4 digits basé sur les interruptions

```
void Affichage4DigitsDirect::Executer() const {
    // unsigned long horloge = micros();

    digitalWrite(A0, HIGH);
    // if (this->m_microsDernierChangement + this->m_dureeAffichageDigit < horloge) {
        digitalWrite(A1, HIGH);
        // this->m_microsDernierChangement = horloge;
        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOff);

        this->m_digitCourant = (this->m_digitCourant + 1) % 4;

        this->EnvoyerValeur(this->m_cache[this->m_digitCourant]);

        digitalWrite(this->m_pinD[this->m_digitCourant], this->m_digitOn);
        digitalWrite(A1, LOW);
    // }
    digitalWrite(A0, LOW);
}
```

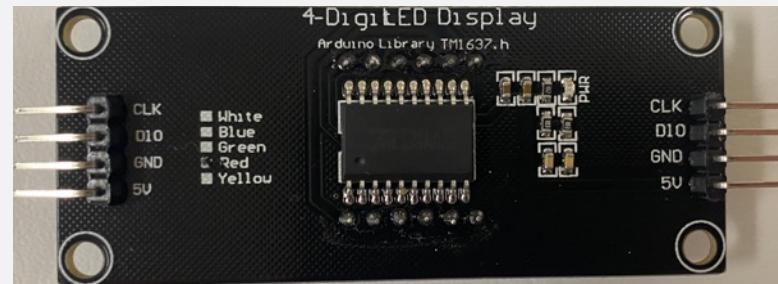
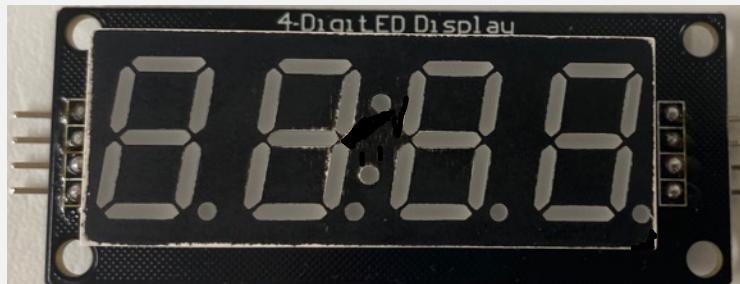


TM1637

- Le TM1637 est un circuit intégré (CI) qui permet de contrôler des DELs et des claviers
- Il contient une interface de communication
- Il est souvent utilisé pour les affichages des réveils, des micro-ondes, etc.
- Dans votre module, il gère l'affichage 4 digits que vous avez
- Il fait l'équivalent de ce que vous avez vu plus haut :
 - Gestion du temps
 - Capable d'afficher une configuration reçue en binaire

Module 4digits avec TM1637

- Vous avez un module qui utilise ce circuit intégré :



- Pour communiquer avec ce CI, vous pouvez utiliser la bibliothèque :
<https://github.com/avishorp/TM1637>
- Pour cela ajoutez là dans votre « platformio.ini » :

```
lib_deps =  
    https://github.com/avishorp/TM1637
```

Module 4digits avec TM1637 – Exemple d'affichage

```
#include <TM1637Display.h>

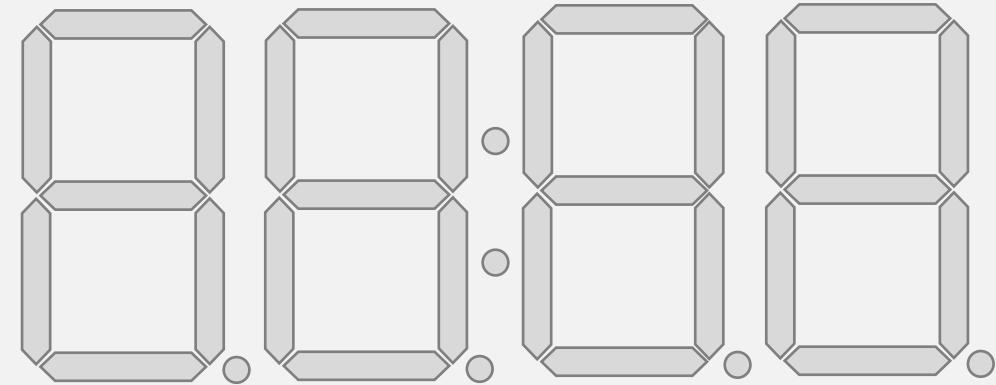
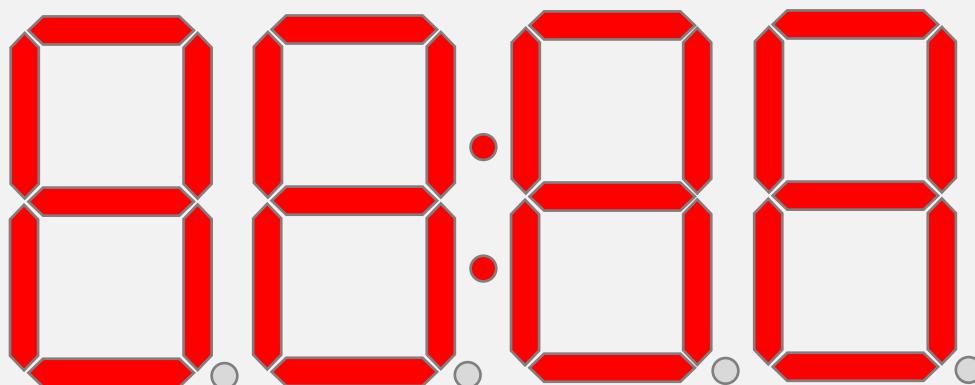
#define CLK 2
#define DIO 3

TM1637Display display = TM1637Display(CLK, DIO);

const uint8_t tousAllumes[] = {0xff, 0xff, 0xff, 0xff};
const uint8_t tousEteints[] = {0x00, 0x00, 0x00, 0x00};

display.setSegments(tousAllumes);
delay(500);

display.setSegments(tousEteints);
delay(500);
```



Module 4digits avec TM1637 – Exemple d'affichage

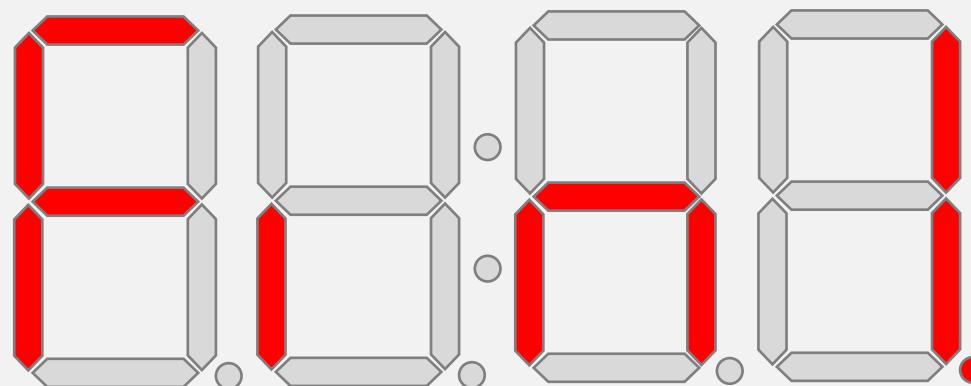
```
#include <TM1637Display.h>

#define CLK 2
#define DIO 3

TM1637Display display = TM1637Display(CLK, DIO);
```

```
const uint8_t fin[] = {
    SEG_A | SEG_E | SEG_F | SEG_G, // F
    SEG_E, // i
    SEG_C | SEG_E | SEG_G, // n
    SEG_B | SEG_C | SEG_DP // !
};
```

```
display.setSegments(fin);
```



TM1637Display.h

- La classe comprend d'autres méthodes que vous avez à explorer :

```
void setBrightness(uint8_t brightness, bool on = true);

void setSegments(const uint8_t segments[], uint8_t length = 4, uint8_t pos = 0);

void clear();

void showNumberDec(int num, bool leading_zero = false, uint8_t length = 4, uint8_t pos = 0);
void showNumberDecEx(int num, uint8_t dots = 0, bool leading_zero = false, uint8_t length = 4, uint8_t pos = 0);
void showNumberHexEx(uint16_t num, uint8_t dots = 0, bool leading_zero = false, uint8_t length = 4, uint8_t pos = 0);

static uint8_t encodeDigit(uint8_t digit);
```

Les méthodes sont documentées dans le fichier d'entête

Références

- <https://www.tinkercad.com/things/7PynAsakWfQ> : Montage et programme de test
- <https://www.tinkercad.com/things/1tZqUFVHNU9> : Montage et chiffre de 0 à 1
- <https://datasheetspdf.com/pdf-file/949036/G-NOR/GNS-5611AS/1> : Datasheet du 5611AS
- <https://www.tinkercad.com/things/7Ha0tayTvkk> : Programme de test et montage de base 4 digits
- <https://www.tinkercad.com/things/gQ3CXT26zba> : Programme pour afficher 1234

Références

- <https://www.youtube.com/watch?v=ZXcmeEMK730> : Affichage 4 digits – Cyrob partie 1
- <https://www.youtube.com/watch?v=ZXcmeEMK730> : Suite (avancée) - Cyrob partie 2
- <https://github.com/avishorp/TM1637>